René KRENN

Design and Development of a Web-Based Clinical Trial Management System

Master's Thesis

Graz University of Technology

Institute for Information Systems and Computer Media
Head: Univ.-Prof. Dipl-Ing. Dr.techn. Frank KAPPE

Supervisor: Assoc. Prof. Dr. Andreas HOLZINGER, PhD, MSc, MPh, BEng, CEng, DipEd, MBCS

Magdalensberg, May 2014

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, ______________________ ______________________

Date Signature

Eidesstattliche Erklärung[1]

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am ______________________ ______________________

Datum Unterschrift

[1]Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008; Genehmigung des Senates am 1.12.2008

Abstract

Clinical trials represent a compulsory and labour-intensive part in the course of the approval process of new diagnostic or therapeutic measures. Trials are conducted by specialized staff at trial sites such as the Clinical Research Center (CRC), a dedicated facility for Phase I and II clinical trials of the Medical University of Graz. Clinical Trial Management System (CTMS) is a term for enterprise applications, which map the particular processes of trial sites. This work introduces the Phoenix CTMS, a tailored Java web application of medium size, which covers all major processes at the CRC. Its mutli-tiered architecture is based on a Spring/Hibernate/JSF technology stack. While a model-driven approach is used for backend layers, generated artefacts are avoided for the extensive Rich Internet Application (RIA) presentation layer, which utilizes a state-of-the-art JSF component suite. Elaborated user requirements are implemented in multiple modules and range from group-ware features such as web calendars, document management and creation up to complete Electronic Data Capture (EDC) capabilities with input form composition and scripting plus editors for ad-hoc queries, allowing to formulate subject eligibility criteria. Compliance with relevant regulations (Good Clinical Practice, Data Privacy Act) is considered by an exceptional number of data security measures, including: subject de-identification and application-level encryption of data at rest, audit trail and digital signatures, configurable user privileges and host-based access restriction. It is shown how this business-critical application was designed, implemeted and rolled out at the level of commercial software products, while keeping costs minimal.

Keywords: clinical trial, medical documentation, Good Clinical Practice, Java Enterprise, model-driven development, database encryption, PrimeFaces

ÖSTAT classification: 3907, 1147, 1156

ACM classification: D.2 - Software Engineering, H.3.3 - Information Search and Retrieval, H.4.1 - Office Automation, J.3 - Life and Medical Sciences

Kurzfassung

Klinische Studien stellen einen vorgeschriebenen und arbeitsintensiven Teil im Zuge der Zulassung von neuen diagnostischen oder therapeutischen Methoden dar. Sie werden von spezialisiertem Personal in Studienzentren wie dem Clinical Research Center (CRC) durchgeführt, einer festen Einrichtung der Medizinischen Universiät Graz für Phase I und II Studien. Als Studienmanagementsysteme (Clinical Trial Management Systems) werden Unternehmensanwendungen bezeichnet, welche die besonderen Prozesse von Studienzentren abbilden. In dieser Arbeit wird das Phoenix CTMS vorgestellt, eine maßgeschneiderte Java Webanwendung, die alle wesentlichen Prozesse des CRC abdeckt. Seine mehrschichtige Architektur basiert auf einem Spring/Hibernate/JSF Technologie-Stack. Während für die Backend-Schichten ein modellgetriebener Ansatz zum Einsatz kommt, enthält die Rich Internet Application (RIA) Präsentationschicht keine generierten Artefakte und ist mittels eines aktuellen JSF Komponentenframeworks realisiert. Erarbeitete Benutzeranforderungen sind in mehreren Modulen umgesetzt und reichen von Funktionen zur Kollaboration wie Web-Kalender, Dokumentverwaltung und -Erzeugung bis hin zur voll ausgeprägten, elektronischen Datenerfassung (EDC) mit der Möglichkeit zur Erstellung und dem Scripting von Eingabemasken sowie Editoren für ad-hoc Abfragen, um Kriterien bei der Probandenauswahl zu formulieren. Konformität mit geltenden Reglementierungen (Gute Klinische Praxis, Datenschutzgesetz) wird durch eine außerordentliche Reihe von Maßnahmen zur Datensicherheit berücksichtigt, einschließlich: Geheimhaltung von direkt personenbezogenen Probandendaten und Verschlüsselung abgelegter Daten auf Applikationsebene, Revisionssicherheit und digitale Signaturen, definierbare Benutzerberechtigungen und Host-basierte Zugriffsbeschränkung. Es wird gezeigt, wie diese geschäftskritische Anwendung auf dem Niveau von kommerziellen Softwareprodukten entworfen, implementiert und ausgerollt wurde, während die Kosten minimal gehalten wurden.

Schlüsselwörter: klinische Studie, medizinische Dokumentation, Good Clinical Practice, Java Enterprise, modellgetriebene Entwicklung, Datenbank-Verschlüsselung, PrimeFaces

ÖSTAT Klassifizierung: 3907 - Medizinische Dokumentation, 1147 - Sicherheit in der Informationstechnik (IT-Sicherheit), 1156 - Web-Engineering

ACM Klassifizierung: D.2, H.3.3, H.4.1, J.3

Contents

Acronyms

ACID	Atomicity, Consistency, Isolation, Durability
ACL	Access Control List
ACM	Association for Computing Machinery
AE	Adverse Event
AES	Advanced Encryption Standard
AGES	Agentur für Gesundheit und Ernährungssicherheit (Austrian Agency for Health and Food Safety)
AJAX	Asynchronous JavaScript and XML
AJP	Apache JServ Protocol
AKH Wien	Allgemeines Krankenhaus der Stadt Wien (Vienna General Hospital)
Alpha-ID	DIMDI Alpha-ID identification code for diagnoses
AMG	Arzneimittelgesetz (Austrian Medicines Act)
ANTLR	Another Tool for Language Recognition
AOP	Aspect-Oriented Programming
API	Application Programming Interface
Apache POI	Apache "Poor Obfuscation Implementation"
BASG	Bundesamt für Sicherheit im Gesundheitswesen (Austrian Federal Office for Safety in Health Care)
BIC	Bank Identification Code
BIFF	Microsoft's Binary Interchange File Format
BLOB	Binary Large Object
BMI	Body Mass Index
CAPTCHA	Completely Automated Public Turing test to tell Computers and Humans Apart
CBC	Cipher Block Chaining mode of operation
CDISC	Clinical Data Interchange Standards Consortium
CDMS	Clinical Data Management System
CFR	Code of Federal Regulations
CHAP	Common Hybrid Agent Platform or Collective Human-Agent Paradigm
ClaML	Classification Markup Language
CLI	Command Line Interface
CLOB	Character Large Object
CMS	Content Management System
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
CRA	Clinical Research Associate
CRC	Clinical Research Center core facility (of the Center for Medical Research)
CRF	Case Report Form
CRO	Contract Research Organisation
CRUD	Create, Update and Delete record manipulations
CSS	Cascading Style Sheets
CSV	Comma- or Character-Separated Values format
CTMS	Clinical Trial Management System
CV	Trial Site Staff and Specialized Medical Personnel Curriculum Vitae
DAO	Data Access Object

DAS	Disease Activity Score
DDL	SQL Data Description Language
DI	Dependency Injection
DIMDI	Deutsches Institut für Medizinische Dokumentation und Information (German Institute of Medical Documentation and Information)
DML	SQL Data Manipulation Language
DMS	Document Management System
DoB	Date of Birth
DOM	Document Object Model
DoS	Denial-of-Service attack
dpi	Dots Per Inch
DSG	Datenschutzgesetz (Austrian Data Privacy Act)
DSL	Domain Specific Language
DST	Daylight Saving Time
DTD	Document Type Definition
DTO	Data Transfer Object
DTS	Data Transfer Service
DVR	Datenverarbeitungsregister (Austrian Data Protection Authority)
E.164	International Public Telecommunication Numbering Plan
EAV	Entity-Attribute-Value
EBM	Evidence-Based Medicine
EBS	Amazon Elastic Block Store
EC2	Amazon Elastic Compute Cloud
ECMA	Ecma (European Computer Manufacturers Association) International
EDC	Electronic Data Capture
eGFR	Estimated Glomerular Filtration Rate
EJB	Enterprise JavaBeans
EL	JSF Expression Language
EMF	Eclipse Modeling Framework
EU	European Union
EUDAMED	European Databank on Medical Devices
EudraCT	European Union Drug Regulating Authorities Clinical Trials
ext4	Fourth Extended Filesystem
FDA	U.S. Food and Drug Administration
FOP	Apache Formatting Objects Processor
FPFV	First-Patient-First-Visit milestone
FSM	Finite State Machine
GB	Gigabyte
GCP	Good Clinical Practice
GxP	collective term for Good Clinical|Manufacturing|... Practice
HbA_{1c}	glycated hemoglobin concentration
HIS	Hospital Information System
HQL	Hibernate Query Language
HSM	Hardware Security Module
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
i18n	numeronym for Internationalisation
IC	Informed Consent
ICD-10	International Classification of Diseases
ICH	International Conference on Harmonisation of Technical Requirements for Registration of Pharmaceuticals for Human Use
ID	Identifier

IDE	Integrated Development Environment
IE	Microsoft's Internet Explorer
IFCC	International Federation of Clinical Chemistry and Laboratory Medicine
INSO	Industrial Software (TUW research group)
InVO	"in" Value Object
I/O	Input/Output
IP	Internet Protocol
ISO	International Organization for Standardization
IT	Information Technology
ITU-T	International Telecommunication Union - Telecommunication standardization sector
IV	Initialisation Vector
IWRS	Interactive Web Response System
JAX-RS	Java API for RESTful Web Services
JCP	Java Community Process
JDBC	Java DataBase Connectivity
JEE	Java Platform, Enterprise Edition
JMS	Java Message Service
JPEG	Joint Photographic Experts Group file interchange format (JFIF)
JR Health	JOANNEUM RESEARCH HEALTH – Institut für Biomedizin und Gesundheitswissenschaften (Institute for Biomedicine and Health Sciences)
Js	JavaScript
JSF	JavaServer Faces
JSON	JavaScript Object Notation
JSP	JavaServer Pages
JSR	Java Specification Request
JVM	Java Virtual Machine
l10n	numeronym for Localisation
LDAP	Lightweight Directory Access Protocol
LIMS	Laboratory Information Management System
LLC	Limited Liability Company
LOC	lines of code
MB	Megabyte
MD5	Message-Digest Algorithm 5
MDA	Model-Driven Architecture
MDD	Model-Driven Development
MDR	NetBeans Metadata Repository
MDRD	Modification of Diet in Renal Disease formula
MIME-type	(Multipurpose Internet Mail Extensions) content type (internet media type)
MPG	Medizinproduktegesetz (Austrian Medical Devices Act)
mTAN	Mobile Transaction Authentication Number
MUG	Medizinische Universität Graz (Medical University of Graz)
MUW	Medizinische Universität Wien (Medical University of Vienna)
MVC	Model-View-Controller architectural pattern
MVCC	Multiversion Concurrency Control
NFR	Non-Functional Requirement
NGSP	U.S. National Glycohemoglobin Standardization Program
NoSQL	Not Only Structured Query Language
OCL	Object Constraint Language
ODM	Operational Data Model
OPE	Order-Preserving (symmetric) Encryption
OPS	Operationen- und Prozedurenschlüssel (german procedure classification)
ORM	Object-Relational Mapping/Mapper

OS	Operating System
ÖSTAT	Bundesanstalt Statistik Österreich (STATISTICS AUSTRIA)
OTS	Off-The-Shelf
OutVO	"out" Value Object
PBE	Password-Based Encryption
PBX	Private Branch Exchange
PDF	Adobe System's Portable Document Format
PHP	PHP: Hypertext Preprocessor
PI	Principal Investigator
PII	Personally Identifiable Information
PIM	Platform Independent Model
PiP	Point-in-Polygon test algorithm
PKCS	Public-Key Cryptography Standard
POJO	Plain Old Java Object
PPR	Partial Page Refresh
PRG	Post/Redirect/Get HTTP request pattern
PRNG	Pseudo-Random Number Generator
PRO	Patient Reported/Recorded Outcome
PSF	Pagination, Sorting and Filtering (of tabular data)
PSM	Platform Specific Model
PVR	Partial View processing and Rendering
QM	Quality Management
RAC	Oracle Real Application Clusters
RAD	Rapid Application Development
RAID	Redundant Array of Independent Disks
RAM	Random Access Memory
RBAC	Role-Based Access Control
RCT	Randomized Controlled Trial
RDA	Remote Data Aquisition
RDBMS	Relational Database Management System
REDCap	Research Electronic Data Capture
RegEx	Regular Expression
REST	Representational State Transfer
RFC	IETF Request For Comments publication
RHEL	Red Hat Enterprise Linux
RIA	Rich Internet Application
RMI	Remote Method Invocation
RPN	Reverse Polish Notation (postfix notation)
RSA	asymmetric cryptography algorithm by Ron Rivest, Adi Shamir and Leonard Adleman
SaaS	Software as a Service
SAE	Serious Adverse Event
SAS	Statistical Analysis Systems
SAX	Simple API for XML
SDF	Source Data Form
SDLC	Systems Development Life Cycle
SHA-1	Secure Hash Algorithm 1
SLES	SUSE Linux Enterprise Server
SMS	Short Message (Service)
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
SoC	System-on-a-Chip

SOP	Standard Operating Procedure
SOUP	Software Of Uncertain Provenance
SPA	Single-Page Interface/Application
SPICS	Secure Platform for Integrating Clinical Services
SPSS	IBM SPSS (Statistical Package for the Social Sciences) Statistics
SQL	Structured Query Language
SSA	Signature Scheme with Appendix
SSL	Secure Sockets Layer
SUSAR	Suspected Serious Adverse Reaction
SVG	Scalable Vector Graphics
TCP	Transmission Control Protocol
TTF	TrueType Font
UAT	User Acceptance Test
UI	User Interface
UML	Unified Modelling Language
URL	Uniform Resource Locator
US	United States of America
UTF	Unicode Transformation Format
UUID	Universally Unique Identifier
VAS	Visual Analog Scale
VAT	Value Added Tax
VO	Value Object
VTL	Velocity Template Language
WebDAV	Web-based Distributed Authoring and Versioning standard
W.H.A.T	SPICS Wound Healing Analyzing Tool
WYSIWYG	What You See Is What You Get
XHR	XMLHttpRequest
XHTML	eXtensible HyperText Markup Language
XMI	XML Metadata Interchange
XML	eXtensible Markup Language
XSL-FO	eXtensible Stylesheet Language - Formatting Objects
YUI	Yahoo User Interface Library
ZID	Zentraler Informatikdienst (information technology services)
ZIP code	postal code (Zone Improvement Plan)
ZMF	Zentrum für Medizinische Grundlagenforschung (Center for Medical Research)

1. Introduction and Motivation

Evidence-Based Medicine (EBM) claims the empirical proof of any diagnostic or therapeutic measure applied to patients. A significant evidence can be achieved with the systematic approach of a study. To evaluate efficacy and tolerance, the first application of a promising intervention should take place in the experimental setup of a trial (Schumacher and Schulgen, 2002). As a consequence, regulations obligate clinical trials as part of the approval process of new drugs, medical procedures or devices.

Clinical trials are conducted by specialized staff at trial sites, which represent departments of a commercial or academic institution. MUG identified clinical trials as an essential business segment and considered a dedicated trial site (CRC) with the establishment of the Zentrum für Medizinische Grundlagenforschung (ZMF) in 2004. Educational concerns are taken into account by providing an opportunity for student workers to get in contact with clinical research. The (ZMF-) Clinical Research Center (CRC) is a growing, well-equipped trial site with capacities for phase I and II clinical trials and a focus on the medical domain of diabetology. Since its launch, the annual number of ongoing projects increased from 7 (2004) to 21 (2012), with a ratio of contract research to public funding of 60:40 (Organisationseinheit für Forschungsinfrastruktur (O-FIS), 2014, p. 21). Further development aims at gaining international recognition of the CRC (Medical University of Graz, 2012b, p. 37). Competitiveness with Contract Research Organisations (CROs) and offered services attractive to renowned trial sponsors are essential prerequisites. This implies to adhere to standards, as claimed by national and international regulations. Therefore, a quality offensive was started in 2011 with the International Organization for Standardization (ISO) 9001:2008 certification of CRC processes.[1] Finally, the strategy comprises the introduction of a comprehensive database system.

The database application can ensure consistent data, procedures and a detailed documentation of processes. When looking forward, the electronic mapping of relevant processes will enable to handle an increasing number of extensive (commercial) trials while reducing costs (Eisenstein et al., 2008, p. 83). To consider requirements in detail but keep investment costs low, stakeholders at the CRC decided against a Off-The-Shelf (OTS) software product. Instead, a tailored solution for long-term use should be developed "in-house" and deployed in the course of a student master thesis. The level of detail and implementation quality has to match established OTS software, since an incomplete, erroneous or prototypical result will impede productive and business-critical operation. The expected volume of this medium-sized software project[2] to build will require hands-on experience and endurance to reach goals. This work gives detailed insight into all development stages of the Phoenix CTMS to show how this challenge was successfully mastered.

[1]The certification project was named "Phoenix" internally.
[2]100k-1M lines of code (LOC)

2. Background

2.1. Clinical Trials

2.1.1. Study Types and Attributes

Decades of pre-clinical and clinical research precede the introduction of a particular new drug or method of treatment. During this period, it is inquired in a sequence of investigations, which can be classified according to common attributes and regulations.

Basic and pre-clinical research Research studies without human subjects take place initially when identifying mechanisms or developing techniques and substances:

Theoretical: modeling and simulation
Applied: in-vitro, ex-vivo and in-vivo experimental research

Diagnostics accuracy studies are in-vitro studies related to diagnostic methods (Smidt, 2008). Animal testing is an in-vivo pre-clinical investigation focused on the evaluation of toxicity and efficacy.

Interventional clinical research Human probands are to be exposed to the *investigational product* during interventional studies, which therefore imply concerns on health, safety and ethics. In Austria, (*Medizinproduktegesetz (MPG)* 1996) and (*Arzneimittelgesetz (AMG)* 1983) regulations apply to interventional trials, depending on the type of the investigated product. Both obligate the aforegoing approval by ethical review committees. (*Arzneimittelbetriebsordnung 2009* 2008) obligates International Conference on Harmonisation (ICH) Good Clinical Practice (GCP) (*ICH E6 (R1) Guideline for Good Clinical Practice: Consolidated Guidance* 1996) adherence as well as requirements regarding regulated environments, which also covers *computerized systems* (*Good Manufacturing Practice - Medicinal Products for Human and Veterinary Use Annex 11: Computerised Systems* 2011).

The specific term "clinical trial" refers to in-vivo interventional studies on human subjects, which are mandatory in the course of the approval process of a new drug (table 2.1).

clinical trial type	**research goal**	**magnitude of subjects**
phase I	tolerability	$10^1 - 10^2$
phase II	efficacy	$< 10^2$
phase III	confirmation	$10^2 - 10^4$
phase IV	pharmacovigilance	$> 10^3$

Table 2.1.: Categories of clinical trials correspond to phases of the drug approval process, adopted from (Wikipedia Contributors, 2014a).

Results of pre-clinical research are a mandatory precondition for clinical trials. While the handling of subject data is basically regulated by (*Datenschutzgesetz 2000 (DSG 2000)* 1999), AMG and MPG introduce additional restrictions and liabilities regarding data transfer and retention.

Non-interventional research The generic term "study" often refers to non-interventional studies. In contrast to clinical trials, no investigational product is applied in experimental setups. Instead, data sets are created by pure observation or data-mined if already existent. Typically, observational studies cover a long time period (prospective or retrospective) or involve a large number of subjects as representative subset of a target population.

cross-sectional studies: Cross-sectional studies are characterized by data collection from a population at one specific point in time.
longitudinal studies: Longitudinal studies are characterized by repeated observations of the same variables over long periods of time.
cohort studies: Cohort studies are longitudinal studies of a population cohort. A (population) cohort is a set of subjects sharing predicates (e.g. smoking habit) or events (e.g. birth) during a given period of time.
case-control studies: Case-control studies compare outcomes of two populations (exposed/not exposed).

(Wikipedia Contributors, 2014b)

Trial design Clinical trial protocols propose a *trial design*, which can be characterized by general attributes below:

controlled: In a controlled trial, subjects are separated into intervention groups (verum groups) and control groups (placebo or reference groups).

parallel trial: A subject belongs to the same group during the whole trial.
crossover trial: A *study arm* defines an sequence of alternating group allocation over *stages* of the trial (Sibbald and Roberts, 1998).

randomized: Subjects are randomly assigned to either a verum group or control group in the case of parallel trials, or a study arm in the case of crossover trials.
blinded: The randomization result is kept secret from:

- neither subjects nor investigators (*open*)
- rater[1] only (*rater blinded*) (Gaus and Chase, 2008, p. 36)
- subjects only (*blinded*)
- both subjects and investigators (*double-blinded*)

Control groups allow to formulate clear conditions to falsify or show proof of the trial hypothesis, while randomization and blinding will reduce systematic errors (*bias*). Randomized Controlled Trials (RCTs) yield the most accurate results (D. J. Torgerson and C. J. Torgerson, 2008, p. 1) and therefore represent the *gold standard*.

Contracting Aside hybrid forms, the trial sponsoring type falls into two basic categories:

commercial: In commercial trials, the sponsor is a (pharmaceutical) company, e.g. in the act of introducing a new drug.
academic: The sponsor of academic trials is an academic institution or non-profit organization. If sponsor and investigator represent the same corporate body, the trial is referred to as *investigator-initiated*.

Applicable regulations are almost identical for both commercial and academic trials (Gaus and Chase, 2008, p. 12).

[1]e.g. sponsor or persons involved with data analysis

Scale A trial is designed to be either carried out by a single trial site (mono-centre trial) or multiple sites (multi-centre trial), essentially depending on the number of subjects planned and the statistical diversity required (D. Wang and Bakhai, 2006, p. 153). Multi-centre trials with sites located in different countries will face regional regulations of each country (investigator regulations). Likewise, Food and Drug Administration (FDA) regulations of an US sponsor or coordinating center (sponsor regulations) may also apply to a local site such as the CRC.

2.1.2. Nomenclature

This section gives a short overview of the nomenclature, actors, processes, events and documents involved with clinical trials.

Trial registration and approval When initiating a clinical trial, multiple registrations and approvals are mandatory. A registration at an authority is usually bound to deadlines and yields some *file number* returned by the authority. A remarkable written permit is the *ethics committee vote*, which clarifies ethic concerns and commensurability and has a limited validity period.

At the end of the trial, the sponsor has to provide a *final study report* to authorities. *Trial registries* such as clinicaltrials.gov allow searching for ongoing and closed trials.

Trial site The local organization or department, whose personnel effectively performs investigations with subjects will be denoted as *trial site* or center. A trial site can be a department of a pharmaceutical company, clinic, medical school or an (interdisciplinary) *CRO*. Trial site staff comprises of employees and contractors with different professions, who all represent the user audience of the Phoenix CTMS:

- physicians
- trial coordinators
- trial monitors and Quality Management (QM) managers
- study nurses
- student employees
- scientists and engineers for the investigational product
- data managers
- data analysts (biostatisticians)

For the evidence of qualification, GCP obligates to provide *Curricula Vitae (CV's)* and relevant documentation upon request by the sponsor, the ethical review committees, or the regulatory authorities (*ICH E6 (R1) Guideline for Good Clinical Practice: Consolidated Guidance* 1996, section 4.1.1).

Sponsor *GCP defines the sponsor as "an individual, company, institution, or organization that takes responsibility for the initiation, management, and/or financing of clinical research."* (*ICH E6 (R1) Guideline for Good Clinical Practice: Consolidated Guidance* 1996, section 1.53)

Investigator According to the AMG, the investigator is a *physician* (or dentist) in charge of carrying out a clinical trial's protocol. In the case of a trial site team, the responsible investigator is the lead investigator[2].

Trial protocol A *trial protocol* is a document provided by the sponsor, which defines the trial's procedures in detail:

[2]also known as *Principal Investigator (PI)*

- trial design
- expected results and their interpretation (hypothesis)
- subject eligibility criteria
- Adverse Event (AE) policy
- visit schedule

The visit schedule defines the chronological sequence of investigation visits. Typically, detailed work instructions and a Case Report Form (CRF) are attached for each visit.

Eligibility criteria Eligibility criteria comprises *inclusion and exclusion criteria* for participants of a concrete trial, which are examined during the initial visit (*screening visit*). *"[Inclusion and exclusion Criteria] are based on such factors as age, gender, the type and stage of a disease, previous treatment history, and other medical conditions."* (Wikipedia Contributors, 2013)

Recruitment The recruitment is the systematic establishment of a candidate stock with a volume required for the trial. When registering a candidate, storing Personally Identifiable Information (PII) such as contact details will require a *data privacy consent* signed by the candidate. Recruitment can be started for a particular trial by means of dedicated contacting or public appeals, such as advertisement campaigns. Since sponsors may choose a contracting site according to the volume of their established registry of readily available candidates, the CRO's recruitment strategy may rely on a permanent recruitment process.

Candidate, Subject A *candidate* is a person interested in participating trials, who is registered during recruitment. If eligibility and regulatory criteria are met, the candidate is enrolled for a particular trial. At this point, the person is denoted as *subject* or proband.[3]

Informed Consent After education about the trial's process, details and risks and the proband's rights, a candidate approves his knowledge and agreement by signing the Informed Consent (IC). A signed IC is a precondition for participation, usually expressed as a particular inclusion criterion. The IC may also stipulate a financial compensation for participation and expenditures, if allowed by regional regulations.[4]

SOP Standard Operating Procedures (SOPs) are written work orders for general tasks, device usage or established procedures at the trial site and represent an essential QM instrument. Specific SOPs may be created for a particular trial or even provided as part of the trial protocol.

Visits Trial site staff performs actual interventions and examinations during a sequence of *investigation visits* (e.g. screening visit, treatment or dosing visits), under surveillance of the investigator.

Pseudonymisation PII of subjects is visible to trial site staff only. It must not be exported to an external party (e.g. sponsor) and will therefore not be part of CRFs. Instead, a pseudonym (*subject ID*) is generated and assigned for subsequent reference when enrolling a candidate. The subject ID is an alphanumeric string, which can be generated randomly or according to a scheme[5] The subject ID is valid for the particular trial only and must not be confused with a permanent identifier such as the patient ID of a Hospital Information System (HIS).

[3]The terms *candidate*, *subject* and *proband* are used synonymously throughout this work.

[4]For example, this is currently restricted in Italy.

[5]A scheme may include *trailing* initials, coded gender or age group, a sequence number, ...

Source data, CRF *Source data* and completed *CRFs* represent the valuable outcome of investigations. Source data like questionnaire answers, medical assessments or lab reports is collected according to visit SOPs. Finally, the source data is aggregated and entered into CRF documents, attached to the trial protocol. *Coding* is the task of assessing or normalizing data according to scores or schemes (e.g. ICD-10) in the course of infilling CRFs. Both source data and CRFs are prepared for delivery when completed. In terms of a CDMS, the CRF data entry will be locked and exported at this point (*database release*). The sponsor receives data in raw paper and/or electronic format for subsequent review at *reading centers* and analysis. The workflow for tracking caveats and inconsistencies of CRFs follows a structured resolution workflow, which is sometimes referred to as *query management*.

Audit Audits are performed by internal *monitor* personnel (internal audits) as well as external *Clinical Research Associates (CRAs)* (external audits) or commissioners (authoritative audits[6]). External and authoritative audits can be scheduled by a sponsor or regulator on a regular basis or without announcement. An audit should confirm the proper adherence to the trial protocol and regulations such as GCP. Violations identified during an authorative audit have to be reported (*finding*) and may inflict a penalty.

Drop-out During ongoing trials, the event of a drop-out means the exclusion of a participant from further investigations and captured data. Aside the proband's decision to leave, a proband has to be excluded if *drop-out criteria* apply, as specified in the trial protocol. Aside frequent drop-out criteria such as insufficient or missed data or the subject's lack of cooperation, medical factors such as pregnancy can be considered in inclusion/exclusion criteria as well.

AE, SAE, SUSAR An *AE* is defined as encountered abnormality of any kind regarding the condition of a proband that is not anticipated or excluded in the trial protocol, whether associated with the investigational product or not. CRFs will include sections for logging AEs.

Serious Adverse Events (SAEs) are AEs that yield death, hospital stay, birth defects or disablements. The investigator has to notify the sponsor immediately, which in turn is liable to inform authorities.

A *Suspected Serious Adverse Reaction (SUSAR)* is a SAE, which is possibly related with the investigational product. SUSAR reports require to reveal randomisation results (*unblinding*).

(Gaus and Chase, 2008, pp. 61-62)

2.2. Related Work

Software systems designated to support conducting clinical trials represent RDBMS-based applications for mapping workflows and managing data relevant to the trial site (the "customer"). When deciding to move from spreadsheet- or paper-based workflows to a database system, available options can be considered as related work in the broader sense.

[6] also known as *inspection*

2.2.1. Standard and Custom Software

Standard software Aside office software packages, a large number of available Commercial Off-The-Shelf (COTS) solutions by numerous players (figure 2.1) evolved due to the diversity of processes involved with clinical trials. As stated in (marketsandmarkets.com, 2012), the global market is lead by the US and was valued at $576.22 million in 2010.

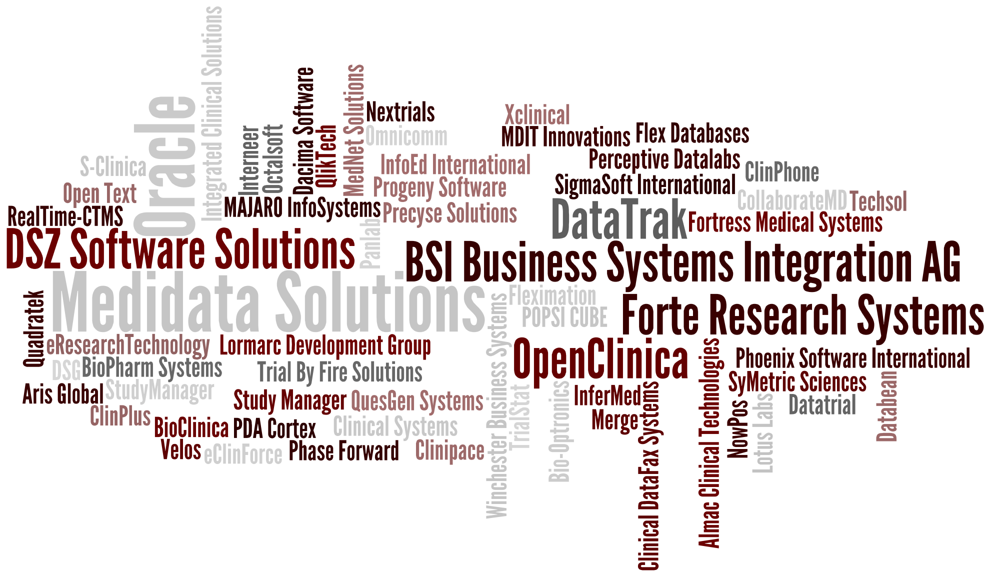


Figure 2.1.: Clinical trial software vendors gathered from eclinicalhealth.com and capterra.com.

The acquisition of a software product represents a major investment (Zubatch, 2006) and leads to some degree of a vendor lock-in.[7] Standardized data interchange formats such as proposed by (CDISC, 2014) were a fundamental step towards basic compatibility, but will not provide universal interoperability. Business models of vendors include training programs and customization such as enterprise integration services, e.g. interface development. Free software solutions came into existence and gain popularity due to reduced costs. Successful free software will typically be be backed by a company that offers commercial support options and coordinates the structured development claimed by regulations for clinical trial software. COTS and established free software solutions may be summarized by the term OTS or *standard software*.

OTS systems focus on key aspects that will satisfy either a large or very specific group of customers. With the advent of fast and widely accessible internet connectivity, the trend towards web-based systems emerged. They turned out as particular useful in the context of multi-center trials involving multiple distant trial sites. Many vendors focus on the business of multi-center trial projects to satisfy customers with increased demand on software assistance and higher budgets.

A growing number of vendors offer rental models for their software solutions (SaaS). However, customer's concerns regarding subject data privacy and intellectual property are justified when storing information externally in the cloud or on dedicated servers. With the traditional, but more costly on-site deployment, the system's servers are installed and maintained by a customer locally.

[7]This was confirmed by a requested offer for a CTMS COTS product purchase license, rated at $50k (10 users) for universities/not-for-profit organisations. Maintenance is available separately, reated at $8k/year.

Custom software In the occasional case available OTS systems are inappropriate, the customer can opt for a tailored solution. In this case, the software should be adapted instead of the customer and can be developed with respect to applicable regulations, specific requirements of the trial site or even a particular trial. The range of custom software starts from spreadsheets containing custom forms and macros, which can be linked to separate RDBMS data sources. For more complex requirements such as editing a data source's content or basic multi-user access, databases created with Rapid Application Development (RAD) tools such as Microsoft Access can be a suitable approach. The upper end of custom software includes the combination and customization of available OTS solutions or even developing an integrated, full-stack enterprise application from scratch, such as the Phoenix CTMS.

Irrespective of standard or custom software, the customer is ultimately responsible for the *validation* of deployed "computerized systems"[8] utilized for clinical trials, *"e.g. data/information systems relating to coding, randomisation, distribution, product recalls, clinical measures, patient records, donation sources, laboratory data, etc."* (PIC/S, 2007, paragraph 23.7). In Austria, the obligatory validation is inspected by the Bundesamt für Sicherheit im Gesundheitswesen (BASG) (BASG/AGES, 2012) and essentially includes:

1. assessment of risks exposed by the system (subject safety, data quality)
2. Good Clinical|Manufacturing|... Practice (GxP) conformance
3. validation results (e.g. test reports, error logs)
4. software development process and QM
5. requirement coverage

Presumably, OTS solutions that successfully passed validation in various deployments before will mean reduced expenses when validating the customer's concrete deployment. Since customization of OTS is considered as custom software coming along with more complex validation procedures, there is a thin line between configuration and customization.[9] The customer has to rely on assistance and documentation in the case of an external supplier. The term Software Of Uncertain Provenance (SOUP) is used if this support is not available (e.g. due to closed source), which could be another reason to go with custom software development. (Redmill, 2001, p. 121)

[8]hardware, software and operational environment

[9]The following question gives an example of recurring discussions: if data entered by an end user will be executed by the system, should this be considered as a configuration aspect or application logic customization?

2.2.2. CDMS and CTMS

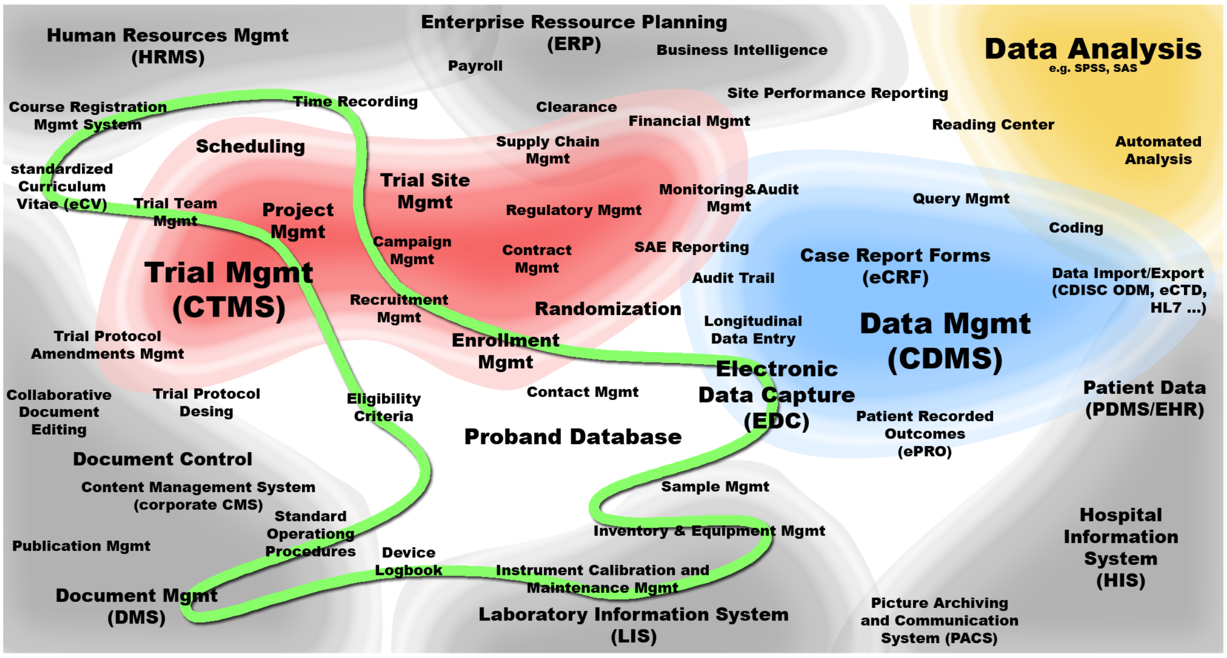


Figure 2.2.: This map shows identified software systems and their related features in the scope of clinical trials and research studies. Features enclosed by the green line are covered by the Phoenix CTMS.

A plethora of software features can be identified which address processes in the course of clinical trials. According to the size of the institution that hosts the trial site, the functionality is provided by a number of involved systems forming a system landscape (figure 2.2). Two terms for types of vertical applications can be considered in the immediate context of clinical trial software:

CDMS A Clinical Data Management System (CDMS) is designed to support the most essential task of populating CRFs by means of Electronic Data Capture (EDC). An EDC software UI displays input forms to site staff (end users), allowing them to enter clinical data per subject. Since data is available in electronic form (eCRF) suitable for subsequent processing (data analysis), there is no need of intermediate paper-based Source Data Forms (SDFs).[10] Similar to eCRF, electronic Patient Reported Outcome (PRO) capabilities support trial designs which require probands to enter data themselves.

CTMS Clinical Trial Management System (CTMS) can be considered as a broader term for database systems covering processes immanent to the planning and management of studies and trial sites. The assessed risk of these processes tends to be lower compared to CDMSs. In crontrast to CDMSs with the clear focus on EDC, the wider scope of supported processes leads to a variety of available CTMS solutions and custom software development. A CTMS may interface with a separate CDMS instance or even incorporate CDMS functionality.

For workflows including subject interaction such as initial contacting, a trial site has to store subject PII at some point. If supported by a system, retention of subjects across trials requires to maintain a proband database. This again presumes storing PII such as contact details instead of pseudonymous data only and will effectively face data privacy regulations. CDMS will therefore avoid storing PII at all. A

[10]In this work, the term EDC is not restricted to the context of CRF data.

solution to reduce this limitation is to link to fully qualified patient records existing in a database of an interconnected system, such as an HIS.

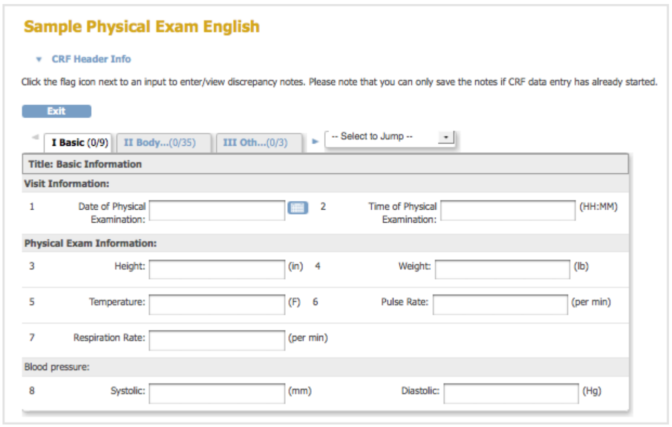
2.2.3. Examples of Existing Systems

This sections outlines concrete OTS and custom software utilized by Austrian medical schools. In addition, an example of a commercial product (SimpleCTMS) is juxtaposed.

OpenClinica OpenClinica is the most prominent open-source CDMS, developed and maintained by OpenClinica, LLC since 2005 (OpenClinica, LLC, 2014). Aside the free community edition, OpenClinica LLC provides commercial support and an enterprise edition with extended features. OpenClinica features a validated software lifecycle and implements data exports compliant with FDA Code of Federal Regulations (CFR) 21 Part 11 regulations, including:

- digital signatures for database records
- audit trail

Aside lower costs, the hassle-free validation of environments that utilize OpenClinica and the built-in Clinical Data Interchange Standards Consortium (CDISC) Operational Data Model (ODM) import/export capabilities are reasons for the widespread acceptance. OpenClinica claims a strongly growing user base of currently 18000 users around the world, including CRC staff. The CRC outsources data management tasks such as CDMS setup and configuration for particular trials to JOANNEUM RESEARCH HEALTH – Institut für Biomedizin und Gesundheitswissenschaften (JR Health). JR Health moved from an CDMS infrastructure developed in-house (Beck et al., 2006) to OpenClinica.


Sample Physical Exam English

▾ CRF Header Info

Click the flag icon next to an input to enter/view discrepancy notes. Please note that you can only save the notes if CRF data entry has already started.

Title: Basic Information

Visit Information:

1 Date of Physical Examination: 2 Time of Physical Examination: (HH:MM)

Physical Exam Information:

3 Height: (in) 4 Weight: (lb)

5 Temperature: (F) 6 Pulse Rate: (per min)

7 Respiration Rate: (per min)

Blood pressure:

8 Systolic: (mm) Diastolic: (Hg)

Figure 2.3.: OpenClinica screens, taken from docs.openclinica.com.

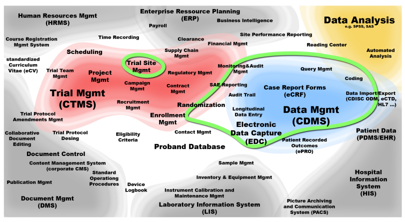
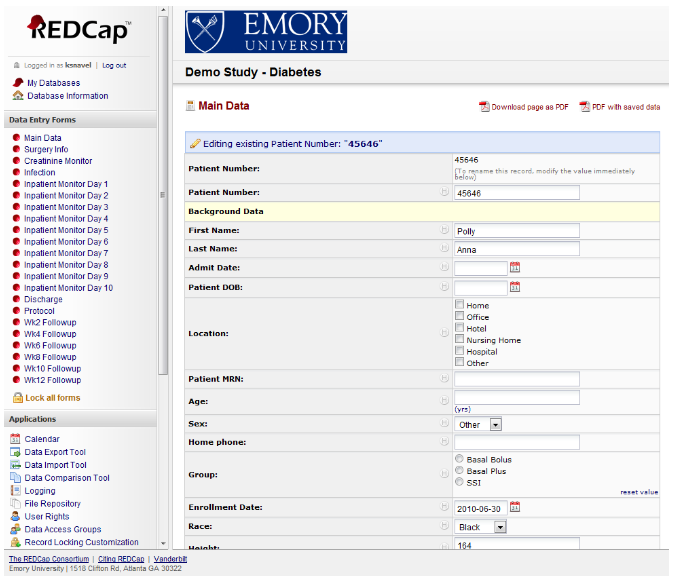


Figure 2.4.: OpenClinica features.

REDCap Research Electronic Data Capture (REDCap) provides flexible EDC capabilities, aiming at (non-interventional) research studies and clinical trials (Harris et al., 2009). It is developed by Vanderbilt University since 2004 and is widely used in the academic field with a claimed user base of 132000, including MUG institutes (e.g. Oswald et al., 2013). REDCap is not open-source software but provided at no charge to "REDCap Consortium" member institutions for non-commercial use.

Built on the lightweight PHP: Hypertext Preprocessor (PHP) stack, REDCap will be easier to deploy and maintain compared to Java applications. Another success factor is REDCap's rapid and intuitive workflow to set up a study project. In contrast to OpenClinica, it still lacks CDISC ODM export capabilities which can be a drawback since sponsors frequently request this data format for deliverables from their commissioned CROs. It is nevertheless capable of producing data output as claimed by FDA Part 11.



Demo Study - Diabetes

Main Data

Editing existing Patient Number: "**45646**"

Patient Number:	45646 (To rename this record, modify the value immediately below)
Patient Number:	45646
Background Data	
First Name:	Polly
Last Name:	Anna
Admit Date:	
Patient DOB:	
Location:	☐ Home ☐ Office ☐ Hotel ☐ Nursing Home ☐ Hospital ☐ Other
Patient MRN:	
Age:	(yrs)
Sex:	Other
Home phone:	
Group:	○ Basal Bolus ○ Basal Plus ○ SSI
Enrollment Date:	2010-06-30
Race:	Black
Height:	164

Figure 2.5.: REDCap screens taken from (Claypool, Szabrak, and Weaver, 2011).

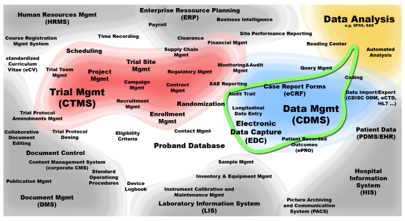
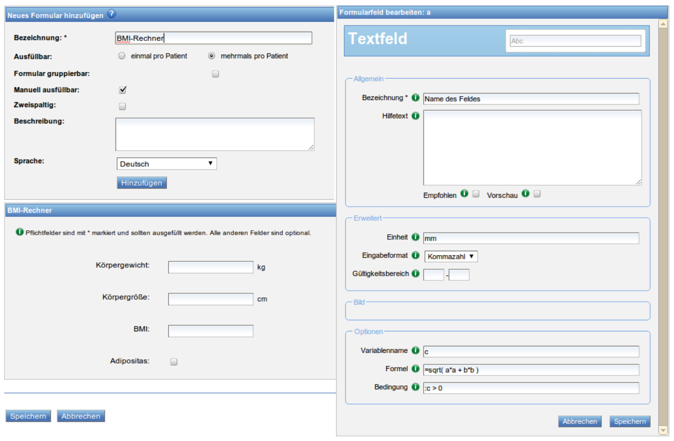


Figure 2.6.: REDCap features.

SPICS The Secure Platform for Integrating Clinical Services (SPICS) is a EDC platform for multiple applications such as clinical studies and medical case documentation. It is developed by Industrial Software (INSO) in collaboration with medical experts and available with extensions for specific medical sectors (product lines SPICSsoul, SPICSvasc, SPICSwound) (Research Industrial Systems IT Engineering (RISE) GmbH, 2013). Based on yet another composition of Java Platform, Enterprise Edition (JEE) technology, SPICS comes with a JavaServer Faces (JSF) UI which utilizes the RichFaces component library.

In contrast to other systems, data privacy concerns are considered by the data model in detail. Subject de-identification is addressed by providing a patient identifier only (*W.H.A.T. (Wound Healing Analyzing Tool) Benutzerhandbuch* 2011, p. 6). The identifier may refer to an external record[11] which can be resolved by direct request of the priviledged user's browser (Lemmé, 2012, p. 78). Hence the system does neither relay nor store a subject's PII at any given time, while it is still able to display screens with fully qualified patient information.

[11]e.g. stored in the HIS


Neues Formular hinzufügen
Bezeichnung: * BMI-Rechner
Ausfüllbar: einmal pro Patient [✓ mehrmals pro Patient]
Formular gruppierbar: ☐
Manuell ausfüllbar: ✓
Zweispaltig: ☐
Beschreibung:
Sprache: Deutsch
Hinzufügen

BMI-Rechner
Pflichtfelder sind mit * markiert und sollten ausgefüllt werden. Alle anderen Felder sind optional.
Körpergewicht: kg
Körpergröße: cm
BMI:
Adipositas: ☐
Speichern Abbrechen

Formularfeld bearbeiten: a
Textfeld Abc
Allgemein
Bezeichnung * Name des Feldes
Hilfetext
Empfohlen ☐ Vorschau ☐
Erweitert
Einheit mm
Eingabeformat Kommazahl
Gültigkeitsbereich -
Bild
Optionen
Variablenname c
Formel =sqrt(a*a + b*b)
Bedingung :c > 0
Abbrechen Speichern

Figure 2.7.: SPICS screens, taken from (Lemmé, 2012)

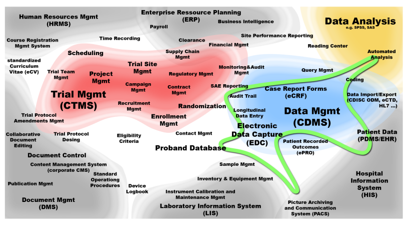
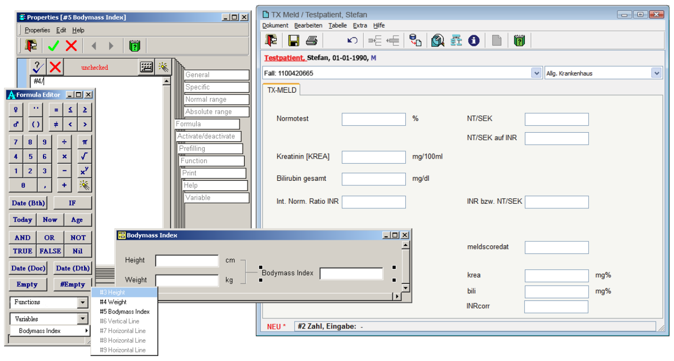

Human Resources Mgmt (HRMS)
Enterprise Ressource Planning (ERP)
Business Intelligence
Payroll
Data Analysis e.g. SPSS, SAS
Course Registration Mgmt System
Time Recording
Clearance
Site Performance Reporting
Financial Mgmt
Reading Center
Scheduling
Supply Chain Mgmt
Automated Analysis
standardized Curriculum Vitae (eCV)
Trial Team Mgmt
Project Mgmt
Trial Site Mgmt
Regulatory Mgmt
Monitoring&Audit Mgmt
Query Mgmt
Campaign Mgmt
Contract Mgmt
Coding
SAE Reporting
Case Report Forms (eCRF)
Trial Mgmt (CTMS)
Audit Trail
Data Import/Export (CDISC ODM, eCTD, HL7 ...)
Recruitment Mgmt
Randomization
Longitudinal Data Entry
Data Mgmt (CDMS)
Trial Protocol Amendments Mgmt
Enrollment Mgmt
Electronic Data Capture (EDC)
Collaborative Document Editing
Trial Protocol Desing
Eligibility Criteria
Contact Mgmt
Patient Data (PDMS/EHR)
Patient Recorded Outcomes (ePRO)
Proband Database
Document Control
Content Management System (corporate CMS)
Sample Mgmt
Standard Operationg Procedures
Hospital Information System (HIS)
Inventory & Equipment Mgmt
Publication Mgmt
Device Logbook
Instrument Calibration and Maintenance Mgmt
Document Mgmt (DMS)
Laboratory Information System (LIS)
Picture Archiving and Communication System (PACS)

Figure 2.8.: SPICS features.

ArchiMed ArchiMed is a mature two-tier database application for EDC, medical case documentation and data analysis. Data analysis capabilities relevant to academic research study projects are supported by means of Statistical Analysis Systems (SAS) Integration Technology. ArchiMed consists of a primary Visual Works (Smalltalk-based) client application (Gall et al., 1999), allowing users to directly connect to an Oracle Database RDBMS backend. The client does not implement local caching, but requires a continuous server connection, as opposed to existing fat client solutions for offline data capture such as clincase[12]. In 2004, ArchiMed is used at Vienna and Graz university hospitals by a user base in the order of 100 clinicians (Duftschmid and Wrba, 2004). MUW started to develop ArchiMed 1995 and caught up again by adding a separate JSF presentation layer to support web-based data entry for multi-center trials in 2007 (Gerhold, 2007). While under continuing development, Laboratory Information Management System (LIMS) aspects were fostered. With the beginning of 2011, it evolved into the "Remote Data Aquisition (RDA) platform", a major component of the MUW/Allgemeines Krankenhaus der Stadt Wien (AKH Wien) IT infrastructure.



#4/

Bodymass Index
Height cm
Weight kg
Bodymass Index

TX Meld / Testpatient, Stefan
Testpatient, Stefan, 01-01-1990, M
Fall: 1100420665
Allg. Krankenhaus
TX-MELD
Normotest %
NT/SEK
NT/SEK auf INR
Kreatinin [KREA] mg/100ml
Bilirubin gesamt mg/dl
Int. Norm. Ratio INR
INR bzw. NT/SEK
meldscoredat
krea mg%
bili mg%
INRcorr
NEU * #2 Zahl, Eingabe: -

Figure 2.9.: ArchiMed/RDA screens, taken from (Duftschmid and Wrba, 2004) and (Wrba et al., 2011).

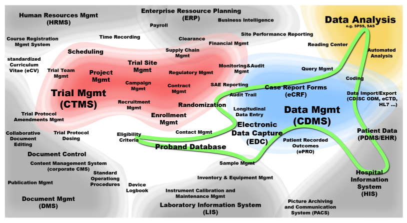

Human Resources Mgmt (HRMS)
Enterprise Ressource Planning (ERP)
Business Intelligence
Payroll
Data Analysis e.g. SPSS, SAS
Course Registration Mgmt System
Time Recording
Clearance
Site Performance Reporting
Financial Mgmt
Reading Center
Scheduling
Supply Chain Mgmt
Automated Analysis
standardized Curriculum Vitae (eCV)
Trial Team Mgmt
Project Mgmt
Trial Site Mgmt
Regulatory Mgmt
Monitoring&Audit Mgmt
Query Mgmt
Campaign Mgmt
Contract Mgmt
Coding
SAE Reporting
Case Report Forms (eCRF)
Trial Mgmt (CTMS)
Audit Trail
Data Import/Export (CDISC ODM, eCTD, HL7 ...)
Recruitment Mgmt
Randomization
Longitudinal Data Entry
Data Mgmt (CDMS)
Trial Protocol Amendments Mgmt
Enrollment Mgmt
Electronic Data Capture (EDC)
Collaborative Document Editing
Trial Protocol Desing
Eligibility Criteria
Contact Mgmt
Patient Data (PDMS/EHR)
Patient Recorded Outcomes (ePRO)
Proband Database
Document Control
Content Management System (corporate CMS)
Sample Mgmt
Standard Operationg Procedures
Hospital Information System (HIS)
Inventory & Equipment Mgmt
Publication Mgmt
Device Logbook
Instrument Calibration and Maintenance Mgmt
Document Mgmt (DMS)
Laboratory Information System (LIS)
Picture Archiving and Communication System (PACS)

Figure 2.10.: ArchiMed features.

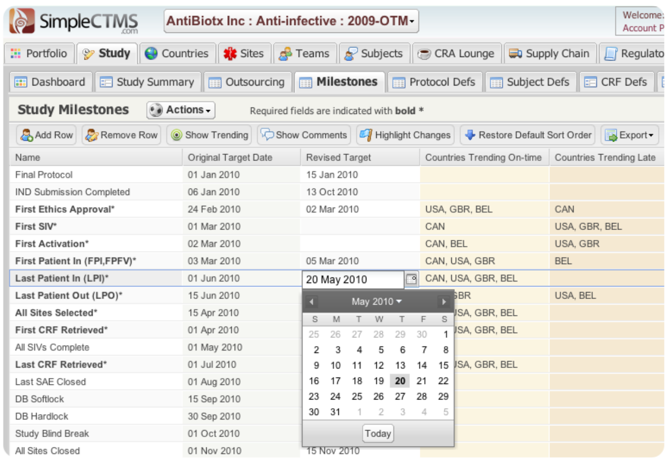
[12]Quadratek clincase is another commercial CDMS product, which was introduced by MUG and MUW 2012 according to (Wrba, 2014).

SimpleCTMS SimpleCTMS is an example of a "fresh" (2010) commercial product aiming at trial sponsor institutions with a focus on planning and managing large multi-centered clinical trials. It is a web-based CTMS built upon Ruby on Rails and MySQL RDBMS and provided as Software as a Service (SaaS). The Rich Internet Application (RIA) UI is created using the ExtJS UI component and AJAX framework library (Sencha Inc., 2014) - an alternative considered for the Phoenix CTMS as well.

SimpleCTMS allows to track CRF completion status and subject enrollment down to the site visit level, but does not provide EDC functionality. It is therefore presumed to combine or interface SimpleCTMS with external CDMS or Interactive Web Response System (IWRS), e.g. located at individual trial sites. Nevertheless, FDA Part 11 regulations are considered by supporting audit trail and digital signatures for data managed in SimpleCTMS. As an CTMS, supported features address remaining aspects of trials such as managing

- sites: subject enrollment progression, budgets, . . .
- project schedules: milestones, regulatory submissions, . . .
- inventory: supply chain, inventory per subject, . . .
- trial site staff: staff training status, team roster, . . .

Notably, some of these features rely on the integrated Document Management System (DMS) capabilities. User access control supports blinded and unblinded user types, which is required for screens for managing medication kit shipments for verum and reference subject groups to trial sites around the globe (supply chain management). (Trial By Fire Solutions, LLC, 2014)



Study Milestones

Required fields are indicated with **bold** *

Name	Original Target Date	Revised Target	Countries Trending On-time	Countries Trending Late
Final Protocol	01 Jan 2010	15 Jan 2010		
IND Submission Completed	06 Jan 2010	13 Oct 2010		
First Ethics Approval*	24 Feb 2010	02 Mar 2010	USA, GBR, BEL	CAN
First SIV*	01 Mar 2010		CAN	USA, GBR, BEL
First Activation*	02 Mar 2010		CAN, BEL	USA, GBR
First Patient In (FPI,FPFV)*	03 Mar 2010	05 Mar 2010	CAN, USA, GBR	BEL
Last Patient In (LPI)*	01 Jun 2010	20 May 2010	CAN, USA, GBR, BEL	
Last Patient Out (LPO)*	15 Jun 2010		GBR	USA, BEL
All Sites Selected*	15 Apr 2010		SA, GBR, BEL	
First CRF Retrieved*	01 Apr 2010		SA, GBR, BEL	
All SIVs Complete	01 May 2010			
Last CRF Retrieved*	01 Jul 2010		SA, GBR, BEL	
Last SAE Closed	01 Aug 2010			
DB Softlock	15 Sep 2010			
DB Hardlock	30 Sep 2010			
Study Blind Break	01 Oct 2010			
All Sites Closed	01 Nov 2010	15 Nov 2010		

S	M	T	W	T	F	S
25	26	27	28	29	30	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	1	2	3	4	5

Figure 2.11.: SimpleCTMS screens, taken from (Trial By Fire Solutions, LLC, 2014).

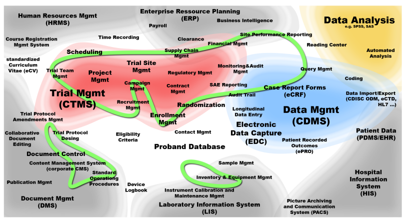


Figure 2.12.: SimpleCTMS features.

2.2.4. Comparison: EDC Support

Although the Phoenix CTMS and each existing systems outlined in the previous section were designed for different purposes, they have EDC capabilities in common (except SimpleCTMS). A comparison of EDC related implementation details is given in table 2.2 and 2.3, which will be encountered throughout sections of the implementation chapter (4):

- "Entity-Attribute-Value (EAV)" (4.1.1.3)
- "Encryption" (4.2.3)
- "Digital Signatures" (4.3.3)
- "Electronic Data Capture (EDC)" (4.8)
- "Query Designer" (4.9)

	OpenClinica	**REDCap**	**SPICS**	**ArchiMed**	**Phoenix CTMS**
general					
application type	web-based	web-based	web-based	database client/web-based	web-based
RDBMS	PostgreSQL (default)	MySQL	PostgreSQL	Oracle Database	PostgreSQL (default)
technology stack	Spring, Hibernate, JSP	PHP	Seam, JSF 1.2, RichFaces	Visual Works/JSF, JDBC	Spring, Hibernate, JSF 2, PrimeFaces
EDC purpose	eCRF, eligibility criteria	eCRF, subject registry	medical case documentation, subject registry	medical case documentation, subject registry, eligibility criteria, statistical analysis	subject registry, eligibility criteria, enrollment progress
EDC					
metamodel	"trial" - "CRFs" - (data) "items"	"database" - "instruments" - "fields"	"wound"[12] - "documents" - "form elements"	"form" - "data fields"	1. "trial" - "inquiries" - "input field", 2. "trial" - "proband list columns" - "input field"
longitudinal form versioning model	(visit) "event"	(visit) "event"	"data sheet"	"document"	✗
data entry control (release, lock)	"event" status	draft and production "database" status	✗	✗	trial status
form composition	`.xls` import	UI editor, `.xls` import	predefined "documents" by product line, UI editor	UI editor	UI editor
multilingual UI/forms	✓/CRF per language	✓/instrument per language	✓/multilingual form elements	✓/	✓/predefined multilingual input fields
preview and test forms	✓	✓	✓		✓
base data types[12]	✓	✓	✓	✓	✓
strongly-typed	✓	✗		✓	✓
additional data types	✗	matrix data type	image data type	✗	sketch data type
file data type	✓	✓	✓	✗	DMS feature
field/form library	✓	✓		✓	✓
input validation	server-side	client-side	server-side	client application	server-side
range check options	RegExs, min-max	predefined RegExs, min-max	min-max, DSL expressions	normal and total min-max	RegExs, min-max
conditional branching	✓	✓	✗	✓	✗
calculated fields	✓	✓	✓	✓	✓
form sections/-variable groups	✓/✓	✓/✗	✓/✗	✓/✓	✓/✗
query-based variable values	✗	✗	✗	predefined queries	autocomplete suggestions

Table 2.2.: Comparison of EDC implementations, part 1.

[12]text, number, date, time, ID/code
[12]SPICSwound Wound Healing Analyzing Tool (W.H.A.T)

	OpenClinica	**REDCap**	**SPICS**	**ArchiMed**	**Phoenix CTMS**
form scripting					
"formulas"	DSL for arithmetic and conditional expressions	JavaScript-based statements	DSL statements for arithmetic and conditional expressions	DSL statements for arithmetic and conditional expressions, predefined functions, predefined queries	anonymous JavaScript functions
iteration constructs[12]	✗	✗	✗	✗	✓
execution	server-side	client-side	server-side	client application	client-side
application	branching logic, field value calculation, creation of discrepancy note records, sending emails	branching logic, field value calculation	field value calculation, range constraints	branching logic, field value calculation	field value calculation, output labels (unit conversion, range checks)
expression entry	XML upload, editor UI (enterprise)	`.xls` upload, editor UI	editor UI	editor UI	editor UI
extensability (compile time)	additional result actions		new operators and functions	predefined functions	JavaScript (Js) library export
data retrieval					
ad-hoc database queries	✗	report builder UI		SAS integration	query editor UI
set operations		✗			✓
data export	CSV, CDISC ODM, SPSS, SQL (enterprise)	CSV, SAS, R, SPSS, STRATA	`.xls`, PDF		`.xls`
reporting charts	✗	cross-sectional charts	annual reports	histograms	✗
web service API	SOAP	✓	✓		REST
security and regulatory compliance					
subject de-identification	✗	export filters, date shifting	HIS record reference	✗	PII encryption
encryption (data at rest)	✗	✗	✗	✗	application-level file and database column encryption
audit trail	✓	✓	✓	✓	✓
digital signatures	✓	✓	✗	✗	✓
special features					
highlight	data discrepancy resolution workflow	REDCap Data Transfer Service (DTS)	image analysis	WYSIWYG form editor	use of scanned paper forms

Table 2.3.: Comparison of EDC implementations, part 2.

[12] A plausible use case of form scripting is using field value calculations to compute statistical parameters. In combinatorics, algorithms for calculating basic parameters rely on iteration constructs. As an example, the binomial coefficient can be expressed using the factorial operator: $\binom{a}{b} = \frac{n!}{k!(n-k)!}$. An exact algorithm for calculating $n!$ requires iterations (or recursions).

2.3. Architectural Style

Web-based CDMS and CTMS can be considered as full-fledged enterprise web applications serving up to hundreds of users. An appreciable fraction of this user base will be "online" at the same time during daily business. Web applications designed for this magnitude of system load still tolerate spending a relatively high amount of system resources per user session. A classical RDBMS data store layer is preferred over the weak consistency guarantees of a Not Only Structured Query Language (NoSQL) approaches (Cattell, 2011). A strong presentation layer will provide a complex and highly interactive UI with a rigorous level of detail, as known from desktop applications. In contrast to traditional page transitions, the browsers' HTTP requests are caused by Asynchronous JavaScript and XML (AJAX) requests when *partially updating* a page's Document Object Model (DOM). These are the well known properties of modern *RIA* User Interfaces.

All these characteristics also hold for the system presented in this work. This section will outline a rigorous architecture for achieving these properties, which is dictated by the cross-platform Java technology stack used.

2.3.1. Multi-Tiered Architecture

Two-tiered client-server systems distinguish between a presentation and data store layer. In this basic case, event listener code of UI components will contain business logic such as input validation and calculations as well as data store read/write operations directly. Three-tier architectures enclose business logic in a separate layer in between, at the cost of a more extensive implementation effort. The service layer is a textbook example for the *separation of concerns*. It provides advantages such as physical separation of components and introduces joint points for swapping out more detailed implementations aspects into additional layers. In this case the term *multi-tiered architecture* might be more accurate.

The application's architectural style corresponds to the multi-tiered architecture illustrated in figure 2.13.

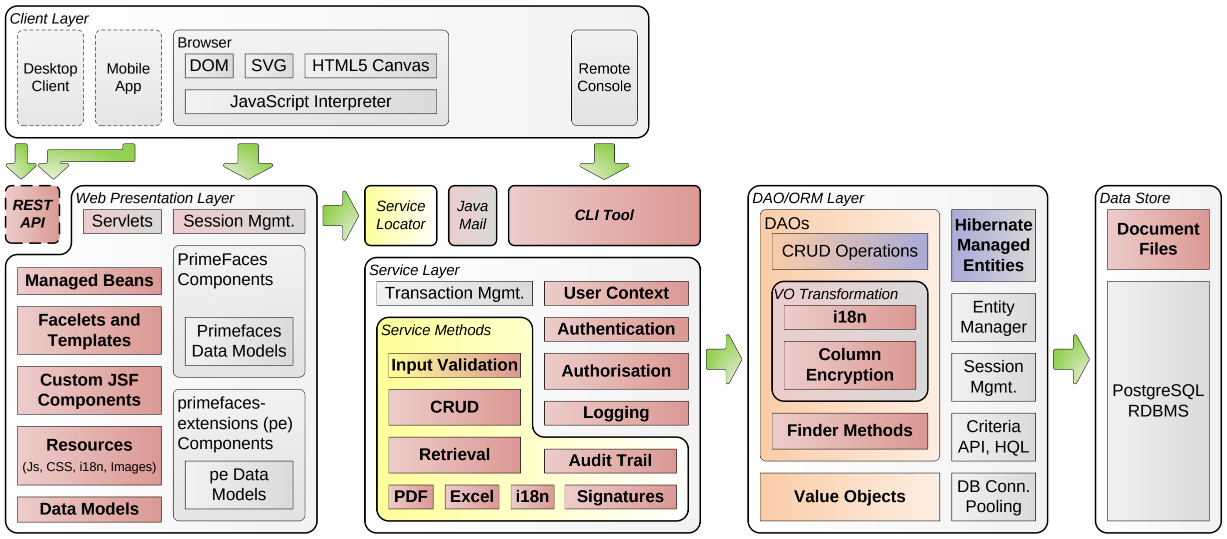


Figure 2.13.: The Phoenix CTMS has a multi-tier application architecture design. Gray elements denote infrastructure provided by standard software and frameworks used, while application-level components are illustrated by a colored background. Components in a color other than rose include source code *generated* in the course of MDD.

2.3.1.1. Data Store Layer

RDBMS technology evolved over decades and there are successful products available as free software and proven in regulated environments. Data is stored in rows of relational tables (records) of a backing RDBMS. The application will connect to the RDBMS using a single database user by means of Java DataBase Connectivity (JDBC), which is mastered by the Hibernate ORM framework. Hibernate provides an abstraction for subsequent application layers, which allows choosing different RDBMS without significant changes to the application. To avoid breaking this portability, *stored procedures* will not be used. For this project, it was decided to go with PostgreSQL, which supports *Atomicity, Consistency, Isolation, Durability (ACID) transactions* by means of *Multiversion Concurrency Control (MVCC)* (Momjian, 2012, p. 58). Aside application logic, referential integrity will be enforced by *constraints* of the database schema[13]. The application will however *not* rely on *cascade triggers*[14], which could be set up in the database schema otherwise. PostgreSQL will run at the default *READ˙COMMITTED* transaction isolation level. This is a happy medium in terms of performance and data consistency with concurrent transactions, although phantom read situations will have to be identified and prevented by explicit row locking. Another essential measure is reducing the frequency of trivial request such as fetching single rows, which is addressed by Hibernate's lazy-loading technique and built-in caching strategies. To increase performance of particular Structured Query Language (SQL) queries, multi-column indexes and indexes for non-key or non-unique table columns are added manually as required.

The application is expected to store considerable amounts of binary data such as document files or images. This can be done by either saving file contents in *Binary Large Object (BLOB)* table columns or physical files stored in the file system separately. Since both options will be supported, the data store layer will also comprise raw files managed by the application.

Aside the data store layer, the term *backend* usually refers to the service layer and DAO layer described next.

2.3.1.2. Data Access Object Layer

Service and DAO layers are based on a *Spring/Hibernate* archetype. Its structure is dictated by the *AndroMDA* code generator, which is the essential tool for the MDD approach described in section 2.3.2. Hibernate is a major ORM persistence framework, while Spring provides Dependency Injection (DI) and Aspect-Oriented Programming (AOP) to wire Hibernate's infrastructure. Spring DI allows expressing *object instantiation and configuration* by either *annotating* Plain Old Java Object (POJO) fields[15] in Java source files or in accompanying XML files. Hibernate enables handling of RDBMS records by *mapping* table columns to fields of POJOs. The mapping is defined by annotations or in separate XML files. The POJO classes represent the application's *domain entities* and are referred to as managed entities in terms of Hibernate. The main idea of Object-Relational Mapping/Mapper (ORM) is to synchronize database table contents with the state of managed entity instances, manipulated by the application logic.
Create, Update and Delete (CRUD) operations rely on Hibernate's entity manager, which will automatically generate and run SQL Data Manipulation Language (DML) statements[16] matching managed entity fields. SQL statements are sent to the RDBMS over TCP/IP connection after a session is created. Since each database connection takes time to set up and causes the RDBMS to allocate memory, they are managed by the framework to reuse established connections by means of *connection pooling*. SQL queries for record retrieval are enclosed by *finder methods*, which may utilize a query abstraction (Criteria API, Hibernate Query Language (HQL)). Both CRUD operations and finder methods are encapsulated by the Data Access Object (DAO) of a corresponding domain entity. Because managed entities are sensitive regarding

[13] `FOREIGN KEY`, `UNIQUE`, `NOT NULL`
[14] e.g. `ON DELETE CASCADE`
[15] A *field* refers to a class member variable and/or their corresponding accessor methods.
[16] `INSERT INTO ...`, `UPDATE ...`, `DELETE FROM ...`

manipulation and serialization, the Data Transfer Object (DTO) pattern intends to decouple application layers by passing *DTOs*[17] instances instead of entity instances (Fowler, 2002, pp. 401). In general, VOs can be defined as projections of the original managed entities. When instantiating a VO, it has to be populated by copying entity field values, which is performed by VO transformation methods part of the DAO.

2.3.1.3. Service Layer

The application's business logic is broken down into service methods, which are exposed to remaining components by interfaces of the service layer. Service method invocations with short latency facilitate the request-response pattern inherent to consuming layers. Each service method call is expected to execute in its own request thread context. This requires a thread-safe implementation which is simplified by staying with stateless service classes and methods. Service method implementations will propagate or combine CRUD and finder methods by utilizing injected DAO instances for relevant domain objects. Input parameters and return types of service methods are primitive data types and/or VOs, which are converted or created using VO transformation methods. Integrating input validation into the service layer is advantageous since it has to be implemented only once instead of in each consuming layer.

Use cases of multi-user systems such as enterprise applications may require identifying and checking legitimation of requesting users instead of allowing anonymous access. The service layer performs user authentication and authorisation for each service method call and leaves session management up to consuming layers. During the authentication operation, a user context variable visible to the scope of the request thread is populated with credentials of the requesting user[18] at the beginning of a service method invocation. This will simplify authorisation logic, personalisation and cryptographic operations, because credentials need not be passed using additional method parameters.

To keep concurrent data access from multiple users under control, the application is required to utilize ACID transactions supported by the RDBMS. This is achieved by demarcating a transaction for the scope of each service method. A transaction starts when entering the service method and will be committed upon returning result values. The service method implementation may throw an exception to roll the entire transaction back. If service methods invoke each other, the more complex situation of *transaction propagation* arises. The management of transactions is part of Hibernate and wired using Spring AOP.

Service singleton instances are managed and exposed to consumers using a *service locator* object. Aside the web presentation layer, the Command Line Interface (CLI) tool is a remarkable consumer component. It allows starting the Java interpreter to run an executable entry point (`void main`), which can provide features for maintenance task automation and asynchronous processing.

2.3.1.4. Web Presentation Layer

JavaServer Faces JavaServer Faces (JSF) is the JEE stack's well-established Model-View-Controller (MVC) (A. Holzinger, Struggl, and Debevc, 2010) framework for building a web presentation layer. *Mojarra* (Oracle America, Inc., 2012) is the JSF reference implementation according to Java Specification Request (JSR)-314 of the Java Community Process (JCP). JSF introduces a component-based approach for rendering dynamic web pages. A page displayed by the browser consists of UI components such as containers (e.g. panel), input components (e.g. checkbox) and others (e.g. button, datatable). The hierarchy of components displayed in the page is defined by XHTML files[19] (*MVC view*). Templates allow organising and reusing facelet fragments. A *servlet container* is used to run the JSF main *servlet*

[17] also known as *Value Objects (VOs)*
[18] sometimes referred to as *security principal*
[19] referred to as *facelets*

for processing incoming HTTP requests (*MVC controller*). It maintains an in-memory representation of the page's component tree for manipulation and transformation into HTML markup (*rendering*). The generated markup of basic JSF components will comprise standard HTML DOM elements. Advanced components may utilize extended browser infrastructure such as JavaScript (Js), HTML5, Scalable Vector Graphics (SVG) and browser plug-ins. Aside full page load requests (HTTP GET) and HTML form submissions (POST), JSF 2 supports AJAX interactions by the concept of Partial View Rendering (PVR). In this case, an AJAX request[20] will update desired component tree branches only, as defined for the originating component in the page facelet.

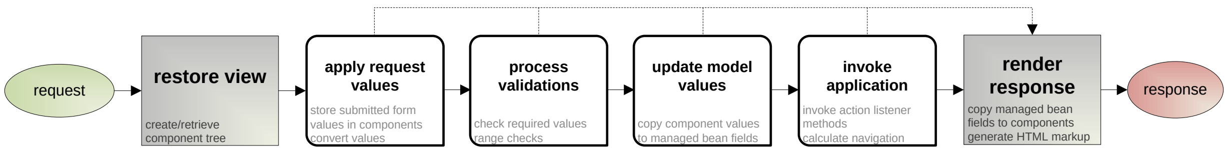

request
restore view
create/retrieve component tree
apply request values
store submitted form values in components convert values
process validations
check required values range checks
update model values
copy component values to managed bean fields
invoke application
invoke action listener methods calculate navigation
render response
copy managed bean fields to components generate HTML markup
response

Figure 2.14.: The lifecylce of HTTP requests comprises a sequence of phases. While "restore view" and "render response" are system level phases implemented by the JSF framework, phases in between represent application level phases. (Hightower, 2005)

A *managed bean* is a POJO representing a buffer for data displayed by UI components on the one hand and DTO instances for service method input parameters and return types on the other hand (*MVC model*). JSF maintains HTTP session cookies in order to keep data held by managed bean instances across HTTP requests. During the restore view phase of the JSF lifecycle (figure 2.14), managed bean instances are either created or looked up in a map of active HTTP sessions.[21] The instance lifetime of a particular managed bean is defined at design time using *scope* annotations (table 2.4).

managed bean lifetime scope	**description**
`@ApplicationScoped`	The same instance of an `@ApplicationScoped` managed bean is accessible within request cycles from any client during the JSF application's entire lifetime in the servlet container.
`@SessionScoped`	An instance of a `@SessionScoped` managed bean is accessible in request cycles referring to a particular HTTP session.
`@ViewScoped`	An instance of a `@ViewScoped` managed bean is accessible in repeating PPR request cycles triggered from the page loaded in a particular browser window or tab.
`@RequestScoped`	An instance of a `@RequestScoped` managed bean exists for the scope of the ongoing request only.
`@NoneScoped`	An instance of a `@NoneScoped` managed bean is created upon every single property access during a request's lifecycle phases.

Table 2.4.: The JSF infrastructure supports different lifetime scopes for managed bean instances. RIA applications will heavily rely on `@ViewScoped` managed beans. An application session controller can be implemented using a `@SessionScoped` managed bean holding credentials required for service method calls.

[20]The Asynchronous JavaScript and XML (AJAX) request is a HTTP POST request, often refererd to as Partial Page Refresh (PPR) in terms of JSF. The server responds with data containing HTML page fragments, encoded in XML format.

[21]This consideration assumes the JSF configuration parameter `javax.faces.STATE_SAVING_METHOD` set to "server" in deployment descriptor.

In facelets, managed bean instances are referenced in *Expression Language (EL)* expressions of component attribute values (*binding*). The expressions are evaluated during application level phases of the JSF request lifecycle. As a result, managed bean fields are accessed (getter and setters) and managed bean methods bound to UI components (*action listeners*) will be invoked. Action listeners represent the effective link to the service layer; they will invoke service methods exposed by the service locator (figure 2.15).

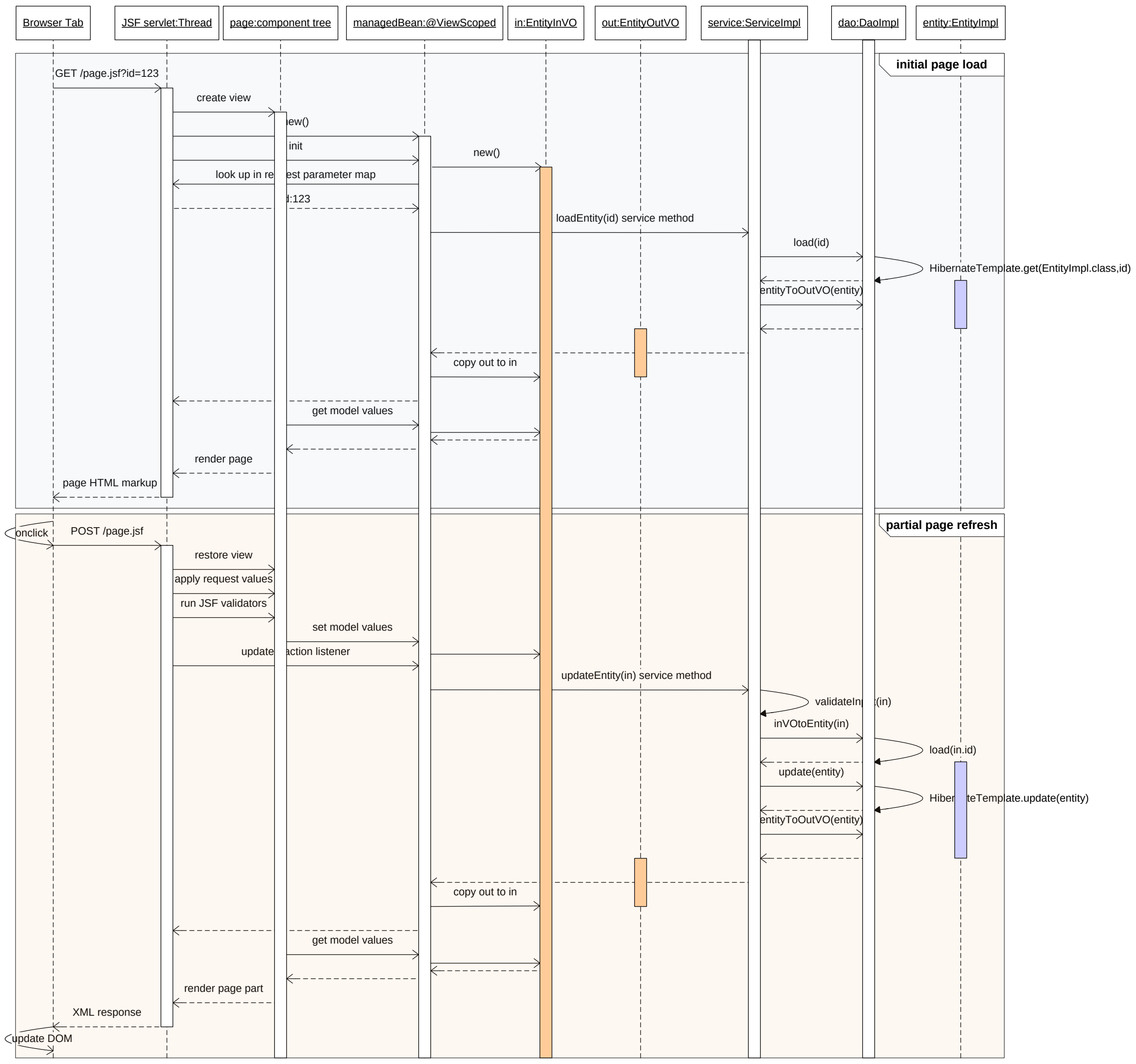


Figure 2.15.: This sequence diagram summarizes the propagation of requests across all layers of the proposed RIA architecture. `@ViewScoped` JSF managed beans are created upon the initial page load to hold DTOs across subsequent AJAX requests. The user's browser displays the rendered page showing populated DTO field values. The user can now start editing the loaded record. A PPR is triggered by pressing a button (e.g. "save") and modified form values are submitted. The action listener will invoke the corresponding CRUD service method and passes the updated DTO (InVO). The service method return value (OutVO) reflects the modified record after updating the database. The PPR completes after copying OutVO field values back into InVO fields and rendering the page components to be refreshed. The browser's DOM is updated to display the refreshed page fragments and the UI is ready for subsequent editing again.

Component suites While JSF comes with a basic set of components[22], it is possible to develop composite and *custom components*. Popular component suites such as *PrimeFaces* (Çivici et al., 2014) provide replacements for native components with extended capabilities and enhanced visual appearance, as well as numerous new components. Complex UI components will typically require to provide managed bean fields with predefined data structures (*data models*). *primefaces-extensions* is an add-on for PrimeFaces contributed by a separate group (Varaksin et al., 2013), providing even more components. PrimeFaces and primefaces-extensions represent collections of custom JSF components, most of which relying on complementary *client-side*[23] implementations (*widgets*) and third-party JavaScript (Js) libraries.

Web service An optional Representational State Transfer (REST) web service allows to publish relevant service methods, which is considered for future extensions such as a desktop client application. *Jersey* (Oracle Corporation, 2014) is the reference implementation of the Java API for RESTful Web Services (JAX-RS) specified in JSR-311. Jersey provides a separate request servlet for handling URLs mapped to annotated methods of web service *resource* classes. These methods represent lightweight wrappers for service methods to be exposed. Input arguments are de-serialized from the HTTP request body carrying data in JavaScript Object Notation (JSON), XML, multipart or other formats. Web service requests complete by serializing return values for HTTP response bodies. For the sake of a simple deployment, the web service servlet may run aside the JSF request servlet in the same servlet container.

2.3.2. Code Generation with AndroMDA

AndroMDA (Bohlen et al., 2012) is the successful, free code generator tool, which will be used to facilitate the development of the system's DAO and service layers. AndroMDA promotes a Model-Driven Development (MDD) process, which corresponds to the "Model-Driven Architecture (MDA)-light" approach described in (Gruhn, Pieper, and Röttgers, 2006, 49,pp. 178):

1. create a Platform Independent Model (PIM) using a Unified Modelling Language (UML) modelling tool of choice
2. enrich the model with additional information required for the code generation
3. automatic generation of a skeleton from the model
4. manual completion of the missing business logic code
5. iterate steps before

This outlines a forward engineering scheme, which allows to rapidly create a working prototype for subsequent refinement. AndroMDA uses either NetBeans Metadata Repository (MDR) API (UML 1.4) or the Eclipse Modeling Framework (EMF) UML2 API (UML 2) to create an in-memory abstraction (*matafacades*) of parsed UML models. Output files (*artefacts*) such as Java source files are generated from the loaded model using Apache Velocity *templates*. AndroMDA supports generating artefacts for various target technology stacks using (interdependent) plug-ins (*cartridges*). Aside the metafacades implementation, cartridge resources comprise templates, which can be overriden with modified copies part of the specific application to create. Another cartridge tuning option are *merge mappings*, a mechanism for inserting application-specific fragments into generated artefacts (e.g. Spring configuration) at predefined placeholders (*merge-points*). The skeleton created by the Spring/Hibernate/Java cartridge combination dictates a proven backend architecture, which considers a plethora of implementation details out-of-the-box:

[22]also known as *native components*

[23]*Client-side* denotes program code executed by the browser's Js interpreter. In contrast, the *server-side* Java application code is executed by the Java Virtual Machine (JVM) running the servlet container.

Maven build automation:

- hierarchy of Maven projects
- versioning and dependencies on external libraries per project
- built lifecycle including test suite and AndroMDA invocation
- creation of SQL Data Description Language (DDL) scripts
- pre-/post-processing of source/target files

AndroMDA configuration:

- cartridge resource merge mappings and template overrides
- UML model excludes
- generator "outlets"
- POJO field naming and table column naming options

Spring/Hibernate configuration:

- Hibernate-related defaults (lazy-loading, transaction propagation, ORM inheritance strategy, Java collection type for association end points)
- Hibernate wiring (transaction manager, session creation)
- Java and column data type mapping
- (custom) Hibernate user types (enumeration, BLOB)
- SQL dialect
- JDBC connection parameters
- caching strategy configuration

Layer infrastructure code:

- main Spring configuration including merge-points for extensions
- principal store (user context)
- service locator
- a simple abstraction for constructing HQL query statements

The proposed application structure comprises three interdependent Maven projects (figure 2.16). The Spring/Hibernate cartridge will create artefacts for both the *core tier* and *common* projects, which are compiled to separate `.jar` libraries each. Basically, hand-crafted implementations are a mandatory part of the core tier project only.

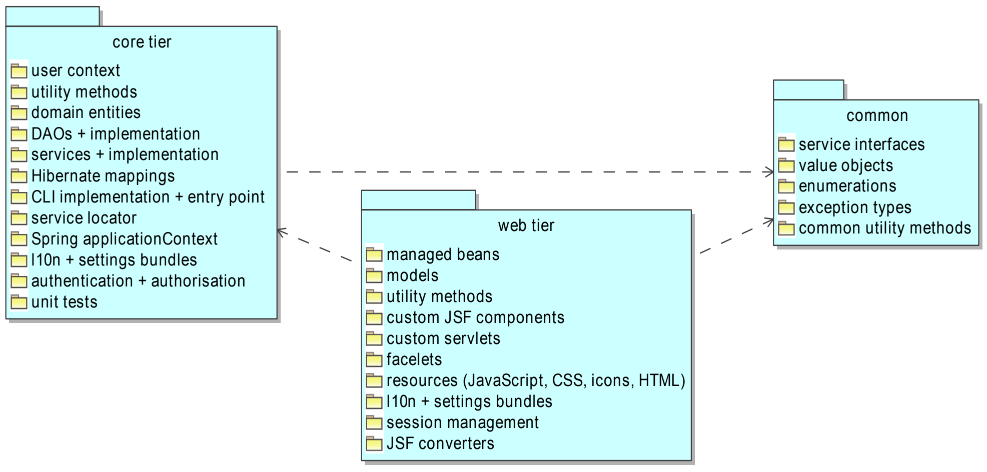


Figure 2.16.: The core tier project houses both the service and DAO layers. Essential type definitions are externalized into the common project, which is required when distributing layers on different physical servers.

AndroMDA comes with additional cartridges such as "BPM4Struts" and "JSF" for generating the web tier based on UML use case and activity diagrams. However, a "hand-crafted" approach was preferred over a generated UI for various reasons. Most notably, a custom cartridge would be required to generate the Maven project for a JSF presentation layer based on a third-party JSF component suite. Even if such a catridge exists, expressing complex sequences of PPR interactions with UML is considered not practicable. UML modelling therefore reduces to creating the PIM[24] AndroMDA requires for generating the backend tiers. This model will be a (partitioned) UML 2 *class diagram* (figure 2.17). The UML diagram contains class model elements, which relate to one another according to *dependency*, *association* and *generalisation* model elements.

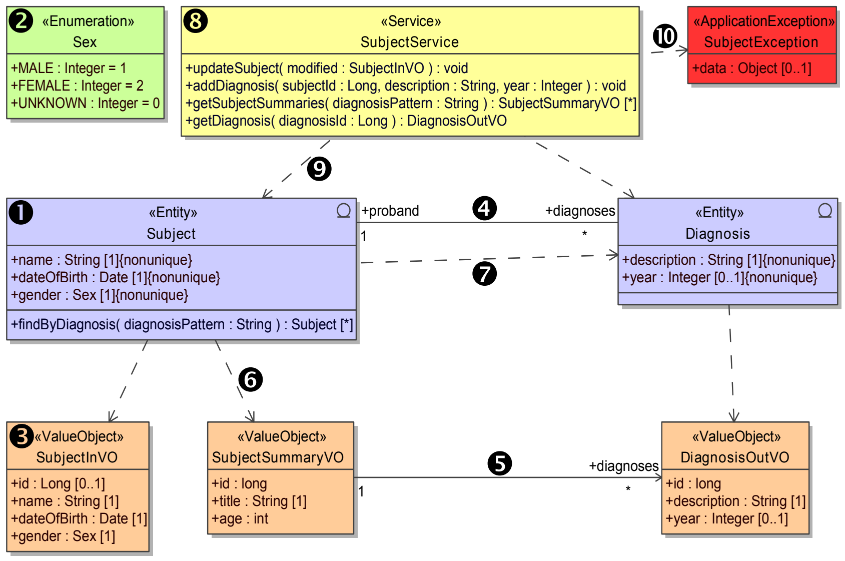

❷ «Enumeration» Sex
+MALE : Integer = 1
+FEMALE : Integer = 2
+UNKNOWN : Integer = 0
❽ «Service» SubjectService
+updateSubject(modified : SubjectInVO) : void
+addDiagnosis(subjectId : Long, description : String, year : Integer) : void
+getSubjectSummaries(diagnosisPattern : String) : SubjectSummaryVO [*]
+getDiagnosis(diagnosisId : Long) : DiagnosisOutVO
❿
«ApplicationException» SubjectException
+data : Object [0..1]
❾
❶ «Entity» Subject
+name : String [1]{nonunique}
+dateOfBirth : Date [1]{nonunique}
+gender : Sex [1]{nonunique}
+findByDiagnosis(diagnosisPattern : String) : Subject [*]
+proband
1
❹
+diagnoses
*
❼
«Entity» Diagnosis
+description : String [1]{nonunique}
+year : Integer [0..1]{nonunique}
❻
❸ «ValueObject» SubjectInVO
+id : Long [0..1]
+name : String [1]
+dateOfBirth : Date [1]
+gender : Sex [1]
«ValueObject» SubjectSummaryVO
+id : long
+title : String [1]
+age : int
❺
1
+diagnoses
*
«ValueObject» DiagnosisOutVO
+id : long
+description : String [1]
+year : Integer [0..1]

Figure 2.17.: This minimal example describes a system for storing subjects and their patient history. The UML class diagram gives an overview of most relevant model elements used when creating service and DAO layer skeletons using AndroMDA. Dependencies are indicated by dashed lines, while associations are indicated by solid lines.

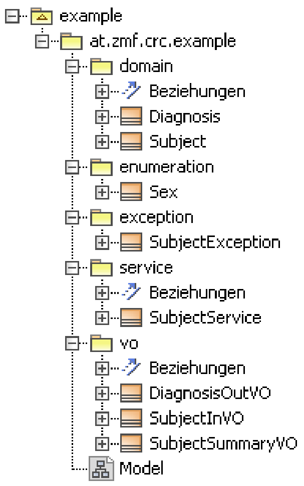


Figure 2.18.: Package names are defined in MagicDraw's containment tree.

Domain entities Domain entities are UML classes tagged with the ≪Entity≫ stereotype (1). AndroMDA will use domain entity definitions to generate Hibernate's managed entity artefacts. This includes mappings for Hibernate, which in turn allows generating DDL SQL scripts for removing and creating the *database schema* (listing 2.1 and 2.2). The example in figure 2.17 does not utilize *ORM inheritance*, which could be defined using generalisation relationships between domain entities.

```
example=# \d+ subject
                    Table "public.subject"
   Column    |          Type          |Modifiers|Storage
-------------+------------------------+---------+--------
id           |bigint                  |not null |plain
name         |character varying(1024)|not null |extended
date_of_birth|date                    |not null |plain
gender       |integer                 |not null |plain
Indexes:
    "subject_pkey" PRIMARY KEY, btree (id)
Referenced by:
    TABLE "diagnosis" CONSTRAINT "diagnosis_proband_fkc"
        FOREIGN KEY (proband_fk) REFERENCES subject(id)
```

Listing 2.1: Database table mapped by the `Subject` domain entity from figure 2.17.

```
example=# \d+ diagnosis
                  Table "public.diagnosis"
  Column   |          Type          |Modifiers|Storage
-----------+------------------------+---------+--------
id         |bigint                  |not null |plain
description|character varying(1024)|not null |extended
year       |integer                 |         |plain
proband_fk |bigint                  |not null |plain
Indexes:
    "diagnosis_pkey" PRIMARY KEY, btree (id)
Foreign-key constraints:
    "diagnosis_proband_fkc" FOREIGN KEY (proband_fk)
        REFERENCES subject(id)
```

Listing 2.2: Database table mapped by the `Diagnosis` domain entity from figure 2.17.

[24]Since the PIM is enriched with Hibernate/Spring or or Java language specific information such as *tagged values* and differentiated stereotypes, it is actually considered as a hybrid PIM/Platform Specific Model (PSM) model in terms of MDA.

Since UML field types have to be mapped to both Java data types and table column types, the UML *profile* included with AndroMDA's UML model template provides platform-independent data types. Custom enumerations (2) can be defined to be used as field or method parameter types as well. Stubs for finder methods in DAO artefacts are generated from static method declarations in domain entity model elements, such as `findByDiagnosis` in figure 2.17. AndroMDA also supports generating the complete database query implementation according to an *Object Constraint Language (OCL) expression* tagged to the finder method.

Value Objects VOs are UML classes tagged with the ≪ValueObject≫ stereotype (3). AndroMDA will use VO definitions to generate DTO artefacts, which is the preferred term for independent POJOs. The term Value Object (VO) is preferred for the majority of DTOs, which represent entity projections (6). In this case, transformation methods for converting between managed entity and VO instances will be added to DAO artefacts. Unfortunately, generalisation relationships between VO cannot be used to specify Java class inheritance without cartridge template extensions.

Service interfaces Service interfaces are UML classes tagged with the ≪Service≫ stereotype (8). AndroMDA will use service interface definitions to generate service layer artefacts. As with other artefacts, an abstract base class is extended to put hand-crafted implementations into place (figure 2.19). The application's business logic is essentially provided by service method implementations, located in separate source files (listing 2.4). Multiplicity checks for service method input parameters of VO types are genererated according to the multiplicity specified for VO fields. This rudimentary input validation is located in service base class implementations.

Entity-to-entity associations Entity associations (4) allow to define relationships mapped by the database schema (table 2.5).

association type	`Subject`	`Diagnosis`	**interpretation**
one-to-one	0..1	0..1	A diagnosis record may link to one subject record at most. A subject record may link to one diagnosis record at most.
many-to-one	*	0..1	A diagnosis record may link to many subject records. A subject record may link to one diagnosis record at most.
one-to-many	0..1	*	A diagnosis record may link to one subject record at most. A subject record may link to many diagnosis records.
many-to-many	*	*	A diagnosis record may link to many subject records. A subject record may link to many diagnosis records.

Table 2.5.: Possible types of associations are determined by the *multiplicity* of their end points. In the example, the one-to-many association might fit the real-life situation modelled in figure 2.17 best, while keeping the implementation effort low.

Entity relations manifest in foreign key columns and constraints. A many-to-many association additionally requires a separate association table. Managed entities reflect association ends using single- and collection-valued fields, if the end is marked *navigatable* in the UML diagram. If the same entity is used for association ends, the association is referred to as *reflexive*. The *graph* given by associated entities will be referred to as domain model or *data model* for short.

VO-to-VO associations VO associations (5) allow constructing VO graphs. These may be of particular relevance in the case of complex service method return types. VO associations manifest in POJO references of VO instance graphs. If VO graphs are considered as projections of entire entity graphs, VO associations will typically correspond to entity associations.

Entity-to-VO dependencies Entity-to-VO dependencies (6) are used to generate VO transformation methods in the entity's DAO artefact. Created code for both transformation directions is organized into two groups of VO transformation methods (entity-to-VO and VO-to-entity). Generated VO transformation method code will include copying member variable values of basic types from managed entity to VO instances and vice versa, if fields of entities and VOs have a matching name and data type. This essentially excludes VO association end fields, which therefore have to be populated manually in DAO implementation classes (listing 2.3).

Entity-to-entity dependencies Entity-to-entity dependencies (7) can be used to model *DAO DI*. The implementation of a VO transformation method with association transformations mentioned before will depend on the DAO instance of the associated entity. As an example, this will be encountered when implementing `SubjectDaoImpl.toSubjectSummaryVO` (listing 2.3). This VO transformation method populates a `SubjectSummaryVO` from a given `Subject` instance, which includes populating the output's `SubjectSummaryVO.diagnoses` field (5). This field represents the one-to-many association end for holding a `Collection` of the "many" `DiagnosisVO` objects, given by the subject's `Subject.diagnoses` field (4). This introduces a dependency between `SubjectDao` and `DiagnosisDao`, since populating the collection requires to invoke `DiagnosisDao.toDiagnosisVO` (`DiagnosisDao.toDiagnosisVOCollection`).

```java
// Generated by: hibernate/SpringHibernateDaoImpl.vsl in andromda-spring-cartridge.
// license-header java merge-point
/**
 * This is only generated once! It will never be overwritten.
 * You can (and have to!) safely modify it by hand.
 */
package at.zmf.crc.example.domain;

public class SubjectDaoImpl extends SubjectDaoBase {

    ...

    public void toSubjectSummaryVO(Subject source, SubjectSummaryVO target) {

        super.toSubjectSummaryVO(source, target); //copy trivial entity fields to VO fields
        //the message below is placed by the generator:
                // WARNING! No conversion for target.diagnoses (can't convert source.getDiagnoses():at.zmf.crc.example.domain.
                    Diagnosis to at.zmf.crc.example.vo.DiagnosisOutVO

        //thus, the "diagnoses" association has to be transformed manually:
        Collection diagnoses = source.getDiagnoses();
        this.getDiagnosisDao().toDiagnosisOutVOCollection(diagnoses); //dependency on DiagnosisDao
        target.setDiagnoses(diagnoses);

        //calculate remaining VO fields:
        target.setAge(Utils.calculateAge(source.getDateOfBirth()));
        target.setTitle(MessageFormat.format("{0}, {1}", source.getGender().name(), target.getAge())); //e.g. "MALE, 34"
    }

    ...
```

Listing 2.3: DAO transformation method implementation example of `SubjectDaoImpl.toSubjectSummaryVO` from figure 2.19.

Service-to-entity dependencies *DAO DI* into services is inherent to the architecture and modeled using Service-to-entity dependencies (9). They represent the basic link between DAO and service layer. Service method implementations (listing 2.4) will effectively utilize methods exposed by DAOs and therefore rely on DAO instances injected into the service instance.

Service-to-exception dependencies A service base class wires manual service method implementations (handlers, listing 2.4) to add generated multiplicity checks and wrap exceptions in order to restrict exception types. The business logic may throw custom exception types to abort transactions and propagate problem information to consuming layers. Custom application exception types are UML classes declared using the ≪ApplicationException≫ stereotype. They yield generated Java exception types, which are added to service method stub signatures by drawing a Service-to-exception dependency (10). These linked exception types are propagated without wrapping when thrown in service method implementations. Generalisation relationships between exception types are supported to define a inheritance hierarchy of Java exception types.

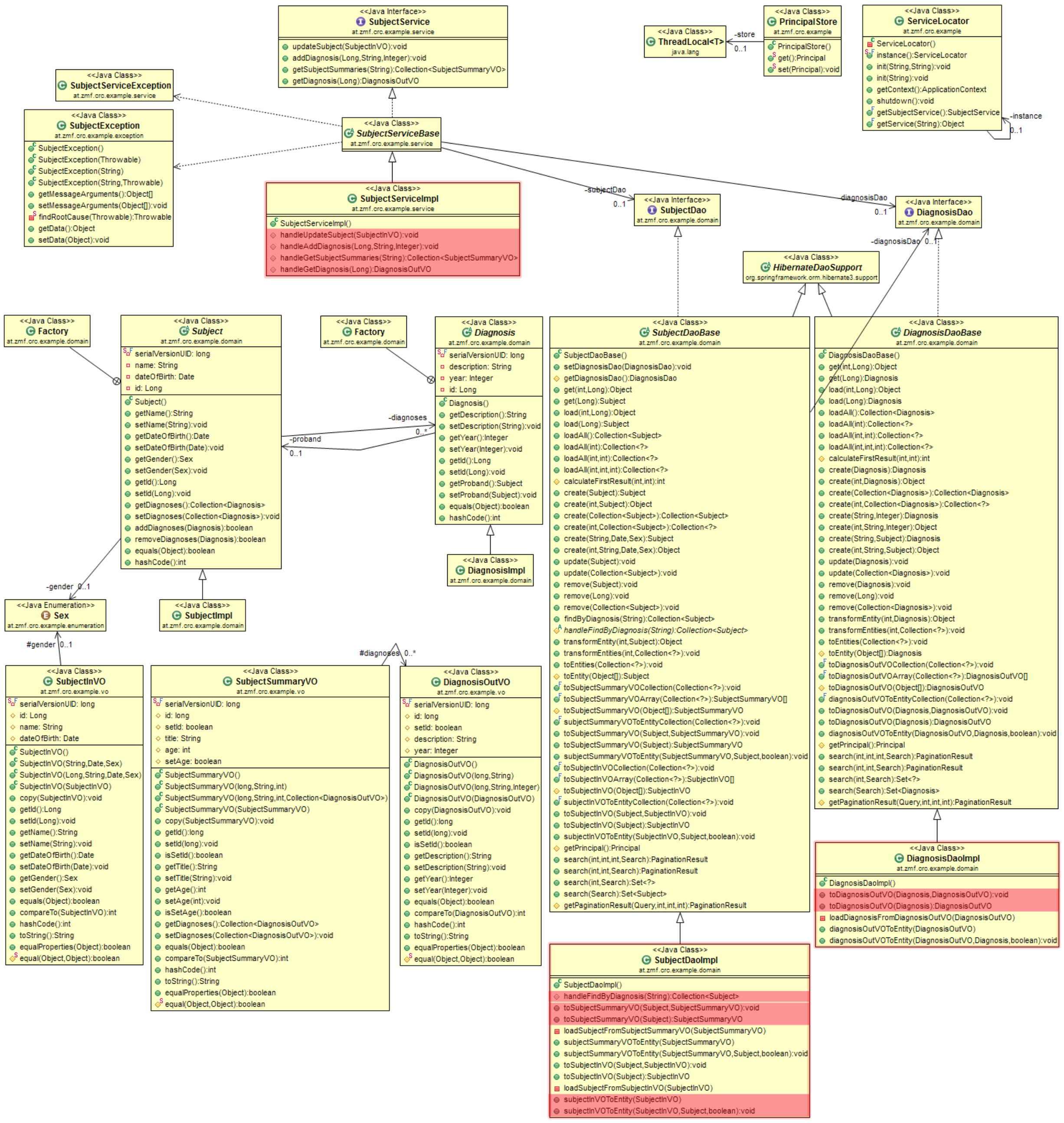
The model is created and exported into XML Metadata Interchange (XMI) format using a UML model editor such as MagicDraw (No Magic, 2014). AndroMDA loads and validates the model before starting to generate artefacts of different type:

- Java source files (figure 2.19)
- Hibernate mappings (`.hbm` files)
- Spring configuration files (`applicationContext.xml`)

The generated code base shown in figure 2.19 represents the application's complete service and DAO layers. Although it is consistent and can be compiled directly, generated service method handler stubs will throw `UnsupportedOperationExceptions` when invoked. A MDD iteration is completed by replacing stub code with hand-crafted application logic. A simple service method implementation is given by the example of `SubjectServiceImpl.addDiagnosis` in listing 2.4.

```java
/**
 * This is only generated once! It will never be overwritten.
 * You can (and have to!) safely modify it by hand.
 * TEMPLATE:     SpringServiceImpl.vsl in andromda-spring cartridge
 * MODEL CLASS:  Data::example::at.zmf.crc.example::service::SubjectService
 * STEREOTYPE:   Service
 */
package at.zmf.crc.example.service;

public class SubjectServiceImpl extends SubjectServiceBase {

    ...

    //handler for the SubjectService.addDiagnosis service method
    protected void handleAddDiagnosis(Long subjectId, String description, Integer year) throws Exception {

        //business logic for inserting another Diagnosis record of an existing subject:

        Diagnosis newDiagnosis = Diagnosis.Factory.newInstance(); //create new managed entity instance
        newDiagnosis.setId(null); //the entity ID will be generated according to the generator strategy Hibernate is configured
              to (UUID, PostgreSQL sequence, etc.)
        newDiagnosis.setDescription(description);
        newDiagnosis.setYear(year);

        //for bidirectional navigatable associations, both associaion ends have to be 'maintained':
        Subject subject = this.getSubjectDao().load(subjectId);
        if (subject == null) { //input validation
            throw new SubjectException("unknown subject with ID " + subjectId);
        }
        newDiagnosis.setProband(subject);
        subject.addDiagnoses(newDiagnosis);

        this.getDiagnosisDao().create(newDiagnosis);
        Logger.getLogger(SubjectService.class).info("new diagnosis record created - ID " + newDiagnosis.getId());

    }

    ...
```

Listing 2.4: Business logic will actually persist data passed by service layer consumers. Executing this service method implementation example of `SubjectServiceImpl.addDiagnosis` from figure 2.19 results in inserting another diagnosis record for an existing subject into the `public.diagnosis` database table from listing 2.2.



<<Java Interface>> SubjectService at.zmf.crc.example.service
<<Java Class>> ThreadLocal<T> java.lang
-store 0..1
<<Java Class>> PrincipalStore at.zmf.crc.example
<<Java Class>> ServiceLocator at.zmf.crc.example
-instance 0..1
<<Java Class>> SubjectServiceException at.zmf.crc.example.service
<<Java Class>> SubjectException at.zmf.crc.example.exception
<<Java Class>> SubjectServiceBase at.zmf.crc.example.service
<<Java Class>> SubjectServiceImpl at.zmf.crc.example.service
-subjectDao 0..1
<<Java Interface>> SubjectDao at.zmf.crc.example.domain
-diagnosisDao 0..1
<<Java Interface>> DiagnosisDao at.zmf.crc.example.domain
<<Java Class>> HibernateDaoSupport org.springframework.orm.hibernate3.support
-diagnosisDao 0..1
<<Java Class>> Factory at.zmf.crc.example.domain
<<Java Class>> Subject at.zmf.crc.example.domain
<<Java Class>> Factory at.zmf.crc.example.domain
<<Java Class>> Diagnosis at.zmf.crc.example.domain
-diagnoses 0..*
-proband 0..1
<<Java Class>> SubjectDaoBase at.zmf.crc.example.domain
<<Java Class>> DiagnosisDaoBase at.zmf.crc.example.domain
<<Java Class>> DiagnosisImpl at.zmf.crc.example.domain
-gender 0..1
<<Java Enumeration>> Sex at.zmf.crc.example.enumeration
<<Java Class>> SubjectImpl at.zmf.crc.example.domain
#gender 0..1
#diagnoses 0..*
<<Java Class>> SubjectInVO at.zmf.crc.example.vo
<<Java Class>> SubjectSummaryVO at.zmf.crc.example.vo
<<Java Class>> DiagnosisOutVO at.zmf.crc.example.vo
<<Java Class>> DiagnosisDaoImpl at.zmf.crc.example.domain
<<Java Class>> SubjectDaoImpl at.zmf.crc.example.domain

Figure 2.19.: This Java class diagram shows service and DAO layer artefacts, as they are generated by a vanilla[25] AndroMDA setup when processing the UML model from figure 2.17. Source files for classes with rose frames are generated only once, if they do not exist in outlet folders yet. Relevant method stubs (rose background) have to be completed by adding hand-crafted code. The organisation into Java packages is derived from package names specified in the model (figure 2.18).

[25] AndroMDA 3.4.14978 without template overrides and merge mappings.

3. Design

3.1. Requirement Analysis

The conceptual formulation of the project described in this work states the mission to create and introduce a custom database application for mapping enterprise processes in the domain of clinical trials. More specifically, the system should cover all major processes at the CRC trial site, which are harmonized with relevant legislation and defined in the course of the ISO certification. Starting from scratch (*green field*), the system will be powered by a conventional RDBMS and accessed via a web UI. These were the fundamental starting points for the requirement analysis, with the goal of compiling a preliminary description of the desired system, irrespective of a concrete technology stack or the extent of an implementation.

In the beginning, workflows of CRC employees were analyzed during a few scheduled team sessions in order to gain an overview of the domain and the current situation. Between sessions, the information gathered was assessed. Conceptual models and corresponding data required to map the processes were identified and described in textual form using a clear language, including the vision of usage and limitations. These results were discussed and refined in upcoming sessions to reveal more precise requirements finally. Aside regular job activities, CRC staff became aware of the project size, their roles as prospective users and stakeholders in a software project. In the end, a requirement analysis document (appendix B) with a moderate level of detail was available as a consensus and basis of the subsequent development steps. It outlines basic characteristics of the software project:

Domain-driven design An abstract or physical object of the real world relevant to the business will be effectively represented by a record of a database table. In terms of UML class diagrams, these objects (*domain entities*) relate to each other, forming *data models*. Data models are clear compartments in a hierarchy representing the system's overall data model, e.g.:

- A proband has zero or more addresses.
- An employee has zero or more assigned duties.
- A trial record has n (zero or more) investigational visits and m (zero or more) subject groups. A regular trial might have nm appointments for visits.

Data models should be simple and delimitable to be intuitive to users, since they will manipulate the contained data in order to document, coordinate and overview activities according to their particular job responsibilities.

Data-driven application A data-driven application is characterized by its main purposes:

1. manipulation of records by means of CRUD operations
2. retrieval and presentation of records in an appropriate format such as lists, documents, charts, calendar views . . .

Data processing such as input validation or record selection might require a high level of detail but relies on straight-forward techniques after all.

No dedicated user roles The CRC is still a growing department with changing and shared responsibilities of employees. Roles in projects may vary for each clinical trial conducted. User roles were therefore hard to separate, notably at the time of the beginning of the Phoenix CTMS project. However, privileged ("admin") and non-privileged ("regular" user) roles were considered for the authorisation of data manipulations per data model.

No use case and activity diagrams Most of the workflows supported comprise a sequence of data entries, which are allowed to be incomplete or performed in parallel or varying order. The reason for this is to provide freedom for users and flexibility regarding pre-conditions such as the extent or type of a particular clinical trial. This would result in complex UML activity diagrams, which are less useful for documentation purposes. The same holds for UML use case diagrams due to the lack of definite role names.

Generalisation A specification is considered as a detailed definition of a requirement including constraints, e.g.

- *"The system must support unified CV documents of employees, as given by examples of CV's from employees A, B or C . . . "*
- *"The system must allow user to select recruited subjects according to a trial's eligibility criteria, as given by examples of trial protocols from trial X, Y or Z . . . "*

The requirement analysis document denotes details of such specifications for clarification of particular requirements, but anticipates variations and incompleteness. According to the "decide as late as possible" credo of *lean development* (M. Poppendieck and T. Poppendieck, 2003, p. 52), the application should therefore be adjusted by either application users themselves or by means of *configuration*, when final specifications are known exactly or come into effect. Apart from that, an overall design goal kept in mind is flexibility. Satisfying future change requests or adaption to other trial sites should be possible with low effort. These aspects mentioned imply the tendence towards generalisation of program logic wherever applicable, e.g. by using elaborately techniques such as *metamodels* (e.g. user-defined input forms) and Domain Specific Languages (DSLs) (e.g. query language, form scripting).

3.2. System Modules

The requirement analysis introduces a global separation of features into *system modules*, each grouping related data models. From the very beginning, users see modules as partitions of the whole database schema, with a corresponding scope of responsibility. There is a high cohesion between domain entities of a module, but relations between entities of different module exists as well (figure 3.1).

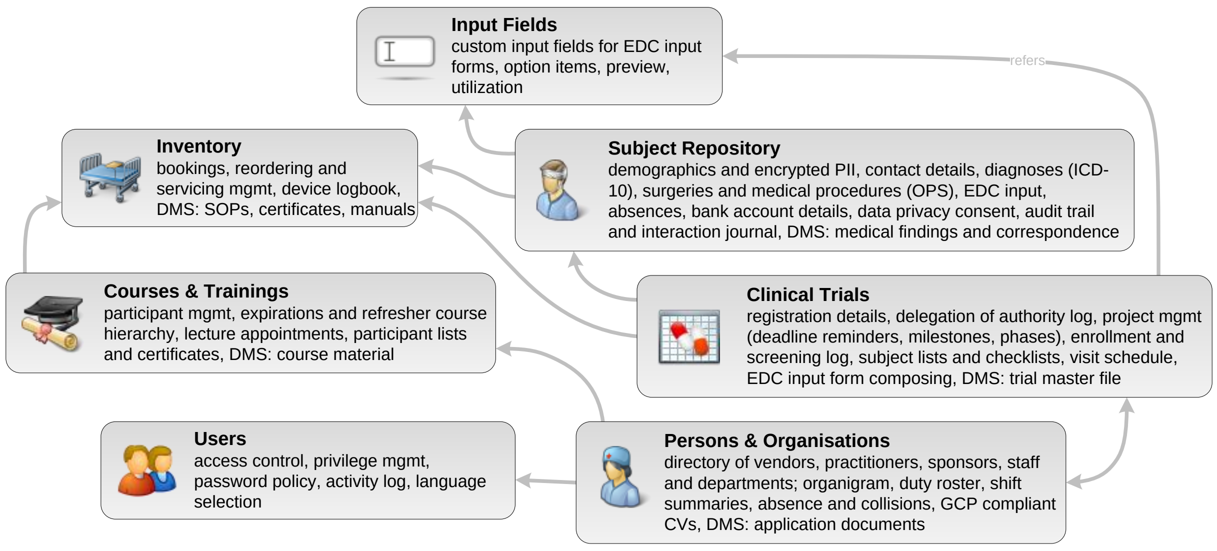

Input Fields
custom input fields for EDC input forms, option items, preview, utilization
refers
Inventory
bookings, reordering and servicing mgmt, device logbook, DMS: SOPs, certificates, manuals
Subject Repository
demographics and encrypted PII, contact details, diagnoses (ICD-10), surgeries and medical procedures (OPS), EDC input, absences, bank account details, data privacy consent, audit trail and interaction journal, DMS: medical findings and correspondence
Courses & Trainings
participant mgmt, expirations and refresher course hierarchy, lecture appointments, participant lists and certificates, DMS: course material
Clinical Trials
registration details, delegation of authority log, project mgmt (deadline reminders, milestones, phases), enrollment and screening log, subject lists and checklists, visit schedule, EDC input form composing, DMS: trial master file
Users
access control, privilege mgmt, password policy, activity log, language selection
Persons & Organisations
directory of vendors, practitioners, sponsors, staff and departments; organigram, duty roster, shift summaries, absence and collisions, GCP compliant CVs, DMS: application documents

Figure 3.1.: The Phoenix CTMS consists of an extensible number of database modules. Each module integrates features related to a pivotal domain entity (root entity) such as a proband or clinical trial. Gray lines denote major referrals between modules. For instance, a trial's list of participating subjects is mapped by referencing registered subjects.

The following sections will give a compact, retrospective overview of system modules in English language. In contrast to the requirement analysis document, descriptions reflect the final implementation, after refinement iterations. Approximately each sentence reflects a detailed requirement.

3.2.1. Inventory

The inventory module allows managing premises, materials, devices, and other equipment available to the trial site. Inventory objects can be structured by a tree hierarchy. Each item can be tagged with relevant details such as product name, model, serial number or other identifiers. If the item is marked "bookable", reservations can be created and displayed in an interactive calendar view. By default, a bookable inventory item's limit for multiple occupancy (overlapping bookings) is set to 1. A booking can be associated with a proband and/or a trial and/or a course. Reminders for recurrent maintenance measures such as inspections or calibrations can be registered to notify the responsible person via email in time. A status can be logged if a device is not disposable due to repair, loan etc. During this period, users are advised of an item's unavailability the when creating bookings. Documents such as manuals, service certificates, bills or SOPs with device usage instructions can be uploaded. While SOP documents might be provided by a corporate Content Management System (CMS), manuals or customer support information can be available from the vendor's web site. Corresponding hyperlinks can be bookmarked per inventory item in these cases.

3.2.2. Persons and Organisations

A database for involved persons and organisations maps employees, departments and institutions. A tree hierarchy is used to reflect the organisational structure. Aside, third parties such as contractors, vendors or practitioners can be registered. Persons can be tagged with relevant details such as badge number or job description, while different properties apply for organisations, such as a Value Added Tax (VAT) number for companies. It is possible to store contact details such as phone number or email addresses, which can be marked to receive relevant notification emails generated by the system. If a person is marked as employee, absences (vacation, sick leave etc.) can be tracked. If the employee is marked “allocable”, a dispatcher can assign duties of ongoing trials. Employees are notified of new, swapped or withdrawn assignments via email. Shift summaries show overviews of working hours to facilitate a fair assignment by weighting holiday and night shifts. The duty roster is presented by a separate calendar view, where collisions with employee absences are indicated and users are able to assign themselves to duties planned in advance. For persons, a uniform CV document in PDF format can be generated for download. CV positions are specified in a structured way. One of the postal addresses provided can be marked as business address to be printed on the CV, unless the address of a superseding person or organisation is to be used. The system’s web UI supports capturing a photo for the CV directly, if a web cam is available. Documents such as application letters or certificates can be uploaded using the file manager and retrieved as a merged PDF document.

3.2.3. Courses and Trainings

Courses and trainings are held by educated trial site staff in order to prepare remaining employees (students) in charge for trial specific tasks or teach applicable regulations. Some courses may have a limited validity period, requiring participants to attend a refresher course when expired. The emerging hierarchy of refresher courses is depicted by tree views. In order to schedule lecture appointments, rooms can be booked. A denominated institution and the training team members are registered and printed on the attendance certificate document, which can be generated for each participant. Course participations are managed by an invitation-acknowledge protocol, which stops with an obligatory assessment result (passed/failed). Training team members and participants are notified of participation state transitions by email. The course can be set up to show up in the participants’ CV positions unless failed or cancelled. A participant list can be printed, which is to be own-hand signed per participant per lecture appointment. If the course is marked to allow self-registration, it shows up in a list of upcoming courses, where participants can register themselves. Course material can be uploaded or bookmarked if available online.

3.2.4. Clinical Trials

The trial module is designed for managing of independent, controlled clinical trials, conducted at a trial site in parallel. The lifecycle of each trial is mapped by a walkthrough of predefined states. A state transition can cause an action to be performed such as creating a digital signature value of the trial’s entire object graph. Lockdown states will render the UI in readonly mode. Trials can be categorized by the type of investigation/investigational product and the sponsoring type. Each trial can be tagged with details such as registration codes assigned during the formal trial registration processes. The trial team comprises involved persons and organisations. For the team member list to meet the “delegation of authority log”, roles are assigned and access to the trial is controlled per member in the style of an Access Control List (ACL). The project management aspect is considered by predefined and individual events and phases with an optional reminder notification each. Email notification of trial state changes and event reminders can be enabled per team member. Events, phases and visit appointments of concurrent trials can be surveyed in an interactive timeline view. After a number of visits (as specified by the trial protocol) and a number of proband groups (as considered by the trial coordinators[1]) are set up, a visit schedule can be created. The visit schedule consists of visit appointments, defining a time period for each combination of visit and group. Aside the timeline view, visit appointments are displayed in the duty roster calendar view in order to align created duties. Every single duty associated with the trial can be marked to allow self-allocation of employee users. The trial’s self-allocable duties can be fixated up

[1]Groups of subjects serve an organisational purpose and are not to be mistaken with verum and control groups, whose allocation is kept secret by the sponsor in the case of double-blinded trials.

to a given date. This date will be increased by the dispatching user in stages, whenever a new batch of duties is going to be prepared for the trial. Inventory allocated to the trial is listed in another inventory bookings view.

The enrollment of subject is effectively managed by the "proband list". Candidate subjects are selected by querying the proband database and can be added in *reproducible random order* subsequently. If a blocking period is specified for the trial, the system will reject probands, who participated in other trials managed by the system during this time period before. The query for candidates typically reflects a coarse definition of the trial's eligibility criteria given by the trial protocol. Since criteria attributes will vary from trial to trial, an individual inquiry input form can be composed per trial. A preview allows testing the input form in advance. When registering a new proband entry during recruitment, data is captured by entering answers to questions and values of parameters presented in the inquiry form (first EDC use case). After adding a matching subject to the trial's proband list, it will walk through enrollment states, starting with the "candidate" state. The enrollment progress is visualized by an enrollment chart, showing the distribution of the subjects' status over time. Each subject is assigned to a proband group, so relevant appointments of the visit schedule can be listed and checked for collision with a proband's availability. User-defined checklists are supported by means of individual columns, which can be added to the trial's proband list (second EDC use case).

Hyperlinks can be stored to bookmark SOPs or other trial-related material maintained in the corporate CMS. Selected (or all) documents of the "trial master file" can be made available online to Phoenix CTMS users by uploading or a file system import.

3.2.5. Subject Repository

The proband module is a database for subjects recruited by the trial site. PII captured comprises basic data such as

- name and Date of Birth (DoB)
- a (web cam) photo for proof of identity
- contact details such as email addresses and phone numbers
- postal addresses

Entering postal addresses works in the same manner as for persons and organisations and can be backed by a directory of street names and postal codes (ZIP codes), if available. Views for a proband's absences and appointments of the visit schedule indicate collisions when the proband is participating a trial. Aside individual inquiry forms of trials, patient history can be coded by means of International Classification of Diseases (ICD-10) diagnoses and surgeries/medical procedures (Operationen- und Prozedurenschlüssel (OPS)). Information about bank institutions can be backed by a corresponding directory when entering bank account information. The bank account information will be used for printing final invoice documents. Medical findings and scanned documents such as the own-hand signed data privacy consent can be uploaded. The data privacy consent is part of the proband letter, which can be generated as a mail merge document. Subjects registered recently will be deleted by the system after four weeks, unless the returned data privacy consent was saved. Pending deletions are notified by email in time and can be overviewed and dismissed temporarily in a separate list view. In these cases, the aforegoing interactions with the proband (phone, email) can be manually logged in a journal.

3.2.6. Input Fields

Input fields used for composing inquiry forms (first EDC use case) and appending individual columns to the proband list (second EDC use case) are managed by the input field module. To define questions or parameter inputs, users can choose from various input types such as text, integer, multiple choice etc. Range checks and user-defined error messages are supported for any field type. Option items of selection input types can be created and updated as required. A special feature is the sketch input field type for the use of pen-enabled devices. A sketch input allows marking user-defined regions of an uploaded background image in order to choose options when capturing data such as skin lesions or sites, which can be queried for afterwards. A preview allows testing a created input field in advance. The utilization of an input field such as the number of values captured with it is broken down in list views, which can be reviewed before manipulating items that already have entered values associated to it.

To automate populating dependent values when capturing data using the prepared input form, the UI allows to develop compact JavaScript functions to express relationships of variables (form scripting). They are used to calculate values, e.g. the Body Mass Index (BMI) from given body weight and size inputs.

3.2.7. Users

User accounts are managed separately in the user module. Each user account can be optionally linked to a person or organisation, which is referred to as *identity*[2]. Each user account has its own language, timezone and UI theme setting. User settings and password can be changed from a menu accessible in any UI screen. Configurable password policies can be adjusted to meet common corporate guidelines. An impending password expiration is displayed to the user and notified by email. Permissions are pre-configured to constitute user roles appropriate for the trial site and can be assigned or revoked with instant effect at any time.

3.2.8. Universal Features

- As mentioned, most modules facilitate uploading and managing of document files. These features can be accessed by file managers as known from modern Document Management Systems (DMSs). The file manager UI allows handling large and numerous files of various file types of file *repositories*. A repository represents the virtual file system of a module's root entity item. Preset folder structures can be pre-configured per system module. Files can be instantly searched for and merged for download in PDF format. Data consistency is always assured in face of simultaneous access. Instead of uploading, directories containing a large number of files can be imported from the server file system in order to make them available online.
- A query editor is provided to compose and execute complex queries instantly. It constrains structured SQL-like database queries using criterion term as building blocks and guides users with syntax highlighting and locations of syntactic or semantic errors. Virtually any object attribute is available as a query criterion, meaning users can search for anything they ever entered into the system. Queries can be saved in order to provide them to other users. The proband module's query editor allows expressing a trial's eligibility criteria to retrieve matching subject candidates.
- A journal available to items of each module shows the history of changes, which are logged by the system (audit trail). In addition, manual entries can be created to log events related to the item.
- Independent of journal system messages, notifications of events such as reminders are generated for individual recipients. Selected notifications will be sent to enabled email addresses of recipient identities. The history of notifictions can be browsed and obsolete notifications are revoked.
- Multi-client capability is supported throughout all modules by the separation into *departments* and the option to configure global and department-level user roles. Each department corresponds to a trial site. The sovereignty of PII stored in the proband database is given by cryptographic means for each department.

[2]By design, a person (e.g. an employee) or an organisation (e.g. an institution) can therefore have multiple user accounts.

3.3. Non-Functional Requirements

category	**Non-Functional Requirement (NFR)**	**addressed by**
regulatory compliance	data privacy	data privacy consent for trial subjects
	traceability	audit trail (system messages), error logging
	structured development	development process, MDD
data security	access control	password-based authentication, session timeout
	password requirements, validity	password policy
	authorisation	user privileges, host-based, ACL, instant revocation
	secure storage	database and document file encryption
	secure data transfer	Secure Sockets Layer (SSL) with server certificate
usability	Internationalisation (i18n)	UI language and timezone per user
	consistent UI look and feel	page templates, UI component framework, UI themes, date formatting, uniform naming, icons
	conformance	UI component framework, master-detail views, response messages
	self-describing	tooltip texts
compatibility	platform independency	JVM (OS), Hibernate abstraction (RDBMS)
	state-of-the-art standard software	de-facto standard frameworks and libraries
	open-source software	PostgreSQL over Oracle Database
risk management	prevention of misuse	input validation, range checks, privileges
	avoidance of misuse	UI highlighting, preview for input fields
	data theft	Password-Based Encryption (PBE), host-based authorisation
	Denial-of-Service (DoS)	corporate perimeter firewall, fronting Apache HTTP server
accessability	reachable from the internet	public HTTP server
	support for all major browsers	fallbacks provided by the UI component framework, custom fallbacks
	no mandatory browser plug-ins	HTML5, SVG support
architecture	extendability	additional system modules, exports (Excel, PDF), web service for client applications
	testable	test data generator, code generation for unit test stubs
	multi-client capability	logical separation of multiple trial sites (departments)
	scaling options	multi-tiered architecture
	single server	monolithic default deployment
adaptability	user-definable	custom input fields, composing input forms, query editor
	configuration	privilege definitions, criterion term column definitions (query editor), state diagrams, actions for item status transitions
reliability	24/7 availability	Nagios monitoring, maintenance window announcements
	disaster recovery plan	history of database dumps, daily backup, setup manual (appendix C)
	external dependencies	asynchronous processing when connecting to services (e.g. Simple Mail Transfer Protocol (SMTP) server), local master data instead of external web services
	self-recovery	operational state after server booting
capacity	controlled data growth	production data growth of $\tilde{1}$ GB per year per department[3], scheduled cleanup tasks
	stable performance	constant UI response times (lazy loading)

Table 3.1.: Table of considered NFRs. While some of them are already covered by the design of modules, others can be found for example in (Firesmith, 2003), (*ISO/IEC 9126-1 —Software Engineering —Product Quality* 2001) or (*IEEE Recommended Practice for Software Requirements Specifications* 1998).

[3]Assumptions: (1) the trial master file directory is not managed using the DMS feature, (2) a reference department comprises 10-20 of simultaneously active users

Apart from functional requirements of the system's modules, a couple of Non-Functional Requirements (NFRs) were presumed (table 3.1). Implementing these NFRs will ensure the achievment of goals listed below and constitute a software system of high quality after all:

1. regulatory compliance
2. user satisfaction and smooth operation
3. suffice general accepted standards and guidelines
4. suffice standards and guidelines dictated by CRC, Zentraler Informatikdienst (ZID) and JR Health

3.4. Development Process

An agile development process was established with the initial requirement analysis as a starting point. The extensive implementation was performed in iterations for each requirement or consolidated group of related requirements. Iterations include the continuous involvement of user feedback typical for the *user-centered design* approach (Hussain, Slany, and Andreas Holzinger, 2009, p. 424). The development process represents the essential part of the entire Systems Development Life Cycle (SDLC) shown in figure 3.2.

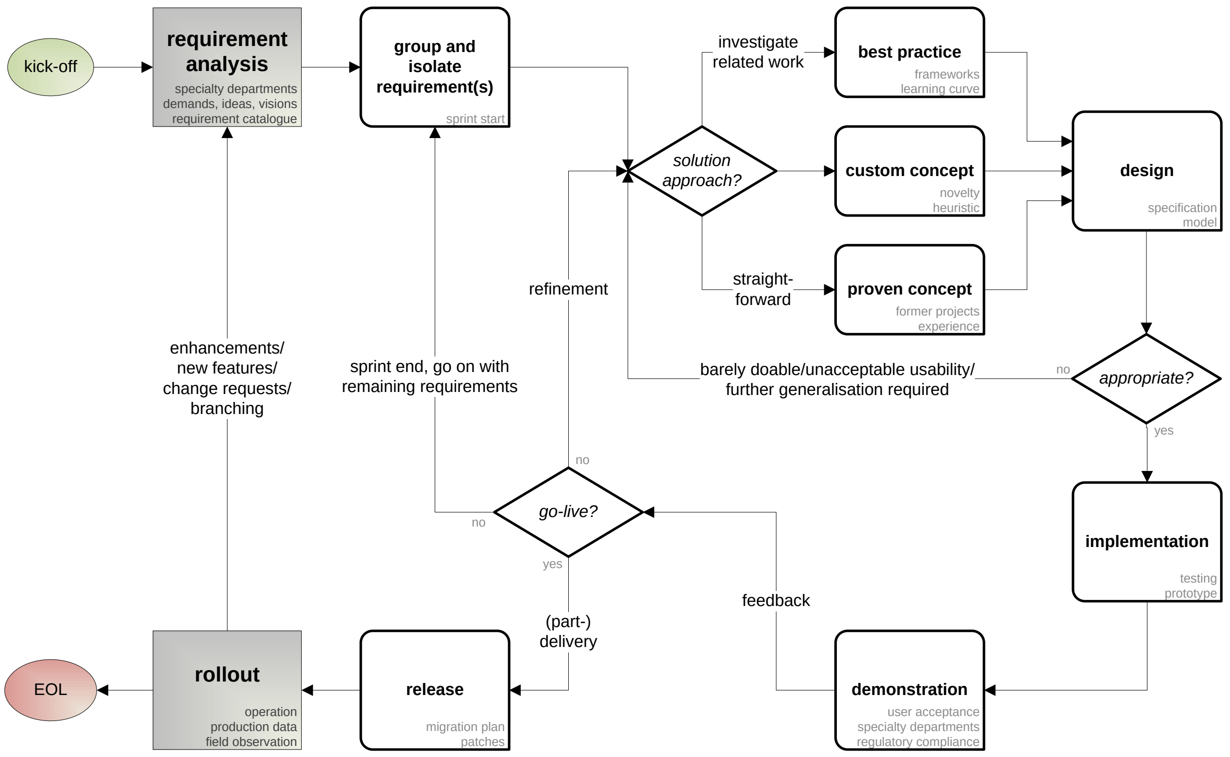


Figure 3.2.: The release cycles on top of the development process start with a new or updated requirement catalogue and ends with the rollout of the new version as a result.

A development iteration starts by picking associated requirements from the requirement analysis document (catalogue). Representative examples given below show approaches when elaborating the design for a detail solution:

1. best practice:
   - AOP/interceptors
   - ICD-10 for encoding diagnoses
2. custom concepts:
   - service method level authorisation
   - a grammar for database queries capable of expressing common eligibility criteria
3. proven concepts:
   - calculating weighted shift durations
   - consumer-producer pattern for asynchronous operations

Detail solutions are already considered in the requirement analysis document to some degree. A design for a particular solution essentially represents to some kind of model (e.g. domain entities, processing model), which implies a detailed specification (e.g. entity fields, algorithm) coming along with potential limitations. An implementation attempt starts if it is considered doable and the design's level of generalisation will cope with expected requirement details. A *vertical prototype* is developed in form of fully functional UI screens and demonstrated in a team meeting. Use cases are shown by walkthroughs with the prototype and discussed. In particular, the compliance with regulatory requirements is validated at this point. The prototype might be refined or even completely refactored according to detailed requirements the team became aware of during discussions.[4] A new cycle starts by moving on to the next requirements otherwise. Although it requires some experience to hold steady, this communication process using prototypes helped maximizing the comprehension with a minimal number of required team meetings and was therefore preferred by stakeholders.

The integrated prototype grows incrementally with each requirement achieved. At some point, it was bundled into an initial release and rolled out for users to try it out themselves. While requirements are still in development, users can evaluate the system and formulate concrete incidents and change requests, which are tracked using Atlassian Jira (Atlassian, 2014). After exposing the system to a larger number of users, a continuous field observation (Andreas Holzinger, 2005) is established to identify misuse and bugs by means of system log analysis. With the joint decision to move on to production operation, a migration of production data must be considered whenever rolling out new releases. Rollouts can be implemented overnight most times, since it is required to upgrade the web application's system server(s) only, but not employee workstations. With the beginning of the operational phase, the development process cools down. When the system will reach end-of-life finally, decommissioning strategies can be supported by means of the exporter infrastructure provided by the application. The overall project timeline recorded (figure 3.3, 3.4, 3.5 and 3.6) shows rollouts and major iterations with varying duration.

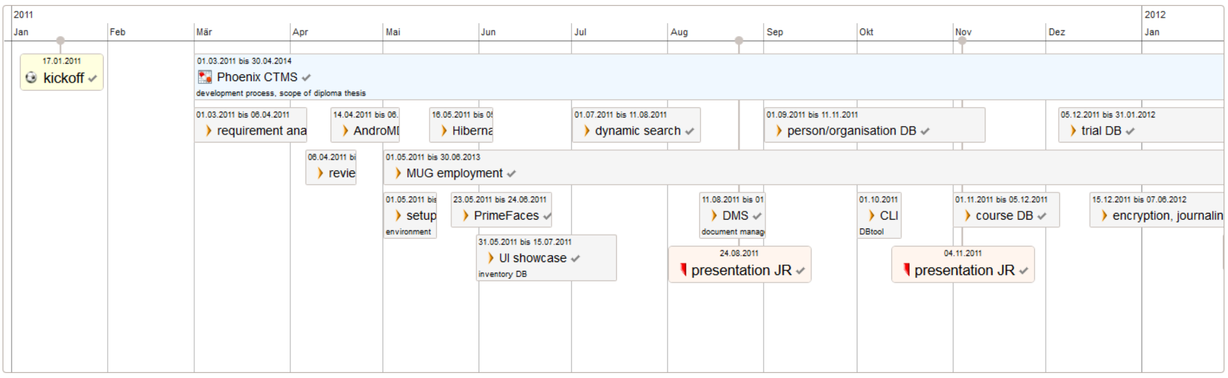


Figure 3.3.: Project timeline (2011).

[4]This is also known as *requirement churn* (Armour and Miller, 2001, pp. 282). For example, a remarkable number of requirement churns were encountered until a satisfying solution for subject data privacy consents was found.

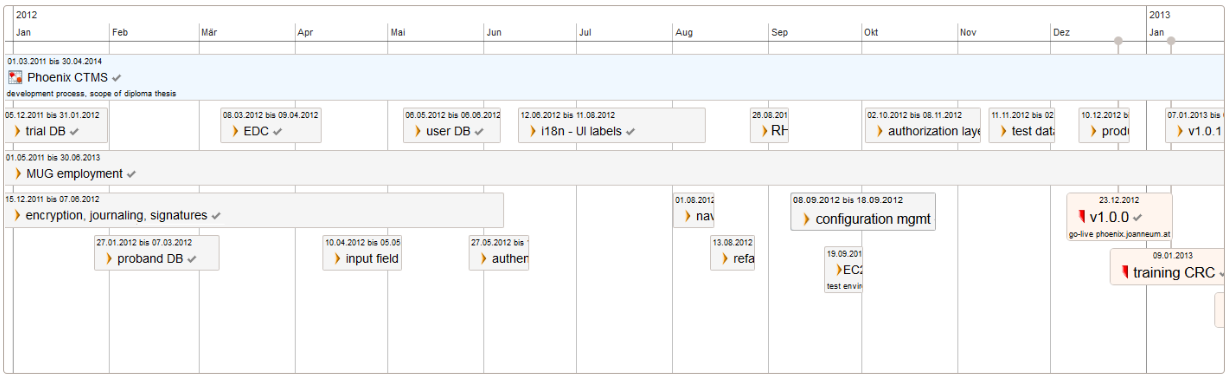


Figure 3.4.: Project timeline (2012).

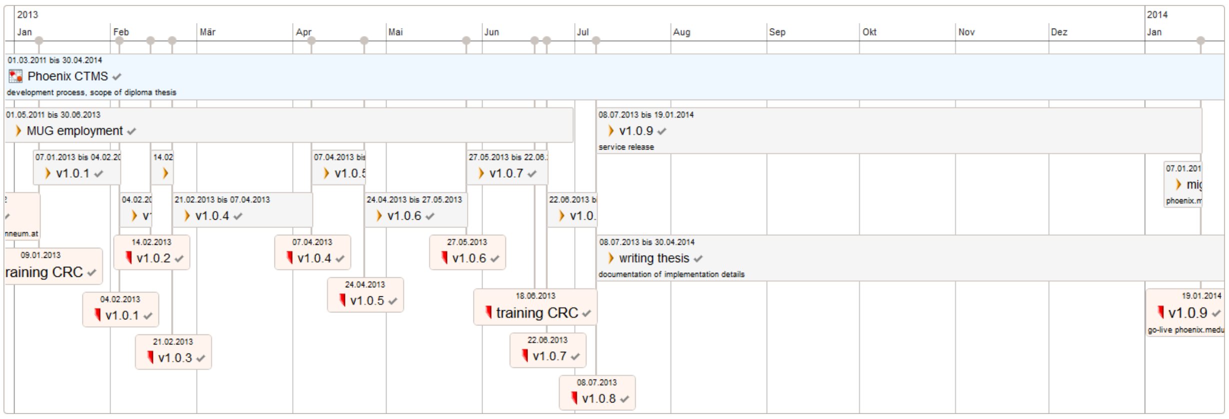


Figure 3.5.: Project timeline (2013).

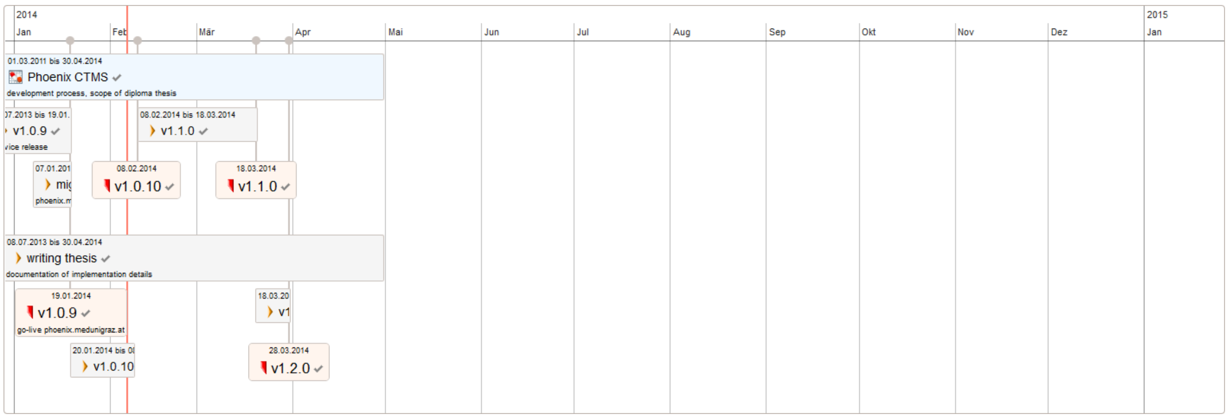


Figure 3.6.: Project timeline (2014).

4. Implementation

The implementation chapter represents the main part of this work and gives an in-depth system description. Sections 4.1-4.6 cover concepts and structures that form the foundation of the Phoenix CTMS application. Since an exhaustive documentation is not possible in the scope, most interesting features are described in detail in sections "Query Designer" (4.9), "Electronic Data Capture (EDC)" (4.8) and "Document Management" (4.7).

The following remarks and conventions apply throughout the document:

UML diagrams Extracted UML class diagrams with the background color scheme for stereotypes below are used to depict domain models, VO graphs and service interfaces:

- *Lavender blue* : domain entity
- *light orange* : VO/DTO
- *light yellow* : service interface
- *red* : application exception
- *light green* : enumeration

Class diagrams Java class diagrams (pale yellow background) are used to outline the organisation of hand-crafted application code related to a particular part of the implementation.

Listings Code excerpts are inevitable for a detailed understanding. They are simplified and therefore do not reflect actual production code in most cases. The convention below is used for background colors in order to emphasize the language of a listing:

- *pale blue* : Java code
- *pale magenta* : XML markup (Spring `applicationContext.xml`, Facelet, SVG, HTML)
- *pale orange* : Js code and JSON data
- *white* : SQL or HQL statement
- *light gray* : console command transcript

Data privacy Accompanying screenshots show the actual UI, taken from a test environment. They might depict less delicate production data such as user or employee names. However, any information regarding trial subjects is always *imaginary* since data is randomly generated.

Version Unless stated otherwise, any information refers to versions 1.0.8, 1.0.9, 1.1.0 and 1.2.0 of the system, which were operational and under development at the time of writing.

4.1. Domain Model

4.1.1. Entities and Association Topology

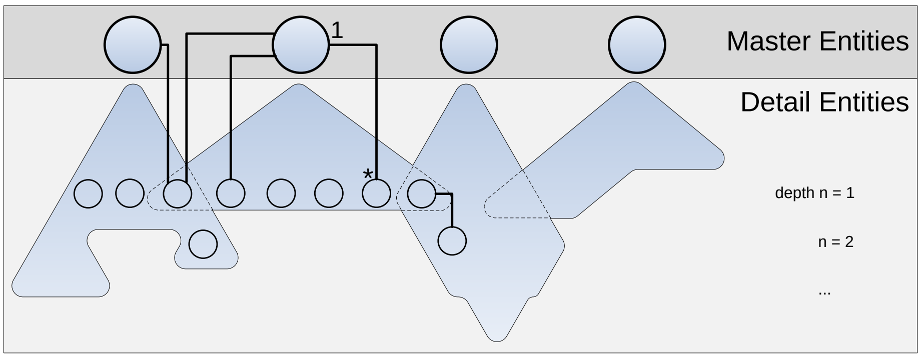


Figure 4.1.: The overall domain model's schematic topologogy shows a master-details tree per system module.

Domain entities of the Phoenix CTMS data model are structured according to its system modules. Relationships of entities belonging to a module form a coarse master-details tree (figure 4.1). A master entity is enriched with multiple *detail entities* to organize the data to be managed (table 4.1). These master entities are refered to as *root entities*.

module	**root entity**	**master-details tree**
inventory database	`Inventory`	figure 4.2
person/organisation database	`Staff`	figure 4.3
course database	`Course`	figure 4.4
trial database	`Trial`	figure 4.5
proband database	`Proband`	figure 4.6
input fields	`InputField`	figure 4.7
users	`User`	figure 4.8
database queries	`Criteria`	figure 4.9

Table 4.1.: Phoenix CTMS system modules and their corresponding root entities.

The majority of detail entities (such as `ProbandAddress`, `ProbandContactDetailValue`) refer to a particular root entity (such as `Proband`). This gives one-to-many associations that are typical of master-detail relationships (Singh et al., 2002). As a major advantage, extending this domain model structure is possible with minimal effort, e.g. by introducing another detail entity or even new root entities for additional system modules.

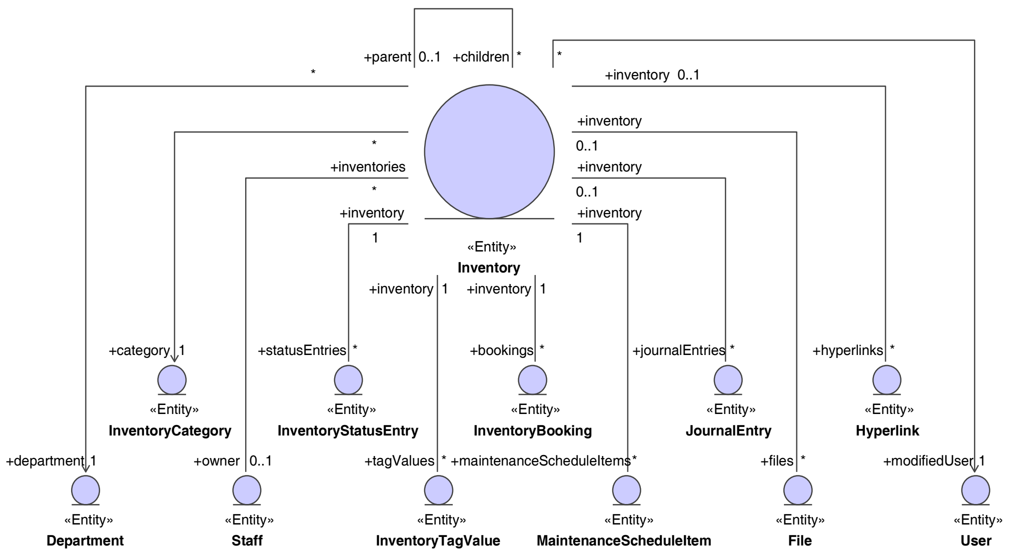


Figure 4.2.: Master-details tree of the `Inventory` root entity ($n = 1$).

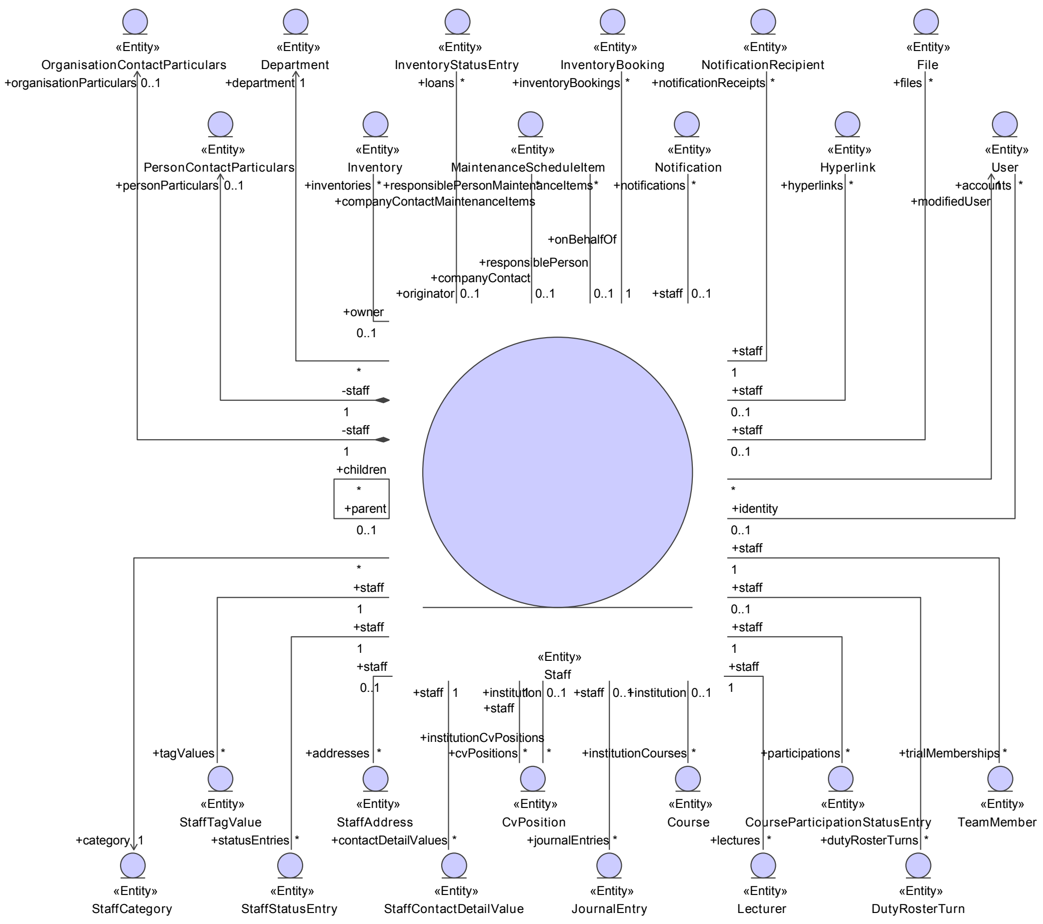


Figure 4.3.: Master-details tree of the `Staff` root entity ($n = 1$).

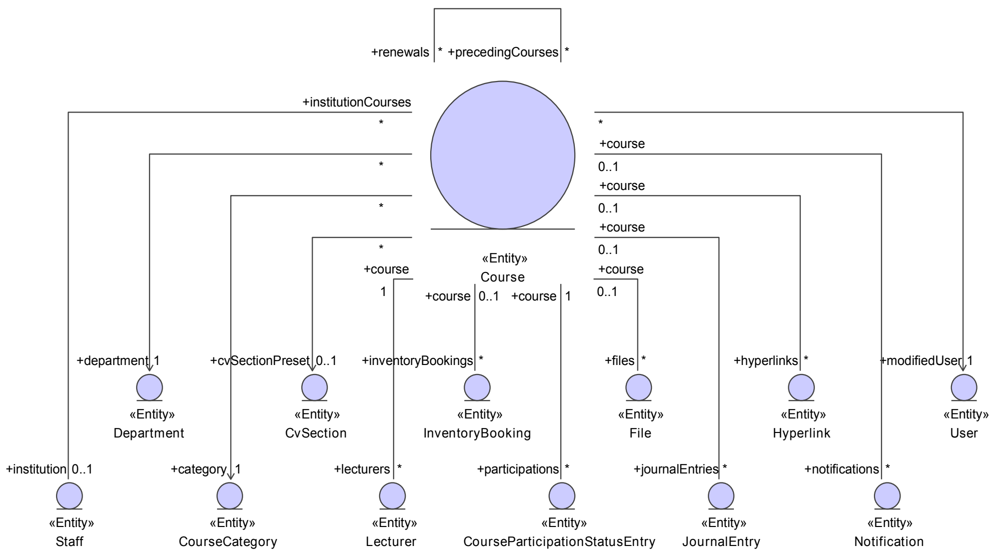


Figure 4.4.: Master-details tree of the `Course` root entity ($n = 1$).

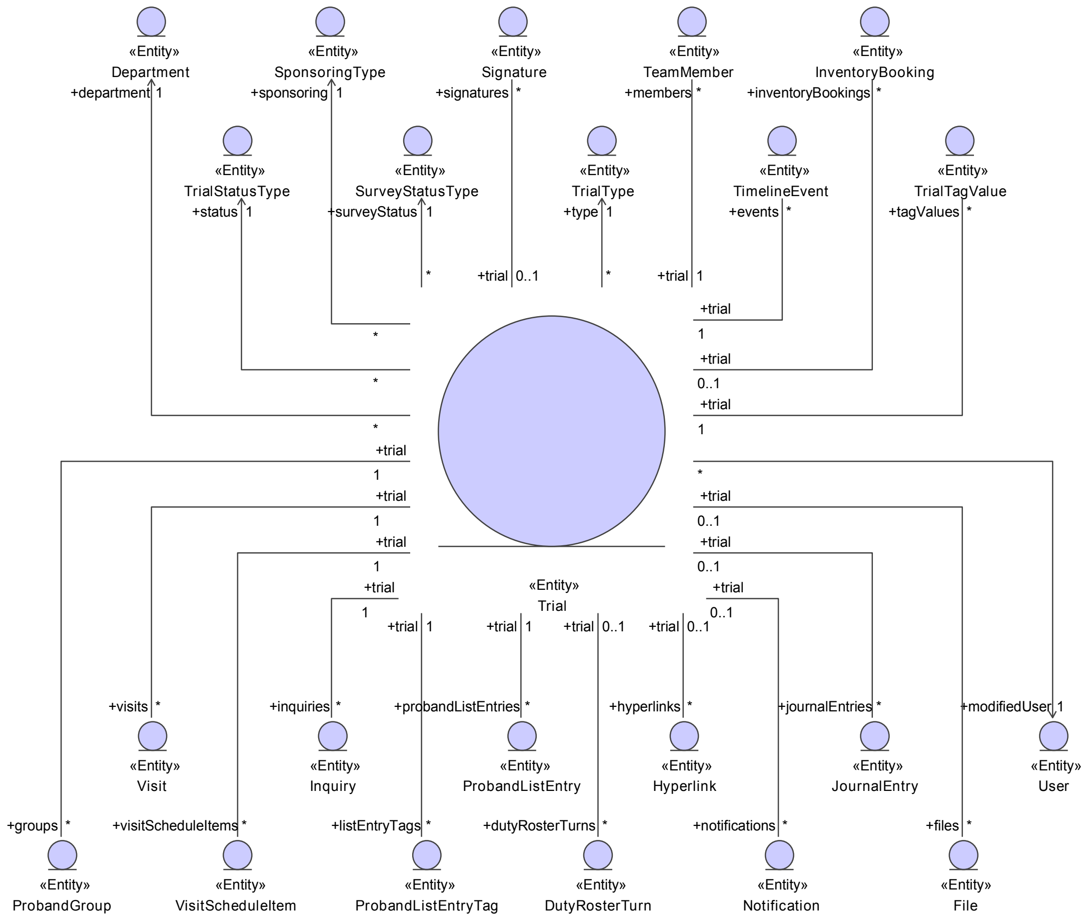


Figure 4.5.: Master-details tree of the `Trial` root entity ($n = 1$).

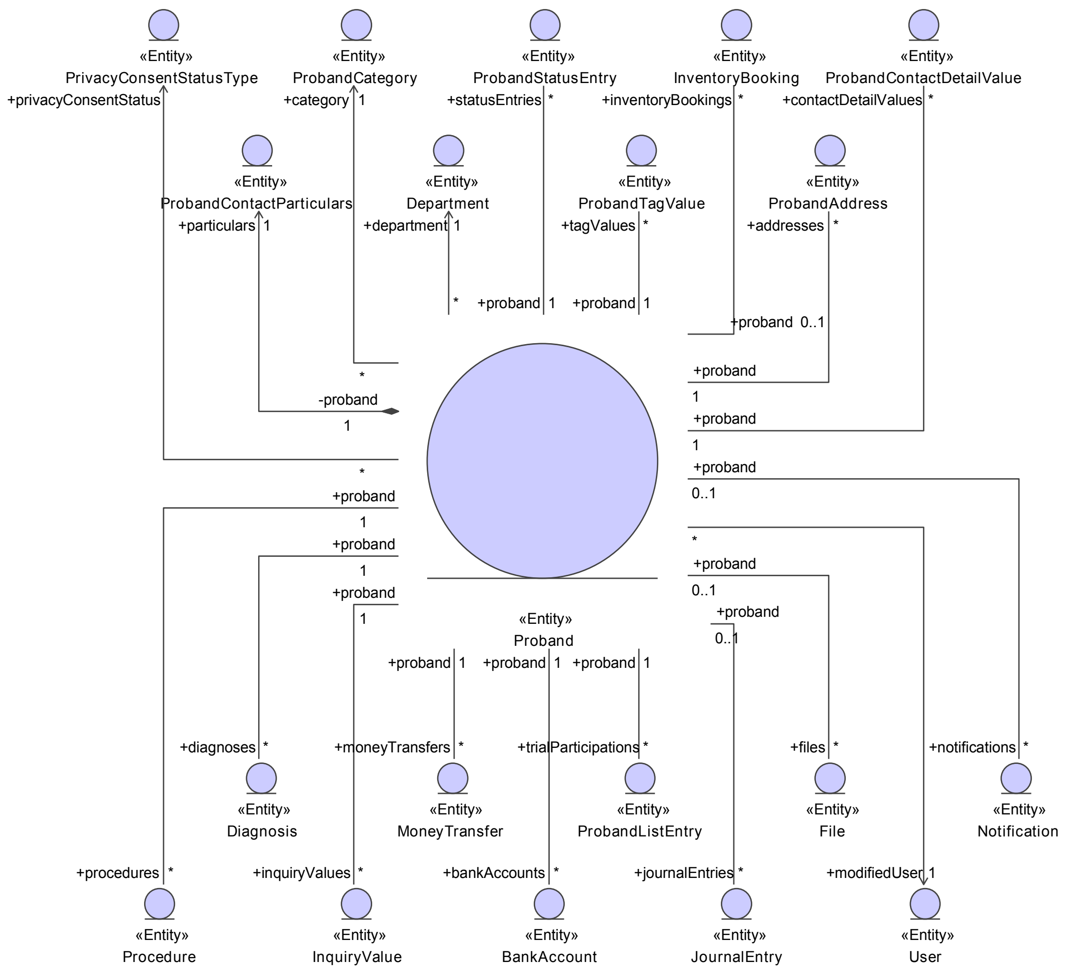


Figure 4.6.: Master-details tree of the `Proband` root entity ($n = 1$).

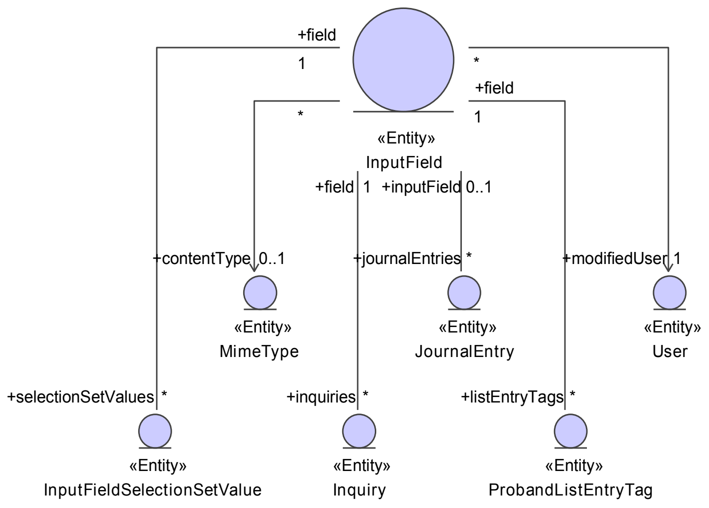


Figure 4.7.: Master-details tree of the `InputField` root entity ($n = 1$).

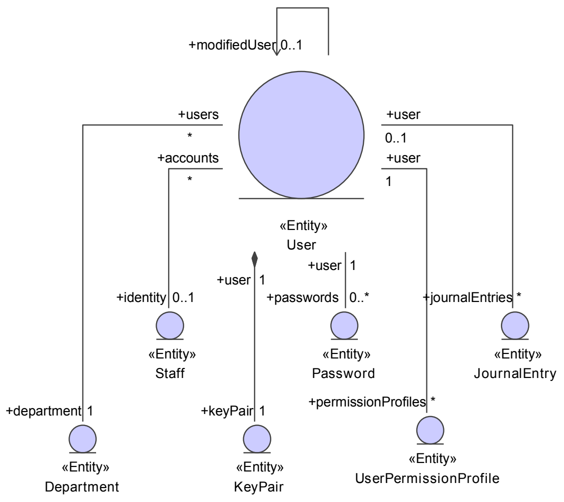


Figure 4.8.: Master-details tree of the `User` root entity ($n = 1$).

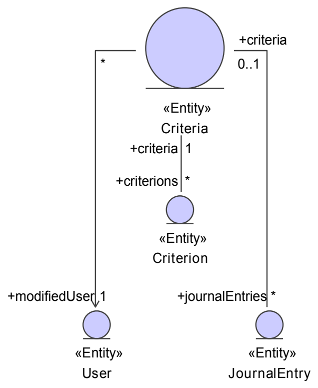


Figure 4.9.: Master-details tree of the `Criteria` root entity ($n = 1$).

The depth n of master-details tree branches will vary but should be kept small in general since database queries require a costly SQL join per involved association. For instance, a query of root entities that involves fields of a detail at depth n will require n joins (figure 4.11). Short cut associations can improve performance in crucial scenarios but mean the storage of redundant information (Prigmore, 2007).

4.1.1.1. Module Interdependency

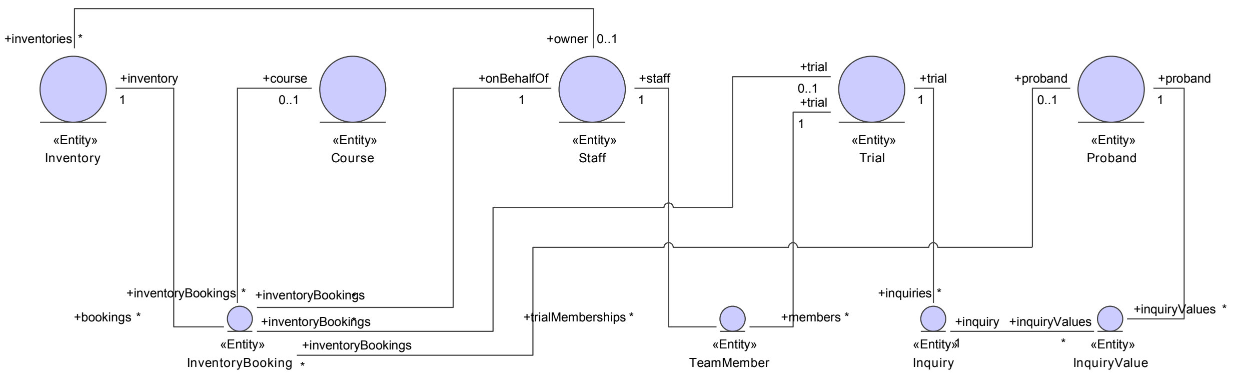


Figure 4.10.: Associations between entities of different master-details trees cause the interdependency of modules.

A closer look at master-details trees (figure 4.10) reveals the inherent interdependency of modules due to associations between them:

1. Root entities can have direct associations to other root entities. For example, to specify an inventory's owner, the inventory can refer to a person/organization.
2. Some detail entities involve two root entities, for instance the `ProbandListEntry` entity refers to a trial and a proband. This corresponds to a many-to-many association between `Trial` and `Proband`.
3. There are detail entities that refer to even more than two root entities. They can be interpreted as *n-ary* associations. An example gives the `InventoryBooking` entity, which refers to the inventory and originator (`Staff`) and optionally to a `Course`, `Trial` and/or `Proband`.

4. Ocassionally, associations exist between detail entities belonging to different root entities. For instance, `Proband`'s `InquiryValue` detail entity refers to `Trial`'s `Inquiry` detail entity. Both detail entities are part of the EAV data model described in section 4.1.1.3.

The coupling would cause problems when the data model should be divided for some reason. In contrast, service interfaces are strictly separated per system module. Nevertheless, the interdependence within the Phoenix CTMS modules is a design element, since it is vital for its use cases. Many real-life queries for finder methods rely on detail entities of category 2-4 from before. For instance, the query in figure 4.11 lists all probands for a given trial. This requires an association path which is spanned between `Proband` and `Trial`.

Query:

"*a trial's subject list*: get probands of trial 12345"

HQL:

```
from at.zmf.crc.ctsms.ProbandImpl as proband where
  proband.trialParticipations.trial.id = 12345
```

SQL:

```
select * from
  proband as p
  join proband_list_entry as ple
  join trial as t
where t.id = 12345
```

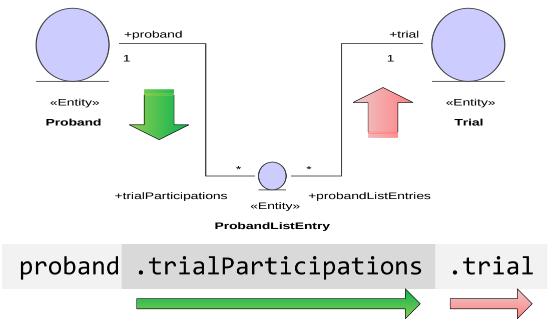


Figure 4.11.: An example query to retrieve probands of a given trial. Joins employ the `ProbandListEntry` entity, which belongs to both the `Trial` and `Proband` master-details trees.

4.1.1.2. Tree Hierarchies

Reflexive associations of root entities are used to model item hierarchies, representing *acyclic digraphs*[1]:

`Inventory` *tree:* An inventory can be a containment for others, e.g. a building contains rooms (figure 4.12). The tree is built by picking the *parent inventory* of an inventory.

`Staff` *tree:* A person/organization can have a superordinate person/organization, forming the hierarchy typically illustrated by an organigram (figure 4.13). The tree is built by picking the *parent person/organisation* of an person/organisation.

`Course` *trees:* An (expiring) course can have multiple refresher courses (tree of renewal courses). On the other hand, the course can represent a refresher course for multiple preceding courses itself (tree of preceding courses). These two trees (figure 4.14) emerge by picking *preceding courses* for the selected course, which represents the root node of both trees. For the absence of cycles across the graph that results when combining both trees, a course child node must appear in either tree exclusively.

A reflexive association represents a cycle in the association graph. It requires special attention with the VO transformation of involved entities, which is analyzed in section 4.2.1.

[1]commonly referred to as *trees*

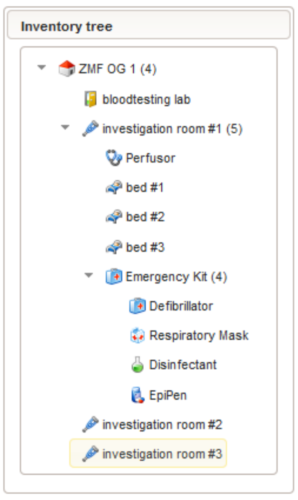


Figure 4.12.: Sample UI detail for the `Inventory` tree structure.

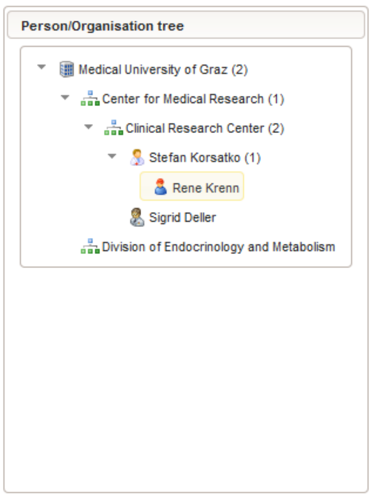

Person/Organisation tree
Medical University of Graz (2)
Center for Medical Research (1)
Clinical Research Center (2)
Stefan Korsatko (1)
Rene Krenn
Sigrid Deller
Division of Endocrinology and Metabolism

Figure 4.13.: Sample UI detail for the `Staff` tree structure.

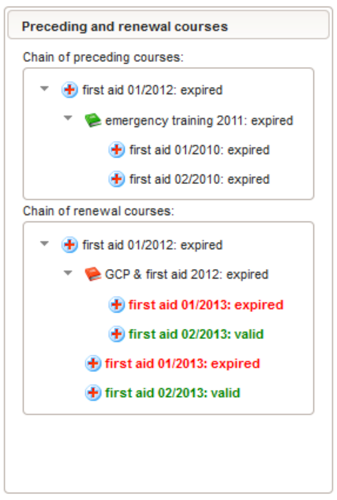

Preceding and renewal courses
Chain of preceding courses:
first aid 01/2012: expired
emergency training 2011: expired
first aid 01/2010: expired
first aid 02/2010: expired
Chain of renewal courses:
first aid 01/2012: expired
GCP & first aid 2012: expired
first aid 01/2013: expired
first aid 02/2013: valid
first aid 01/2013: expired
first aid 02/2013: valid

Figure 4.14.: Sample UI detail for the `Course` tree structures.

4.1.1.3. Entity-Attribute-Value (EAV)

In the context of ORM, a domain entity's fields are statically mapped onto columns of a corresponding database table. Tables are created when initializing the database using DDL SQL statements[2]. In general, most database applications rely on a static table schema. Database transactions of service methods represent the regular operation and are limited to DML and `SELECT ...` SQL statements. Entities are represented by POJO classes, whose field accessor methods cannot be modified by means of Java language *reflection* either. Modification of entity definitions would additionally require a synchronisation of Hibernate's mappings during application runtime.

Basically, entity field definitions are therefore considered to be limited to design time. This can turn out as handicap for the requirement of dynamic or custom entity fields. These custom attributes should be defined by application end users themselves, without the need of software change requests requiring a software developer for the implementation. Database applications in the clinical domain like HIS or CDMS have the characteristic requirement to support user-definable entity attributes (Haas, 2004, pp. 380). For example, a generic data model to map CRF data of clinical trials needs to handle various parameter fields for the trial data gathered, as specified by a trial's protocol. Modern CDMS incorporate form designers to set up a trial's CRFs. For each subject, investigative users fill in the UI form showing a blank CRF (EDC). The generic data model prepared by the form designer before is thereby populated.

Although the Phoenix CTMS is not designed to cover CRF data as a CDMS does, EDC was an identified requirement for proband recruitment. The proband database can be queried to rapidly identify the volume of suitable probands, according to the inclusion/exclusion criteria defined by the protocol of an acquired trial project. A criterion may refer to an arbitrary subject parameter, which has to be set up for data capturing first. While a person's common parameters like demographics[3] may be covered by explicit entity fields, trial-specific inquiries such as relevant medical information[4] require a generic data model similar to the data model for mapping CRFs.

[2]`CREATE TABLE ...`, `ALTER TABLE ...`, etc.
[3]age, sex, ...
[4]BMI, blood pressure, medication, patient history, ...

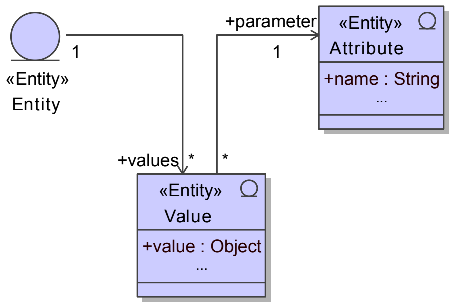


Figure 4.15.: The rudimentary EAV data model.

The EAV data model pattern (figure 4.15) is the intuitive approach when modelling a relational data model capable of handling custom attribute records. An entity instance (a `Proband` record) is linked with a set of attribute values (parameter values), forming an one-to-many relationship. Each value record is described by metadata defined by the associated attribute record (parameter type), forming a many-to-one relationship. Designing a form results in populating the attribute table, while the EDC process results in populating the value table. Depending on the implementation of enhanced structures of values and the UI presentation of input forms, both attribute and value entities may get additional fields:

Attribute: description, default value, position, required flag, data type, range limits for input validation, unit of measurement, ...

Value: modification timestamp, value array[5], reference to the former value, aggregate of multiple values[6], ...

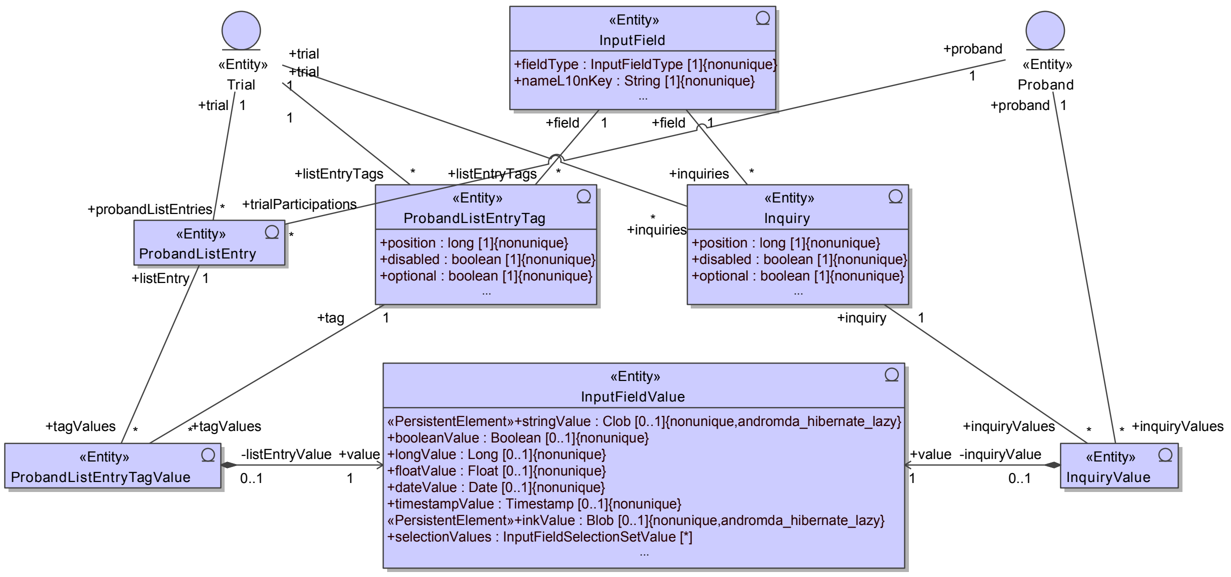

«Entity»
Trial
«Entity»
InputField
+fieldType : InputFieldType [1]{nonunique}
+nameL10nKey : String [1]{nonunique}
«Entity»
Proband
«Entity»
ProbandListEntry
«Entity»
ProbandListEntryTag
+position : long [1]{nonunique}
+disabled : boolean [1]{nonunique}
+optional : boolean [1]{nonunique}
«Entity»
Inquiry
+position : long [1]{nonunique}
+disabled : boolean [1]{nonunique}
+optional : boolean [1]{nonunique}
«Entity»
InputFieldValue
«PersistentElement»+stringValue : Clob [0..1]{nonunique,andromda_hibernate_lazy}
+booleanValue : Boolean [0..1]{nonunique}
+longValue : Long [0..1]{nonunique}
+floatValue : Float [0..1]{nonunique}
+dateValue : Date [0..1]{nonunique}
+timestampValue : Timestamp [0..1]{nonunique}
«PersistentElement»+inkValue : Blob [0..1]{nonunique,andromda_hibernate_lazy}
+selectionValues : InputFieldSelectionSetValue [*]
«Entity»
ProbandListEntryTagValue
«Entity»
InquiryValue
+trial
+proband
+field
+listEntryTags
+inquiries
+probandListEntries
+trialParticipations
+listEntry
+tag
+inquiry
+tagValues
+inquiryValues
-listEntryValue
+value
-inquiryValue

Figure 4.16.: Two EAV patterns can be identified within the system's data model.

Since a parameter can be of any data type, the value entity is required to hold values of any type (weak typing). RDBMSs do not support a variant data type in general, thus nuances of the EAV model

[5]Arrays of values can be used to store *longitudinal data*, depicting a series of values.
[6]e.g. a time interval with `start` and `stop` fields

are effectively used. The Phoenix CTMS data model contains two manifestations of the "hybrid EAV" (Gilchrist et al., 2011) pattern (figure 4.16), which is characterized by a distinct value column per supported data type (strong typing):

Proband Inquiries: Medical information for eligibility criteria is captured by interviewing probands. For the questionnaire, an input form can be set up per trial (EAV attributes). Entered answers and parameter values represent EAV values.

> *Entity:* `Proband`
> *Attribute:* `Inquiry` - `InputField`
> *Value:* `InquiryValue` - `InputFieldValue`

Proband list columns: Lists are used to track the progression per subject during an ongoing trial. Users are supposed to extend a trial's proband list with individual columns (EAV attributes) as required. A corresponding input form allows entering column values (EAV values).

> *Entity:* `ProbandListEntry`
> *Attribute:* `ProbandListEntryTag` - `InputField`
> *Value:* `ProbandListEntryTagValue` - `InputFieldValue`

Parameter definitions are mapped by the `InputField` entity, whose records can be managed globally with the system's input field module. The trial's inquiry form is represented by a set of `Inquiry` items. A set of `ProbandListEntryTag` items extends the trial's proband list by additional columns. To faciliate consolidation of application logic of both EAV models, the shared `InputFieldValue` entity provides table columns for supported value data types. In addition, it allows to retrieve a comprehensive set of suggestion values for autocomplete UI input elements with a single SQL query. A general downside of EAV is the need to join potentially large tables when querying values.

The current implementation of the EAV model covers neither time series of values (longitudinal data) nor value aggregates (complex data types). Although it would not be too difficult to add these capabilities to the EAV model, it was decided to leave them out for now due to the increased comlexity of database queries that users would be actually faced with when creating dynamic queries. Basically, the user has the option to compose input forms with an explicit field per indexed data element. The disadvantage of this approach is the need to anticipate a constant number of field items placed on the form (e.g. "Chronic disease #1:" - "Chronic disease #10:"). Therefore, separate `Proband` detail entities (e.g. `Diagnosis`, `Procedure`) are provided for capturing this kind of baseline information.

4.1.1.4. Master Data Entities

The Phoenix CTMS UI contains a multitude of dropdown and suggestion input elements. The option items displayed are backed by sets (selection sets) of key-value pairs or a plain list of values, representing *master data* records, such as a directory of street names. In contrast to remaining domain objects, master data are basically considered immutable and cannot be modified, at least not by application users themselves. Master data records are stored in the database like the rest of the domain entity records in order to support a fast lookup and retrieval even for large master data tables. Extensive configuration data represents another kind of master data, e.g. access permission definitions or entity column definitions for query criterion terms. A database administrator is able to manipulate data records directly or import them, without the need to modify application source code or configuration files contained in the application's `.jar` packages.

Throughout the application, *master data entities* can be retrieved or referenced only, but not modified since corresponding CRUD operations are not available. Aside records of master data entities, *enumerations* represent *hard-coded* static data. Enumeration types[7] are preferred if there is a tight coupling with

[7] `Sex`, `Weekday`, `Module`, `VariablePeriod` ...

application logic or the number of constants is limited and definite. If a domain entity field represents a collection of enumeration constants, a supplemental master data entity (e.g. `TrialStatusAction`, `ProbandListStatusLogLevel`) holding the enumeration's constants.[8] Selected master data records are populated once during system setup when initializing the blank database schema, while others can be updated frequently using provided *importers*.

scope	**master data entity**	**description/examples**	**association**	**key**	**suggestion**
inventory	`InventoryCategory`	"basic equipment", "laboratory equipment", "expendable", ...	✓		
	`InventoryTag`	custom inventory string properties	✓		
	`InventoryStatusType`	"in repair", "defective", "lent", ...	✓		
	`MaintenanceType`	"inspection", "calibration", "reorder", ...	✓		
person/organisation	`StaffCategory`	"institution", "graduand", "study nurse", ...	✓		
	`StaffTag`	custom person/organisation string properties	✓		
	`StaffStatusType`	"vacation", "sick leave", ...	✓		
	`CvSection`	sections for CV positions	✓		
course	`CourseCategory`	"GCP", "trial-specific", ...	✓		
	`LecturerCompetence`	"trainer", "examiner", ...	✓		
	`CourseParticipationStatusType`	lifecycle states of a course participation	✓		
trial	`TrialStatusType`	lifecycle states of a trial	✓		
	`TrialStatusAction`	trial state transition actions	✓		
	`TrialType`	"AMG trial", "MPG trial", "in-vitro diagnostic product", ...	✓		
	`SurveyStatusType`	customer survey states	✓		
	`SponsoringType`	"commissioned trial", "academic trial"	✓		
	`TrialTag`	custom trial string properties	✓		
	`TeamMemberRole`	"principal investigator", "project staff", "monitor", ...	✓		
	`TimelineEventType`	types of a trial's milestones, deadlines and phases	✓		
	`VisitType`	"screening", "dosing", "final visit", ...	✓		
	`ProbandListStatusType`	lifecycle states of a trial participation	✓		
	`ProbandListStatusLogLevel`	subject list subsets	✓		
proband	`PrivacyConsentStatusType`	lifecycle states of the data privacy consent aquisition	✓		
	`ProbandCategory`	"migrated", "new entry", "test proband"	✓		
	`ProbandTag`	custom proband string properties	✓		
	`ProbandStatusType`	"no time", "sick", ...	✓		

Table 4.2.: Table of master data entities, part 1.

[8]The redundant design of enumeration *and* entity is required to model the many-to-many association, since AndroMDA otherwise fails to validate a model containing entity fields of enumeration type with a multiplicity > 1.

scope	master data entity	description/examples	association	key	suggestion
shared	`Title`	input suggestions for academic degrees			✓
	`AddressType`	"office address", "term address", …	✓		
	`ContactDetailType`	"mobile phone number", "email", …	✓		
	`BankIdentification`	input suggestions for bank names, bank code numbers, BICs			✓
	`FileFolderPreset`	default file manager folder structures			✓
	`MimeType`	file content types	✓		
	`HyperlinkCategory`	"SOP", "customer support site", "online course material"	✓		
	`JournalCategory`	categories of manual journal entries	✓		
	`NotificationType`	notification event types and their recipients' `StaffCategorys`	✓		
	`Department`	organisation departments corresponding to encryption domains	✓		
	`Holiday`	public holiday and commemoration day definitions			
authorisation	`Permission`	service method privilege definitions	✓		
	`ProfilePermission`	privilege profile (role) definitions		✓	
query editor	`CriterionTie`	criterion conjunctions		✓	
	`CriterionProperty`	criterion entity fields		✓	
	`CriterionRestriction`	criterion comparison operators		✓	
ICD-10	`AlphaId`	DIMDI Alpha-ID diagnoses (alphabetical, including synonyms)	✓		
	`IcdSyst`	ICD-10 chapters		✓	
	`IcdSystBlock`	ICD-10 blocks	✓		
	`IcdSystCategory`	ICD-10 categories	✓		
	`IcdSystModifier`	ICD-10 modifiers	✓		
OPS	`OpsCode`	DIMDI OPS procedures (alphabetical)	✓		
	`OpsSyst`	OPS chapters		✓	
	`OpsSystBlock`	OPS blocks	✓		
	`OpsSystCategory`	OPS categories	✓		
	`OpsSystModifier`	OPS modifiers	✓		
addresses	`Country`	input suggestions for country names			✓
	`Zip`	input suggestions for ZIP codes and city names			✓
	`Street`	input suggestions for street names			✓

Table 4.3.: Table of master data entities, part 2.

Master data entities (table 4.2 and 4.3) can be classified by the type of their relationship with domain entities. Each type implies specific advantages when adjusting or maintaining a Phoenix CTMS deployment:

Association In the default case, master data records are linked according to regular associations types from table 2.5. For example, the `Staff` domain entity strictly refers to the `StaffCategory` master data entity by modelling an explicit (many-to-one) association. The associated master data entity may come with a set of attributes for UI presentation like name and icon. While there is no impact when adding new categories, removing a category causes a conflict, once it is referrenced by existing records. Therefore, a `visible` flag can be cleared instead of deleting the master data record and updating any reference. The `visible` flag allows hiding the master data record in option items when editing domain entity records from that point on.

Key In contrast to master data entities linked by regular associations from before, a relationship modeled by an explicit key column can be considered as *unconstrained association*. This key conceptually represents a foreign key, but there is no foreign key constraint created for the column, which would be watched by the RDBMS. Queries employing unconstrained associations require SQL joins with an *explicit* join condition on the key column to relate tables.[9] In order to ensure performance, an unique

[9]Contrary, SQL joins are implicitly generated with a join condition derived from the association when "joining a (regular) association" (e.g. `entity_a JOIN entity_b ON entity_a.b_fk = entity_b.id`) in the context of HQL or the Hibernate Criteria API.

constraint or index is applied to key columns if appropriate. Since no foreign key constraint is generated, it is possible to remove master data records temporarily. However, consistency requires to take care of key value integrity when recreating master data records.

Hibernate ORM is particularly helpful when adding complex master data items programmatically, which need the creation of multiple rows in multiple tables each. This is ideal for master data importers for ICD-10, permission or criterion term column definitions, which are linked to domain entity records with unconstrained associations. Before importing updated master data, related tables can be flushed without the need to handle constraint violations.

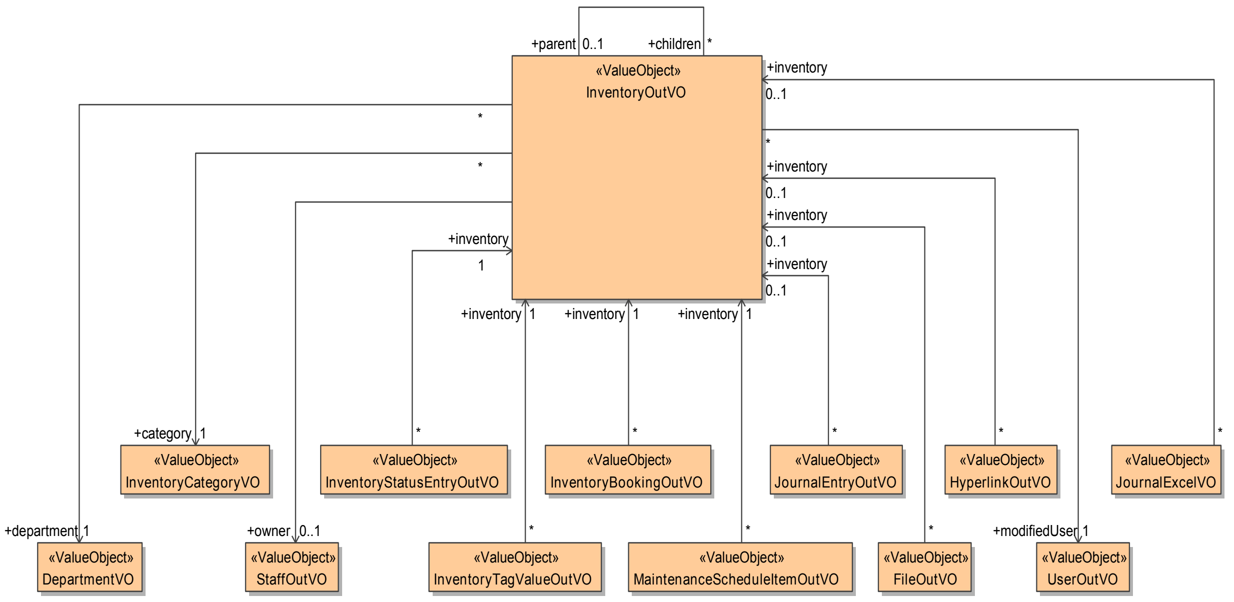
Suggestion For master data that is expected to be incomplete, it was decided to omit traditional entity relationships at all. For example, storing postal addresses using an association would require an absolute directory of addresses. Instead, the domain entity gets dedicated (string) entity fields to store a postal address. In order to facilitate consistent data entry, UI input elements providing *suggestions* are used. Even if a street name entered is not found among suggested items from the backing master data, users may persist the address containing the "unknown" street name.

4.1.2. Association Navigability

Entity association ends need to be named and navigatable in order to be part of association path notations used in HQL statements (figure 4.11) or Hibernate's Criteria API. Except master data associations, any entity associations are therefore modeled *bidirectional navigatable* and with names on both ends (King et al., 2004, chapter 24). This avoids limitations when referring to entity fields in query criteron expressions, on which the infrastructure for dynamic database queries relies on. A drawback is the expense to always *maintain* both association ends in order to persist data correctly in the course of CRUD operations, as shown in listing 2.4.

AndroMDA generates field accessor methods for navigatable association ends of entity POJOs. Collection-valued association fields are *lazy-loaded* by default. In contrast to eager-loading, lazy-loading will populate an entity's private field member variable by retrieving a database record upon the first getter method invocation during a program flow. The creation of an OutVO returned by a service method therefore requires an actual database access for each entity association mapped in the target OutVO when transforming the source entity. For the complex data model presented here, OutVO graphs would have a considerable size, if entity graphs were mapped completely. This would cause a disappointing performance due to the consecutive loading of records that are actually not needed by the web tier or client application in most cases. To reduce size, a VO association should be modeled only if there is a defensible need for it. Bidirectional navigatable associations of VO graphs represent *cycles*. In general, association cycles are another source of loading too big parts from the database. In contrast to modeled entities, navigability directions of OutVOs should therefore be restricted by default.

Taking these arguments into account, an sample OutVO graph will show unidirectional associations (figure 4.17). To promote a uniform structure for VO associations that will meet an acceptable size of VO instance graphs, detail VOs will preferably link towards the root element.



+parent 0..1
+children *
«ValueObject» InventoryOutVO
+inventory 0..1
*
+inventory 0..1
+inventory 0..1
+inventory 0..1
+inventory 1
+inventory 1
+inventory 1
+inventory 1
+category 1
«ValueObject» InventoryCategoryVO
«ValueObject» InventoryStatusEntryOutVO
«ValueObject» InventoryBookingOutVO
«ValueObject» JournalEntryOutVO
«ValueObject» HyperlinkOutVO
«ValueObject» JournalExcelVO
+department 1
«ValueObject» DepartmentVO
+owner 0..1
«ValueObject» StaffOutVO
«ValueObject» InventoryTagValueOutVO
«ValueObject» MaintenanceScheduleItemOutVO
«ValueObject» FileOutVO
+modifiedUser 1
«ValueObject» UserOutVO

Figure 4.17.: The arrangement of this OutVO graph corresponds to the `Inventory` master-details tree (figure 4.2). In contrast to the entity graph, most VO associations are unidirectional.

4.1.3. Entity Field Patterns

4.1.3.1. Entity Generalization

At the moment, Hibernate's ORM inheritance feature is *not used* throughout the data model. This means entity class diagrams do not have any entity generalization design elements. This will reduce the complexity of the large UML class diagrams since there is no generalization hierarchy graph overlay. Each entity exactly reflects its resulting database table and corresponding columns. In addition, entity polymorphism affects database queries, which can require more complex finder methods implementations. This is particularly true for the current implementation of dynamic queries, because additional table joins or SQL union operations would be necessary.

Contrary, a scenario predestinated for generalization that came into mind at first originates from similarities of `Staff` and `Proband` detail entities. In spite of obvious functional congruencies when comparing OutVOs as in the example shown in figure 4.18, the need for encryption of `Proband`'s detail entity fields will render common base types less useful here.

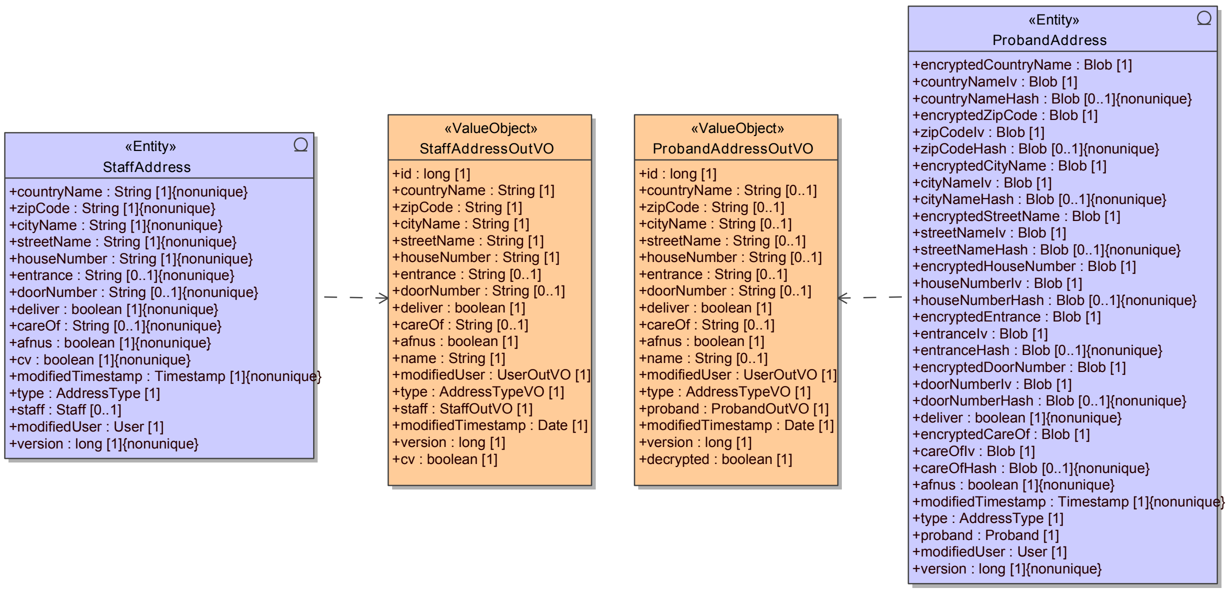

«Entity»
StaffAddress
+countryName : String [1]{nonunique}
+zipCode : String [1]{nonunique}
+cityName : String [1]{nonunique}
+streetName : String [1]{nonunique}
+houseNumber : String [1]{nonunique}
+entrance : String [0..1]{nonunique}
+doorNumber : String [0..1]{nonunique}
+deliver : boolean [1]{nonunique}
+careOf : String [0..1]{nonunique}
+afnus : boolean [1]{nonunique}
+cv : boolean [1]{nonunique}
+modifiedTimestamp : Timestamp [1]{nonunique}
+type : AddressType [1]
+staff : Staff [0..1]
+modifiedUser : User [1]
+version : long [1]{nonunique}
«ValueObject»
StaffAddressOutVO
+id : long [1]
+countryName : String [1]
+zipCode : String [1]
+cityName : String [1]
+streetName : String [1]
+houseNumber : String [1]
+entrance : String [0..1]
+doorNumber : String [0..1]
+deliver : boolean [1]
+careOf : String [0..1]
+afnus : boolean [1]
+name : String [1]
+modifiedUser : UserOutVO [1]
+type : AddressTypeVO [1]
+staff : StaffOutVO [1]
+modifiedTimestamp : Date [1]
+version : long [1]
+cv : boolean [1]
«ValueObject»
ProbandAddressOutVO
+id : long [1]
+countryName : String [0..1]
+zipCode : String [0..1]
+cityName : String [0..1]
+streetName : String [0..1]
+houseNumber : String [0..1]
+entrance : String [0..1]
+doorNumber : String [0..1]
+deliver : boolean [1]
+careOf : String [0..1]
+afnus : boolean [1]
+name : String [0..1]
+modifiedUser : UserOutVO [1]
+type : AddressTypeVO [1]
+proband : ProbandOutVO [1]
+modifiedTimestamp : Date [1]
+version : long [1]
+decrypted : boolean [1]
«Entity»
ProbandAddress
+encryptedCountryName : Blob [1]
+countryNameIv : Blob [1]
+countryNameHash : Blob [0..1]{nonunique}
+encryptedZipCode : Blob [1]
+zipCodeIv : Blob [1]
+zipCodeHash : Blob [0..1]{nonunique}
+encryptedCityName : Blob [1]
+cityNameIv : Blob [1]
+cityNameHash : Blob [0..1]{nonunique}
+encryptedStreetName : Blob [1]
+streetNameIv : Blob [1]
+streetNameHash : Blob [0..1]{nonunique}
+encryptedHouseNumber : Blob [1]
+houseNumberIv : Blob [1]
+houseNumberHash : Blob [0..1]{nonunique}
+encryptedEntrance : Blob [1]
+entranceIv : Blob [1]
+entranceHash : Blob [0..1]{nonunique}
+encryptedDoorNumber : Blob [1]
+doorNumberIv : Blob [1]
+doorNumberHash : Blob [0..1]{nonunique}
+deliver : boolean [1]{nonunique}
+encryptedCareOf : Blob [1]
+careOfIv : Blob [1]
+careOfHash : Blob [0..1]{nonunique}
+afnus : boolean [1]{nonunique}
+modifiedTimestamp : Timestamp [1]{nonunique}
+type : AddressType [1]
+proband : Proband [1]
+modifiedUser : User [1]
+version : long [1]{nonunique}

Figure 4.18.: Entities like `StaffAddress` and `ProbandAddress` are expected to qualify for ORM inheritance when looking at their mostly identical OutVOs. However, column encryption thwarts generalisation since it requires multiple entity fields in `ProbandAddress` per corresponding field of `StaffAddress`.

The decision to omit generalizations was taken at the very beginning of the design phase. However, the option of future refactoring by modeling entity inheritance hierarchies was kept in mind. This means introducing potential generalisations in the UML model results in updated Hibernate mappings and other artefacts, but will not require significant modifications to the existing database schema. Instead of rather specific cases as given by similarities of `Staff` and `Proband` detail entities that would qualify for generalisation, the introduction of generalizations is considered for a series of entity categories presented in the next sections. Types of entities with similar properties were identified and aspects of their design such as field naming and semantics were unified by means of *convention*. Techniques used to minimize code duplication of logic that is bound with entity types will be in focus.

4.1.3.2. Record Versioning and Locking

To encounter the race condition that is inherent to concurrent modifications of records by users, each entity that is subject to CRUD operations has a `version` field. Starting with 0 when the item is created, its integer value is incremented by one with every update that is persisted. InVOs provide a `version` field for passing the version value hold by the service method caller (managed bean). The version value was previously obtained with the retrieval of the existing record that is to be updated. In the course of the service method transaction, the passed version value is compared to the record's current version value. Modifications in the meantime are detected, if the current value is higher than the version value provided. In this case the transaction will be aborted by throwing an exception, which provides an appropriate error message for feedback to the user. Apart from the `version` field, each entity is equipped with a modification "footprint" made up by a `Timestamp` and `User` field. Uniform field names allow to encapsulate implementation of the logic using reflection (listing 4.1).

```
//located in public final class at.zmf.crc.ctsms.util.CoreUtil
//field naming conventions:
private static final String ENTITY_VERSION_GETTER_METHOD_NAME = "getVersion";
private static final String ENTITY_VERSION_SETTER_METHOD_NAME = "setVersion";
private static final String ENTITY_MODIFIED_USER_GETTER_METHOD_NAME = "getModifiedUser";
private static final String ENTITY_MODIFIED_USER_SETTER_METHOD_NAME = "setModifiedUser";
private static final String ENTITY_MODIFIED_TIMESTAMP_SETTER_METHOD_NAME = "setModifiedTimestamp";
private static final String ENTITY_ID_GETTER_METHOD_NAME = "getId";

//check and set a record's update version value and footprint:
public static <E> void updateVersion(E current, E updated, Timestamp now, User user) throws Exception {

    //when an entity is created, its id must be set to null:
    if (current == null) {
        Long id = getEntityId(updated);
        if (id != null) {
            throw L10nUtil.initServiceException(ServiceExceptionCodes.ENTITY_ID_NOT_NULL, id.toString());
        }
    }

    //get the version value provided by the service method caller:
    long providedVersion = ((Long)updated.getClass().getMethod(ENTITY_VERSION_GETTER_METHOD_NAME).invoke(updated)).longValue();

    //prepare the version value of the new/updated entity:
    long newVersion;
    if (original != null) { //updating: compare current and provided version value
        long currentVersion = ((Long) current.getClass().getMethod(ENTITY_VERSION_GETTER_METHOD_NAME).invoke(current))
            .longValue();
        if (providedVersion != currentVersion) { //record was modified in meantime:
            User currentModifiedUser = (User) current.getClass().getMethod(ENTITY_MODIFIED_USER_GETTER_METHOD_NAME)
                .invoke(current);
            throw L10nUtil.initServiceException(ServiceExceptionCodes.ENTITY_MODIFIED_IN_MEANTIME, currentModifiedUser
                .getName());
        }
        newVersion = currentVersion + 1; //increment version value
    } else { //creating: version value is set to 0
        if (providedVersion != 0) {
            throw L10nUtil.initServiceException(ServiceExceptionCodes.ENTITY_VERSION_NOT_ZERO);
        }
        newVersion = 0;
    }

    //modify entity update:
    entity.getClass().getMethod(ENTITY_VERSION_SETTER_METHOD_NAME, long.class).invoke(updated, newVersion);
    entity.getClass().getMethod(ENTITY_MODIFIED_TIMESTAMP_SETTER_METHOD_NAME, Timestamp.class).invoke(updated, now);
    entity.getClass().getMethod(ENTITY_MODIFIED_USER_SETTER_METHOD_NAME, User.class).invoke(updated, user);

}
//alternative method signature for creating new records:
public static void updateVersion(Object newEntity, Timestamp now, User user) throws Exception {
    updateVersion(null, newEntity, now, user);
}
```

Listing 4.1: The `updateVersion` method sets the `version` field and modification footprint of all entities that are subject to CRUD operations. It will fail with an exception, if the record was modified in meantime.

The presented implementation of *optimistic locking* will ensure that users will not override their changes mutually. Hibernate's built-in support for optimistic locking is not used since it can cover associations, which requires additional annotations in the UML model to define which branches of an entity graph should be excluded in order to avoid notorious `OptimisticLockingExceptions`. Basically, a chance of *lost updates* remains with optimistic locking since they are inherent to the MVCC architecture, as used by an RDBMS like PostgreSQL. The lost update problem can come into effect in the rare case when users are editing the *same record* and click the update buttons almost at the same time, causing concurrent database transactions. A solution to this is *pessimistic locking*, which relies on the RDBMS to prevent lost updates by means of acquired *row locks*. Record locking is used in two situations:

- long-lasting transactions
- transactions prone to the *phantom read* anomaly[10] (Reese, 2003, p. 68)

The AndroMDA DAO templates were therefore modified to provide additional `load` operation overloads including an `org.hibernate.LockMode` parameter. It is passed to `org.hibernate.Session.load` in order

[10]PostgreSQL's default transaction isolation level (READ_COMMITTED) is assumed.

to emit SELECT ... *FOR UPDATE* SQL statements for loading entity instances and aquiring a row write lock at once.

4.1.3.3. Entities for Universal Features

The record version control presented in the previous section is modeled by adding the required fields to each supported entity manually. This equals the mapping of a "CRUDRecord" entity base type[11] in *table-per-concrete-class* ORM inheritance mapping scheme (Fowler, 2002, pp. 293).

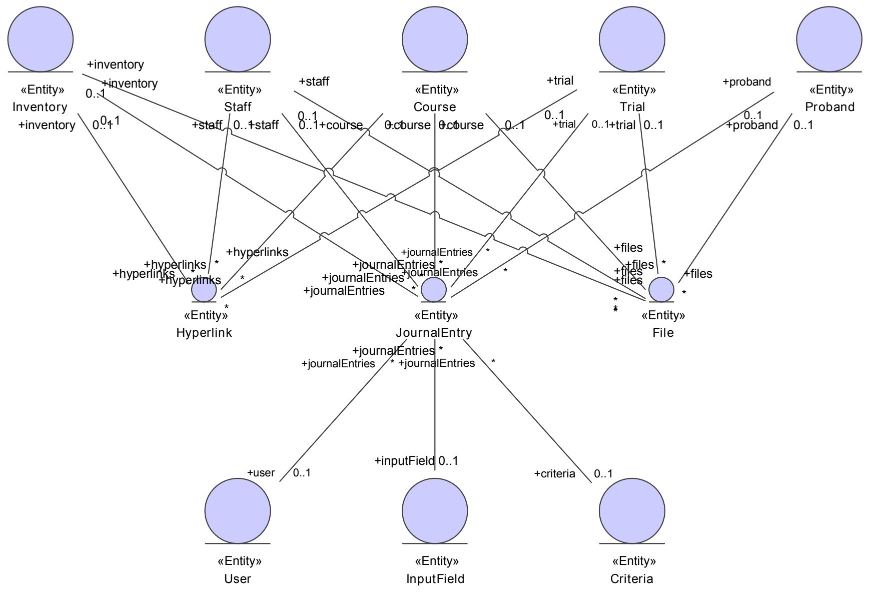


Figure 4.19.: Detail entities such as Hyperlink, File and JournalEntry for universal features have associations to multiple root entities.

In contrast, the detail entities Hyperlink, File and JournalEntry reflect the the mapping of a *table-per-hierarchy* mapping scheme (Fowler, 2002, pp. 278). They belong to entity types used to implement universal features available to multiple Phoenix CTMS modules (section 3.2.8):

Hyperlink: URL bookmarks for inventories, person/organisations, courses and trials.
File: File managers (section 4.7) support an apart document repository per inventory, person/organisation, course, trial and proband.
JournalEntry: An accumulated journal of manual notes and system generated log entries for all root entities (Audit Trail, section 4.3.1).
Notification: Notification for events for a series of root and detail entities are generated and sent via email (Notification messages, section 4.6.3.2).
Department: Root entities except input fields are associated with a department to support multi-client capabilities.

Hyperlink, File and JournalEntry show associations to multiple root entities (figure 4.19). These associations are treated mutual exclusive, thus an instance will have a reference to no more than one root entity record while the remaining association fields are set to null. Corresponding enumeration types (JournalModule, FileModule, HyperlinkModule) provide *discriminator* values in terms of the table-per-hierarchy mapping scheme.

[11]CRUDRecord would contain the common fields (version, modifiedTimestamp and modifiedUser) in this case.

4.1.3.4. Entity Adapters

Mastering recurring entity fields by means of reflection was shown for record versioning in a previous section. The *facade pattern* is applied to access fields of a wide range of entities in order to encapsulate a number of recurring application logic constructs. A class hierarchy of entity adapters reflecting identified entity types is introduced to structure implementations (figure 4.20).

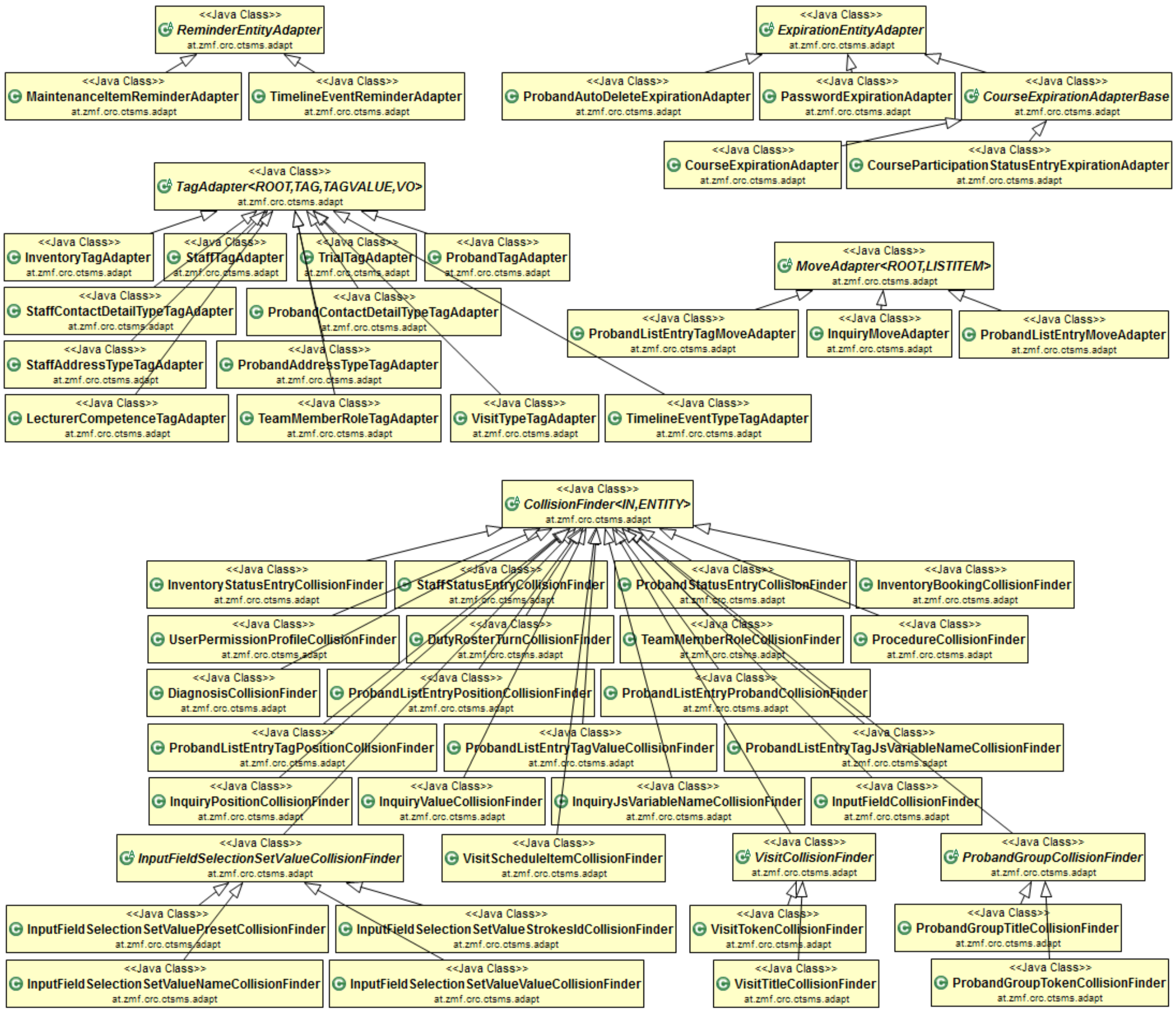
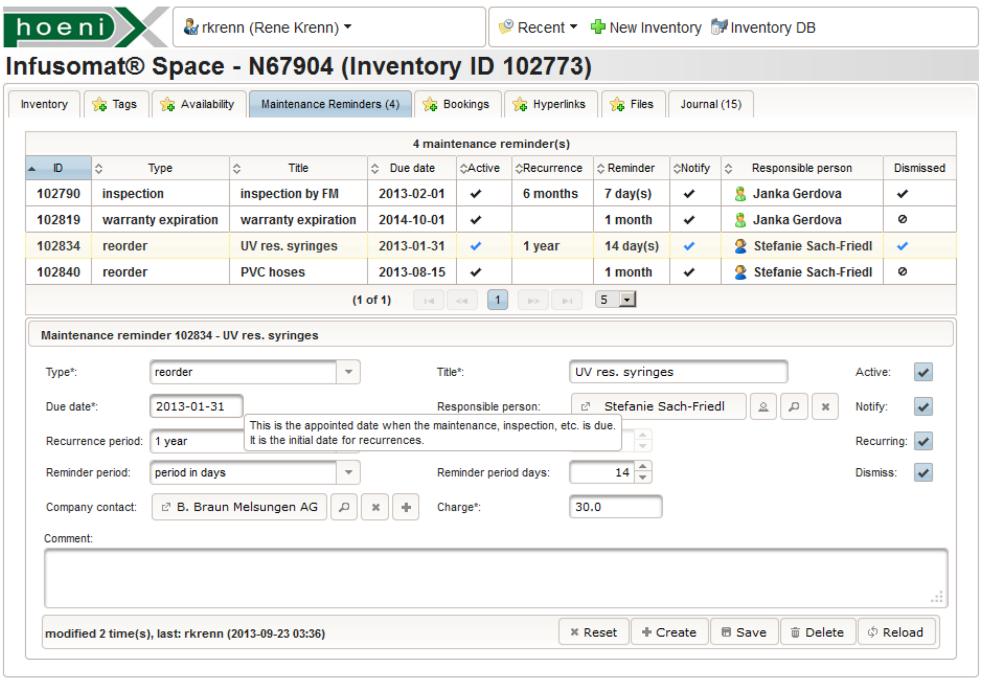

ReminderEntityAdapter
MaintenanceItemReminderAdapter
TimelineEventReminderAdapter
ExpirationEntityAdapter
ProbandAutoDeleteExpirationAdapter
PasswordExpirationAdapter
CourseExpirationAdapterBase
CourseExpirationAdapter
CourseParticipationStatusEntryExpirationAdapter
TagAdapter<ROOT,TAG,TAGVALUE,VO>
InventoryTagAdapter
StaffTagAdapter
TrialTagAdapter
ProbandTagAdapter
StaffContactDetailTypeTagAdapter
ProbandContactDetailTypeTagAdapter
StaffAddressTypeTagAdapter
ProbandAddressTypeTagAdapter
LecturerCompetenceTagAdapter
TeamMemberRoleTagAdapter
VisitTypeTagAdapter
TimelineEventTypeTagAdapter
MoveAdapter<ROOT,LISTITEM>
ProbandListEntryTagMoveAdapter
InquiryMoveAdapter
ProbandListEntryMoveAdapter
CollisionFinder<IN,ENTITY>
InventoryStatusEntryCollisionFinder
StaffStatusEntryCollisionFinder
ProbandStatusEntryCollisionFinder
InventoryBookingCollisionFinder
UserPermissionProfileCollisionFinder
DutyRosterTurnCollisionFinder
TeamMemberRoleCollisionFinder
ProcedureCollisionFinder
DiagnosisCollisionFinder
ProbandListEntryPositionCollisionFinder
ProbandListEntryProbandCollisionFinder
ProbandListEntryTagPositionCollisionFinder
ProbandListEntryTagValueCollisionFinder
ProbandListEntryTagJsVariableNameCollisionFinder
InquiryPositionCollisionFinder
InquiryValueCollisionFinder
InquiryJsVariableNameCollisionFinder
InputFieldCollisionFinder
InputFieldSelectionSetValueCollisionFinder
VisitScheduleItemCollisionFinder
VisitCollisionFinder
ProbandGroupCollisionFinder
InputFieldSelectionSetValuePresetCollisionFinder
InputFieldSelectionSetValueStrokesIdCollisionFinder
VisitTokenCollisionFinder
ProbandGroupTitleCollisionFinder
InputFieldSelectionSetValueNameCollisionFinder
InputFieldSelectionSetValueValueCollisionFinder
VisitTitleCollisionFinder
ProbandGroupTokenCollisionFinder
at.zmf.crc.ctsms.adapt

Figure 4.20.: Entity adapters allow to reuse implementations related to recurring entity field patterns.

The abstract base classes give a coarse-grained classification of identified entity field patterns:

`ReminderEntityAdapter` Provides the calculation of dates based on an enumerated recurrence period (`VariablePeriod`[12]), including the logic for dismissing event reminders such as `MaintenanceScheduleItem` (figure 4.21) and `TimelineEvent`.

[12]`VariablePeriod` constants enumerate named date intervals like `MONTH`, `LUNATION`, `YEAR` etc.



Infusomat® Space - N67904 (Inventory ID 102773)

4 maintenance reminder(s)

ID	Type	Title	Due date	Active	Recurrence	Reminder	Notify	Responsible person	Dismissed
102790	inspection	inspection by FM	2013-02-01	✓	6 months	7 day(s)	✓	Janka Gerdova	✓
102819	warranty expiration	warranty expiration	2014-10-01	✓		1 month	✓	Janka Gerdova	⊘
102834	reorder	UV res. syringes	2013-01-31	✓	1 year	14 day(s)	✓	Stefanie Sach-Friedl	✓
102840	reorder	PVC hoses	2013-08-15	✓		1 month	✓	Stefanie Sach-Friedl	⊘

(1 of 1) 1 5

Maintenance reminder 102834 - UV res. syringes

Type*:	reorder	Title*:	UV res. syringes	Active:	✓
Due date*:	2013-01-31	Responsible person:	Stefanie Sach-Friedl	Notify:	✓
Recurrence period:	1 year			Recurring:	✓
Reminder period:	period in days	Reminder period days:	14	Dismiss:	✓
Company contact:	B. Braun Melsungen AG	Charge*:	30.0		

This is the appointed date when the maintenance, inspection, etc. is due.
It is the initial date for recurrences.

Comment:

modified 2 time(s), last: rkrenn (2013-09-23 03:36)

Figure 4.21.: This sample UI screen shows the tab for managing an inventory's reminders. Each recurrence of a reminder item can be dismissed separately when resolved.

`ExpirationEntityAdapter` Provides the calculation of the expiration date with respect of a reminder period for entities containing expiry information, represented by fields of `Date` and `VariablePeriod` type. `CourseExpirationAdapterBase` child classes extend this basic logic to implement the refresher course hierarchy traversal. `CourseExpirationAdapter` can determine the latest expiration date of a given course with expiration by depth-first search.

`MoveAdapter` Swaps ordered items that provide a `position` field.

`CollisionFinder` Tests if record collides with existing ones. Implementations cover collision conditions like temporal overlapping or uniqueness within a group.

`TagAdapter` Validates a given InVO input for persisisting a corresponding *tag value* detail entity. Tag value entities are made up of a *tag type* and string value field. Tag types define an occurence limit and a Regular Expression (RegEx) constraint at least.

Since detailed explanation of all adapter types goes beyond scope, a closer look at the `TagAdapter` will describe the adapter concept by example. The `TagAdapter` usage classifies a category of detail entities that can store additional information of a root entity in form of string properties in an extendable way. This will be examined for the concrete case of the `TrialTagAdapter`.

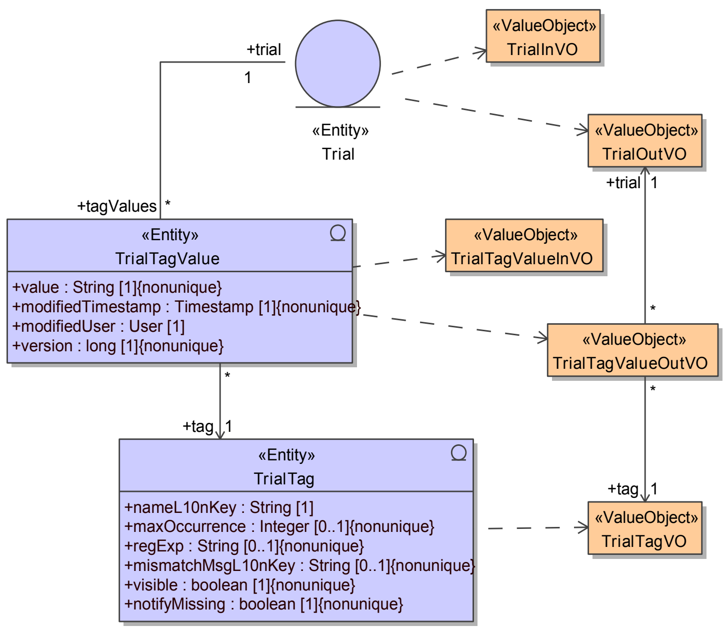

+trial
1
«Entity»
Trial
«ValueObject»
TrialInVO
«ValueObject»
TrialOutVO
+trial
1
+tagValues
*
«Entity»
TrialTagValue
+value : String [1]{nonunique}
+modifiedTimestamp : Timestamp [1]{nonunique}
+modifiedUser : User [1]
+version : long [1]{nonunique}
«ValueObject»
TrialTagValueInVO
*
«ValueObject»
TrialTagValueOutVO
*
*
+tag
1
«Entity»
TrialTag
+nameL10nKey : String [1]
+maxOccurrence : Integer [0..1]{nonunique}
+regExp : String [0..1]{nonunique}
+mismatchMsgL10nKey : String [0..1]{nonunique}
+visible : boolean [1]{nonunique}
+notifyMissing : boolean [1]{nonunique}
+tag
1
«ValueObject»
TrialTagVO

Figure 4.22.: The tag value (`TrialTagValue`) of a root entity (`Trial`) stores a string value for a predefined tag type (`TrialTag`).

The `TrialTagValue` entity (figure 4.22) maps the detail information of a trial listed below, as identified in the requirement analysis. Each property may exist once per trial only at most.

- ethics committee number
- BASG/Agentur für Gesundheit und Ernährungssicherheit (AGES) file number
- European Union Drug Regulating Authorities Clinical Trials (EudraCT) reference number
- ClinicalTrials.gov ID
- European Databank on Medical Devices (EUDAMED) reference number
- internal order number

The set of properties represents master data that is created during system setup in form of trial tag types. Each type optionally defines a maximum number of occurrences (one tag value) and a RegEx the property value must match when persisted. To assure the consistent state after concurrent modifications, the input validation for occurence limitation requires a `Trial` record *row lock* for the transaction. Exceeding the maximum number of occurrences cannot be precluded otherwise, since the `SELECT COUNT(*) ...` SQL statement to get the current number of `TrialTagValues` is prone to the phantom read anomaly. The logic for these input checks covers both create and update situations and is located in the `TagAdapter` abstract base class. In addition, it provides the calculation of a list of remaining available tag types for a given root entity instance, which is required to populate option items of UI dropdowns. Although field naming conventions are used that would qualify for reflection, accessing specific entity fields (and DAO methods) takes place in concrete implementations (`TrialTagAdapter`), which are aware of the entity type and its field accessor methods. To invoke the validation method (`checkTagValueInput`), a suitable adapter instance is instantiated (listing 4.2).

```
//located in public class at.zmf.crc.ctsms.service.trial.TrialServiceImpl
//input validation for addTrialTagValue and updateTrialTagValue service methods:
private void checkTrialTagValueInput(TrialTagValueInVO tagValueIn) throws ServiceException {
    (new TrialTagAdapter(this.getTrialDao(), this.getTrialTagDao())).checkTagValueInput(tagValueIn); //TODO: DAO DI for
            adapters
}
```

Listing 4.2: With `TagAdapters`, input validation for creating and updating a tag value is reduced to a single line of code.

A special advantage of the adapter approach is finally pointed out for advanced detail entities like `TimelineEvent`. It qualifies for a `TagAdapter` because some characteristic milestone events such as First-Patient-First-Visit (FPFV) can exist once per trial only. Since `TimelineEvent` will also contain temporal information that qualifies for a concrete `ReminderEntityAdapter`, the adapter technique provides an elegant way to overcome Java's multiple inheritance restriction which would manifest with ORM inheritance otherwise.

4.1.3.5. Filter by Entity Fields

An entity's *finder methods* implement database queries by utilizing a query abstraction provided by the Hibernate framework. In the case of the Criteria API, a `org.hibernate.Criteria` object represents the database query to run against the entity table in order to retrieve the result set in form of a collection of managed entity instances. `org.hibernate.Criteria` instances are bound to the session and are not designed for repeated manipulation or query parameter binding. However, utility methods are introduced to append specific restrictions[13] to an `org.hibernate.Criteria` object in order to simplify composing extensive queries and to minimize code duplication. Entity fields referred by appended restrictions need stipulated names and meaning for this to work. As an example, listing 4.3 shows a method to query entities with time interval information, characterized by their `start` and `stop` fields.

[13]According to the `org.hibernate.criterion.Restrictions` factory, a query criterion term is referred to as *restriction* in the context of Criteria API.

```
//located in public final class at.zmf.crc.ctsms.query.CriteriaUtil
//setup criteria for a temporal interval query of entities describing left-open, bounded or left-unbounded datetime intervals:
public static void applyStartUnboundedIntervalCriterion(org.hibernate.criterion.Criteria intervalCriteria,
    Timestamp from, Timestamp to) {
    if (intervalCriteria != null) {
        if (from != null  to != null) { //bounded query interval
            intervalCriteria.add(Restrictions.or(
                    Restrictions.or( //partial interval overlapping:
                        Restrictions.and(
                        Restrictions.ge("start", from),
                        Restrictions.lt("start", to)),
                    Restrictions.and(
                        Restrictions.gt("stop", from),
                        Restrictions.le("stop", to))
                    ),
                    Restrictions.or( //total inclusion:
                    Restrictions.and(
                        Restrictions.le("start", from),
                        Restrictions.ge("stop", to)),
                    Restrictions.and(
                        Restrictions.isNull("start"),
                        Restrictions.ge("stop", to))
                    )
                ));
        } else if (from != null  to == null) { //right-unbounded query interval
                intervalCriteria.add(Restrictions.gt("stop", from));
        } else if (from == null  to != null) { //left-unbounded query interval
                intervalCriteria.add(Restrictions.or(
                Restrictions.or(
                        Restrictions.lt("start", to),
                        Restrictions.le("stop", to)),
                    Restrictions.and(
                    Restrictions.isNull("start"),
                    Restrictions.ge("stop", to))
            ));
        }
        //do not narrow result set if query interval is unbounded
    }
}
//setup criteria for a temporal interval query of entities describing left-open, bounded or right-unbounded datetime intervals:
public static void applyStopUnboundedIntervalCriterion(Criteria intervalCriteria, Timestamp from, Timestamp to) { ... }

//setup criteria for a temporal interval query of entities describing events or bounded, closed or left-open datetime intervals
    :
public static void applyStartOptionalIntervalCriterion(org.hibernate.criterion.Criteria intervalCriteria,
    Timestamp from, Timestamp to, boolean includeStop) { ... } //if an entities start field is null, it is interpreted as event
public static void applyStopOptionalIntervalCriterion(org.hibernate.criterion.Criteria intervalCriteria,
    Timestamp from, Timestamp to, boolean includeStop) { ... } //if an entities stop field is null, it is interpreted as event
```

Listing 4.3: Interval entities have `start` and `stop` datetime fields. The methods shown will select interval entities that overlap a given query interval. In detail, the business logic relies on four identified variants of retrieving matching entities. Each corresponding method adds the appropriate restrictions to a given `org.hibernate.Criteria` object.

A database query result is calculated by the RDBMS and therefore basically outperforms a record selection logic implemented in Java application code. In the latter case, a considerable memory consumption and poor performance can result when fetching potentially many records for subsequent filtering. In order to filter items, iterating collections of entity instances wrapped by adapters from section 4.1.3.4 remain the reasonable choice for complex queries however, since Hibernate's query abstractions are basically designed to generate rudimentary `SELECT` SQL statements. Even basic features specified by the SQL-92 standard (*ISO/IEC 9075:1992(E) —Information technology —Database languages —SQL* 1992) can be unsupported, such as the `INTERVAL` data type.[14] Expressing advanced SQL constructs such as control structures, RDBMS-specific functions or mathematical expressions are known to be obstacles, but are easily implemented by entity adapters.

More details on finder method implementations follow in section 4.2.4 and 4.9.2.2.

[14]A custom date calculation logic was implemented as a workaround in this case, which allows to specify date intervals by `VariablePeriod` enumeration constants.

4.1.3.6. Stateful Entities

The stateful entity pattern is introduced to model *workflows* based on entities with a lifecycle aspect:

- trial state
- course participation state, four variants:
  - participant perspective for courses with self-registration
  - participant perspective for courses without self-registration
  - lecturer perspective for courses with self-registration
  - lecturer perspective for courses without self-registration
- proband enrollment state
- proband privacy consent state

As an example, the trial state describes the trial's current status of conducting (figure 4.23).

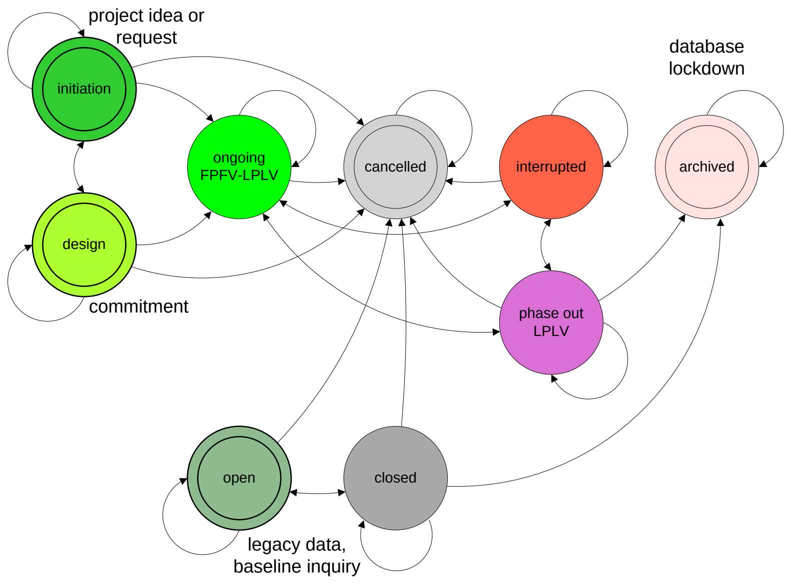
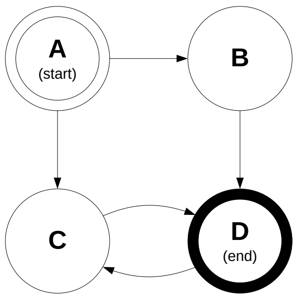


Figure 4.23.: A trial runs through a predefined sequence of states. The current state can be used for application control. For example, states can be configured to switch to read-only mode or to enable inquiry input forms in order to start the EDC process.

A workflow is mapped by a walkthrough of a Finite State Machine (FSM), depicted by a *state diagram* (figure 4.24). The state diagrams can show multiple start and terminal states and will not form *strongly connected* graphs[15] in general. Conditions for allowed state transitions depend on the current state only and are stated in a transition table.

[15]Strongly connected graphs are directed graphs that have a path from each vertex to every other vertex (Chapman and Hall, 1998).



state transitions	
current state	new state
A	**B, C**
B	**D**
C	**D**
D	**C**

Figure 4.24.: For use cases involving a workflow scheme, a stateful entity is modeled to map the relevant item lifecycle. States can be depicted by a state diagram or a transition table. In the sample shown, states A and B can never be reached again, once they were left.

A stateful entity comprises a many-to-one association with a *status type* entity, which in turn maps the state diagram. The transition table is stored by the reflexive many-to-many association of the status type entity (figure 4.25). Thus, state diagrams are effectively not hard-coded but configurable by means of master data setup. VOs for status type entities will not contain the transition association. Instead, option items of an UI dropdown to select the next state are populated with results of a finder method such as `TrialStatusType.findTransitions` which returns reachable new states for a given current state.

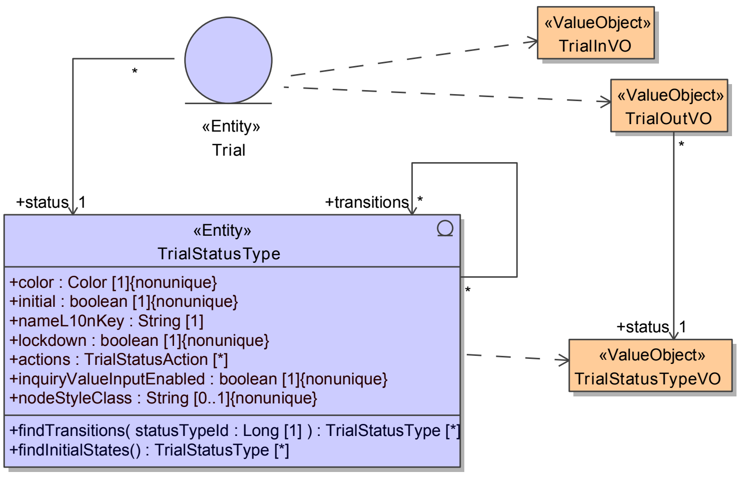

«ValueObject»
TrialInVO
«ValueObject»
TrialOutVO
«Entity»
Trial
*
*
+status 1
+transitions *
*
«Entity»
TrialStatusType
+color : Color [1]{nonunique}
+initial : boolean [1]{nonunique}
+nameL10nKey : String [1]
+lockdown : boolean [1]{nonunique}
+actions : TrialStatusAction [*]
+inquiryValueInputEnabled : boolean [1]{nonunique}
+nodeStyleClass : String [0..1]{nonunique}
+findTransitions(statusTypeId : Long [1]) : TrialStatusType [*]
+findInitialStates() : TrialStatusType [*]
+status 1
«ValueObject»
TrialStatusTypeVO

Figure 4.25.: The many-to-one relationship with a status type entity turns `Trial` into a stateful entity. The transition table is stored by the reflexive many-to-many association of the status type entity (`TrialStatusType`).

Transition actions are an optional pattern applicable to status type entities, which allows triggering prepared program logic when changing a state. At the moment it is implemented for `TrialStatusType` only. As shown in figure 4.25, a sequence of `TrialStatusAction` enumeration constants can be set up per state, defining which actions to perform when entering the state. Each `TrialStatusAction` refers to an implemented procedure, e.g.

`SIGN_TRIAL`: Create a digital signature value of the `Trial` entity graph.
`NOTIFY_MISSING_TRIAL_TAG`: Send notifications about mandatory `TrialTagValues` not entered yet.
`EXPORT_PROBAND_LIST`: Export subject data to an external CDMS instance.

System interconnection is not implemented yet but will typically rely on (mutual) web service interfaces. For example, OpenClinica provides a web service interface to accept subject information exported by external systems (OpenClinica, LLC and Contributors, 2013, chapter 3 - SOAP Web Services). Another option is to automatically generate and import the subject randomization outcomes using a randomization service such as `http://www.randomizer.at` (Ofner-Kopeinig and Errath, 2004).

4.2. Data Access Objects (DAOs)

This section outlines the remarkable challenges and solutions in the scope of DAO implementation classes. More precisely, implementation details of general *finder methods* and *entity-to-OutVO* transformation methods (figure 4.26) are examined.

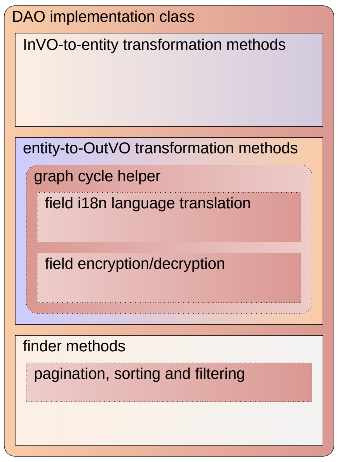

DAO implementation class
InVO-to-entity transformation methods
entity-to-OutVO transformation methods
graph cycle helper
field i18n language translation
field encryption/decryption
finder methods
pagination, sorting and filtering

Figure 4.26.: DAO implementation classes enclose common aspects: internationalisation, encryption and PSF

4.2.1. Graph Cycles

An association cycle can be defined as path along navigatable entity or VO associations with minimal length n, that ends with the same entity it started with. A reflexive association gives a single cycle (unidirectional navigability) or two separate cylces (bidirectional navigability) with $n = 1$ each. The mapped OutVO graph shown in figure 4.27 contains two reflexive associations and a cycle with $n = 2$.

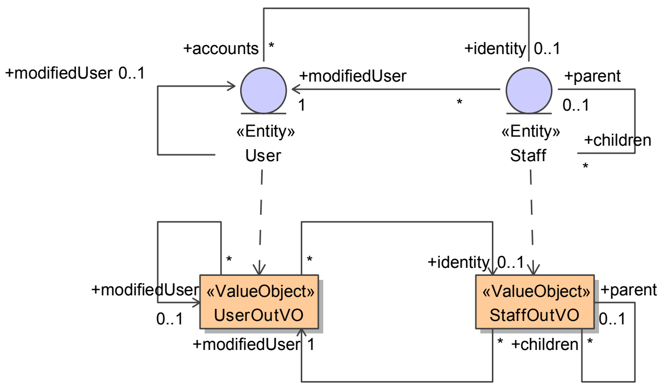


Figure 4.27.: `UserOutVO` and `StaffOutVO` associations form three cycles.

While cycles are no problem when retrieving entity instances due to Hibernate's lazy-loading mechanism[16], transforming entities into OutVO instances need special precautions. Circular references come into effect when populating VO instances and would cause endless loops of DAO transformation methods calling one another.

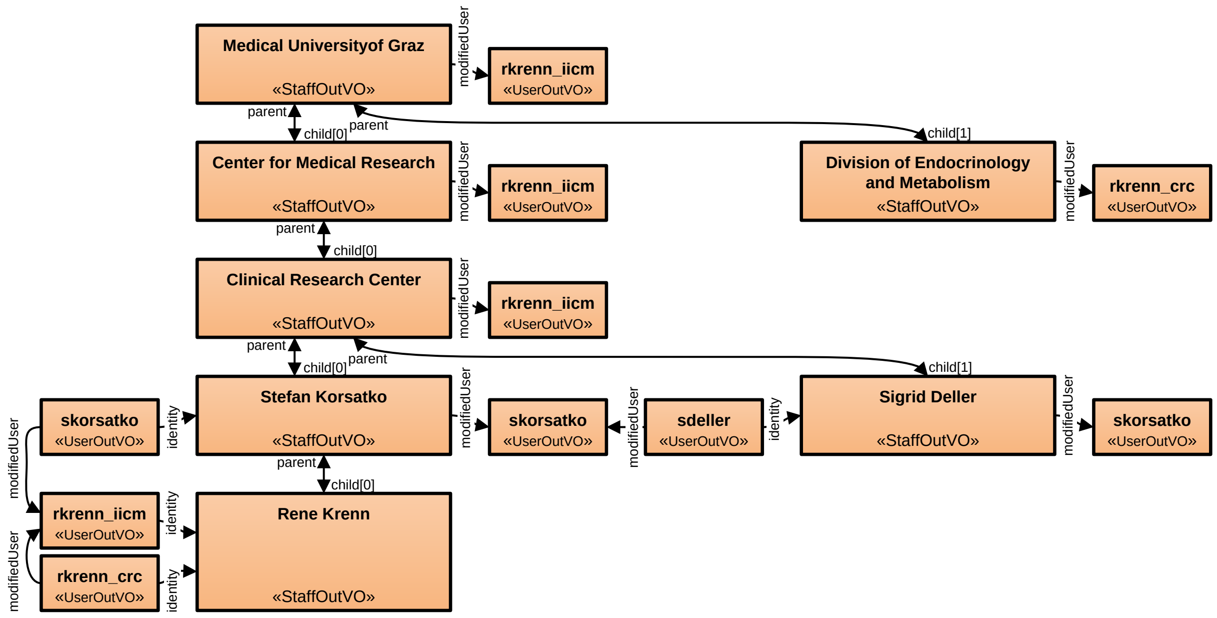


Figure 4.28.: A OutVO instance graph like this could be returned by an invocation of the `getStaff` or `getUser` service method. Effectively, eight person/organisations and four user records are loaded instead of one. While this beneficial for an UI view dedicated to display this entire person/organisation tree, it is a performance drawback for remaining situations when a single person's name is to be displayed only.

Wether introducing OutVO models including cycles or not should be considered carefully, since they come along with both advantages and disadvantages:

Pros:

- OutVO instance graphs allowing circular references reflect complete and therefore correct representations, i.e. there are no items left out in order to cut off circular references.
- The presentation layer's frequency of service method invocations can be reduced by providing "bloated" OutVO graphs for UI screens that need to display a high level of details at once. This is particularly true when displaying the tree hierarchies from figure 4.12, 4.13 and 4.14.
- There is an upper bound of memory allocated for OutVO instance graphs with circular references, since the number of created objects is limited by the number of distinct items. In contrast, lazy-loadig tree hierarchies[17] result in instantiating copies of OutVO objects with the same content again and again.
- Taking circular references into account to consider association cycles will yield a robust implementation. The manifestation of cycles in tree hierarchies is a rare real-life use case however, e.g. an inventory item will not contain itself. The tree hierarchies shown in figure 4.12, 4.13 and 4.14 are supposed to vorbid cycles. However, corresponding OutVO are subject to curcular references, since their reflexive associations are bidirectional navigatable in order to get versatile

[16]Lazy-loading is enabled by default as of Hibernate 3.
[17]Child nodes showing OutVO instances are loaded upon expanding a parent node. This technique is required for large trees such as file directory structures.

OutVOs. Hence, both parent/parents and child/children fields exists and inherently cause circular references.

Cons:

- DAO implementation classes will need a complex transformation logic to construct circular references.
- UI tree components and some libraries used by the web tier such as Gson (Leitch et al., 2014) cannot handle circular references directly.
- Creating an instance graph (figure 4.28) from a corresponding an OutVO model with association cycles (figure 4.27) may cause loading too many or even all existing records.

As a conclusion, VO graphs with cycles should be avoided unless there is an explicit need at the presentation layer side. Therefore, the overall number of anticipated cycles in OutVO models is minimal (figure 4.29).

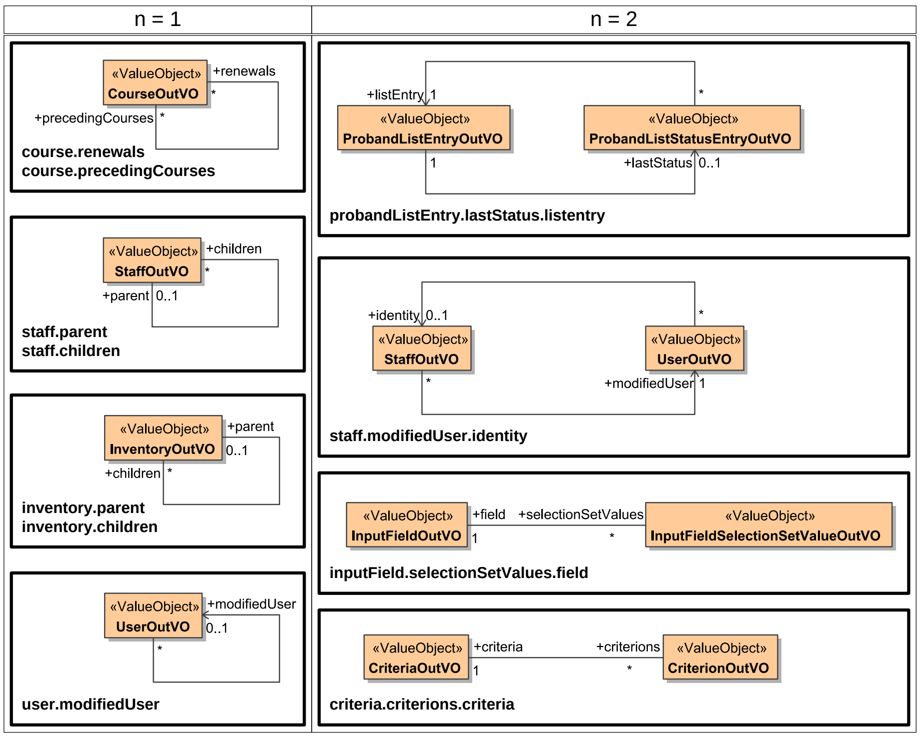
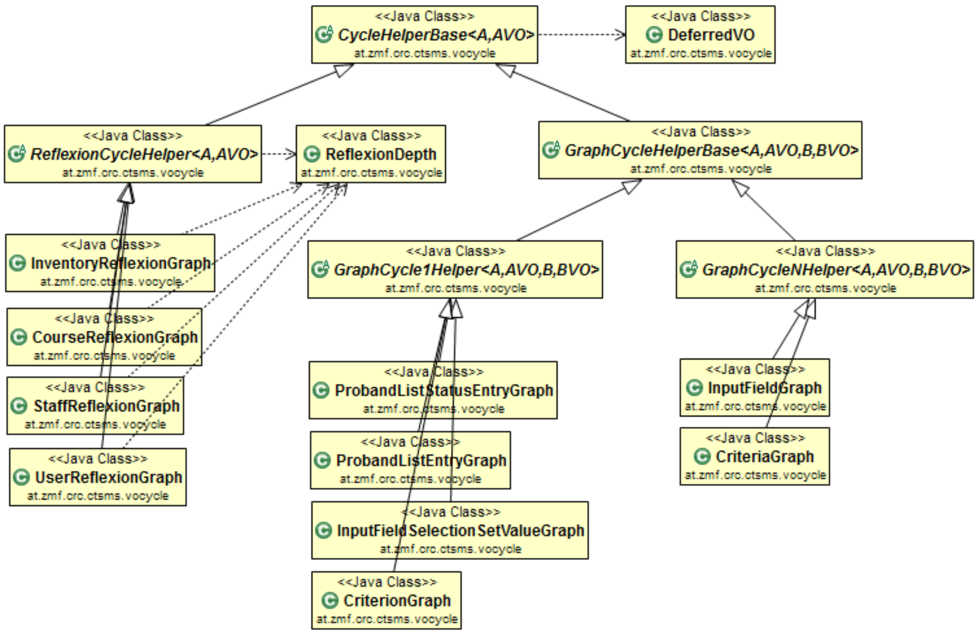


Figure 4.29.: Eight association cycles can be identified among all OutVOs (app. 150). They are formed by reflexive ($n = 1$), bidirectional navigatable associations and pairs of associations ($n = 2$).

Accordingly, the `UserOutVO`/`StaffOutVO` graph from figure 4.27 contains the following cycles:

- `StaffOutVO` reflexive, bidirectional navigatable association:
  - `staff.parent`
  - `staff.children`
- `UserOutVO` reflexive association:
  - `user.modifiedUser`
- `staff.modifiedUser.identity`

The idea for the implementation of VO transformation logic capable of handling cycles is to treat identified cycles as building blocks. *Cycle helpers* are defined and structured by the class hierachy shown in figure 4.30. Association end points are handled with instances of `GraphCycleHelperBase` implementations for the corresponding multiplicity (`GraphCycle1Helper`, `GraphCycleNHelper`). While assembling VO instances, they populate and hand over a `HashMap` (`voMap`) that is used to check VO instances that were already traversed. Concrete implementation classes inherit from a graph cycle helper with the matching multiplicity and adapt to the end point's entity type. While `GraphCycle1Helper` and `GraphCycleNHelper` are used for $n \geq 2$ cycles, there is a integrated variant for reflexive associations with bidirectional navigability (`ReflexionCycleHelper`).



<<Java Class>> CycleHelperBase<A,AVO> at.zmf.crc.ctsms.vocycle
<<Java Class>> DeferredVO at.zmf.crc.ctsms.vocycle
<<Java Class>> ReflexionCycleHelper<A,AVO> at.zmf.crc.ctsms.vocycle
<<Java Class>> ReflexionDepth at.zmf.crc.ctsms.vocycle
<<Java Class>> GraphCycleHelperBase<A,AVO,B,BVO> at.zmf.crc.ctsms.vocycle
<<Java Class>> InventoryReflexionGraph at.zmf.crc.ctsms.vocycle
<<Java Class>> CourseReflexionGraph at.zmf.crc.ctsms.vocycle
<<Java Class>> StaffReflexionGraph at.zmf.crc.ctsms.vocycle
<<Java Class>> UserReflexionGraph at.zmf.crc.ctsms.vocycle
<<Java Class>> GraphCycle1Helper<A,AVO,B,BVO> at.zmf.crc.ctsms.vocycle
<<Java Class>> GraphCycleNHelper<A,AVO,B,BVO> at.zmf.crc.ctsms.vocycle
<<Java Class>> ProbandListStatusEntryGraph at.zmf.crc.ctsms.vocycle
<<Java Class>> ProbandListEntryGraph at.zmf.crc.ctsms.vocycle
<<Java Class>> InputFieldSelectionSetValueGraph at.zmf.crc.ctsms.vocycle
<<Java Class>> CriterionGraph at.zmf.crc.ctsms.vocycle
<<Java Class>> InputFieldGraph at.zmf.crc.ctsms.vocycle
<<Java Class>> CriteriaGraph at.zmf.crc.ctsms.vocycle

Figure 4.30.: Concrete graph cycle helpers represent end points of associations forming cycles.

The AndroMDA DAO template (`SpringDao.vsl`) was modified to generate transformation methods including the `voMap` passthrough parameter, which are required by cycle helper classes. These methods are overridden within a DAO implementation class to put the cycle helpers into place. As an example, listing 4.4 and listing 4.5 shows the wiring for the `inputField.selectionSetValues.field` cycle. Cycle helpers integrate the transformation of all remaining VO fields. On top, DAO transformation methods with default signatures containing the source and target parameters only provide the same interface used for VOs without cycles.

```java
public class InputFieldDaoImpl extends InputFieldDaoBase {

    @Override
    public void toInputFieldOutVO(InputField source, InputFieldOutVO target) {

        toInputFieldOutVO(source, target, new HashMap<Class, HashMap<Long, Object>>());

    }

    @Override
    public void toInputFieldOutVO(Staff source, InputFieldOutVO target, HashMap<Class, HashMap<Long, Object>> voMap) {

        (new InputFieldGraph(this, this.getInputFieldSelectionSetValueDao(), this.getUserDao()))
            .toVOHelper(source, target, voMap); //TODO: DAO DI for cycle helpers

    }

    ...
```

Listing 4.4: The `InputFieldOutVO` transformation method instantiates `InputFieldGraph` `GraphCycleNHelpers` to serve the `selectionSetValues` end point of the `inputField.selectionSetValues.field` cycle.

```
public class InputFieldSelectionSetValueDaoImpl extends InputFieldSelectionSetValueDaoBase {

    @Override
    public void toInputFieldSelectionSetValueOutVO(InputFieldSelectionSetValue source,
        InputFieldSelectionSetValueOutVO target) {

            toInputFieldSelectionSetValueOutVO(source, target, new HashMap<Class, HashMap<Long, Object>>());

    }

    @Override
    public void toInputFieldSelectionSetValueOutVO(InputFieldSelectionSetValue source,
        InputFieldSelectionSetValueOutVO target, HashMap<Class,HashMap<Long,Object>> voMap) {

        (new InputFieldSelectionSetValueGraph(this, this.getInputFieldSelectionSetValueDao(), this.getUserDao()))
            .toVOHelper(source, target, voMap); //TODO: DAO DI for cycle helpers

    }

    ...
```

Listing 4.5: The `InputFieldSelectionSetValueOutVO` transformation method sets up an `InputFieldSelectionSetValueGraph` `GraphCycle1Helper` to serve the `field` end point of the `inputField.selectionSetValues.field` cycle.

The size of object graphs that results from association cycles showed up as critical performance factor for the entire application. Although the number of distinct cycles in OutVO models is kept down, the reason is the frequent occurrence of the particular `UserOutVO` graph from figure 4.27. Most OutVOs include this graph with their `modifiedUser` field, which belongs to the common record versioning footprint (section 4.1.3.2). As a consequence, large object graphs from figure 4.28 would be created as part of result values of most service methods. As a countermeasure, the cycle helper's logic limits instance counts per type. The limits are set to adequate default values per association end point. Individual instance count limits can be passed by an extra parameter (`maxInstances`) of another additional transformation method overload, which is generated by the modified AndroMDA template. `ReflexionCycleHelpers` provide an extra limitation mechanism for the parent/parents and child/children depth. This is what the `ReflexionDepth` class in figure 4.30 is used for. VO transformations use a *depth-fist* traversal order by default. As an alternative, the `DeferredVO` class is used to drive a *breadth-first* traversal. With instance count limitations, breadth-first traversal is more reasonable when displaying the OutVO graph in a tree view since displayed trees are balanced and child nodes can be sorted.

An alternative solution for VO tranformation with respect to association cycles is given by `ValueTreeBuilder`, referenced in (Truskaller, 2003, pp. 40-42). Its single-class approach is based on reflection and "include trees", which are patterns passed in a string format for associations to process. However, it relies on the strict mapping of entity[18] and VO fields, and lacks an instance count limitation mechanism. Instead of a transparent integration in DAO transformation methods, it is designed to be explicitly invoked in service methods.

4.2.2. Internationalisation

Internationalisation (i18n) turned out as must-have requirement for some use cases during refinement iterations. The most prominent example was the user's final preference of German as UI language to reduce the risk of misuse, while exported CV documents have to be uniform documents in English language. This example indicates that various application components are subject to i18n:

- web UI
- PDF export
- Excel export
- email notifications
- audit trail

[18]The original implementation applies to Enterprise JavaBeans (EJB) entities but could be ported to work with Hibernate managed entities.

The Java platform and the JSF framework support i18n with a number of facilities. These were utilized to implement the following i18n aspects throughout the application:

- date/datetime formatting
- time zones
- language translations

Date formats, time zones Date (and Datetime)[19] presentation is an exclusive concern of the web tier. Custom *JSF converters* are implemented to convert `java.util.Date` objects of OutVO fields into strings. `java.text.DateFormat` is utilized to apply configurable format string patterns and a `java.util.TimeZone` argument. Time zones are handled by introducing a *server time zone* to persist and calculate (with) dates consistently as well as a time zone user setting (figure 4.31) to display dates with the desired time zone offset. Some time zones define an additional Daylight Saving Time (DST) offset, which has to be considered when converting between Java and JavaScript date representations. Workarounds were necessary to harmonize Java application code, PrimeFaces components and Js libraries in order to apply the DST offset consistently.

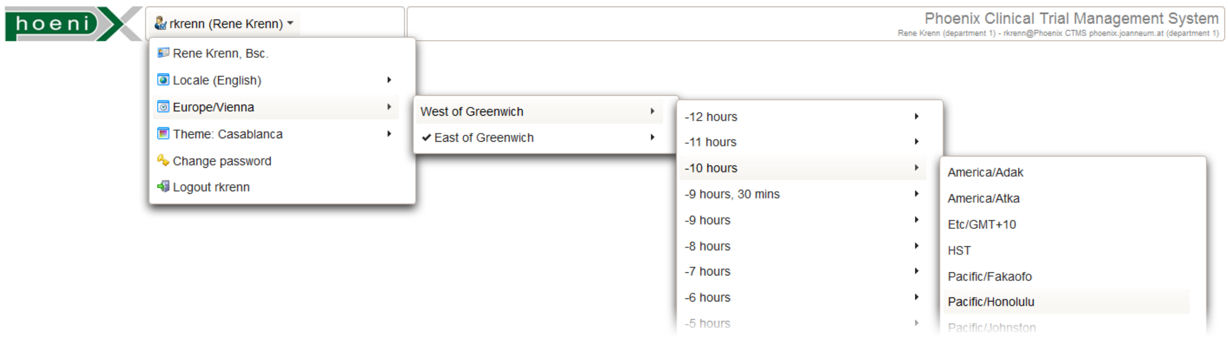


Figure 4.31.: A user can select the UI time zone preferrence from the menu bar available to all screens of the system.

Language translations Basically, translation is the simple lookup of identifiers in a table of string literals prepared for the desired language. There is no substitution hierarchy of recurring terms or phrases, hence the constitution of string literals reduces to a plain string table including redundancies. To structure the large string table, it is split into multiple *bundles* to separate by the type of elements subject to translation. Each element type such as an enumeration type or master data entity gets its own *bundle*. According to the `java.util.Properties` API, a bundle consists of a `<bundle base name>.properties` file that defines the bundle's base name and contains key-value pairs for a default language (English). A separate file is added be per supported language (e.g. `<bundle base name>_de.properties`). This way, concrete *Localisations (l10ns)* for English (`en`) and German (`de`) language are provided.

[19]Throughout the application, there is no entity field with time information only. However, a time input may be required for EDC purposes. In this case, users can prepare a single-line text input field for entering the time of day and specify a custom RegEx validation such as `/^[0-9]{2}:[0-9]{2}$/`.

scope	category of bundle content	#bundles	#bundle keys
DAO	static master data for option items of UI dropdowns	30	319
DAO	enumeration literals	10	73
service method	exception messages	4	541
DAO	preset string values	6	37
DAO	permission profile names	2	82
DAO	Audit Trail: system message titles, comment bodies and field names in record dumps	2	404
DAO	notification message titles and template names[20]	3	57
DAO	column names for criterions displayed by the query editor UI	3	194
Excel export	spreadsheet column names	3	173
PDF export	labels for PDF documents	5	68

Table 4.4.: Volume of bundles covered by the *core tier*.

scope	category of bundle content	#bundles	#bundle keys
facelet	common UI labels	1	285
managed beans	UI messages, preset string values and labels	1	548
facelet	inventory module UI labels and tooltips	1	201
facelet	person/organisation module UI labels and tooltips	1	334
facelet	course module UI labels and tooltips	1	174
facelet	trial module UI labels and tooltips	1	426
facelet	proband module UI labels and tooltips	1	294
facelet	user module UI labels and tooltips	1	88
facelet	input field module UI labels and tooltips	1	135

Table 4.5.: Volume of bundles covered by the *web tier*.

Language translations take place in all application layers. Static labels and tooltip texts of UI pages are externalized into bundles (table 4.5) and can be refererd in JSF facelets directly. Remaining strings are covered by bundles part of the core tier (table 4.4), which are handled with a custom utility. The `L10nUtil` class (listing 4.6) arranges methods for the effective conversion of identifiers into translated, user presentable strings (localized string). It encapsulates the required logic for selecting language locales, loading bundles with Unicode Transformation Format (UTF)8 encoded `.properties` files and retrieving/formatting strings from bundles.

[20] i18n of notification message bodies is implemented separately using Velocity templates.

```
public final class L10nUtil {
    private final static BundleControl BUNDLE_CONTROL = new BundleControl(); //custom ResourceBundle.Control for UTF8 support
         and custom fallback bundle

    private L10nUtil() {}

    //context locales definitions:
    public enum ContextLocale {
        USER,
        JOURNAL, NOTIFICATION, CV_PDF, //...
        DEFAULT
    }
    //variable for storing a configured locale, e.g. for CV PDF documents:
    private static Locale cvPdfLocale;
    @Autowired(required = true) //spring will set a value of e.g. "en" at application start up ...
    public void setCvPdfLocale(String cvPdfLocale) {
        L10nUtil.cvPdfLocale = getLocaleFromString(cvPdfLocale); //... or fails here for invalid language tokens.
    }
    //convert the context locale to its effective java.util.Locale:
    public static Locale getLocale(ContextLocale locale) {
        switch (locale) {
            case USER: //a context locale can map a dynamic locale ...
                return CoreUtil.getUserContext().getLocale();
            case JOURNAL: //... or fixed, configured ones:
                return journalDatabaseWriteLocale;
            case NOTIFICATION:
                return notificationsDatabaseWriteLocale;
            case CV_PDF:
                return cvPdfLocale;
            //...
            case DEFAULT:
            default:
                return Locale.getDefault();
        }
    }

    //loading a localised .property file:
    private static ResourceBundle getBundle(Locale locale, String baseName) {
        if (locale != null) {
            try {
                return ResourceBundle.getBundle(baseName, locale, BUNDLE_CONTROL);
            } catch (MissingResourceException e) {
                //the requested language file (e.g. <baseName>_de.properties) is missing
            }
        }
        //failover to "<baseName>.properties" that should contain the default "en" localisation:
        return ResourceBundle.getBundle(baseName, BUNDLE_CONTROL); //fail here if missing
    }
    private static ResourceBundle getBundle(ContextLocale locale, String baseName) {
        return getBundle(getLocale(locale), baseName);
    }

    //get translated strings:
    private static String getMessage(String l10nKey, ResourceBundle bundle, String defaultString, Object... args) { ... }
    private static String getString(String l10nKey, ResourceBundle bundle, String defaultString) {
        if (l10nKey == null || l10nKey.trim().length() == 0) { //no key, apply a default value:
            return defaultString;
        }
        if (bundle == null) { //no bundle, display the key:
            return l10nKey;
        }
        try { //try to load the key:
            return bundle.getString(l10nKey);
        } catch (MissingResourceException e) { //key not found:
            return l10nKey;
        } catch (ClassCastException e) { //key value not a string?
            return l10nKey;
        }
    }

    //the subsequent pattern is used per particular bundle, e.g. trial status type names:
    private static String trialStatusTypesBundleBasename;   //variable for storing the base name of a .properties file that
                                 //contains the default localisation
    @Autowired(required = true) //Spring will configure to e.g. "ctsms-trialstatustypes" at application start up ...
    public void setTrialStatusTypesBundleBasename(String trialStatusTypesBundleBasename) {
        L10nUtil.trialStatusTypesBundleBasename = trialStatusTypesBundleBasename;
        getBundle(Locales.DEFAULT, trialStatusTypesBundleBasename); //... or fails here if a default localisation is missing
    }
    //the final interface method that converts a given trial status type identifier to a user presentable, translated string:
    public static String getTrialStatusTypeName(ContextLocale locale, String l10nKey) {
            return CommonUtil.getString(l10nKey, getBundle(locale, trialStatusTypesBundleBasename),
                DefaultMessages.TRIAL_STATUS_TYPE_NAME);
    }

    //next bundle ...
}
```

Listing 4.6: The `L10nUtil` utility class.

A typical usage of the `L10nUtil` utility is given in entity-to-VO transformation methods of master data entity DAOs. The appropriate `L10nUtil` method is called when populating a localized VO field (listing 4.7).

```
public class TrialStatusTypeDaoImpl extends TrialStatusTypeDaoBase {

    @Override
    public void toTrialStatusTypeVO(TrialStatusType source, TrialStatusTypeVO target) {

        super.toTrialStatusTypeVO(source, target);

        //wherever the name of a trial status type appears in UI screens, it has to be localized according to the
        //user's language setting. getTrialStatusTypeName looks up a given identifier string (e.g. "phase_out") to get
        //the localized string from the "ctsms-trialstatustypes" bundle:
        //  de: "Abschluss" in ctsms-trialstatustypes_de.properties
        //  en: "phase out" in ctsms-trialstatustypes_en.properties
        target.setName(L10nUtil.getTrialStatusTypeName(ContextLocales.USER, source.getNameL10nKey()));

        //target.set...

    }

    ...
```

Listing 4.7: A DAO transformation method invokes a `L10nUtil` method to populate localized VO field values.

The method parameters of `ContextLocale` enumeration type are an intermediate step to support dynamic UI language selection. Beside the configuration of fixed languages (e.g. `ContextLocale.CV_PDF`), the implementation supports language selection according to a user setting (`ContextLocale.USER`). Users can switch to their preferred UI language directly from any screen in the same way they can choose their time zone. Spring is used to configure fixed locales (e.g. `cvPdfLocale`) and bundle names (listing 4.8). This allows the application to fail during startup if a bundle is missing.

```
 ...
<bean id="l10nUtil" class="at.zmf.crc.ctsms.util.L10nUtil">
    <property name="defaultLocale" value="en" />
    <property name="timeZone" value="Europe/Vienna" />
     ...

    <property name="journalDatabaseWriteLocale" value="de" />
    <property name="notificationsDatabaseWriteLocale" value="de" />
    <property name="cvPdfLocale" value="en" />

    <property name="trialStatusTypesBundleBasename" value="ctsms-trialstatustypes" />
     ...
</bean>
 ...
```

Listing 4.8: i18n settings cover static language settings and the declaration of bundle base names. This definitions are part of the Spring configuration for the `L10nUtil` bean.

In contrast to user settings for language and time zone, the remaining i18n aspects listed below are globally configurable or considered by specific implementations:

Collation When sorting strings, collation determines the ordering of language specific symbols like umlauts. Similar to the character encoding of persisted string values (UTF8), the collation (`de_DE.UTF-8`) is configured globally for the PostgreSQL RDBMS at database level for all contained table columns. These settings cannot be altered once the database is initialized. However, dynamic collation could be supported by the Java application itself using `java.text.Collator` in combination with comparators (`java.util.Comparator`). Dedicated comparators are provided for specific sorting where required only. For example, a user-friendly, "natural" (alphanumeric) sorting is applied to files and folders displayed in directory tree views.

Number formats When numbers are converted to strings in order to be displayed in UI screens, the locale-independent `toString` implementation of the concrete `java.lang.Number` type applies.

Likewise, parsing of entered numbers works with invocation of a complementary string parsing method such as `parseFloat`. Both conversion directions are handled by default JSF converters such as `javax.faces.convert.FloatConverter`. PrimeFaces input components use JSF default converters unless a custom converter is specified. Currently, custom converters are implemented for the date/datetime input components (`p:calendar`) only. According to `java.text.DateFormat`, `java.text.NumberFormat` could be used to implement a custom number converter. In order to implement a user-dependent number format, the number converter would be bound to components for numerical input such as `p:spinner`. A variable number format would introduce additional complexity for the Js-based form scripting implementation and is therefore not supported at the moment.

Measuring units Measuring units are a definitive topic for data entry of medical information. However, it was decided to avoid master data for measurement unit definitions, since a global unit conversion matrix would be cumulative and thus cumbersome to maintain. When setting up input fields for EDC forms, users are supposed to include textual information about the expected measurement unit of a numerical input field type themselves. Form scripting can be used to convert and output values in a unit as required.

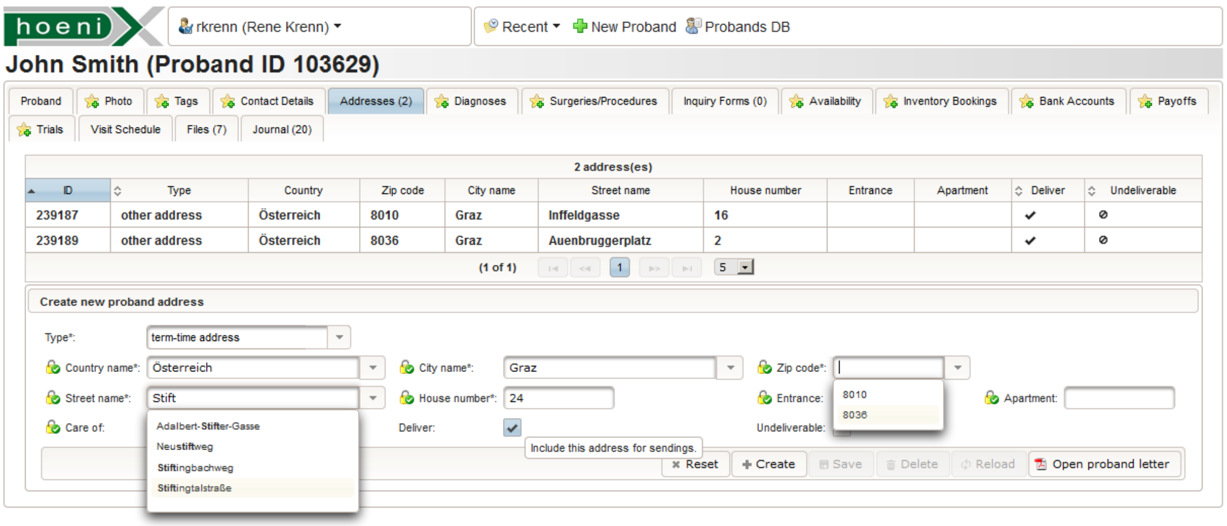
Currency A few UI views contain input elements for storing data representing amounts of money (figure 4.21). Java provides `java.util.Currency` for representing currency names and codes, which is currently not used however. In order to dynamically apply a currency setting, currency conversion would be necessary, which is considered as an overkill for the scope of the application.

Phone numbers Phone numbers are validated by a RegEx to enforce the format specified in the E.164 ITU-T recommendation (*Recommendation E.164: The international public telecommunication numbering plan* 2010, p. 7). This will be advantageous when interfacing with a telephone system/Private Branch Exchange (PBX).

Postal codes and addresses Persisting address information (`StaffAddress`, `ProbandAddress`) is designed to support a postal address format as commonly used in Austria (Österreichische Post AG, 2013). An address is broken down an stored by multiple entity fields of string type:

1. country name
2. city name
3. ZIP code
4. street name
5. house number
6. entrance
7. door number

UI input elements for all address elements except the postal code allow free text input. For the postal code, another RegEx can be configured to restrict the format, e.g. to a four-digit number. To promote correct and consistent spelling, correlated `p:autoComplete` inputs are provided for address elements 1 to 4 (figure 4.32). The input suggestions are backed by master data (list of countries, list of city names with postal codes and street directoy).



John Smith (Proband ID 103629)

2 address(es)

ID	Type	Country	Zip code	City name	Street name	House number	Entrance	Apartment	Deliver	Undeliverable
239187	other address	Österreich	8010	Graz	Inffeldgasse	16			✓	⊘
239189	other address	Österreich	8036	Graz	Auenbruggerplatz	2			✓	⊘

(1 of 1) 1 5

Create new proband address

Type*: term-time address
Country name*: Österreich
City name*: Graz
Zip code*:
Street name*: Stift
House number*: 24
Entrance:
Apartment:
Care of:
Deliver: ✓
Undeliverable:

Adalbert-Stifter-Gasse
Neustiftweg
Stiftingbachweg
Stiftingtalstraße

8010
8036

Include this address for sendings.

Figure 4.32.: This sample UI screen shows the tab for managing a proband's addresses. PrimeFaces' `p:autoComplete` inputs instantly query address master data for a fast and consistent data entry. The suggestions are context-sensitive: if a street name is provided at first, available ZIP codes are presented. Otherwise street name suggestions are restricted to the ZIP codes code entered before.

Holidays Holidays are displayed by calendar views and used to calculate weighted shift durations. Since holidays declarations are a matter of regional regulations, they are can be set up as required in a separate master data table.

4.2.3. Encryption

While a CDMS can manage proband related data by reference to a pseudonym like a subject ID, this is apparently not an option for systems used to manage proband contacts. A proband database will be used to register probands by storing PII such as name, DoB, contact details and addresses. For each recruited subject whose data is to be stored, an own-hand signed agreement is required to exist. This data privacy consent will obligate the secrecy of any PII provided. Access control features of database applications (Authentication and Authorisation) are a fundamental measure to veritably ensure secrecy. The blocking logic keeps unauthorised persons from opening screens showing confidential records. Most applications will settle with this, although there are unresolved issues:

System administrators: A database administrator could actually be an unauthorised person in the context of the privacy consent. Nevertheless a database admin can bypass application logic because of the direct read and write table access required for his/her job.

Hosting location: If the database is hosted on servers in a foreign country, differing data privacy regulation may apply. These may conflict with regional regulations, which is a known issue with SaaS.

Database backups: Database backups like SQL dumps will contain table contents and could be read e.g. by system admins or others that can access the storage location of backup files.

Data theft: A malicious party that manages to compromise a server can potentially access the database in the same way the admin is allowed to. Even worse, it is possible to copy and divert or forge database contents or backups.

An effective countermeasure to these scenarios is table column encryption (figure 4.33).

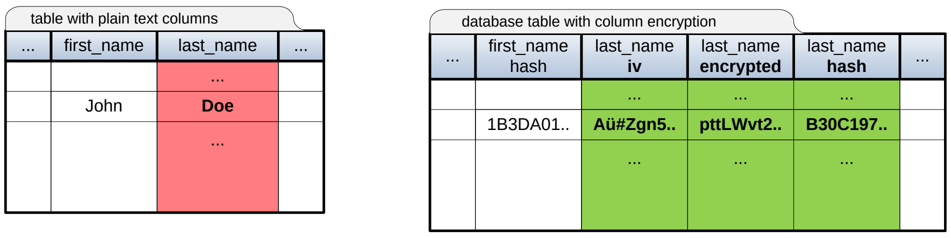


Figure 4.33.: The column encryption scheme shown is used to protect entity field values that contain probands' PII against unauthorized access. Each field column is therefore expanded to provide additional storage of IV values and a hash value required for search queries (optional).

Using a static encryption key that can be found in some configuration file, Hardware Security Module (HSM) or confidential system documentation will have management-related disadvantages at least. The Phoenix CTMS introduces column encryption by means of PBE to utilize its integrated user management. Independent of the system's access control features there is no way to read encrypted columns of raw database content without knowing at least *one* user's password. This implies that it is vital that no single user password is ever revealed. Password safety is therefore enforced by the application according to constraints given by a *password policy*. With the setup of a user account, decryption is basically granted to the account holder. The access control's blocking logic is used for detailed restrictions throughout application modules.

Since multiple users need to be able to encrypt and decrypt, the concept of a *group key* is required. A symmetric encryption algorithm like Advanced Encryption Standard (AES) encrypts and decrypts secret entity field values with a group key that is generated during the initial system setup. When a user's password is set, a new user password entity is persisted that contains the PBE-encrypted group key using the password string. For business logic accessing secret entity fields, the key can be restored by decryption with provided authentication credentials.

Different group keys will be used to support multi-client capabilities, therefore a group key is denoted as *department key*. Departments form encryption domains due to their differing department keys. During the initial system setup, departments need to be created first with interactive procedures (section 4.10.2) provided by the CLI tool. Furthermore, an initial user has to be created per department (listing 4.68) in order to log on and create further user accounts using the web UI. When creating new users via the web UI, the department key can be derived from the executing user's context (listing 4.22). However, each CLI procedure would require to enter the 128 bit department key. To overcome entering the key's non-printable character bytes, PBE is utilized once more. The actual department key is randomly generated when creating a new department but persisted in PBE-encrypted form by specifying a user friendly *department passphrase*. This intermediate step will additionally allow changing the department's passphrase without the time-consuming need to decrypt and re-encrypt any encrypted database contents created so far.

The described encryption and decryption procedures are summarized below. The implementation of referenced encryption, decryption and initialisation methods is shown in listing 4.9.

Create new department (system setup)

1. plaintext department passphrase $P_{DP} \leftarrow$ console input
2. department key K_D = `randomAESKey`
3. department key salt S_{K_D} = `createSalt`
4. encrypted department key C_{K_D}, department key initialisation vector IV_{K_D}:
   $(C_{K_D}, IV_{K_D}) = \mathbf{E}_{PBE:P_{DP},S_{K_D}}(K_D)$[21]
5. S_{K_D}, C_{K_D}, $IV_{K_D} \rightarrow$ `Department` entity $\rightarrow$ persist

Encrypt department passphrase for a user (system setup)

1. P_{DP}, plaintext user password $P_{UP} \leftarrow$ console input
2. department passphrase salt S_{DP} = `createSalt`
3. encrypted department passphrase C_{DP}, department passphrase initialisation vector IV_{DP}:
   $(C_{DP}, IV_{DP}) = \mathbf{E}_{PBE:P_{UP},S_{DP}}(P_{DP})$
4. S_{DP}, C_{DP}, $IV_{DP} \rightarrow$ `Password` entity $\rightarrow$ persist

Encrypt an entity field value (regular operation)

1. S_{K_D}, C_{K_D}, IV_{K_D}, S_{DP}, C_{DP}, $IV_{DP} \leftarrow$ load `Department`, `Password` $\leftarrow$ user credential parameter
2. $P_{UP} \leftarrow$ password credential parameter
3. plaintext field value $P_{FV} \leftarrow$ InVO input parameter
4. $P_{DP} = \mathbf{D}_{PBE:P_{UP},S_{DP}}(C_{DP}, IV_{DP})$[22]
5. $K_D = \mathbf{D}_{PBE:P_{DP},S_{K_D}}(C_{K_D}, IV_{K_D})$
6. encrypted field value C_{FV}, field value initialisation vector IV_{FV}:
   $(C_{FV}, IV_{FV}) = \mathbf{E}_{AES:K_D}(P_{FV})$
7. C_{FV}, $IV_{FV} \rightarrow$ entity $\rightarrow$ persist

Decrypt the entity field value (regular operation)

1. S_{K_D}, C_{K_D}, IV_{K_D}, S_{DP}, C_{DP}, $IV_{DP} \leftarrow$ load `Department`, `Password` $\leftarrow$ user credential parameter
2. $P_{UP} \leftarrow$ password credential parameter
3. C_{FV}, $IV_{FV} \leftarrow$ load entity $\leftarrow$ entity ID input parameter
4. $P_{DP} = \mathbf{D}_{PBE:P_{UP},S_{DP}}(C_{DP}, IV_{DP})$
5. $K_D = \mathbf{D}_{PBE:P_{DP},S_{K_D}}(C_{K_D}, IV_{K_D})$
6. $P_{FV} = \mathbf{D}_{AES:K_D}(P_{FV}, IV_{FV})$
7. $P_{FV} \rightarrow$ OutVO return value

[21] $\mathbf{E}_{PBE:P,S} \equiv \mathbf{E}_{AES:\texttt{createPBEKey(S, P)}}$
[22] $\mathbf{D}_{PBE:P,S} \equiv \mathbf{D}_{AES:\texttt{createPBEKey(S, P)}}$

```java
//final class at.zmf.crc.ctsms.security.CryptoUtil
private static final String PBE_KEY_ALGORITHM = "PBKDF2WithHmacSHA1";
  private static final int PBE_KEY_ITERATIONS = 1000;
public static final String SYMMETRIC_ALGORITHM = "AES";
  private static final String SYMMETRIC_ALGORITHM_MODE = SYMMETRIC_ALGORITHM + "/CBC/PKCS5Padding";
  private static final int SYMMETRIC_KEY_LENGTH = 128;
private static final String SALT_ALGORITHM = "SHA1PRNG";
  private static final int SALT_LENGTH = 16;
//ENCRYPTIONS:
public static CipherText encrypt(byte[] salt, String password, Serializable plainText) throws Exception { //E_PBE
    Cipher cipher = getEncryptionCipher(createPBEKey(salt, password));
    return new CipherText(cipher.getIV(), encrypt(cipher, serialize(plainText)));
}
public static CipherText encrypt(SecretKey key, Serializable plainText) throws Exception { //E_AES
    Cipher cipher = getEncryptionCipher(key);
    return new CipherText(cipher.getIV(), encrypt(cipher, serialize(plainText)));
}
public static CipherText encrypt(Serializable plainText) throws Exception { //department key shortcut for DAOs
    return encrypt(CoreUtil.getUserContext().getDepartmentKey(), plaintext);
}
private static Cipher getEncryptionCipher(SecretKey key) throws Exception { //create encryption mode AES cipher
    Cipher cipher = Cipher.getInstance(SYMMETRIC_ALGORITHM_MODE);
    cipher.init(Cipher.ENCRYPT_MODE, key); //random initialisation vector
    return cipher;
}
private static byte[] encrypt(Cipher cipher, byte[] plainText) throws //encrypt a byte array
    ShortBufferException, IllegalBlockSizeException, BadPaddingException {
    byte[] cipherText = new byte[cipher.getOutputSize(plainText.length)];
    int ctLength = cipher.update(plainText, 0, plainText.length, cipherText, 0);
    ctLength += cipher.doFinal(cipherText, ctLength);
    return Arrays.copyOf(cipherText, ctLength);
}
//DECRYPTIONS:
public static Object decrypt (byte[] salt, String password, byte[] iv, byte[] cipherText) throws Exception { //D_PBE
    return deserialize(decrypt(createPBEKey(salt, password), cipherText));
}
public static Object decrypt (SecretKey key, byte[] iv, byte[] cipherText) throws Exception { //D_AES
    return deserialize(decrypt(getDecryptionCipher(iv, key), cipherText));
}
public static Object decrypt (byte[] iv, byte[] cipherText) throws Exception { //department key shortcut for DAOs
    return decrypt(CoreUtil.getUserContext().getDepartmentKey(), iv, cipherText);
}
private static Cipher getDecryptionCipher(byte[] iv, SecretKey key) throws Exception { //create decryption mode AES cipher
    Cipher cipher = Cipher.getInstance(SYMMETRIC_ALGORITHM_MODE);
    IvParameterSpec spec = new IvParameterSpec(iv);
    cipher.init(Cipher.DECRYPT_MODE, key, spec);
    return cipher;
}
private static byte[] decrypt(Cipher cipher, byte[] cipherText) throws //decrypt a byte array
    ShortBufferException, IllegalBlockSizeException, BadPaddingException {
    int ctLength = cipherText.length;
    byte[] plainText = new byte[cipher.getOutputSize(ctLength)];
    int ptLength = cipher.update(cipherText, 0, ctLength, plainText, 0);
    ptLength += cipher.doFinal(plainText, ptLength);
    return Arrays.copyOf(plainText,ptLength);
}
//utilities and helpers:
public static SecretKeySpec createRandomKey() throws NoSuchAlgorithmException { //createAESKey()
    KeyGenerator kgen = KeyGenerator.getInstance(SYMMETRIC_ALGORITHM);
    kgen.init(SYMMETRIC_KEY_LENGTH);
    return new SecretKeySpec(kgen.generateKey().getEncoded(), SYMMETRIC_ALGORITHM);
}
public static byte[] createSalt() throws NoSuchAlgorithmException { //createSalt()
    SecureRandom rand = SecureRandom.getInstance(SALT_ALGORITHM);
    byte[] salt = new byte[SALT_LENGTH];
    rand.nextBytes(salt);
    return salt;
}
//create a 128 bit AES key from a given password string according to the \myac{pkcs} \#5 v2.0 standard described in RFC 2898:
private static SecretKey createPBEKey(byte[] salt, String password) throws NoSuchAlgorithmException, InvalidKeySpecException {
    if (password == null || password.length() == 0) { //zero-length passwords won't work
        throw new IllegalArgumentException(L10nUtil.getMessage(MessageCodes.PBE_PASSWORD_ZERO_LENGTH_ERROR, DefaultMessages.
               PBE_PASSWORD_ZERO_LENGTH_ERROR));
    }
    SecretKeyFactory factory = SecretKeyFactory.getInstance(PBE_KEY_ALGORITHM);
    PBEKey pbeKey = (PBEKey) factory.generateSecret(new PBEKeySpec(
    password.toCharArray(), salt, PBE_KEY_ITERATIONS, SYMMETRIC_KEY_LENGTH));
    return new SecretKeySpec(pbeKey.getEncoded(), SYMMETRIC_ALGORITHM);
}
//two user context helpers. The user department's passphrase and key will be decrypted and cached within the user context on demand.
public static byte[] decryptDepartmentKey(Department department, String plainDepartmentPassword) throws Exception {
    return decrypt(department.getKeySalt(), plainDepartmentPassword, department.getKeyIv(), department.getEncryptedKey());
}
public static String decryptDepartmentPasswphrase(Password userPassword, String plainPassword) throws Exception {
    return (String) decrypt (userPassword.getDepartmentPasswordSalt(), plainPassword, userPassword.getDepartmentPasswordIv(), userPassword.
          getEncryptedDepartmentPassword());
}
```

Listing 4.9: The `CryptoUtil` class provides an infrastructure of methods used for PBE/AES-based entity field value encryption.

In contrast to an IV-per-column (Mattsson, 2005, p. 9), a random IV is stored per every single value. This will obfuscate identical values, which is appreciated for enumerated field values (e.g. gender[23])

[23]`Proband.gender` is actually not considered as PII and therefore not encrypted.

or identical string values (e.g. salutation). Cipher Block Chaining (CBC) mode of operation will hide patterns within the value itself. The downside of IV-per-value is the impossibility of searching encrypted values the fast, $O(log(n))$ way. It would require the RDBMS to perform a full table scan instead and thereby encrypt the search criterion's comparison value with the corresponding IV per every single row scanned (Agrawal et al., 2004, p. 1). To compensate, a simple option is to populate a separate hash value. This should only be done if really required, since it leaks information about similarity of stored values. If populating hash values is enabled for a particular column, searching for exact matches is possible by predicates with SQL equality comparison operators[24] and the hashed comparison value(s). Other relational operators[25] still cannot work, assuming a *cryptographic* hash function[26] such as Message-Digest Algorithm 5 (MD5)[27] was used to generate hash values. *Range queries* against encrypted fields are therefore not supported by presented scheme at the moment.[28] If a strict separation of departments is desired when searching for exact matches[29], hash values can be additionally encrypted with the department key (IV-per-column).

Since range queries of probands' medical parameters are a definitive requirement and implemented by the system's query editor, the corresponding entity columns are *not encrypted*.

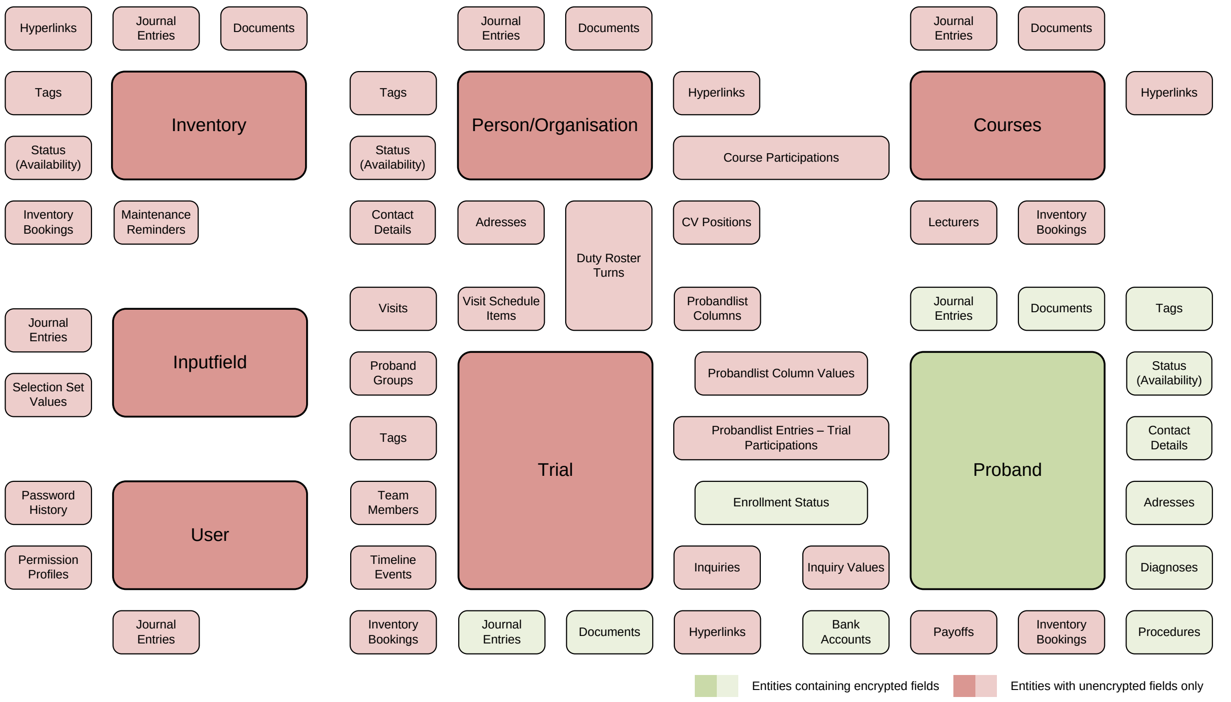


Figure 4.34.: This simplified data model gives an overview of entities with encrypted fields. Only secret fields that store data allowing to personally identify a proband require protection and are encrypted since it is desired to have queryable fields in general.

[24] `=`, `IN`

[25] `<`, `<=`, `>`, `>=`, `BETWEEN`, `LIKE`

[26] By definition, cryptographic hash functions do not preserve the order of preimages.

[27] When hashing column values in order to mask plaintext information, preimage resistance is of interest only. Although they are considered to be broken regarding collision resistance, outdated hash functions such as MD5 remain preimage-resistant (Sasaki and Aoki, 2009, p. 2).

[28] Security tradeoffs are inherent to advanced techniques such as the Order-Preserving Encryption (OPE) scheme (Boldyreva et al., 2009).

[29] With plain hash values, a user y from department Y would be able to search for a proband x of department X. In this case, knowledge about the existence of x is leaked to y at least, for example when y searches for given first and last names.

UI screens emphasize decrypted elements and inputs of encrypted data using icons:

- 🔒✓ A form input that is backed by an encrypted field (table 4.6).
- 🔓✗ A form input that is backed by a plaintext field only, where user might be tempted to enter a probands' PII. This applies to text inputs for unencrypted information where the system cannot prevent users from explicitly referring the proband e.g. by name.
- 🔓 In front of labels of datatable cells or tree nodes, it indicates a successfully decrypted field value.
- 🔒 In front of labels (datatable cells, tree nodes, ...), it indicates the *failed* field value decryption. In this case the label itself is set to string constants like "encrypted proband".
- 🔍 This icon applies to datatable column headers that provide text inputs for filtering. It should make users aware of the limitation when filtering by encrypted field values - in particular search pattern strings cannot work.

entity	**field**	**plaintext**
`Proband`	`prefixedTitle1-3`	✗
	`firstName`	✗
	`lastName`	✗
	`postpositionedTitle1-3`	✗
	`dateOfBirth`	✗
	`citizenship`	✗
`ProbandImage`	`fileName`	✗
	`data`	✗
`ProbandTagValue`	`value`	✗
`ProbandContactDetailValue`	`value`	✗
	`comment`	✗
`ProbandAddress`	`countryName`	✗
	`cityName`	✗
	`zipCode`	✗
	`streetName`	✗
	`houseNumber`	✗
	`entrance`	✗
	`doorNumber`	✗
	`careOf`	✗
`ProbandStatusEntry`	`comment`	✗
`Diagnosis`	`comment`	✗
`Procedure`	`comment`	✗
`BankAccount`	`accountHolderName`	✗
	`accountNumber`	✗
	`bankCodeNumber`	✗
	`bankName`	✗
	`iban`	✗
	`bic`	✗
`ProbandListStatusEntry`	`reason`	✗
`File`	`title`	✓
	`comment`	✓
	`fileName`	✓
	`data`	✓
`JournalEntry`	`title`	✓
	`comment`	✓

Table 4.6.: This table lists entity fields in detail, which store proband-related data that definitely represents or eventually contains PII. Fields for both encrypted and plaintext values are provided for entities relevant to universal features of system modules (section 4.1.3.3) only. For example, encrypted `JournalEntry` fields are used for records of a proband, while plaintext fields are used in the case of an inventory.

When looking at the data stored in tables overall, any field containing PII is encrypted (figure 4.34, table 4.6). In order to support subject eligibility criteria queries including range criteria, `Proband` records referring to encrypted myacpii are linked with records containing (sensitive) medical information in plain text (inquiry values and proband list column values, section 4.1.1.3). This scheme results in the *de-identification* of (medical) *data at rest*, which is reverted by the application by decrypting PII whenever it is displayed to the user. Although the identity of a proband is safe by default, the danger of identifying individuals from unencrypted (medical) records arises. It depends how much information is captured with inquiry forms in the course of EDC, which in turn is up to the users. The risk of deriving a subject's identity depends on a data set's *k-anonymity* (Hauf, 2007). When medical information is exported, subject identities are supposed to be protected by keeping k high (e.g. Simonic and Gell, 2001, p. 10). For instance, to claim a plain table of records is 5-anonymous, filtering it by relevant columns[30] must give 5 matching results at least for any filter values (listing 4.10). The definition considers a single result row as a successful identification of an idividual.

```sql
select min(subquery.group_count) as k_anonymity from
       (select count(*) as group_count from patients t group by t.field_1, t.field_2, ...) as subquery
```

Listing 4.10: A simple SQL statement can be used to check the k-anonymity of a database table's content before exporting it to third parties.

Aside data exports, k-anonymity can be considered in the context of an *inference attack* scenario. A stolen database dump contains the entire database schema, including tables with medical data of subjects. The set of all persons in Austria that e.g. suffer from diabetes is assumed as basic set, which is expected to be available to the thief. This basic set includes PII such as person names as well as any associated medical parameter values one can think of - especially those which are likely to be captured with input forms in the Phoenix CTMS. When users design an inquiry form for proband recruitment, they should keep in mind that the medical data captured holds a lower limit of e.g. $k = 5$ related to the thief's basic set. When the thief extracts a parameter tuple from the dump and queries the basic set, there are always 5 or more individual persons listed for any tuple tried. It cannot be determined for sure, which person names where actually stored in the encrypted table columns of the database dump.

```java
public class ProbandDaoImpl extends ProbandDaoBase {

    ...

    @Override
    public void probandInVOToEntity(ProbandInVO source, Proband target) {
        super.probandInVOToEntity(source, target);
        ...
        try {
            ...
            //AES encryption using the context user's department key:
            cipherText cipherText = CryptoUtil.encrypt(source.getLastName());
            target.setLastNameIv(cipherText.getIv());
            target.setEncryptedLastName(cipherText.getCipherText());
            //if the field really should be searchable:
            //target.setLastNameHash(CryptoUtil.hashForSearch(source.getLastName()));
            ...
        } catch (Exception e) { //errors with encryption are not expected and propagated:
            throw new RuntimeException(e);
        }
    }

    ...
```

Listing 4.11: Encryption of field values is performed in InVO-to-entity transformation methods.

[30]A column is relevant, if its information could eventually be found in an external database. This excludes application specific data like a record ID but includes virtually anything else.

```java
public class ProbandDaoImpl extends ProbandDaoBase {

    ...

    @Override
    public void toProbandOutVO(Proband source, ProbandOutVO target) {
        super.toProbandOutVO(source, target);
        ...
        try {
                ...
            //AES decryption using the context user's department key:
            target.setLastName((String) CryptoUtil.decrypt(source.getLastNameIv(), source.getEncryptedLastName()));
            ...

            target.setDecrypted(true); //all decryption operation were successful
        } catch (Exception e) { //errors with decryption like javax.crypto.BadPaddingException are anticipated:
            //reset all decrypted VO fields:
            ...
            target.setLastName(null);
            ...

            target.setDecrypted(false); //at least one decryption operation failed
        }
    }

    ...
```

Listing 4.12: Decryption operations in entity-to-OutVO transformations.

Field value encryption (listing 4.11) and decryption (listing 4.12) is performed within DAO transformation methods. While errors with encryption (during create and update operations) are unexpected, decryption errors (during retrieval operations) are an anticipated case. For example, if a user lists all registered probands using the query editor, they will belong to various departments and thus each proband's name is encrypted with the corresponding department key. As desired, the UI reveals successfully decrypted names of probands of the executing user's department only, while remaining entries should be displayed as "encrypted proband". This is realized by a `decrypted` flag that is common for all VOs containing decrypted data. The detection of unsuccessful decryption operations due to a wrong key behind is a frequent topic with ciphers. The implementation shown relies on raised `BadPaddingExcpetions` that come from the Public-Key Cryptography Standard (PKCS) #5 padding scheme (RSA Laboratories, 1993). Assuming that an incorrect key results in random bytes for the final AES block, a false positive probability p_{fp} due to the akward case of a valid PKCS5 padding block structure can be determined with a decision tree:

$$p_{fp} = \sum_{i=1}^{16} \frac{1}{256^i} = 0.392\%$$

While this corresponds to a single decryption, the probability drops to an even lower value of p_{fp}^m for VOs containing m decrypted fields. Thus it is unlikely a user will ever see UI labels showing random characters instead of the "encrypted proband" label.[31] For an absolute distinction of successful and unsuccessful decryption operations, a comparison with a checksum is required, which is implemented for storing binary file data (`VerifyMD5Stream`, section 4.7).

The primary goal of the presented encryption scheme is to prohibit unauthorized readout of identity information of probands, which is peristed along with sensitive medical data. The encryption method described applies to entity fields of primitive data types, whose values are saved in RDBMS tablespace files on non-volatile storage (data at rest). To store binary data like document file contents, entity fields with `byte[]` data type can be used. This way, file content encryption can be supported, which is required for document files containing probands' PII. The default setting is to store files in the file system instead of the database. Decryption and encryption of files is implemented using `javax.crypto.Cipher` objects created by factory methods `getDecryptionCipher` and `getEncryptionCipher` from listing 4.9, which are also used for field

[31] A value of $m = 10$ for `ProbandOutVO` results in $p_{fp}^{10} = 8.602 \cdot 10^{-23}\%$

value encryption. Since streams are used to read and write files, `javax.crypto.CipherInputStream` and `javax.crypto.CipherOutputStream` have to be used to utilize Java's `javax.crypto.Cipher` block cipher abstraction. Therefore, the same `AES/CBC/PKCS5Padding` mode used for column encryption also applies to file encryption after all.

4.2.4. Finder Methods

DAO finder methods are the backbone of retrieval operations. Basically, they emit SQL `SELECT` queries to list records to be displayed in the UI finally. This section presents the full pathway of record list retrieval, from the UI layer down to a DAO finder method.

Modern applications are expected to come with a set of pervasive features when presenting record lists:

Pagination: Pagination allows the user to switch between pages with a given number of rows to avoid scrolling long lists. Lazy-loading pagination is a crucial mechanism for application performance when presenting potentially large result sets. For the performance and memory consumption benefit, lazy-loading pagination is expected to utilize RDBMS support by extending a query statement with the SQL `LIMIT` clause. This way, only a range of records of the query's overall result set is loaded, which represent the page items displayed.

Sorting: Users expect the ability to sort by values of a column of interest for orientation. Again, RDBMS sorting using the SQL `ORDER BY` clause is faster than sorting performed by the Java application code. In combination with pagination, RDBMS sorting is mandatory, since only items of the retrieved page would be sorted otherwise, which is typically not what users want.

Filtering: A related feature known from spreadsheet applications is the ability to filter by a column's value in order to display only matching rows. Additional filter criterion terms are appended to a query using the SQL `AND` operator to narrow the result set.

Most UI screens contain *datatable* components for viewing lists of records and are therefore subject to Pagination, Sorting and Filtering (PSF). To master lazy-loading pagination for app. 100 distinct datatable views along with sorting and filtering capabilities for hundreds of columns, a structured approach is required. A dedicated method parameter per field to be sorted or filtered by would result in long parameter lists in finder and service method declarations. Modifications to parameter lists in the UML model are cumbersome to sync across hand-crafted artefacts of the core tier as well as data models and facelets in the web tier. Since PSF can be considered as presentation aspect, it is a reasonable design objective to place definitions for sorting and filtering in facelets and align them with datatable column definitions.

PrimeFaces's `p:datatable` component provides UI control elements for AJAX-based PSF out-of-the-box:

- A paginator bar groups controls for adjusting the page size and performing transitions between record pages.
- Sort order controls are displayed in the column header. They allow exclusive sorting by the selected column's values and toggling the sort order.
- Column headers show input elements for entering filter values. Filtering by values of multiple columns at once is supported.

To actually employ RDBMS support for a constant UI performance independent of the size of the table data, the datatables are configured to *lazy mode*. In this case, a corresponding data model will hold the actual page's rows only. Each PSF interaction like switching to another page will need another service method request that invokes the responsible finder method to return the new page's rows. When triggering a PSF interaction, the user updates PSF parameters, which are propagated across application layers until they are passed to the backing finder method. Since the majority of finder methods are required to support PSF, the `PSFVO` DTO is introduced to encapsulate PSF information (figure 4.35). By convention, the last argument of these finder methods is of type `PSFVO`.

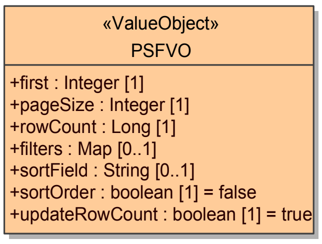
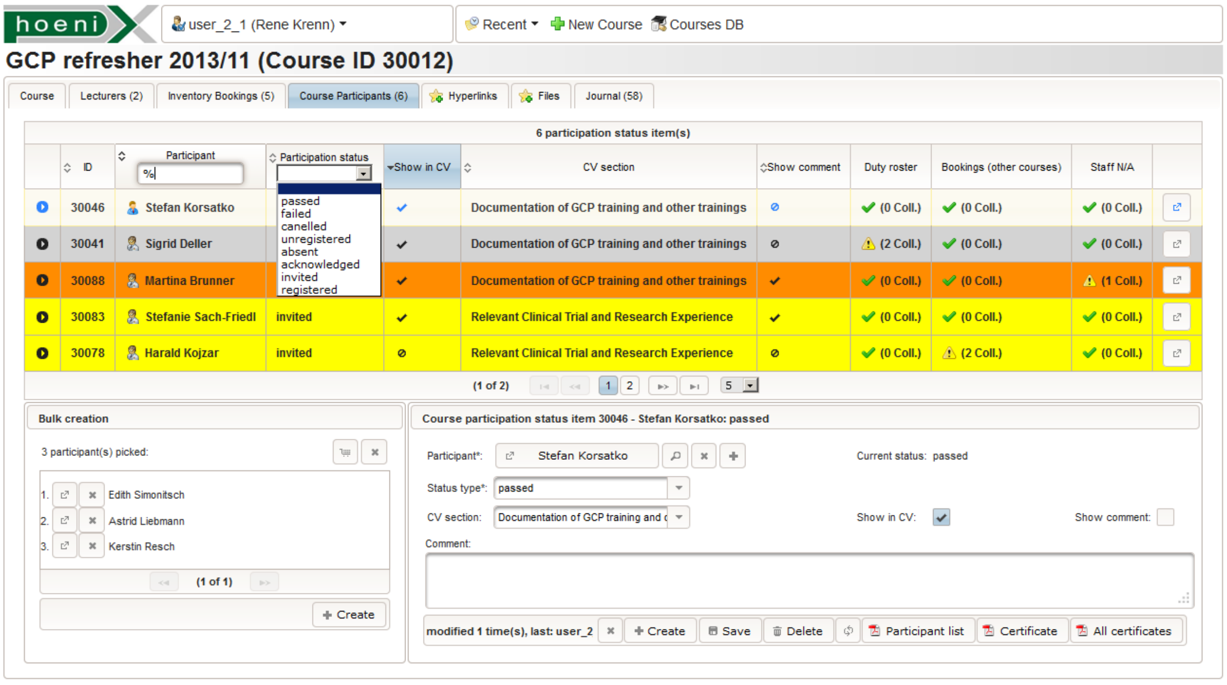


Figure 4.35.: The `PSFVO` encapsulates parameters to derive information for both queries in finder methods and datatable controls: current page, pagesize and number of pages (`first`, `pageSize`, `rowCount`), column to sort and ascending/descending sort mode (`sortField`, `sortOrder`), filter criteria consisting of column and filter value each (`filters`).

The example of the list view for managing a course's participants (figure 4.36) is used to examine the program flow of PSF interactions more in detail.



GCP refresher 2013/11 (Course ID 30012)

6 participation status item(s)

	ID	Participant %|	Participation status	Show in CV	CV section	Show comment	Duty roster	Bookings (other courses)	Staff N/A	
	30046	Stefan Korsatko		✓	Documentation of GCP training and other trainings	⊘	✓ (0 Coll.)	✓ (0 Coll.)	✓ (0 Coll.)	
	30041	Sigrid Deller		✓	Documentation of GCP training and other trainings	⊘	⚠ (2 Coll.)	✓ (0 Coll.)	✓ (0 Coll.)	
	30088	Martina Brunner		✓	Documentation of GCP training and other trainings	✓	✓ (0 Coll.)	✓ (0 Coll.)	⚠ (1 Coll.)	
	30083	Stefanie Sach-Friedl	invited	✓	Relevant Clinical Trial and Research Experience	✓	✓ (0 Coll.)	✓ (0 Coll.)	✓ (0 Coll.)	
	30078	Harald Kojzar	invited	⊘	Relevant Clinical Trial and Research Experience	⊘	✓ (0 Coll.)	⚠ (2 Coll.)	✓ (0 Coll.)	

(1 of 2) 1 2 5

Bulk creation

3 participant(s) picked:

1. Edith Simonitsch
2. Astrid Liebmann
3. Kerstin Resch

(1 of 1)

Create

Course participation status item 30046 - Stefan Korsatko: passed

Participant*: Stefan Korsatko

Current status: passed

Status type*: passed

CV section: Documentation of GCP training and c

Show in CV: ✓

Show comment: ☐

Comment:

modified 1 time(s), last: user_2 | Create | Save | Delete | Participant list | Certificate | All certificates

Figure 4.36.: This sample UI shows the course participants tab of an opened course. It supports sorting by relevant `CourseParticipationStatusEntry` entity fields, beginning with `CourseParticipationStatusEntry.id`[32]. Filter values can be entered to filter by the participant's last name (text input) and participation status (dropdown options). Pagination is controlled by the paginator at the bottom of the datatable.

The facelet fragment in listing 4.13 places and configures the PrimeFaces datatable within the course participants tab UI screen. Sortable columns are specified by defining the `sortBy` attribute of a `p:column` component. Likewise, filterable columns are specified by the `filterBy` attribute. Filter values are basically entered in a text input, but a dropdown is supported as an alternative for selecting filter values from a list of option items. In the latter case, the `filterOptions` attribute binds a managed bean field providing option items. Values for both the `sortBy` and `filterBy` attributes are supposed to denote *association*

[32]Ascending sorting by record IDs gives the order of record creation.

paths, resolvable for the VO type of the datatable row iteration variable (`cpse`). In non-lazy mode, PSF processing is entirely performed by the PrimeFaces datatable implementation. It evaluates the VO paths while iterating all VO instances in order to sort and/or filter rows. The design goal is to perpetuate declaring column sort and filter association paths of datatables configured to lazy mode. As shown in listing 4.13, `sortBy` and `filterBy` paths therefore specify association paths of the domain entity, which may differ from paths for VOs, since entity and VO graphs differ in general. This implies a limitation of datatables in lazy mode: fields available in VOs only cannot be sorted or filtered by, such as `StaffOutVO.age`[33] or calculated appointment collision counts in figure 4.36.

```
...
<p:dataTable lazy="true" <!-- datatable with lazy mode enabled -->
    <!-- paginator bar settings: -->
    paginatorTemplate="{CurrentPageReport}  {FirstPageLink} {PreviousPageLink} {PageLinks} {NextPageLink} {LastPageLink} {
        RowsPerPageDropdown}"
    rowsPerPageTemplate="5,10,20,50"
    paginatorPosition="bottom "
    pageLinks="10"

    var="cpse" <!-- row iteration variable for page records of CourseParticipationStatusEntryOutVO type -->
    value="#{courseParticipationStatusEntryBean.model}" <!-- LazyDataModel for retrieving and buffering page records -->
    selectionMode="single"
    ...>

    <!-- id column: -->
    <p:column sortBy="#{cpse.id}" >
        <f:facet name="header">
            <h:outputText value="#{labels.id_column}" />
        </f:facet>
        <h:outputText value="#{cpse.id}" />
    </p:column>
    <!-- participant column: -->
    <p:column sortBy="#{cpse.staff.personParticulars.lastName}"
        filterBy="#{cpse.staff.personParticulars.lastName}">
        <f:facet name="header">
            <h:outputText value="#{courselabels.[prefix]_last_name_column_key}" />
        </f:facet>
            <ui:include src="../shared/listIcon.xhtml">
            <ui:param name="label" value="#{cpse.staff.name}" />
            <ui:param name="icon" value="#{cpse.staff.category.styleClass}" />
            <ui:param name="rendered" value="true" />
        </ui:include>
    </p:column>
    <!-- participation status column: -->
    <p:column sortBy="#{cpse.status}"
        filterBy="#{cpse.status.id}"
        filterOptions="#{sessionScopeBean.filterCourseParticipationStatusTypes}">
        <f:facet name="header">
            <h:outputText value="#{courselabels.[prefix]_status_column_key}" />
        </f:facet>
        <h:outputText value="#{cpse.status.name}" />
    </p:column>
    ...
</p:dataTable>
...
```

Listing 4.13: Faclet fragment with the definition of the PrimeFaces datatable example from figure 4.36.

Whenever a user triggers a PSF action, the `load` method override of the backing model object is invoked (listing 4.14) on reception of the AJAX request. The method begins by preparing a `PSFVO` instance, which is passed as an argument when invoking the service method. The service method effectively returns the selected page rows together with the total number of records, which is required to update paginator controls. The PPR completes with the AJAX response, which updates the datatable UI to display the requested page of records.

[33]The person's age is derived from the DoB entity field.

```
public class CourseParticipationStatusEntryLazyModel extends
    org.primefaces.model.LazyDataModel.LazyDataModel<CourseParticipationStatusEntryOutVO> { //simplified inheritance hierarchy

    private Long courseId; //the ID of the course root entity currently opened in the course entity view

    //The PrimeFaces datatable component implementeation will trigger the load() method whenever the user performs a PSF
         interaction:
    @Override
    public List<CourseParticipationStatusEntryOutVO> load(
        int first,                      //row index of the displayed page's first element
        int pageSize,                   //number of rows per page
        String sortField,               //entity association path of column to sort by, without leading "cpse" segment,
                                        //  e.g. "status.name"
        SortOrder sortOrder,            //sorting mode: ascending/descending/disabled
        Map<String,String> filters) {   //map of columns with filter values entered:
                                        //  keys:   entity association path of column to filter, without leading "cpse"
                                        //          segment, e.g. "staff.personParticulars.lastName"
                                        //  values: filter value, e.g. "Bru%er"

        PSFVO psf = new PSFVO();
        //prepare a PSFVO DTO instance:
        psf.setFirst(first);
        psf.setPageSize(pageSize);
        psf.setSortField(sortOrder == SortOrder.UNSORTED ? null : sortField);
        psf.setSortOrder(sortOrder == SortOrder.ASCENDING);
        psf.setFilters(filters);
        //service method invocation:
        try {
            //retrieve the specified page from the list of filtered, sorted course participation records:
            List pageRows = (List) WebUtil.getServiceLocator().getCourseService().getCourseParticipationStatusEntryList(
                WebUtil.getSessionScopeBean().getAuthentication(), null, courseId, psf);
            super.setRowCount(psf.getRowCount()); //update the number of total rows
            return pageRows; //return page elements for buffering
        } catch (AuthenticationException e) { //logout, if the authentication fails
            WebUtil.publishException(e);
        } catch (Exception e) { //e.g. no permission

        }
        //empty page:
        super.setRowCount(0);
        return new ArrayList<CourseParticipationStatusEntryOutVO >();
    }

    ...
```

Listing 4.14: The concrete `LazyDataModel` shown is bound to the datatable component and overrides the `load` method to retrieve a page's records for buffering.

Listing 4.15 shows the `getCourseParticipationEntryList` service method, which retrieves a page of `CourseParticipationStatusEntry` entity instances using the `CourseParticipationStatusEntryDao.findByStaffCourseStatus` finder method. The service method transaction includes of parameter checking, finder method invocation and the final VO transformation of the result set page. The `PSFVO.rowCount` field is used as an output parameter, which requires the finder method to perform a separate row count query. Remaining `PSFVO` fields represent input parameters, which are checked for sanity when applied by the finder method.

```
public class CourseServiceImpl extends CourseServiceBase {

    ...

    @Override
    protected Collection<CourseParticipationStatusEntryOutVO> handleGetCourseParticipationStatusEntryList(
        AuthenticationVO auth, Long staffId, Long courseId, PSFVO psf) throws Exception {

        //input validation:
        if (staffId != null) { //check staff id if provided
            ServiceUtil.checkStaffId(staffId, this.getStaffDao());
        }
        if (courseId != null) { //check course id if provided
            ServiceUtil.checkCourseId(courseId, this.getCourseDao());
        }

        CourseParticipationStatusEntryDao dao = this.getCourseParticipationStatusEntryDao();
        //invoke the finder method and pass the PSFVO argument:
        Collection courseParticipations = dao.findByStaffCourseStatus(staffId, courseId, psf);
        //transform and return result set:
        dao.toCourseParticipationStatusEntryOutVOCollection(courseParticipations);
        return courseParticipations;

    }

    ...
```

Listing 4.15: The service method returns actual page records in form of a list of corresponding OutVOs. Each OutVO instance is a POJO graph including related OutVOs. These are useful when displaying details such as background colors and type or category icons of datatable rows.

The finder method implementation (listing 4.16) sets up the actual database query by means of the Hibernate Criteria API. A tree of `org.hibernate.Criteria` objects is used to map joined entity tables and organize criterion terms per joined table in order to generate and emit SQL statements in the end. The logic for processing passed PSF information is encapsulated within the `applyPSFVO` utility method (listing 4.17). `SubCriteriaMap` manages creation and reuse of nested `org.hibernate.Criteria` instances when processing entity association paths of filters. Therefore, `SubCriteriaMap.createCriteria` extends Hibernate's `Session.createCriteria` capabilities, which does not support entity association paths enclosing x-to-many relationships. The work around for this limitation is also implemented for HQL and explained in detail in section 4.9.2.2.

```
public class CourseParticipationStatusEntryDaoImpl extends CourseParticipationStatusEntryDaoBase {

    ...

    @Override
    protected Collection<CourseParticipationStatusEntry> handleFindByStaffCourseStatus(
            Long staffId, Long courseId, PSFVO psf) throws Exception {

        //create a Hibernate Criteria object for the CourseParticipationStatusEntry entity:
        org.hibernate.Criteria courseParticipationStatusEntryCriteria = this.getSession().createCriteria(
            CourseParticipationStatusEntry.class);

        if (staffId != null) { //filter by staff id, if provided
            courseParticipationStatusEntryCriteria.add(Restrictions.eq("staff.id", staffId.longValue()));
        }
        if (courseId != null) { //filter by course id, if provided
            courseParticipationStatusEntryCriteria.add(Restrictions.eq("course.id", courseId.longValue()));
        }

        //prepare a criteria map helper to create and manage criteria objects when resolving PSF filter association paths:
        SubCriteriaMap criteriaMap = new SubCriteriaMap(CourseParticipationStatusEntry.class,
            courseParticipationStatusEntryCriteria);
        //effectively append LIMIT and ORDER BY clauses and filter criterion terms:
        CriteriaUtil.applyPSFVO(criteriaMap, psf, false);

        return courseParticipationStatusEntryCriteria.list(); //execute query

    }

    ...
```

Listing 4.16: The finder method employs a `org.hibernate.Criteria` object to construct and execute the effective database query.

```java
//located in public final class at.zmf.crc.ctsms.query.CriteriaUtil
public static void applyPSFVO(SubCriteriaMap criteriaMap, PSFVO psf, boolean distinct) throws Exception {
    Criteria criteria = criteriaMap.getCriteria();
    //1. Filters:
    Map<String,String> filters = psf.getFilters();
    if (filters != null  filters.size() > 0) { //process filters if any:
        for (Iterator<Map.Entry<String,String>> it = filters.entrySet().iterator(); i.hasNext();) {
            Map.Entry<String,String> filter = (Map.Entry<String,String>) it.next();
            AssociationPath filterFieldAssociationPath = new AssociationPath(filter.getKey());
            //A Hibernate criteria object has to be created to join a association from the filter's
            //association path. The object is to be reused, if the same association is encountered
            //when processing another filter. This is managed by the SubCriteriaMap.createCriteriaForAttribute
            //method. In addition, it populates a map (SubCriteriaMap.getPropertyClassMap) of encountered
            //path's terminal entity field data types:
            Criteria subCriteria = criteriaMap.createCriteriaForAttribute(filterFieldAssociationPath);
            //prepare and append the criterion term:
            applyFilter(subCriteria, filterFieldAssociationPath.getPropertyName(),
                criteriaMap.getPropertyClassMap().get(filterFieldAssociationPath.getFullQualifiedPropertyName()),
                filter.getValue());
        }
    }
    //2. Total number of records:
    if (psf.getUpdateRowCount()) { //true by default
        Long count;
        //perform a separate count query to populate the PSFVO row count field:
        if (distinct) {
            count = (Long) criteria.setProjection(Projections.countDistinct("id")).uniqueResult();
        } else {
            count = (Long) criteria.setProjection(Projections.rowCount()).uniqueResult();
        }
        psf.setRowCount(count);
        criteria.setProjection(null);
        criteria.setResultTransformer(Criteria.ROOT_ENTITY);
    }
    //3. Pagination:
    if (psf.getFirst() != null) {
        criteria.setFirstResult(psf.getFirst());
    }
    if (psf.getPageSize() != null) {
        criteria.setMaxResults(psf.getPageSize());
    }
    //4. Sorting:
    AssociationPath sortFieldAssociationPath = new AssociationPath(psf.getSortField());
    if (sortFieldAssociationPath.isValid()) {
        Criteria subCriteria = criteriaMap.createCriteriaForAttribute(sortFieldAssociationPath);
        subCriteria.addOrder(psf.getSortOrder() ?
            Order.asc(sortFieldAssociationPath.getPropertyName()) :
            Order.desc(sortFieldAssociationPath.getPropertyName()));
    }
    //A DISTINCT modifier can be applied if required, e.g. when a x-to-many association was joined at some point, causing
         duplicate records.
    if (distinct) {
        criteria.setResultTransformer(Criteria.DISTINCT_ROOT_ENTITY);
    }
}
private static void applyFilter(Criteria criteria, String propertyName, Class propertyClass, String value) throws Exception {

    //entity field specified by the filter path is of type ...
    if (propertyClass.equals(String.class)) { //string
        criteria.add(Restrictions.ilike(propertyName, new String(value), MatchMode.ANYWHERE));
    } else if (propertyClass.isArray()  propertyClass.getComponentType().equals(java.lang.Byte.TYPE)) { //byte array
        //this is the support to filter hash columns of encrypted fields, if populated.
        //filtering encrypted columns of string type will work only!
        criteria.add(Restrictions.eq(propertyName, CryptoUtil.hashForSearch(value)));
    } else if (propertyClass.equals(Date.class)) { //date
        criteria.add(Restrictions.eq(propertyName, CommonUtil.parseDate(value,CommonUtil.INPUT_DATE_PATTERN)));
    } else if (propertyClass.equals(Timestamp.class)) { //datetime
        Date date = CommonUtil.parseDate(value,CommonUtil.INPUT_DATE_PATTERN); //filter value contains no time part
        criteria.add(Restrictions.between(propertyName,
            CommonUtil.dateToTimestamp(DateCalc.getStartOfDay(date)), //00:00:00
            CommonUtil.dateToTimestamp(DateCalc.getEndOfDay(date)))); //23:59:59
    } else if (propertyClass.equals(Boolean.class)) { //Boolean
        criteria.add(Restrictions.eq(propertyName, new Boolean(value)));
    } else if (propertyClass.equals(java.lang.Boolean.TYPE)) { //boolean
        criteria.add(Restrictions.eq(propertyName, Boolean.parseBoolean(value)));
    } else if (propertyClass.equals(Long.class)) {
        criteria.add(Restrictions.eq(propertyName, new Long(value)));
    } else if (propertyClass.equals(java.lang.Long.TYPE)) {
        criteria.add(Restrictions.eq(propertyName, Long.parseLong(value)));
    ...
    } else {
        throw new IllegalArgumentException(L10nUtil.getMessage(MessageCodes.CRITERIA_PROPERTY_TYPE_NOT_SUPPORTED,
              DefaultMessages.CRITERIA_PROPERTY_TYPE_NOT_SUPPORTED, new Object[]{propertyClass.toString(),propertyName}));
    }
}
```

Listing 4.17: The `applyPSFVO` method applies the `PSFVO` argument to the provided query.

The `applyPSFVO` method implementation (listing 4.17) benefits from the object-orientated way of handling queries provided by the Criteria API. It processes the `PSFVO` instance step-wise:

1. Each filter is processed to complete the final query criteria:
   Beginning with the processing of filters, each association path and filter value is extracted. A filter is effectively applied by appending a criterion term using a meaningful, hard-coded comparison operator (restriction) for the filter value (comparison value). The `applyFilter` method in listing 4.17 shows the different handling for filter values of string, date, datetime types etc. The data type of the entity field specified by the filter association path is determined by `SubCriteriaMap`. The filter association path is registered with the `SubCriteriaMap`, which analyses the path and creates nested `org.hibernate.Criteria` objects mapping required table joins. In terms of the Criteria API, adding a criterion means appending it to the query statement using the SQL `AND` operator.
2. A row count query is executed to get the total number of records:
   After all filters were processed, the `org.hibernate.Criteria` object is considered complete. At this point, the query can be used to obtain the total number of records, which is required by the UI datatable component to calculate the total number of pages.
3. Pagination parameters are applied:
   The row number offset of the page's first record and page size are specified for the root `org.hibernate.Criteria` object to append a SQL `LIMIT` clause.
4. Sorting parameters are applied:
   PrimeFaces supports sorting by a single column only. Thus, a single `org.hibernate.criterion.Order` element is added to the `org.hibernate.Criteria` object. It specifies the sort order and the table column given by a separate entity association path. As a result, a corresponding `ORDER BY` clause will be appended. Due to a common RDBMS limitation, sorting a *distinct* result set[34] by columns not part of the SQL `SELECT` statement's projection column list is not possible. The association path must therefore not specify an associated entity's field in this case.
5. The updated query is executed to retrieve the result set page:
   After pagination and sorting was applied, the query is run once again to retrieve the finder method's result list finally.

Hibernate provides two abstraction methods (Criteria API, HQL) to implement database queries, which both are used for finder methods. Native SQL queries are possible as well, but entail major drawbacks:

- It is likely to introduce a dependency on a specific RDBMS.
- AndroMDA derives table and column names, which should never affect hand-crafted artefacts. Native SQL statements have to refer table and column names literally, which requires to sync the UML model and SQL statement strings manually after changes.

Hibernate's Criteria API is utilized for "regular" finder methods such as shown before, which are designed and optimized to fit the business logic and specific needs of the web presentation layer. They can be critical to performance or show a complex query structure, as pointed out by examples below:

"my *n* recently modified items": Find the first-*n* root entities of latest related journal system messages produced by a given user (section 4.6.3.1, System Modules Overview).

"(virtual) file sub-directories": Find the folder path names of files merged with preset folder names for a given root entity and file path depth (section 4.7, Document Management).

HQL is used for a particular type of finder method only, capable of executing database queries with dynamic structure as formulated by users. Since set operations are not supported by Hibernate at all, user-defined query expressions require an approach based on HQL statement translation. HQL-based finder methods are supposed to support PSF in the same way as presented for finder methods implemented using Criteria API. Aside general capabilities and restrictions of Hibernate's query abstractions, there can

[34]When joining a x-to-many association, duplicate result records can be filtered by applying the `DISTINCT_ROOT_ENTITY` result set transformer.

be specific reasons to either prefer Criteria API or HQL. Handling objects instead of statement fragment strings might be more convenient in cases. Criteria API organizes table joins itself, but explicit joins as required by unconstrained associations can be expressed in HQL only. A workaround using subqueries (`org.hibernate.criterion.DetachedCriteria`, listing 4.27) can cause a performance penalty.

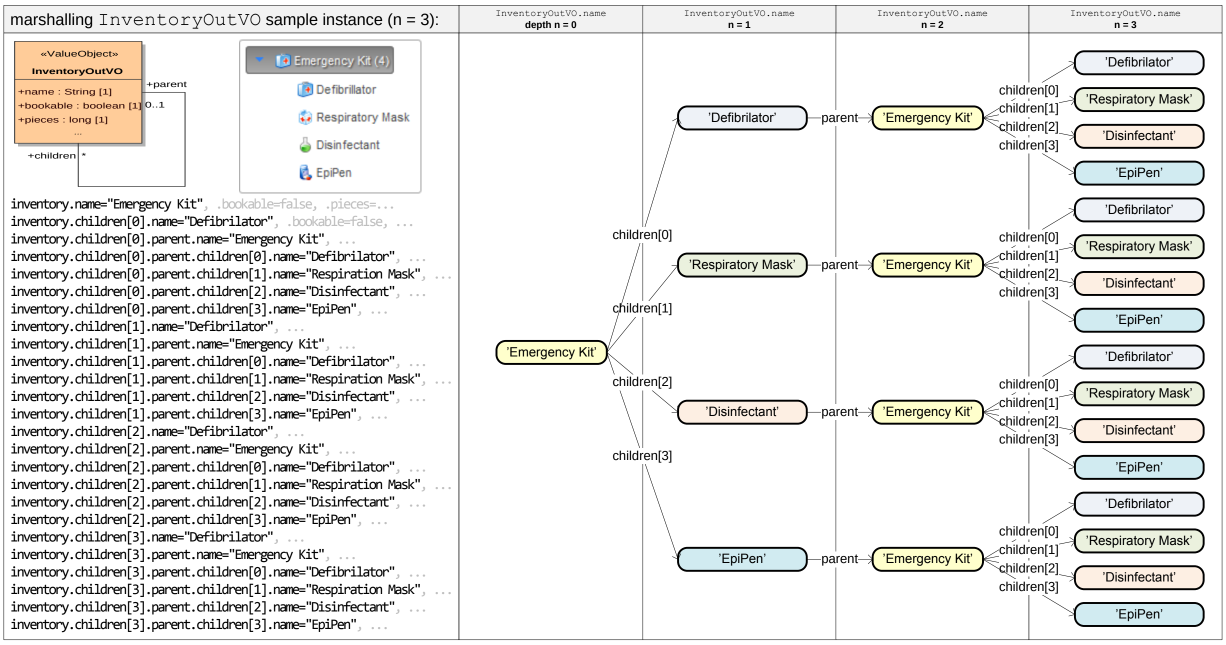
Hibernate's query abstractions turned out as helpful for very basic queries only. Complex queries were actually more difficult to develop using the Criteria API than with HQL which in turn was still more difficult than elaborating the required SQL statement. Native SQL statements are not used however. AndroMDA's Translation Library plug-in can generate HQL finder method implementations which are specified in the UML model by means of OCL, which implies even less flexibility and space for optimizing finder queries after all.

4.3. Object Graph Marshallers

The implementation of features presented in this section requires the traversal of entity or VO instances up to a given depth. This results in the conversion of a POJO graph into an iterable fieldname-value list. A generic approach of this transformation will require an instance graph traversal that employs the listing and invocation of field getter methods via reflection. The getter return type equals the field's type and belongs to one of four basic categories:

Reference: single entity or VO reference
Collection: `java.util.Collection` with elements of reference, collection, map or terminal type
Map: `java.util.Map` with values of reference, collection, map or terminal type
Terminal: remaining types such as Java primitives and wrapper types, arrays, enumeration types, `String`, `Date`, `Timestamp` ...

Single reference fields originate from x-to-one associations. While terminal and reference types are easy to handle, special treatment is required for collections and maps since they are multi-valued. Fields of map type are a rare case, but collection types are frequent since they come from x-to-many associations. For a given depth, the traversal volume will depend upon the actual sizes of map/collection fields for a given instance input. Several multi-valued fields encountered along a processed association path gives the cross product of the collection index or map key sets as shown in figure 4.37.



```
inventory.name="Emergency Kit", .bookable=false, .pieces=...
inventory.children[0].name="Defibrilator", .bookable=false, ...
inventory.children[0].parent.name="Emergency Kit", ...
inventory.children[0].parent.children[0].name="Defibrilator", ...
inventory.children[0].parent.children[1].name="Respiration Mask", ...
inventory.children[0].parent.children[2].name="Disinfectant", ...
inventory.children[0].parent.children[3].name="EpiPen", ...
inventory.children[1].name="Defibrilator", ...
inventory.children[1].parent.name="Emergency Kit", ...
inventory.children[1].parent.children[0].name="Defibrilator", ...
inventory.children[1].parent.children[1].name="Respiration Mask", ...
inventory.children[1].parent.children[2].name="Disinfectant", ...
inventory.children[1].parent.children[3].name="EpiPen", ...
inventory.children[2].name="Defibrilator", ...
inventory.children[2].parent.name="Emergency Kit", ...
inventory.children[2].parent.children[0].name="Defibrilator", ...
inventory.children[2].parent.children[1].name="Respiration Mask", ...
inventory.children[2].parent.children[2].name="Disinfectant", ...
inventory.children[2].parent.children[3].name="EpiPen", ...
inventory.children[3].name="Defibrilator", ...
inventory.children[3].parent.name="Emergency Kit", ...
inventory.children[3].parent.children[0].name="Defibrilator", ...
inventory.children[3].parent.children[1].name="Respiration Mask", ...
inventory.children[3].parent.children[2].name="Disinfectant", ...
inventory.children[3].parent.children[3].name="EpiPen", ...
```

Figure 4.37.: In this example, the `InventoryOutVO` instance "Emergency Kit" represents a containment for four inventory items. Dumping the object graph would never halt due to `parent` and `children` fields from the bidirectional navigatable, reflexive association. Limiting to a depth of $n = 3$ results an output size of 25 `name` field values.[35]

The conversion logic is encapsulated within the abstract `GraphMarshaller` class (figure 4.38). Its static `appendField` method performs the essential depth-first traversal of a given Java class (root) and populates a list of instances of a concrete `GraphMarshaller` implementation class as result. Each `GraphMarshaller` object represents a "compiled" field value *accessor*, capable of determining and managing the chain of collection indexes and map keys for a given instance of the root class. When a `GraphMarshaller` instance is created, base class logic populates a list of `java.lang.reflect.Method` objects. The list represents the chain of getter methods that corresponds to the field's association path.

[35] $\sum_{i=0}^{n} \prod_{j=0}^{i} \begin{pmatrix} |\texttt{parent}|..if\ j\ even \\ |\texttt{children}|..if\ j\ odd \end{pmatrix} = 25$. The size of serialized object graphs containing circular references grows exponentially with the serialization depth n.

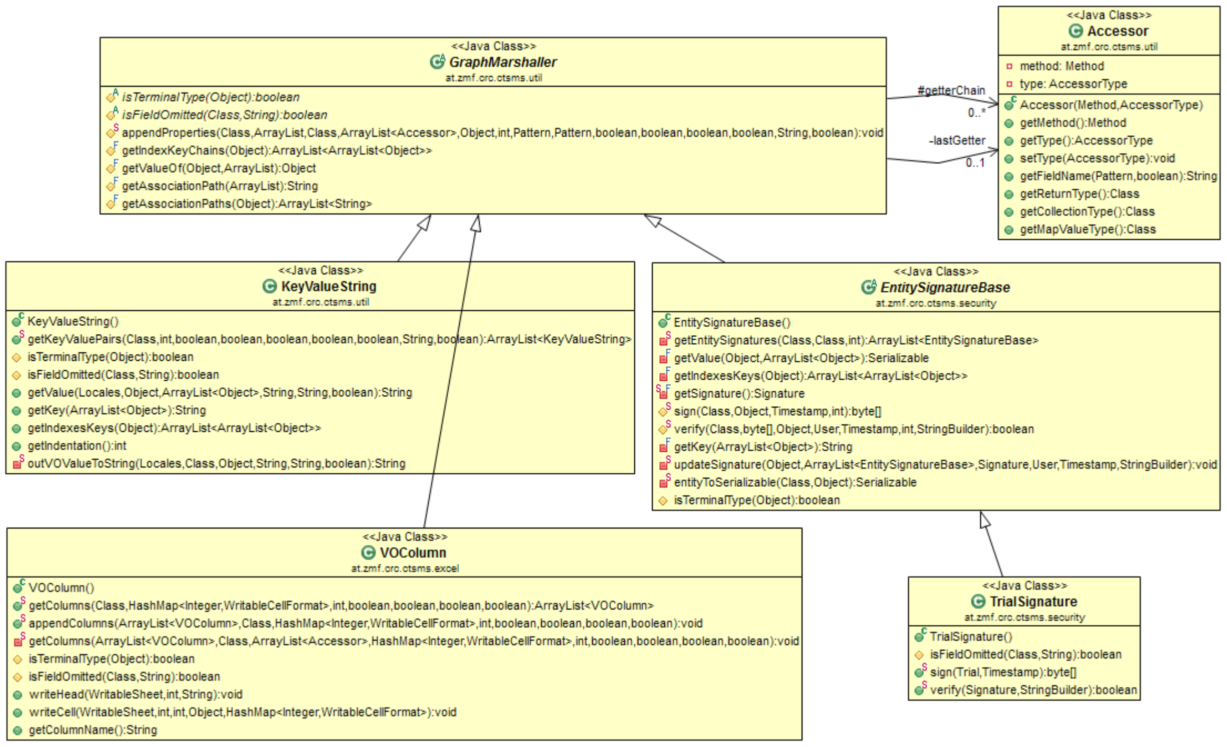

<<Java Class>>
GraphMarshaller
at.zmf.crc.ctsms.util
isTerminalType(Object):boolean
isFieldOmitted(Class,String):boolean
appendProperties(Class,ArrayList,Class,ArrayList<Accessor>,Object,int,Pattern,Pattern,boolean,boolean,boolean,boolean,String,boolean):void
getIndexKeyChains(Object):ArrayList<ArrayList<Object>>
getValueOf(Object,ArrayList):Object
getAssociationPath(ArrayList):String
getAssociationPaths(Object):ArrayList<String>
#getterChain
0..*
-lastGetter
0..1
<<Java Class>>
Accessor
at.zmf.crc.ctsms.util
method: Method
type: AccessorType
Accessor(Method,AccessorType)
getMethod():Method
getType():AccessorType
setType(AccessorType):void
getFieldName(Pattern,boolean):String
getReturnType():Class
getCollectionType():Class
getMapValueType():Class
<<Java Class>>
KeyValueString
at.zmf.crc.ctsms.util
KeyValueString()
getKeyValuePairs(Class,int,boolean,boolean,boolean,boolean,boolean,String,boolean):ArrayList<KeyValueString>
isTerminalType(Object):boolean
isFieldOmitted(Class,String):boolean
getValue(Locales,Object,ArrayList<Object>,String,String,boolean):String
getKey(ArrayList<Object>):String
getIndexesKeys(Object):ArrayList<ArrayList<Object>>
getIndentation():int
outVOValueToString(Locales,Class,Object,String,String,boolean):String
<<Java Class>>
EntitySignatureBase
at.zmf.crc.ctsms.security
EntitySignatureBase()
getEntitySignatures(Class,Class,int):ArrayList<EntitySignatureBase>
getValue(Object,ArrayList<Object>):Serializable
getIndexesKeys(Object):ArrayList<ArrayList<Object>>
getSignature():Signature
sign(Class,Object,Timestamp,int):byte[]
verify(Class,byte[],Object,User,Timestamp,int,StringBuilder):boolean
getKey(ArrayList<Object>):String
updateSignature(Object,ArrayList<EntitySignatureBase>,Signature,User,Timestamp,StringBuilder):void
entityToSerializable(Class,Object):Serializable
isTerminalType(Object):boolean
<<Java Class>>
VOColumn
at.zmf.crc.ctsms.excel
VOColumn()
getColumns(Class,HashMap<Integer,WritableCellFormat>,int,boolean,boolean,boolean,boolean):ArrayList<VOColumn>
appendColumns(ArrayList<VOColumn>,Class,HashMap<Integer,WritableCellFormat>,int,boolean,boolean,boolean,boolean):void
getColumns(ArrayList<VOColumn>,Class,ArrayList<Accessor>,HashMap<Integer,WritableCellFormat>,int,boolean,boolean,boolean,boolean):void
isTerminalType(Object):boolean
isFieldOmitted(Class,String):boolean
writeHead(WritableSheet,int,String):void
writeCell(WritableSheet,int,int,Object,HashMap<Integer,WritableCellFormat>):void
getColumnName():String
<<Java Class>>
TrialSignature
at.zmf.crc.ctsms.security
TrialSignature()
isFieldOmitted(Class,String):boolean
sign(Trial,Timestamp):byte[]
verify(Signature,StringBuilder):boolean

Figure 4.38.: The abstract `GraphMarshaller` class and implementation classes.

To actually use graph marshallers, the "precompiled" list of graph marshaller instances is iterated for a given entity or VO instance (listing 4.18). Specific processing logic that is contained in concrete implementations of `GraphMarshaller` can finally create output (e.g. a stringified field value) or perform operations (e.g. update some state like a checksum). Furthermore, implementations define, which classes to *shortcut* (`isTerminalType`) or skip entirely (`isFieldOmitted`) during traversal. Shortcut means using a representative field like a natural key (e.g. `UserOutVO.name`) instead of processing the whole trailing object (e.g. `UserOutVO`).

```
//retrieve some inventory records to output as text:
Collection inventoryList = inventoryDao.loadAll();
inventoryDao.toInventoryOutVOCollection(inventoryList);

//compile list of marshallers for the InventoryOutVO graph:
ArrayList<KeyValueString> fields = KeyValueString.getKeyValuePairs(InventoryOutVO.class, ...); //depth = 3, field key format
      options, ...

StringBuilder output = new StringBuilder();
//iterate the inventory list:
for (Iterator i = inventoryList.iterator(); i.hasNext();) {
    InventoryOutVO inventory = (InventoryOutVO) i.next();
    output.append("######################################################################"); //separator line
    //iterate marshallers to append "field=value" lines to the output:
    for (Iterator f = fields.iterator(); f.hasNext();) {
        KeyValueString field = (KeyValueString) f.next();
        //for association paths containing multi-valued fields, iterate over the cartesian product of their indexes and map
              keys:
        for (Iterator ik = field.getIndexesAndKeys(reminder); ik.hasNext();) {
            ArrayList indexesAndKeys = (ArrayList) ik.next();
            //add a output line:
            output.append(field.getKey(indexesAndKeys));
            output.append("=");
            output.append(field.getValue(indexesAndKeys, ...)); //i18n options
            output.append("\n");
        }
    }
}
```

Listing 4.18: This basic example shows how a `GraphMarshaller` list can effectively be used for a comprehensive conversion of OutVO objects into a string. It would produce the output shown in figure 4.37.

`KeyValueString` The special processing of the `KeyValueString` graph marshaller lies in string serialization and l10n of OutVO field values. This class is used to dump records, required to generate system messages of audit trails as described in section 4.3.1.

`VOColumn` The processing logic of the `VOColumn` graph marshaller is related to that of `KeyValueString`, but focuses on writing formatted Excel spreadsheet cells directly using jExcelApi (`writeCell`). It disables the unfolding of collections and maps entirely in order to maintain a fixed number of columns over a given list of OutVO instances (spreadsheet rows). In the case of including a multi-valued field as single spreadsheet column, its values have to be aggregated into a single cell using the shortcut technique.

`EntitySignatureBase` `EntitySignatureBase`'s processing will update a checksum state instead of creating serialized output. This is required for the calculation of a digital signature value (section 4.3.3) that includes relevant field values of an entity. In contrast to `KeyValueString` and `VOColumn`, `EntitySignatureBase` processes entities instead of VOs and is therefore independent of i18n parameters.

4.3.1. Audit Trail

Obligated regulations like EU Annex 11 or the comparable FDA Part 11 claim audit trail functionality for computerized systems used for clinical trials. *"It is important to keep track of all changes made to information in the electronic records that document activities related to the conduct of the trial (audit trails)."* (*Guidance for Industry - Computerized Systems Used in Clinical Investigations* 2007)

The audit trail should be implemented as a read-only, system generated, human readable log of *all* CRUD operations applied to a *linked record*. Each log entry should contain the following information (Tedstone, 2010):

- reference to the linked record
- original data
- changed data
- timestamp
- originator
- reason for change

In spite of this more precise understanding, still questions about details arise when planning an concrete software implementation:

Relevant data: Entities relevant for audit trails should be identified. While this might be easy in the case of CDMSs covering CRF data only, it could make sense to differentiate in the case of the Phoenix CTMS in order to avoid logs getting polluted with unimportant information.

Confidentiality: Confidentiality claimed by data privacy regulations must be considered. In example, an audit trail must not contain probands' PII, since the audit trail can be exported and handed out to a third party.

Durable storage: The claim of durable storage of the audit trail faces the claim of a unrecoverable deletion of subject-related records, e.g. in the case a registered proband revokes his privacy data agreement.

Reason for change: Forcing users to enter a reason for every single edit operation will not be practicable.

The implemented journal feature can satisfy the identified formal requirements of audit trails. The system supports automated logging for all records it handles - root entities and their detail records.

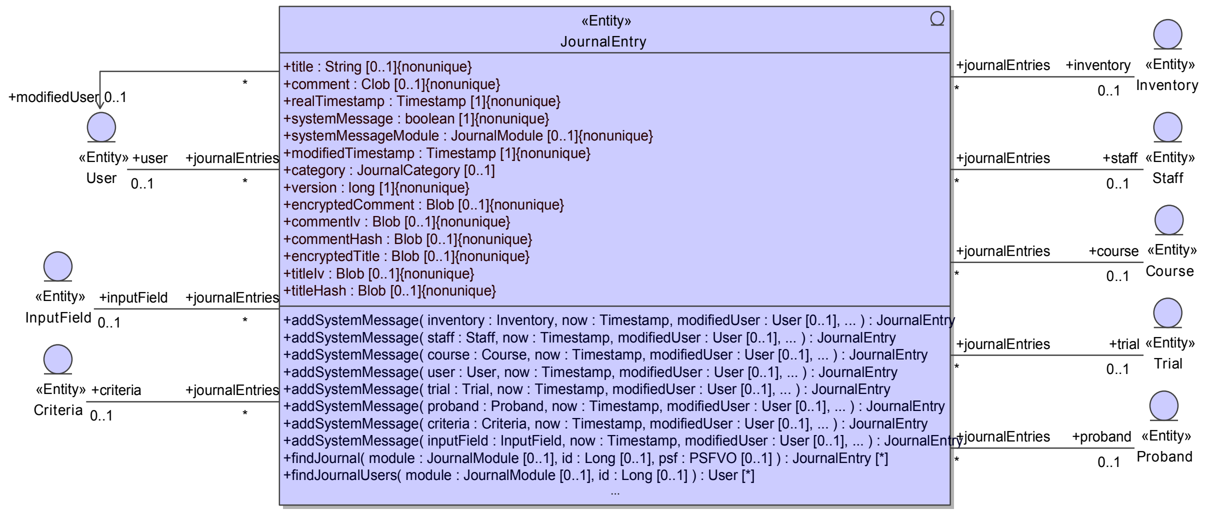
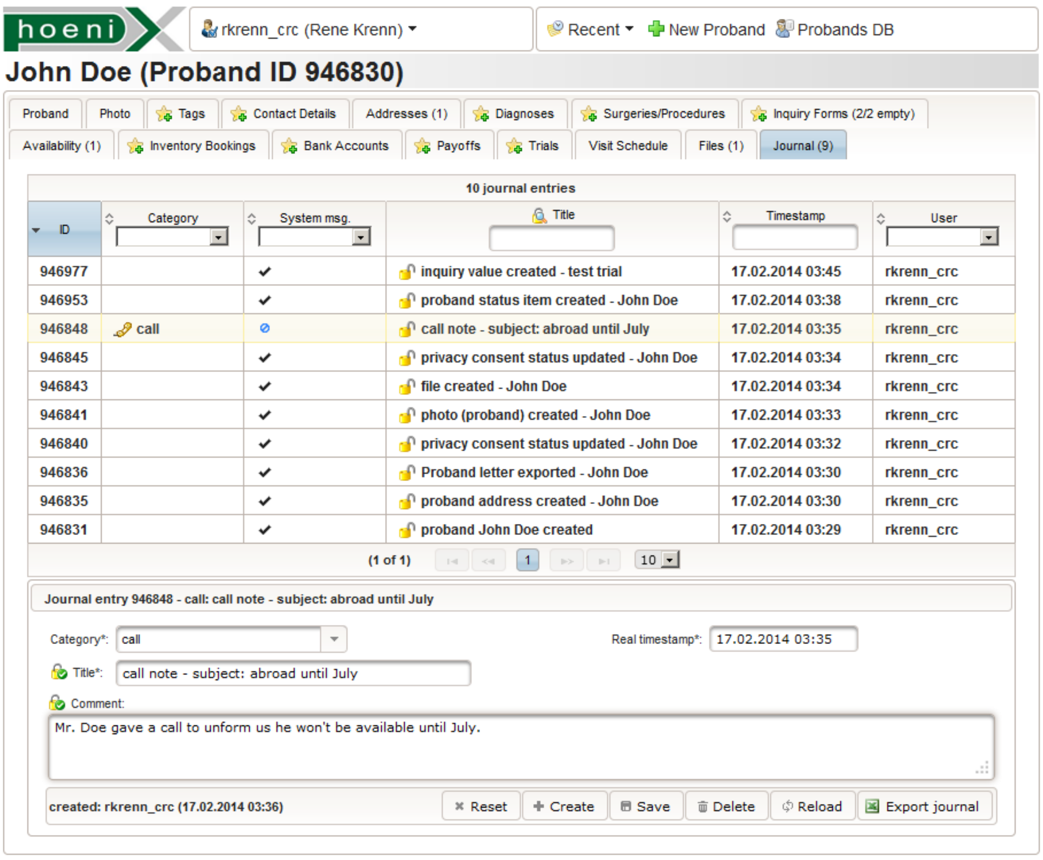

«Entity»
JournalEntry
+title : String [0..1]{nonunique}
+comment : Clob [0..1]{nonunique}
+realTimestamp : Timestamp [1]{nonunique}
+systemMessage : boolean [1]{nonunique}
+systemMessageModule : JournalModule [0..1]{nonunique}
+modifiedTimestamp : Timestamp [1]{nonunique}
+category : JournalCategory [0..1]
+version : long [1]{nonunique}
+encryptedComment : Blob [0..1]{nonunique}
+commentIv : Blob [0..1]{nonunique}
+commentHash : Blob [0..1]{nonunique}
+encryptedTitle : Blob [0..1]{nonunique}
+titleIv : Blob [0..1]{nonunique}
+titleHash : Blob [0..1]{nonunique}
+addSystemMessage(inventory : Inventory, now : Timestamp, modifiedUser : User [0..1], ...) : JournalEntry
+addSystemMessage(staff : Staff, now : Timestamp, modifiedUser : User [0..1], ...) : JournalEntry
+addSystemMessage(course : Course, now : Timestamp, modifiedUser : User [0..1], ...) : JournalEntry
+addSystemMessage(user : User, now : Timestamp, modifiedUser : User [0..1], ...) : JournalEntry
+addSystemMessage(trial : Trial, now : Timestamp, modifiedUser : User [0..1], ...) : JournalEntry
+addSystemMessage(proband : Proband, now : Timestamp, modifiedUser : User [0..1], ...) : JournalEntry
+addSystemMessage(criteria : Criteria, now : Timestamp, modifiedUser : User [0..1], ...) : JournalEntry
+addSystemMessage(inputField : InputField, now : Timestamp, modifiedUser : User [0..1], ...) : JournalEntry
+findJournal(module : JournalModule [0..1], id : Long [0..1], psf : PSFVO [0..1]) : JournalEntry [*]
+findJournalUsers(module : JournalModule [0..1], id : Long [0..1]) : User [*]
...
+modifiedUser 0..1
*
«Entity» User
+user 0..1
+journalEntries *
«Entity» InputField
+inputField 0..1
+journalEntries *
«Entity» Criteria
+criteria 0..1
+journalEntries *
+journalEntries * +inventory 0..1 «Entity» Inventory
+journalEntries * +staff 0..1 «Entity» Staff
+journalEntries * +course 0..1 «Entity» Course
+journalEntries * +trial 0..1 «Entity» Trial
+journalEntries * +proband 0..1 «Entity» Proband

Figure 4.39.: The `comment` field of the `JournalEntry` holds the linked record's snapshot content in text format. Since this content can contain PII, additional columns are provided for encryption.

A *journal* is provided for each root entity (the linked record, figure 4.39). It accumulates journal entries that are entered manually by users (e.g. an interaction note) as well as *system messages* (figure 4.40). System messages are named messages that are generated for CRUD manipulations of the root entity itself or its detail records.



John Doe (Proband ID 946830)

10 journal entries

ID	Category	System msg.	Title	Timestamp	User
946977		✓	inquiry value created - test trial	17.02.2014 03:45	rkrenn_crc
946953		✓	proband status item created - John Doe	17.02.2014 03:38	rkrenn_crc
946848	call	⊘	call note - subject: abroad until July	17.02.2014 03:35	rkrenn_crc
946845		✓	privacy consent status updated - John Doe	17.02.2014 03:34	rkrenn_crc
946843		✓	file created - John Doe	17.02.2014 03:34	rkrenn_crc
946841		✓	photo (proband) created - John Doe	17.02.2014 03:33	rkrenn_crc
946840		✓	privacy consent status updated - John Doe	17.02.2014 03:32	rkrenn_crc
946836		✓	Proband letter exported - John Doe	17.02.2014 03:30	rkrenn_crc
946835		✓	proband address created - John Doe	17.02.2014 03:30	rkrenn_crc
946831		✓	proband John Doe created	17.02.2014 03:29	rkrenn_crc

Journal entry 946848 - call: call note - subject: abroad until July

Category*: call
Real timestamp*: 17.02.2014 03:35
Title*: call note - subject: abroad until July
Comment:
Mr. Doe gave a call to unform us he won't be available until July.

created: rkrenn_crc (17.02.2014 03:36)

Figure 4.40.: Sample UI for managing the journal of the imaginary proband "John Doe". The screen shows the proband's history of journal entries, comprising generated system messages and a manual entry. When entering a manual journal entry like a call note, the category can be selected from a predefined set. Configurable preset strings for the title (and comment) can be applied when switching the category.

This design allows logging of detail *record deletions*. The deletion of a root entity itself is logged in the executing user's journal. Aside invasive operations, the following events are recorded:

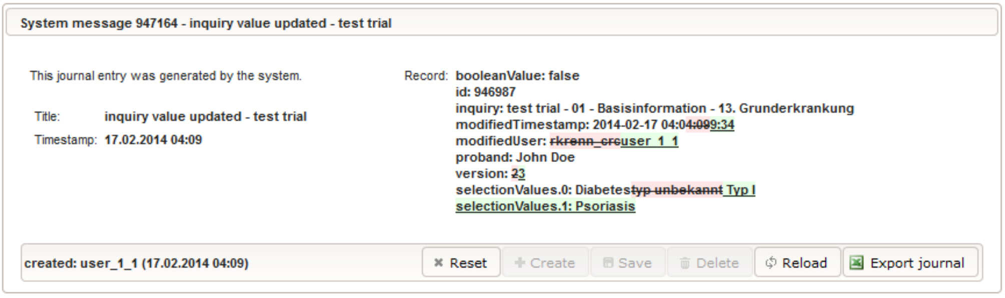
- user logon attempts
- data exports (Excel, PDF)
- execution of dynamic database queries

Exports and query executions may expose PII of a considerable number of probands and are logged in order to track data leaving the system.[36] Journal entries of entities potentially containing PII are encrypted.

From the implementation point of view, a system message `JournalEntry`'s most remarkable fields are `title` and `comment`. For system generated journal entries, the comment is populated with the record dump of an entity including data manipulations. The dump is a textual representation reflecting the entity's state when persisted. A first implementation attempt was utilizing XStream (Walnes, Schaible, and Committers, 2013) to dump entire Java object graphs in XML format. The serialization capabilities of XStream are complete and can handle circular references, but actually produce dumps which are too bloated and not really human readable. This was the original motivation to build the `GraphMarshaller` after all. System message comments are created using the `KeyValueString` graph marshaller configured

[36] A proband's PII must not leave the institution (CRC) that is explicitly accredited by the proband's signed privacy consent.

with a reasonable graph depth in the way outlined in listing 4.18. The resulting record dump shows a line-by-line transcript of the VO instance graph's fieldname-value pairs. Both i18n of field names and filtering of fields enumerated by the graph marshaller are possible.[37] In order to detect changes of update operations for highlighting (figure 4.41), *Myers' "diff" algorithm* (Myers, 1986) is applied to record dumps created before and after persisting changes of an entity.



System message 947164 - inquiry value updated - test trial

This journal entry was generated by the system.

Title: inquiry value updated - test trial
Timestamp: 17.02.2014 04:09

Record: booleanValue: false
id: 946987
inquiry: test trial - 01 - Basisinformation - 13. Grunderkrankung
modifiedTimestamp: 2014-02-17 04:04~~:09~~9:34
modifiedUser: ~~rkrenn_crc~~user_1_1
proband: John Doe
version: ~~2~~3
selectionValues.0: Diabetes~~typ unbekannt~~ Typ I
selectionValues.1: Psoriasis

created: user_1_1 (17.02.2014 04:09)

Figure 4.41.: Changes of update operations are highlighted by identifying differences of the record dump texts, as shown by this system message example.

A system message is created as final step of most CRUD service method by persisting the `JournalEntry` record using the `addJournalEntry` DAO method. In order to produce a comprehensive journal per root entity item, selected operations such as deletions potentially require to generate a multitude of journal entries for associated items. For example, when removing a `Proband` record graph, a system message is created for any trial the subject participated in. While proband list entries of affected trials are effectively deleted in this case, inventory bookings linked to the proband to remove are not deleted but updated by setting the `InventoryBooking.proband` field to `null` and generating a corresponding system message regarding the record update. Specifics like this disqualify the usage of `ON DELETE CASCADE` triggers in the database schema as well as a generic approach for creating system message entries using *interceptors*.

By design, a separate input element for specifying the reason for data changes is not provided by the UI in general. However, there are situations where providing a reason is explicitly required and enforced. As an example, a mandatory text input element requires users to specify the reason for enrollment status changes (figure 4.43).

4.3.2. Excel Export

Export and subsequent download of tabular data in Microsoft's Excel format was a recurring requirement. Currently, 18 different exports in Excel 2003 format[38] (`.xls`) are supported. Generating Excel file content and subsequent download is initiated by pressing buttons found in related screens across the UI.

This section describes available exports and the extensible implementation design used to convert VO collections into localized Excel worksheets using the jExcelApi library (Khan, 2013).

[37]l10n bundles with labels and translations for OutVO field names are not completed yet.
[38]Binary Interchange File Format (BIFF)

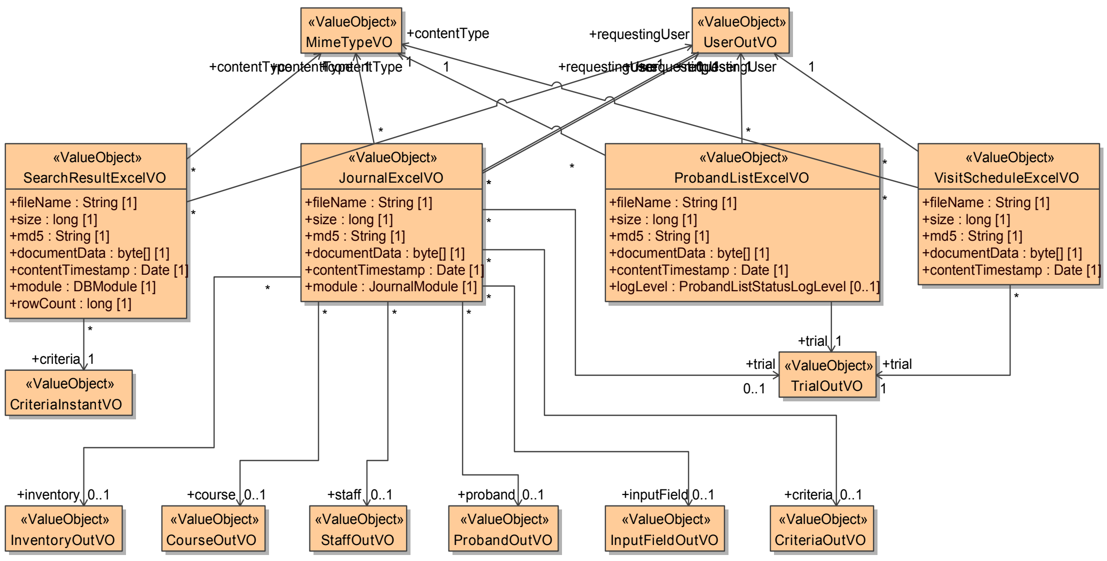

«ValueObject» MimeTypeVO
«ValueObject» UserOutVO
+contentType
+requestingUser
«ValueObject» SearchResultExcelVO
+fileName : String [1]
+size : long [1]
+md5 : String [1]
+documentData : byte[] [1]
+contentTimestamp : Date [1]
+module : DBModule [1]
+rowCount : long [1]
«ValueObject» JournalExcelVO
+fileName : String [1]
+size : long [1]
+md5 : String [1]
+documentData : byte[] [1]
+contentTimestamp : Date [1]
+module : JournalModule [1]
«ValueObject» ProbandListExcelVO
+fileName : String [1]
+size : long [1]
+documentData : byte[] [1]
+contentTimestamp : Date [1]
+logLevel : ProbandListStatusLogLevel [0..1]
«ValueObject» VisitScheduleExcelVO
+fileName : String [1]
+size : long [1]
+md5 : String [1]
+documentData : byte[] [1]
+contentTimestamp : Date [1]
«ValueObject» CriteriaInstantVO
«ValueObject» TrialOutVO
«ValueObject» InventoryOutVO
«ValueObject» CourseOutVO
«ValueObject» StaffOutVO
«ValueObject» ProbandOutVO
«ValueObject» InputFieldOutVO
«ValueObject» CriteriaOutVO
+criteria
+trial
+inventory
+course
+staff
+proband
+inputField

Figure 4.42.: An extendible number of DTOs are used as return types for service methods that produce Excel exports.

UI requests to generate Excel exports are processed synchronously by single service method invocations. They return a `*ExcelVO` DTO (figure 4.42), which carries the created Excel content in the `documentData` `byte[]` field. Since the magnitude of content size should not exceed MBs, *streaming* is not implemented here. Additional fields provide information such as:

- creation timestamp
- requesting `User`
- the regular content type (MIME-type)[39]
- associated master record and specific details

Excel DTOs represent categories of currently implemented exports:

Query results: Dynamic database queries retrieve root entity records matching user-defined criteria. Exporting the presented result set in Excel format was desired in order to use it for e.g. printable checklists.

Journals: The list of `JournalEntry` records of a given root entity instance is exportable for offline review in the course of audits.

Visit schedule: The visit schedule comprises a trial's `VisitScheduleItems`, showing all appointments of investigation visits per proband group. It can be exported in order to hand it out to probands as a part of the *subject information* material.

Subject lists: A trial's subject list consists of stateful `ProbandListEntry` records. Each `ProbandListEntry` item is associated with a proband and a current enrollment status type. The probands' enrollment status changes with the progression of the trial. An evolving enrollment status history of a proband is shown in figure 4.43. The current configuration distinguishes between four subject lists by a filter mechanism:

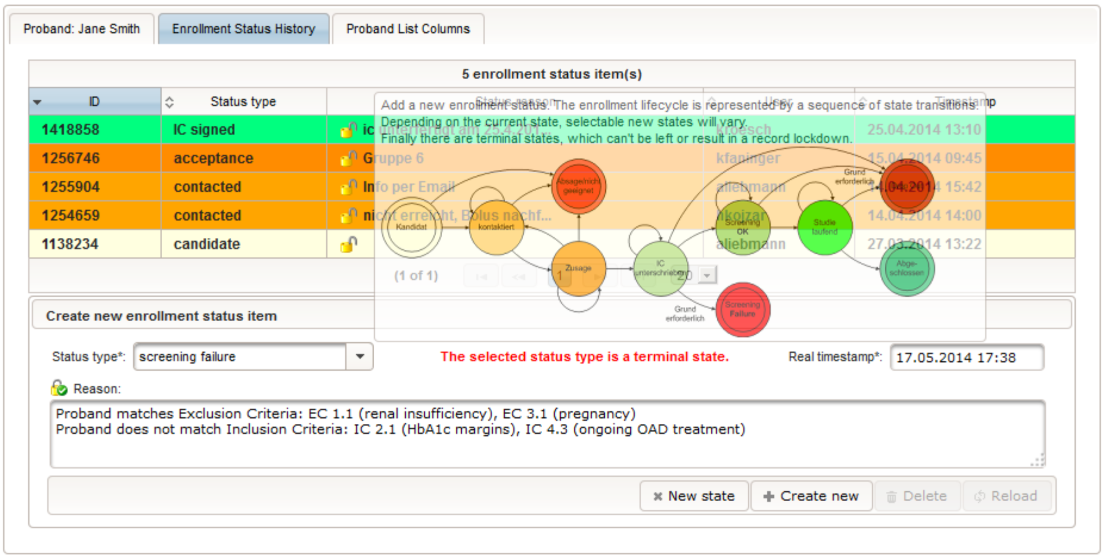
Pre-screening log: Probands that passed or are currently in the initial "candidate" state (all probands).

Screening log: Probands that passed or are currently in the "screening OK" or "screening failure" state.

Enrollment log: Probands that passed or are currently in the "trial ongoing" state.

Entire proband list: For the current configuration, this is the same as Pre-screening log.

[39] `application/vnd.ms-excel`



Proband: Jane Smith | Enrollment Status History | Proband List Columns

5 enrollment status item(s)

ID	Status type	Status reason	User	Timestamp
1418858	IC signed	ic unterfertigt am 25.4.201...	kroesch	25.04.2014 13:10
1256746	acceptance	Gruppe 6	kfaninger	15.04.2014 09:45
1255904	contacted	Info per Email	aliebmann	14.04.2014 15:42
1254659	contacted	nicht erreicht, Bolus nachf...	nkojzar	14.04.2014 14:00
1138234	candidate		aliebmann	27.03.2014 13:22

(1 of 1)

Add a new enrollment status. The enrollment lifecycle is represented by a sequence of state transitions.
Depending on the current state, selectable new states will vary.
Finally there are terminal states, which can't be left or result in a record lockdown.

Create new enrollment status item

Status type*: screening failure

The selected status type is a terminal state.

Real timestamp*: 17.05.2014 17:38

Reason:

```
Proband matches Exclusion Criteria: EC 1.1 (renal insufficiency), EC 3.1 (pregnancy)
Proband does not match Inclusion Criteria: IC 2.1 (HbA1c margins), IC 4.3 (ongoing OAD treatment)
```

Figure 4.43.: This UI detail is part of the proband list tab (figure 4.97) and shows the enrollment status history of a selected proband list entry.

The creation of Excel DTOs is performed in two steps. At first, program logic of the service method retrieves result records using some finder method. The transformed list of OutVOs represent spreadsheet rows. The list is passed as input for a corresponding `SearchResultExcelWriter`, `JournalExcelWriter`, `ProbandListExcelWriter` or `VisitScheduleExcelWriter` (figure 4.44) instance, which generates the Excel document content and populates DTO fields in the second step.

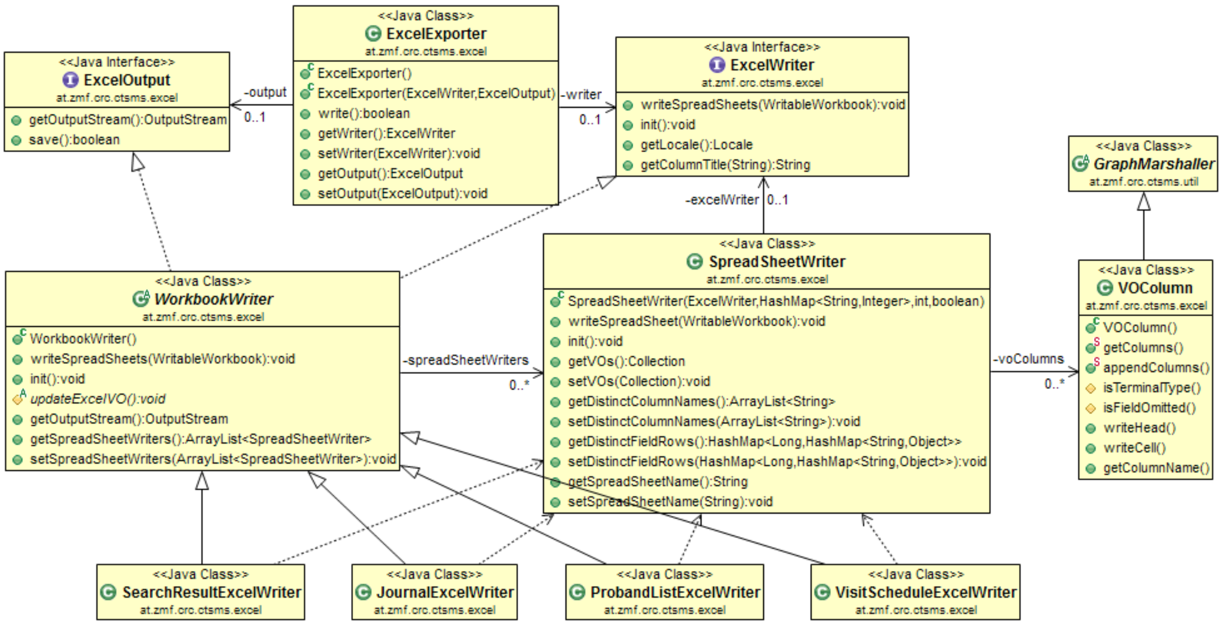


Figure 4.44.: `*ExcelWriter` classes are fed with a VO list to produce an Excel spreadsheet from it. `SpreadSheetWriter` takes over the main work and utilize `VOColumn` graph marshallers.

The generic approach to create Excel contents shown in figure 4.44 is based on a composite pattern.

The `ExcelExporter` base class allows combinations of two aspects: what content to write (interface `ExcelWriter`) and how to save the generated output (interface `ExcelOutput`). At the moment, writing into a `byte[]` variable is implemented only, which is handled within the `WorkbookWriter` class using a `ByteArrayOutputStream`. In addition, `WorkbookWriter` can manage the structure of an Excel file which organizes multiple spreadsheets within a single *workbook*.[40]

`SpreadsheetWriter` contains the more complex parts. It compiles `VOColumn` graph marshallers for the type of the first element of passed spreadsheet rows in form of a VO `Collection`. `VOColumns` are arranged according to a dedicated order, defined by the specific `WorkbookWriter` implementation class. Each `VOColumn` maps a single-valued VO field to a spreadsheet output column. Field path keys are used as keys in l10n bundles for column header labels. Beside ordering, a columns configuration allows excluding specific entity fields explicitly.

Basically, available columns represented by populated `VOColumn` graph marshallers will enclose OutVO terminal type fields and single references (x-to-one associations) only. The conceptual constraint of a fixed number of columns in spreadsheets requires the default processing of graph marshaller to suppress multi-valued fields. However, multi-valued fields are inherent to master-details domain models and require workarounds, such as expanding detail records with maximum occurence limits (section 4.1.3.4) and/or aggregated cell string values. These come into effect for dedicated requirements in the case of subject list and query result exports. `SpreadsheetWriter` provides setters to specify additional columns and corresponding cell values explicitly. The calling service method is in charge of building strings for cell values and has to provide a sorted list of additional column names, passed by the `setDistinctColumnNames` method. Corresponding row data containing cell values has to be passed subsequently in form of a `Map<Long, Map<String, Object>>` data structure. In this data structure, the superior map assigns each row VO `id` value a map of column name-value pairs.

Excel content is written cell-by-cell by iterating `VOColumns` per iterated VO instance. Each terminal field type is mapped to one of the Excel column data types, which are exposed by jExcelApi:

- `jxl.write.Boolean`
- `jxl.write.Label`
  - `jxl.write.NumberFormats.TEXT`
- `jxl.write.Number`
  - `jxl.write.NumberFormats.INTEGER`
  - `jxl.write.NumberFormats.FLOAT`
- `jxl.write.DateTime`[41]
  - `jxl.write.DateFormat(ExcelUtil.EXCEL_DATE_PATTERN`[42]`)`
  - `jxl.write.DateFormat(ExcelUtil.EXCEL_DATE_TIME_PATTERN`[43]`)`

When the row iteration is finished, the stream and handles are closed, which is covered by the `ExcelExporter` base class. Finally, the `ExcelOutput.save` override flushes the data to an implemented target such as a file or `byte[]` field.

The Excel export infrastructure presented processes generic OutVO collections using the `VOColumn` graph marshaller. Integrating an additional export can be done in minutes by providing another `WorkbookWriter` implementation class. Aside details such as column ordering or i18n, it allows declaring desired columns using a field association path notation for OutVO object graphs. Since there were no extraordinary style and file format requirements for generated Excel files, the lightweight jExcelApi qualified as sufficient. A reason for moving on to Apache "Poor Obfuscation Implementation" (Apache POI) in a

[40]Current exports use single spreadsheet workbooks only.
[41]The server's time zone is applied irrespective of the user's time zone setting.
[42]`"yyyy-MM-dd"`
[43]`"yyyy-MM-dd HH:mm:ss"`

future version could be the requirement to generate `.xlsx` files (European Computer Manufacturers Association (ECMA)-376 Office Open XML File Format).

4.3.3. Digital Signatures

Some document types used in processes of clinical trials constitute legally binding documents. In paper form, they require the date and own-hand signature of the competent person (*signee*) who is responsible for the document's content. In this context, the electronic mapping of the document's contents results in *regulated records* that need an adequate treatment. Digital signatures for such records are the way to go in order to provide an unforgeable proof of the signee's identity reference and the record's content authenticity. The use of digital signatures for data exports to authorities is claimed in FDA Part 11 and EU Annex 11 and is implemented e.g. in the CDISC ODM XML format (CDISC, 2013, chapter 3.1.4.1.3).

At the moment, the Phoenix CTMS does not directly manage data, which is considered as regulated records in the common sense (e.g. SDF and CRF documents, application forms, final reports, . . .). However, copies of these documents will be stored in the file repository of a trial. Furthermore, the question arises to what extent inquiry and proband list data entered in the course of the Phoenix CTMS EDC use cases have to be considered as regulated records.[44]

To response to doubts, infrastructure and a full featured use case for the calculation and verification of *digital signatures* for *object graphs* is prepared. The `Trial` graph was chosen as a starting point since it includes `File` records with document content worthy of protection as well as records of EAV models (section 4.1.1.3), created in the course of EDC. The capability to calculate and verify digital signature values of entity graphs should address export formats that could be supported in the course of future extensions to the system.

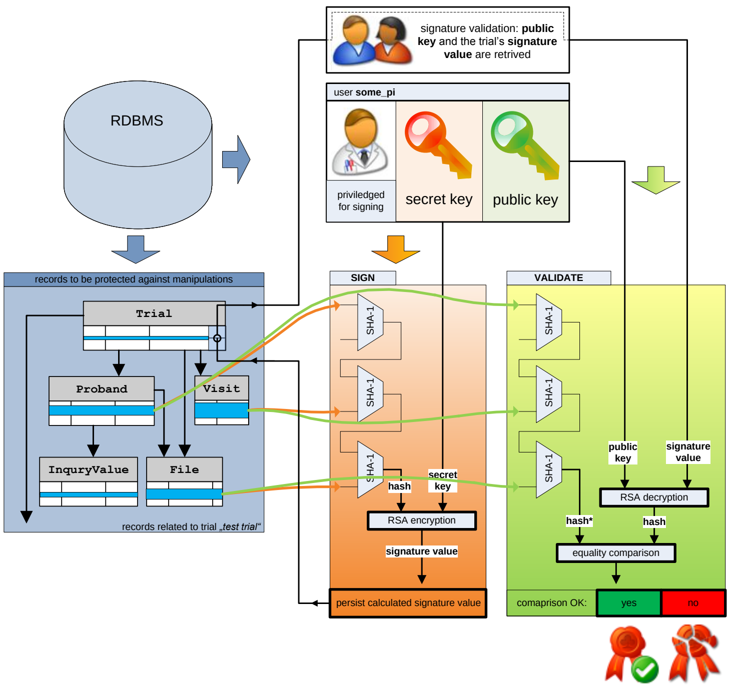


Figure 4.45.: Overview of signing and validation flows for digital signatures of records.

[44]Eligibility criteria queries run against inquiry question answers and captured medical parameters of subjects, which therefore definitely influence the decision of participation in trials.

The use case of signing and signature verification is outlined in figure 4.45. When a new signature is to be created, the entity instance graph checksum is calculated internally within the `TrialSignature` graph marshaller at first. A suitable traversal depth will cover all relevant detail entities. Specific fields that are of no relevance (e.g. previous signatures) or that are likely to render a signature invalid due to anticipated modifications (e.g. `Notification` delivery status change) are skipped explicitly (listing 4.19).

```
public class TrialSignature extends EntitySignatureBase {

   ...

   @Override
   protected boolean isFieldOmitted(Class graph, String field) {
       //traversal encounters ...
       if (Trial.class.isAssignableFrom(graph)) { //... a trial entity:
           if ("signatures".equals(field)) {
               //if trial status actions are configured to create a signature per every single trial status
               //transition, a multitude of Signature instances is persisted for this trial over time. However,
               //these instances need not be included in the calculation of this new, most recent Signature:
               return true;
           } else if ("notifications".equals(field)) {
               //associated Notification instances get their delivery status changed when the deferred email
               //sending job will run. This would therefore cause subsequent signature validations to fail,
               //although this is of no significance:
               return true;
           }
       //uncomment the following to exclude the trial's journal:
       //} else if (JournalEntry.class.isAssignableFrom(graph)) { //... a JournalEntry entity:
       //  return true;
       }
       return false;
   }

   ...
```

Listing 4.19: This method override in the `TrialSignature` graph marshaller class (figure 4.38) defines which records to exlude when calculating the signature value of a `Trial` record graph.

The signee is represented by a `User` entity (figure 4.46). In general, a user is optionally associated with a person/organization, which in turn contains personal details like the full name. With the creation of a new user, a asymmetric cryptography algorithm by Ron Rivest, Adi Shamir and Leonard Adleman (RSA) key pair is generated. Its secret key is encrypted using PBE as described in section 4.2.3. While the public key is used to verify existing signatures, the secret key is used to calculate new signature values. Business logic can obtain both keys using accessor methods of the user context object (listing 4.22).

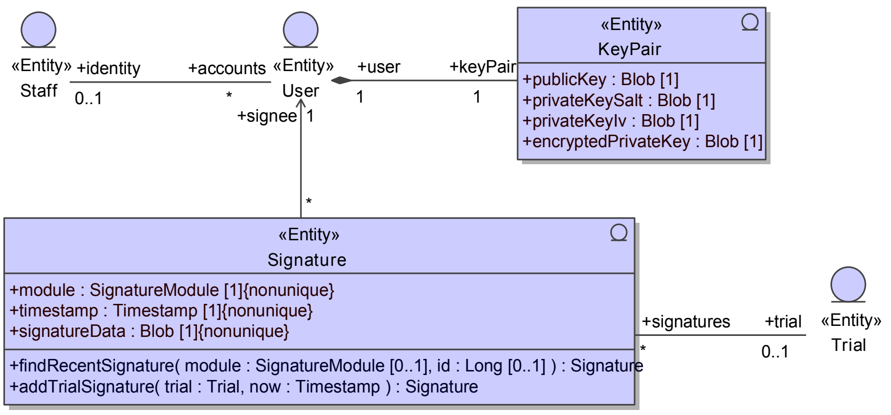

«Entity»
Staff
+identity
0..1
+accounts
*
«Entity»
User
+signee
1
+user
1
+keyPair
1
«Entity»
KeyPair
+publicKey : Blob [1]
+privateKeySalt : Blob [1]
+privateKeyIv : Blob [1]
+encryptedPrivateKey : Blob [1]
*
«Entity»
Signature
+module : SignatureModule [1]{nonunique}
+timestamp : Timestamp [1]{nonunique}
+signatureData : Blob [1]{nonunique}
+findRecentSignature(module : SignatureModule [0..1], id : Long [0..1]) : Signature
+addTrialSignature(trial : Trial, now : Timestamp) : Signature
+signatures
*
+trial
0..1
«Entity»
Trial

Figure 4.46.: The `Signature` entity comprises signature value and timestamp fields and refers to the signee `User` which in turn can refer to a `Staff` identity. Calculation and persisting a signature is implemented for `Trial` entities only. Both steps are performed by invoking the `addTrialSignature` DAO method.

Binary data of the trial's associated files is factored into the checksum by the `File.md5` field values. Final rounds append the elements below to prevent tampering:

Signature timestamp: Signature timestamp manipulations must render the signature invalid.
Signee user: If a username is modified, all signatures issued by the user should be invalid.
Signee identity: If a user's associated identity `Staff` is changed, all signatures issued by the user should be invalid.

In principle, the Secure Hash Algorithm 1 (SHA-1) checksum result is encrypted with the signee's secret key using an asymmetric cryptographic algorithm. The encryption result is the binary signature value which is finally persisted in a `Signature` record. These steps are performed by a `TrialSignature` graph marshaller instance within the `addTrialSignature` DAO method. It employs a `java.security.Signature` instance configured to `SHA1withRSA` mode, which encapsulates outlined cryptographic operations to create and verify RSA Signature Scheme with Appendix (SSA) PKCS1 v1.5 (Kaliski, 1998; Jonsson and Kaliski, 2003) compliant digital signatures.

Details of the signing and verification procedures are summarized below:

Create keypair for a user (system setup)

1. plaintext user password $P_{UP} \leftarrow$ input parameter
2. public key K_P, secret key K_S: (K_P, K_S) = `createRSAKeyPair`
3. secret key salt S_{K_S} = `createSalt`
4. encrypted secret key C_{K_S}, secret key initialisation vector IV_{K_S}:
   $(C_{K_S}, IV_{K_S}) = \mathbf{E}_{PBE:P_{UP},S_{K_S}}(K_S)$
5. K_P, S_{K_S}, C_{K_S}, IV_{K_S} $\rightarrow$ `KeyPair` entity $\rightarrow$ persist

Create entity signature (regular operation)

1. signee $\leftarrow$ load `User` $\leftarrow$ user credential parameter
2. $P_{UP} \leftarrow$ password credential parameter
3. serialized signee P_{SU}[45], serialized signee identity P_{SI}[46] $\leftarrow$ signee
4. current timestamp t_S = `currentTimestamp`
5. S_{K_S}, C_{K_S}, IV_{K_S} $\leftarrow$ load `KeyPair` $\leftarrow$ signee
6. entity instance graph G with m terminal field values P_{FV_i} $\leftarrow$ `GraphMarshaller` $\leftarrow$ load entity $\leftarrow$ entity ID input parameter
7. graph hash $H_G = H_{G_m} \leftarrow$ `for i = 1 ..` m: H_{G_i} = `SHA1(`$H_{G_{i-1}}$`, `P_{FV_i}`)`
8. H_G = `SHA1(SHA1(SHA1(`H_G`, )`t_S`, `P_{SU}`), `P_{SI}`)`
9. $K_S = \mathbf{D}_{PBE:P_{UP},S_{K_S}}(C_{K_S}, IV_{K_S})$
10. graph signature value $S_G = \mathbf{E}_{RSA:K_S}(H_G)$[47]
11. S_G, t_S, signee $\rightarrow$ `Signature` entity $\rightarrow$ persist

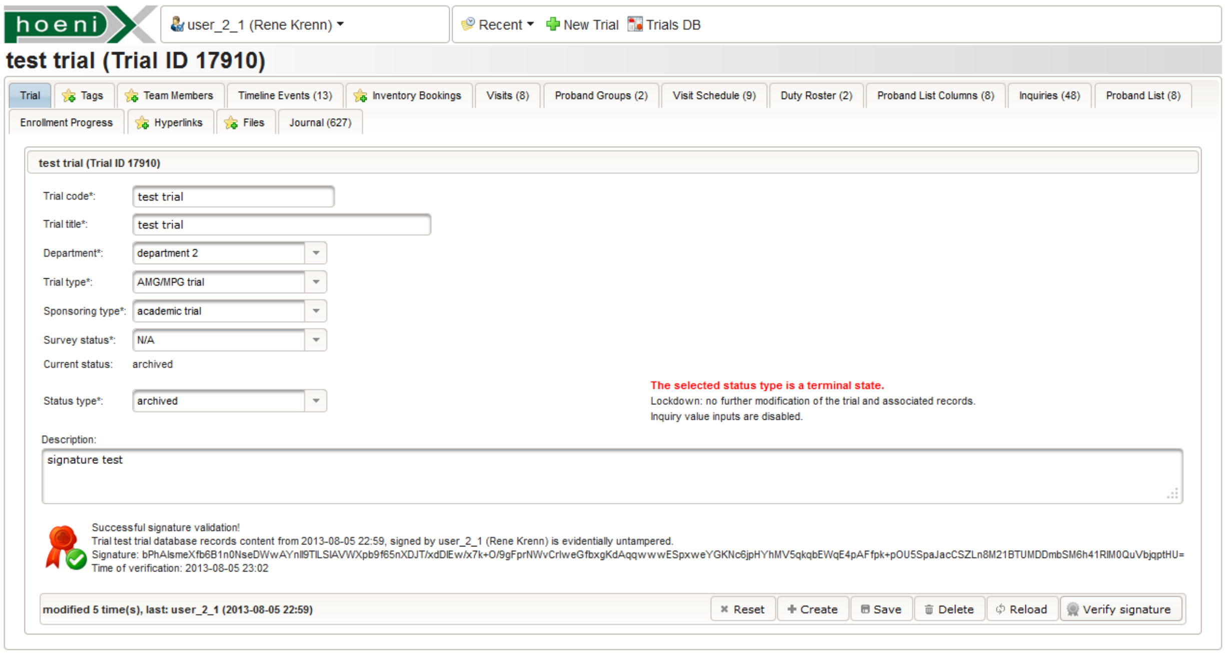
Verify entity signature (regular operation)

1. S_G, entity, t_S, signee $\leftarrow$ load `Signature` $\leftarrow$ entity ID input parameter
2. P_{SU}, P_{SI} $\leftarrow$ signee
3. $K_P \leftarrow$ load `KeyPair` $\leftarrow$ signee
4. modified entity instance graph G^* with m^* terminal field values $P^*_{FV_i}$ $\leftarrow$ `GraphMarshaller` $\leftarrow$ entity
5. modified graph hash $H_{G^*} = H_{G^*_{m^*}} \leftarrow$ `for i = 1 ..` m^*: $H_{G^*_i}$ = `SHA1(`$H_{G^*_{i-1}}$`, `$P^*_{FV_i}$`)`
6. H_{G^*} = `SHA1(SHA1(SHA1(`H_{G^*}`, )`t_S`, `P_{SU}`), `P_{SI}`)`
7. $H_G = \mathbf{D}_{RSA:K_P}(S_G)$[47,48]
8. $H_{G^*} \overset{?}{=} H_G \rightarrow$ boolean return value

[45] `[id, name]`
[46] `[id, firstName, lastName, dateOfBirth]`
[47] Prepending the signature algorithm identifier, encoding and padding as proposed by the RSASSA PKCS1 v1.5 standard is ommitted for simplification.
[48] $\mathbf{E}_{RSA} \equiv \mathbf{D}_{RSA}$

Since the signature value calculation is performed by the application itself and not an external *trusted service*, electronic signatures produced cannot be considered as *qualified electronic signatures* in the (Austrian) legal sense (*Signaturgesetz - SigG* 1999). The supplemental proof required for the *extended electronic signatures* created rely on secret signee user passwords, untampered application `.jar` files and correct server time, which is supposed to be examined in the course of the system validation. At the moment, qualified electronic signatures are not claimed by regulations however (Q-FINITY Quality Management, 2014).

Privileged users create trial signatures when switching the trial's status to "archived". This behavior is configurable by means of trial status transition actions as outlined in section 4.1.3.6). With the archived state, the trial related records are marked as locked and any further changes are prevented. Therefore the UI now shows the button ("Verify signature") to any user that is privileged to view the trial (figure 4.47). The UI lockdown logic can not exclude subsequent table content manipulations, caused e.g. by database administrators. Therefore, the digital signature allows to prove records were not altered after the signature's timestamp.



test trial (Trial ID 17910)

test trial (Trial ID 17910)

Trial code*:	test trial
Trial title*:	test trial
Department*:	department 2
Trial type*:	AMG/MPG trial
Sponsoring type*:	academic trial
Survey status*:	N/A
Current status:	archived
Status type*:	archived

The selected status type is a terminal state.
Lockdown: no further modification of the trial and associated records.
Inquiry value inputs are disabled.

Description:
signature test

Successful signature validation!
Trial test trial database records content from 2013-08-05 22:59, signed by user_2_1 (Rene Krenn) is evidentially untampered.
Signature: bPhAlsmeXfb6B1n0NseDWwAYnll9TILSIAVWXpb9f65nXDJT/xdDlEw/x7k+O/9gFprNWvCrlweGfbxgKdAqqwwwESpxweYGKNc6jpHYhMV5qkqbEWqE4pAFfpk+pOU5SpaJacCSZLn8M21BTUMDDmbSM6h41RIM0QuVbjqptHU=
Time of verification: 2013-08-05 23:02

modified 5 time(s), last: user_2_1 (2013-08-05 22:59)

Figure 4.47.: This sample UI screen shows the the trial entity view's main tab, which incorporates the verification of the digital signatures. Users can trigger the verification, if a signature is available after the trial's state was set to "archived".

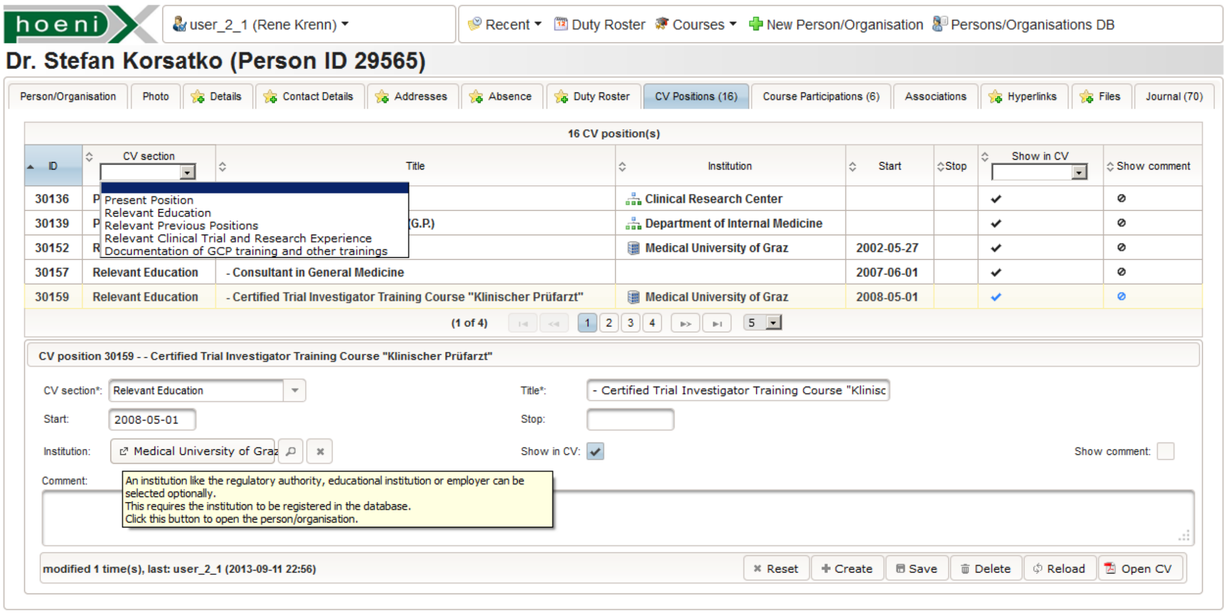
The size of `Trial` entity instance graphs is expected reach tens of thousands of records if system message journal entries are included. Signature calculation and validation can cause noticeable response times in this case and should therefore be performed asynchronously in a future version.

4.4. PDF Export

4.4.1. Document Types

The Phoenix CTMS is currently capable of generating four types of PDF documents outlined in this section. The UI provides corresponding buttons to request the creation of document files for subsequent download and printing. Files are generated in PDF 1.5 format using the PDFBox library (Lehmkühler and PDFBox Contributors, 2013).

CV The investigator is required to provide an up-to-date Curricula Vitae (CV's) in order to "document qualifications and eligibility to conduct [a] trial and/or provide medical supervision of subjects" (*ICH E6 (R1) Guideline for Good Clinical Practice: Consolidated Guidance* 1996, section 8.2.10). This also holds for sub-investigators, monitor personnel and auditors as well - thus for virtually any trial site staff working on the clinical trial.



Dr. Stefan Korsatko (Person ID 29565)

16 CV position(s)

ID	CV section	Title	Institution	Start	Stop	Show in CV	Show comment
30136	P		Clinical Research Center			✓	⊘
30139	P	G.P.)	Department of Internal Medicine			✓	⊘
30152	R		Medical University of Graz	2002-05-27		✓	⊘
30157	Relevant Education	- Consultant in General Medicine		2007-06-01		✓	⊘
30159	Relevant Education	- Certified Trial Investigator Training Course "Klinischer Prüfarzt"	Medical University of Graz	2008-05-01		✓	⊘

Present Position
Relevant Education
Relevant Previous Positions
Relevant Clinical Trial and Research Experience
Documentation of GCP training and other trainings
(1 of 4)
1 2 3 4
5

CV position 30159 - - Certified Trial Investigator Training Course "Klinischer Prüfarzt"

CV section*:	Relevant Education	Title*:	- Certified Trial Investigator Training Course "Klinisc
Start:	2008-05-01	Stop:	
Institution:	Medical University of Graz	Show in CV:	✓
Comment:		Show comment:	

An institution like the regulatory authority, educational institution or employer can be selected optionally.
This requires the institution to be registered in the database.
Click this button to open the person/organisation.

modified 1 time(s), last: user_2_1 (2013-09-11 22:56)

Figure 4.48.: This sample UI screen shows the tab for managing a person's CV positions. CV section and title are mandatory fields for a position entry. An institution can be selected if it is registered in the person/organisation database. This resolves the problem to ensure uniform spellings like "Medical University of Graz" in CV's.

A CV document can be generated for every person that is registered as `Staff` record (figure 4.49). It represents a uniform CV in tabular format with relevant CV positions only which are categorized by predefined sections. Users are enabled to maintain their own CV positions using the person/organisation entity view tab shown in figure 4.48. Additionally, successfully attended courses that are managed by the Phoenix CTMS can automatically be endorsed. Since the number of CV positions and course endorsements is not limited basically, the document can span multiple pages. On top the first page, a separated paragraph block shows general details about the employee:

- full name
- DoB
- title/academic degree[49]
- an optional photo[50]
- business address

If the employee has not explicitly marked one of his/her addresses as CV business address, the `Staff` tree hierarchy (figure 4.13) is searched upwards until an organisation with a marked CV address is found. Traversed organizations are listed as part of the CV address to resemble the department hierarchy information known from corporate email signature blocks. Administrative users can finally list employees using the database query editor to retrieve a single PDF file that aggregates their CV's.

[49]Users can manually provide the translation of titles/academic degrees for the CV document language using a dedicated field.
[50]The UI supports uploading a photo or directly capturing it from a webcam attached to the user's workstation computer.

Course participant list Investigators are furthermore required to provide additional documents that prove CV positions and course endorsements. While scanned application papers like diplomas can be archived using file manager of the person/organisation module, participant lists with own-hand signatures of participants and the performing trainer have to be provided as evidence for attendance of specific courses or trainings. The course participant list is a multi-page signature list (figure 4.50), supposed to be printed out, signed and optionally scanned to store it in the course's file repository finally. Rows with names of registered participants for each lecture appointments[51] are rendered. The adjustable page orientation turned out as a helpful configuration option to reduce the number of pages in the case of many lecture appointments.

Course certificate To provide a discrete evidence for the attendance of a managed course, the course certificate document can be generated for a single participant or all participants of a course (figure 4.51). Details are arranged in a formal style that is unlikely to exceed a single-page:

- course date or duration
- course description
- validity period
- course lecturer and trainer names
- related clinical trial
- issuing authority (person/organisation)

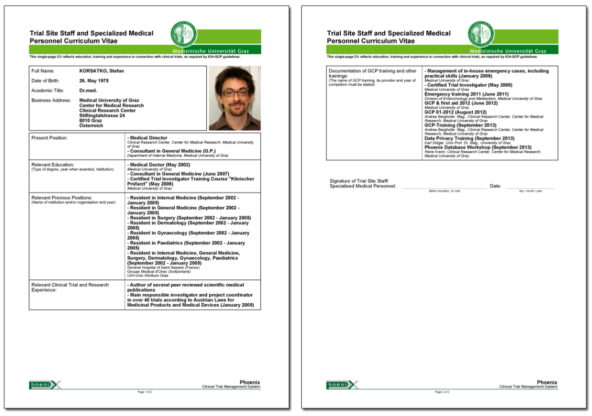
Aside the regular case of sending generated documents to authorities and sponsor organisations, the stamped certificate printout can be used by the certificate recipient for job applications in the future. This is in particular relevant for the many student workers.

Proband letter When first contact was established with a trial candidate by phone, his/her name and an address is preliminary entered into the Phoenix CTMS. Subsequentially, a personal letter document can be generated. This "proband letter" is created by imprinting rudimentary elements on a given PDF template:

- candidate's name
- current date
- postal address and delivery instructions ("care of")
- salutation with respect of gender and titles/academic degrees

The three-page template contains a welcome page and two copies of the data privacy consent (figure 4.52). When the printed proband letter is sent to the candidate, a copy is signed and sent back using the enclosed return envelope for scan-and-store (figure 4.142). During an intense recruitment phase, proband letter printing is simplified by *mail merge* support. Probands can be listed by the query editor to get an aggregated PDF file containing the individual proband letters for printing.

[51]Room reservations are supported by means of `InventoryBookings`. For simplification, these bookings are treated as lecture appointments.



Trial Site Staff and Specialized Medical Personnel Curriculum Vitae

Medizinische Universität Graz

This single-page CV reflects education, training and experience in connection with clinical trials, as required by ICH-GCP guidelines.

Full Name:	**KORSATKO, Stefan**
Date of Birth:	**26. May 1975**
Academic Title:	**Dr.med.**
Business Address:	**Medical University of Graz Center for Medical Research Clinical Research Center Stiftingtalstrasse 24 8010 Graz Österreich**

Present Position:	**- Medical Director** *Clinical Research Center, Center for Medical Research, Medical University of Graz* **- Consultant in General Medicine (G.P.)** *Department of Internal Medicine, Medical University of Graz*
Relevant Education: (Type of degree, year when awarded, institution)	**- Medical Doctor (May 2002)** *Medical University of Graz* **- Consultant in General Medicine (June 2007)** **- Certified Trial Investigator Training Course "Klinischer Prüfarzt" (May 2008)** *Medical University of Graz*
Relevant Previous Positions: (Name of institution and/or organisation and year)	**- Resident in Internal Medicine (September 2002 - January 2005)** **- Resident in General Medicine (September 2002 - January 2005)** **- Resident in Surgery (September 2002 - January 2005)** **- Resident in Dermatology (September 2002 - January 2005)** **- Resident in Gynaecology (September 2002 - January 2005)** **- Resident in Paediatrics (September 2002 - January 2005)** **- Resident in Internal Medicine, General Medicine, Surgery, Dermatology, Gynaecology, Paediatrics (September 2002 - January 2005)** General Hospital of Saint Nazaire (France) Groupe Medical d'Onex (Switzerland) LKH-Univ.Klinikum Graz
Relevant Clinical Trial and Research Experience:	**- Author of several peer reviewed scientific medical publications** **- Main responsible investigator and project coordinator in over 40 trials according to Austrian Laws for Medicinal Products and Medical Devices (January 2005)**

hoeni **Phoenix** Clinical Trial Management System

Page 1 of 2

Trial Site Staff and Specialized Medical Personnel Curriculum Vitae

Medizinische Universität Graz

This single-page CV reflects education, training and experience in connection with clinical trials, as required by ICH-GCP guidelines.

Documentation of GCP training and other trainings: (The name of GCP training, its provider and year of completion must be stated)	**- Management of in-house emergency cases, including practical skills (January 2006)** *Medical University of Graz* **- Certified Trial Investigator (May 2008)** *Medical University of Graz* **Emergency training 2011 (June 2011)** *Division of Endocrinology and Metabolism, Medical University of Graz* **GCP & first aid 2012 (June 2012)** *Medical University of Graz* **GCP 01-2012 (August 2012)** *Andrea Berghofer, Mag., Clinical Research Center, Center for Medical Research, Medical University of Graz* **GCP-Training (September 2013)** *Andrea Berghofer, Mag., Clinical Research Center, Center for Medical Research, Medical University of Graz* **Data Privacy Training (September 2013)** *Karl Stöger, Univ.Prof. Dr. Mag., University of Graz* **Phoenix Database Workshop (September 2013)** *Rene Krenn, Clinical Research Center, Center for Medical Research, Medical University of Graz*

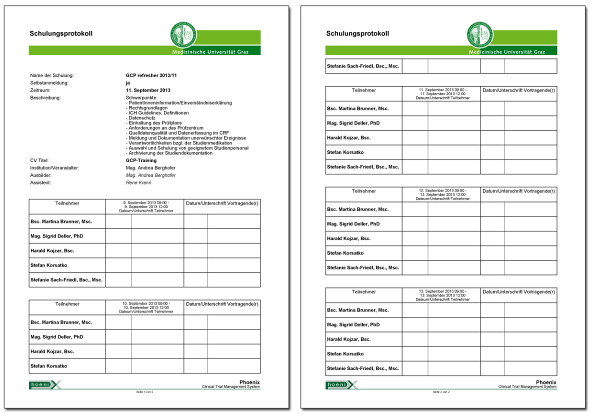
Signature of Trial Site Staff/ Specialised Medical Personnel: ______________ Date: ______________

Stefan Korsatko, Dr.med. day / month / year

hoeni **Phoenix** Clinical Trial Management System

Page 2 of 2

Figure 4.49.: CV sample document.



Schulungsprotokoll

Medizinische Universität Graz

Name der Schulung:	**GCP refresher 2013/11**
Selbstanmeldung:	**ja**
Zeitraum:	**11. September 2013**
Beschreibung:	Schwerpunkte: - PatientInneninformation/Einverständniserklärung - Rechtsgrundlagen - ICH Guidelines, Definitionen - Datenschutz - Einhaltung des Prüfplans - Anforderungen an das Prüfzentrum - Quelldatenqualität und Datenerfassung im CRF - Meldung und Dokumentation unerwünschter Ereignisse - Verantwortlichkeiten bzgl. der Studienmedikation - Auswahl und Schulung von geeignetem Studienpersonal - Archivierung der Studiendokumentation
CV Titel:	**GCP-Training**
Institution/Veranstalter:	Mag. Andrea Berghofer
Ausbilder:	*Mag. Andrea Berghofer*
Assistent:	*Rene Krenn*

Teilnehmer	9. September 2013 09:00 - 9. September 2013 12:00 Dateum/Unterschrift Teilnehmer		Datum/Unterschrift Vortragende(r)	
Bsc. Martina Brunner, Msc.				
Mag. Sigrid Deller, PhD				
Harald Kojzar, Bsc.				
Stefan Korsatko				
Stefanie Sach-Friedl, Bsc., Msc.				

Teilnehmer	10. September 2013 09:00 - 10. September 2013 12:00 Dateum/Unterschrift Teilnehmer		Datum/Unterschrift Vortragende(r)	
Bsc. Martina Brunner, Msc.				
Mag. Sigrid Deller, PhD				
Harald Kojzar, Bsc.				
Stefan Korsatko				
Stefanie Sach-Friedl, Bsc., Msc.				

hoeni **Phoenix** Clinical Trial Management System

Seite 1 von 2

Schulungsprotokoll

Medizinische Universität Graz

Teilnehmer	11. September 2013 09:00 - 11. September 2013 12:00 Dateum/Unterschrift Teilnehmer		Datum/Unterschrift Vortragende(r)	
Bsc. Martina Brunner, Msc.				
Mag. Sigrid Deller, PhD				
Harald Kojzar, Bsc.				
Stefan Korsatko				
Stefanie Sach-Friedl, Bsc., Msc.				

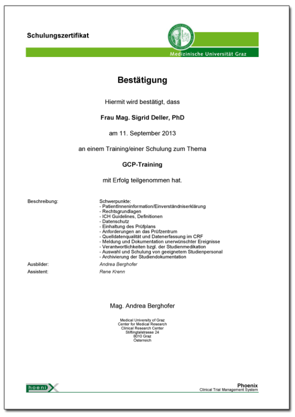
Teilnehmer	12. September 2013 09:00 - 12. September 2013 12:00 Dateum/Unterschrift Teilnehmer		Datum/Unterschrift Vortragende(r)	
Bsc. Martina Brunner, Msc.				
Mag. Sigrid Deller, PhD				
Harald Kojzar, Bsc.				
Stefan Korsatko				
Stefanie Sach-Friedl, Bsc., Msc.				

Teilnehmer	13. September 2013 09:00 - 13. September 2013 12:00 Dateum/Unterschrift Teilnehmer		Datum/Unterschrift Vortragende(r)	
Bsc. Martina Brunner, Msc.				
Mag. Sigrid Deller, PhD				
Harald Kojzar, Bsc.				
Stefan Korsatko				
Stefanie Sach-Friedl, Bsc., Msc.				

hoeni **Phoenix** Clinical Trial Management System

Seite 2 von 2

Figure 4.50.: Course participant sample document.



Schulungszertifikat

Medizinische Universität Graz

Bestätigung

Hiermit wird bestätigt, dass

Frau Mag. Sigrid Deller, PhD

am 11. September 2013

an einem Training/einer Schulung zum Thema

GCP-Training

mit Erfolg teilgenommen hat.

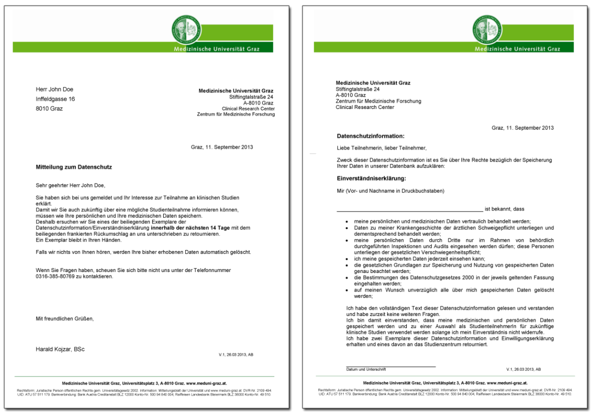
Beschreibung:	Schwerpunkte: - PatientInneninformation/Einverständniserklärung - Rechtsgrundlagen - ICH Guidelines, Definitionen - Datenschutz - Einhaltung des Prüfplans - Anforderungen an das Prüfzentrum - Quelldatenqualität und Datenerfassung im CRF - Meldung und Dokumentation unerwünschter Ereignisse - Verantwortlichkeiten bzgl. der Studienmedikation - Auswahl und Schulung von geeignetem Studienpersonal - Archivierung der Studiendokumentation
Ausbilder:	*Andrea Berghofer*
Assistent:	*Rene Krenn*

Mag. Andrea Berghofer

Medical University of Graz
Center for Medical Research
Clinical Research Center
Stiftingtalstrasse 24
8010 Graz
Österreich

hoeni **Phoenix** Clinical Trial Management System

Figure 4.51.: Course certificate sample document.



Medizinische Universität Graz

Herr John Doe
Inffeldgasse 16
8010 Graz

Medizinische Universität Graz
Stiftingtalstraße 24
A-8010 Graz
Clinical Research Center
Zentrum für Medizinische Forschung

Graz, 11. September 2013

Mitteilung zum Datenschutz

Sehr geehrter Herr John Doe,

Sie haben sich bei uns gemeldet und Ihr Interesse zur Teilnahme an klinischen Studien erklärt.
Damit wir Sie auch zukünftig über eine mögliche Studienteilnahme informieren können, müssen wie Ihre persönlichen und Ihre medizinischen Daten speichern.
Deshalb ersuchen wir Sie eines der beiliegenden Exemplare der Datenschutzinformation/Einverständniserklärung **innerhalb der nächsten 14 Tage** mit dem beiliegenden frankierten Rückumschlag an uns unterschrieben zu retournieren.
Ein Exemplar bleibt in Ihren Händen.

Falls wir nichts von Ihnen hören, werden Ihre bisher erhobenen Daten automatisch gelöscht.

Wenn Sie Fragen haben, scheuen Sie sich bitte nicht uns unter der Telefonnummer 0316-385-80769 zu kontaktieren.

Mit freundlichen Grüßen,

Harald Kojzar, BSc

V.1, 26.03.2013, AB

Medizinische Universität Graz, Universitätsplatz 3, A-8010 Graz. www.meduni-graz.at.

Rechtsform: Juristische Person öffentlichen Rechts gem. Universitätsgesetz 2002. Information: Mitteilungsblatt der Universität und www.meduni-graz.at. DVR-Nr. 2109 494. UID: ATU 57 511 179. Bankverbindung: Bank Austria Creditanstalt BLZ 12000 Konto-Nr. 500 94 840 004, Raiffeisen Landesbank Steiermark BLZ 38000 Konto-Nr. 49 510.

Medizinische Universität Graz

Medizinische Universität Graz
Stiftingtalstraße 24
A-8010 Graz
Zentrum für Medizinische Forschung
Clinical Research Center

Graz, 11. September 2013

Datenschutzinformation:

Liebe Teilnehmerin, lieber Teilnehmer,

Zweck dieser Datenschutzinformation ist es Sie über Ihre Rechte bezüglich der Speicherung Ihrer Daten in unserer Datenbank aufzuklären:

Einverständniserklärung:

Mir (Vor- und Nachname in Druckbuchstaben)

______________________________ ist bekannt, dass

- meine persönlichen und medizinischen Daten vertraulich behandelt werden;
- Daten zu meiner Krankengeschichte der ärztlichen Schweigepflicht unterliegen und dementsprechend behandelt werden;
- meine persönlichen Daten durch Dritte nur im Rahmen von behördlich durchgeführten Inspektionen und Audits eingesehen werden dürfen; diese Personen unterliegen der gesetzlichen Verschwiegenheitspflicht;
- ich meine gespeicherten Daten jederzeit einsehen kann;
- die gesetzlichen Grundlagen zur Speicherung und Nutzung von gespeicherten Daten genau beachtet werden;
- die Bestimmungen des Datenschutzgesetzes 2000 in der jeweils geltenden Fassung eingehalten werden;
- auf meinen Wunsch unverzüglich alle über mich gespeicherten Daten gelöscht werden;

Ich habe den vollständigen Text dieser Datenschutzinformation gelesen und verstanden und habe zurzeit keine weiteren Fragen.
Ich bin damit einverstanden, dass meine medizinischen und persönlichen Daten gespeichert werden und zu einer Auswahl als StudienteilnehmerIn für zukünftige klinische Studien verwendet werden solange ich mein Einverständnis nicht widerrufe.
Ich habe zwei Exemplare dieser Datenschutzinformation und Einwilligungserklärung erhalten und eines davon an das Studienzentrum retourniert.

____
Datum und Unterschrift

V.1, 26.03.2013, AB

Medizinische Universität Graz, Universitätsplatz 3, A-8010 Graz. www.meduni-graz.at.

Rechtsform: Juristische Person öffentlichen Rechts gem. Universitätsgesetz 2002. Information: Mitteilungsblatt der Universität und www.meduni-graz.at. DVR-Nr. 2109 494. UID: ATU 57 511 179. Bankverbindung: Bank Austria Creditanstalt BLZ 12000 Konto-Nr. 500 94 840 004, Raiffeisen Landesbank Steiermark BLZ 38000 Konto-Nr. 49 510.

Figure 4.52.: Proband letter sample document (copy of the data privacy agreement left out).

4.4.2. Renderer Implementation

The extendable implementation is characterized by the following generic features:

Document layout and style configuration Document layouts are based on vertically arranged, rectangular paragraph blocks with dynamic content each. Although configuration of the order and visibility of blocks can be supported, block contents are rendered using hard-coded PDFBox API operations. Each document type externalizes settings into a `.properties` bundle that accumulates parameters for adjusting aspects of the document layout:

Template: Instead of drawing onto blank pages, a template PDF file specified by a filepath can be loaded. The document output results in a copy of the template pages with generated content imprinted. The template can provide multiple pages which are consumed in their order when generating output pages. The last template page is reused if the output page count exceeds the template page count. Templates allow to adopt to the given corporate design guidelines for printable media (Medical University of Graz, 2012a). Exact wordings are provided by the CRC and were reviewed by the legal department as required.

Page: Each document type has a consistent page format. Page size and orientation can be configured independent of the template page format.

Margins and spacings: Exposed constants include various drawing dimensions:

- page margins
- block and border spacings
- min. and max. block height/width limits
- indentations

...

Values are of float type and based on the (virtual) *drawing space*[52] resolution of 72 Dots Per Inch (dpi). Thus, a distance d need to be converted to drawing space units:

$$d[drawing\,space\,unit] = \frac{72[\text{dpi}]}{25.4[mm]} d[mm]$$

Frames and lines: The line width of optional paragraph block frames or other lines drawn is adjustable and taken into account with spacing calculations.

Fonts: Font styles are managed with variables for a limited number of font types that are used to render a document type. In most cases, the PDF built-in typefaces ((*PDF Reference, sixth edition: Adobe Portable Document Format version 1.7* 2006, p. 40)) are sufficient. Alternatively, the path of a TrueType Font (TTF) file can be specified to comply with corporate identity guidelines. Enumerated font sizes with corresponding line spacings cover required text sizes and promote a uniform look of all document types. PDF font metrics and string measurement give dimensions in *glyph units* (*PDF Reference, sixth edition: Adobe Portable Document Format version 1.7* 2006, p. 203) which need to be converted to drawing space units for the given font size f:

$$d[drawing\,space\,unit] = \frac{f[drawing\,space\,unit]}{1000} d[glyph\,unit]$$

Colors: Each document handles a limited number of color variables to define text and line colors. Colors are defined in the `Color` enumeration type, which is used to provide consistent color schemes throughout the application. It defines constants for the 147 named CSS colors (Çelik, Lilley, and Baron, 2011).

Word wrap Utility methods combine low-level PDFBox API drawing methods to introduce layout building blocks. For instance, `PDFUtil.renderMultilineText` utilizes `PDPageContentStream.drawString` to draw multi-line text boxes. The implemented word wrapping uses a *greedy* algorithm for a minimum number of lines (Sharnoff, Pierce, and Perl Contributors, 2009) based on string measurement. It does not support hyphenation but breaks lines at word boundary characters[53]. Textbox height calculation requires API methods to obtain font metrics of the given font type. Text alignments like `TOP_LEFT`, `MIDDLE_CENTER`, `BOTTOM_RIGHT`, ... are are provided by the `PDFUtil.Alignments` enumeration type.

Page break Elements like textboxes build the content of paragraph blocks which will therefore vary in size depending on the document data. Blocks are rendered one below the other, until a block will not fit any more. A page break is inserted to start a new page. This simple technique for page break calculation represents another greedy pattern.

Images To embed images, they are resampled and converted into JPEG format using `javax.imageio.ImageIO`. For a satisfying print quality, the resampling size (w_r, h_r) for an image of size (w, h) is calculated from the images' drawing space dimensions (w_{PDF}, h_{PDF}) to yield the given image resolution value r (e.g. $r = 140\text{dpi}$):

$$(w_r, h_r)\,[pixel] = \frac{r[\text{dpi}]}{72\text{dpi}} (w_{PDF}, h_{PDF})\,[drawing\,space\,unit] \left(\frac{1}{w}, \frac{1}{h}\right) [pixel]$$

[52]also known as PDF *user space*
[53]`' ', '\t', '/', '-'`

Mail merge Mail merge is the aggregation of multiple documents of same type into a single, large PDF output file. In simple terms, the logic generates the entire file at once by inserting a page break and resetting the document and template page counters, before starting with the next document. However, the implementation turned out more complex since it has to ensure loading of embedded images and TTF font files only once at the very beginning. Interference with resource streams from the imported template PDF was a related difficulty to solve.

i18n While translations of dynamic document content is covered by DAOs transformation methods, separate l10n bundles are provided for static labels in the document content. The format of rendered date and datetime labels can be configured using format pattern strings for `java.text.DateFormat`. Unless document types such as CV are configured to use a fixed language, additional PDF template files are required per supported language.

Generating PDF documents UI requests to generate PDF documents are processed synchronously by single service method invocations. They return a dedicated DTO per document type with a structure similar to those for Excel exports (figure 4.42). PDF data is put into a `documentData` `byte[]` field each.

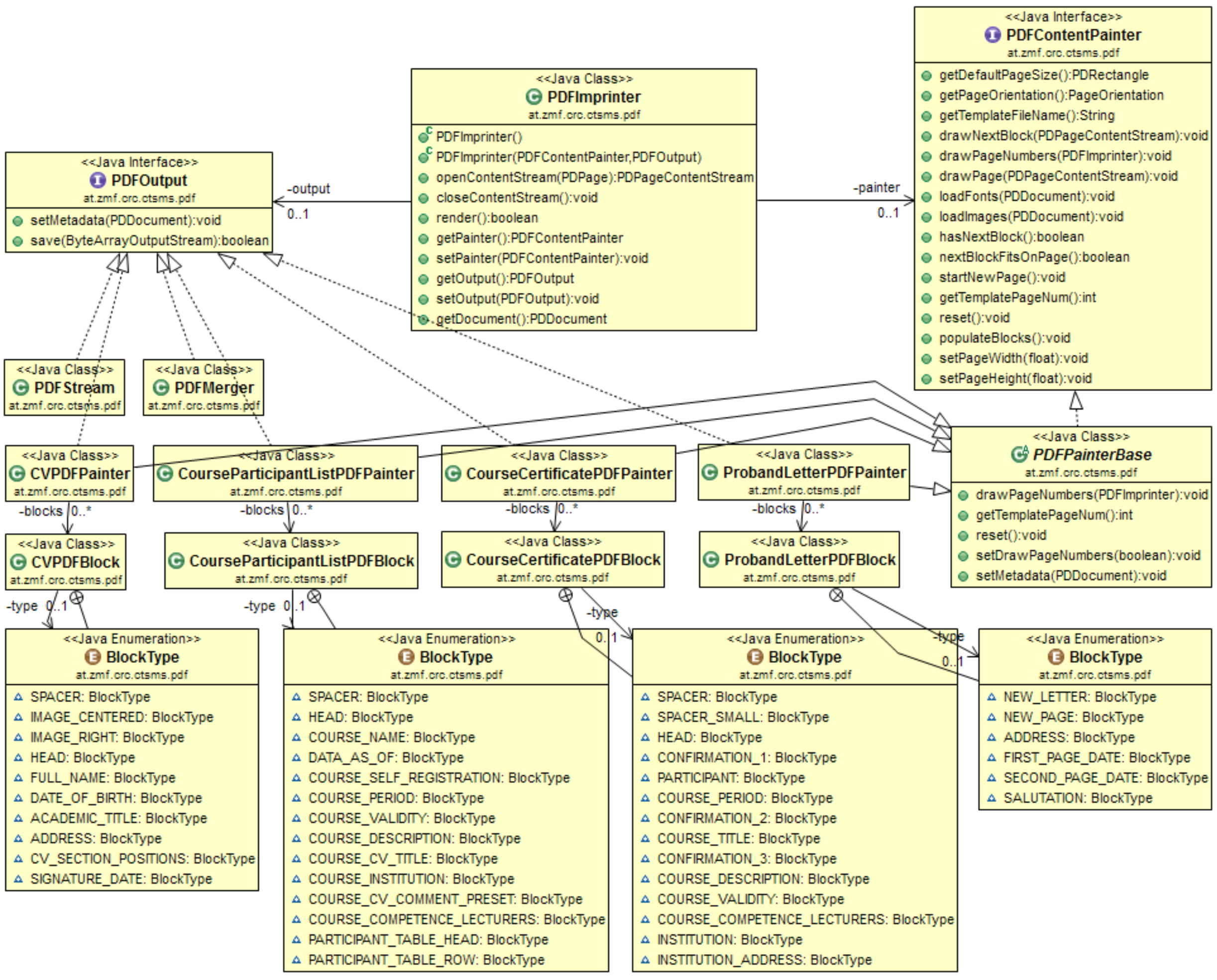


Figure 4.53.: In order to render documents, `*PDFPainter` classes are fed with required document data in form of misc. VOs items. VO fields are processed by `*PDFBlock` classes to draw page content block-wise.

According to the implementation of Excel exporters from section 4.3.2, the generic approach to create PDF contents shown in figure 4.53 is based on a composite pattern. The `PDFImprinter` base class allows combinations of both aspects - what content to write (interface `PDFContentPainter`) and how to save the generated output (interface `PDFOutput`). Output target methods are defined by the `PDFOutput` interface. `PDFOutput` implementations may comprise writing PDF document files to disk, as required for bulk exports to the file system in order to interface with a printing system queue or export to a corporate DMS in the future. At the moment, writing to the file system is provided by `PDFStream` only, which is used by the test data generator (section 4.10.1.3). Writing PDF output into a DTO `byte[]` field variable is implemented by the `*PDFPainter` implementation classes using a `ByteArrayOutputStream`. `PDFPainterBase` takes over common aspects like page numbering and populating PDF metadata. Apart from generating dynamic documents, `PDFMerger` allows aggregating multiple `File` records of uploaded PDF files into a single document using `org.apache.pdfbox.util.PDFMergerUtility`.

```
public class StaffServiceImpl extends StaffServiceBase {

    //This staff service method generates the CV document for a given Staff:
    @Override
    protected CvPDFVO handleRenderCvPDF(AuthenticationVO auth, Long staffId) throws Exception {
        //input validation:
        Staff staff = ServiceUtil.checkStaffId(staffId, this.getStaffDao());
        if (!staff.isPerson()) {
            throw L10nUtil.initServiceException(ServiceExceptionCodes.CV_POSITION_STAFF_NOT_PERSON);
        }
        //like all PDF painters, CVPDFPainter supports mail merge by passing a list of Staff instances, but
        //this time a single Staff is of interest only:
        ArrayList<StaffOutVO> staffVOs = new ArrayList<StaffOutVO>();
        staffVOs.add(staffDao.toStaffOutVO(staff));
        //CVPDFPainter creation and setup includes retrieval of various Staff detail records:
        CVPDFPainter painter = ServiceUtil.createCVPDFPainter(staffVOs,this.getStaffDao(),this.getCvSectionDao(),this.
            getCvPositionDao(),this.getCourseParticipationStatusEntryDao(),this.getStaffAddressDao());

        (new PDFImprinter(painter,painter)).render(); //creates the PDF and populates a CvPDFVO result
        //[...] log the PDF export by persisting a journal system message
        return painter.getPdfVO();
    }

    ...
```

Listing 4.20: This service method generates the CV PDF document for a given `Staff`. `PDFImprinter` allows combining a `PDFContentPainter` implementation with a supported output target, specified by a `PDFOutput` implementation.

For the use of a PDF painter (listing 4.20), miscellaneous VO objects will be prepared. As an example, the VOs related to an employee listed below represent the document data for a CV document:

- `StaffOutVO`
- `StaffAddressOutVO`
- `CVSectionOutVOs`
- `CVPositionOutVOs`
- `CourseParticipationStatusEntryOutVOs`

The `PDFContentPainter` instance prepares a list of `CVPDFBlock` instances (`PDFContentPainter.populateBlocks`) to arrange the document structure. `CVPDFBlock` contains all paragraph block implementations, each of which identified by a constant of the enclosed `BlockType` enumeration. Most block implementations will draw text elements containing formatted and localized VO field values. The painter iterates the list of blocks two times in order to produce correct page numbers. The first run performs no rendering but allows calculating the total number of document pages. The final run enables rendering and effectively produces output pages.

The programmatic drawing of PDF documents using low-level API methods is the fundamental concept promoted by PDFBox. The presented PDF export implementation uses a hybrid approach of basically hard-coded layouts, which supports paragraph floating, page breaks and tweaking of exposed parameters. An alternative solution is Apache Formatting Objects Processor (FOP) (Apache FOP Contributors, 2013), which introduces a precise document model, allowing to describe documents using XML. It implements a XML $\overset{XSLT}{\rightarrow}$ XSL-FO $\overset{FOP}{\rightarrow}$ PDF transformation flow to produce PDF output. Utilizing a dedicated type setting system[54] would provide an even more flexible way to generate documents, at the cost of an even lower performance.

4.5. Interceptors

A few *aspects* common to most service method implementations are hard-coded. These might be deeply interwoven with business logic code or are part of generated code. Their extraction or generalization will be less feasible for the scope of this project:

Input null checks: Code for basic checks for `null` and empty string arguments is generated by AndroMDA according to the modeled multiplicity of service method parameters and InVO fields.

Input validation: Each service method checks arguments and InVO fields referring to a record ID by trying to load the referenced entity instance. In general, sanity checks like this are a specific part of the business logic implemented by a service method.

DTO transformation: The transformation of entity instances from and into DTOs is done by explicit DAO method invocations. However, the DAO layer encapsulates aspects such as decryption, i18n and handling of graph cycles systematically.

System messages: Delete and other complex CRUD operations require multiple `JournalEntry` records to be created. The aspect of explicit `addJournalEntry` invocations is not extracted.

Notification messages: Many operations cause a user notification that is displayed on the UI start page and/or sent via email. Concerned service methods enqueue notifications by persisting `Notification` entity instances. The trigger logic is typically interwoven with the service method's business logic.

However, there are remaining aspects that affect the majority of service methods and qualify for an abstraction:

Authentication: Users identify themselves by a unique username and their chosen secret, the user password.

Authorisation: Each user account has privilege information assigned that defines which operation is granted or restricted for the authenticated user.

Logging: Field observation methods based on log analysis require to log events such as executed operations and raised exceptions.

It is common practice to extract these *cross cutting concerns* from above by means of the AOP (Andreas Holzinger, Brugger, and Slany, 2011). Spring AOP allows to wire a chain of *interceptors* for a method. Interceptors are specified by implementing different types of *advice* interfaces (Johnson et al., 2009, chapter 9.3 Advice API in Spring). A concrete advice implementation will encapsulate program logic that is to be spliced with the method invocation flow in order to minimize code duplication.

`BeforeMethodAdvice`: Program logic is inserted *before* the intercepted method's code by implementing

```
void before(Method m, Object[] args, Object target)
```

It is used to inspect or *instrument* arguments.

`AfterReturningAdvice`: Program logic is placed *after* the intercepted method's code returns by implementing

```
void afterReturning(Object returnValue, Method m, Object[] args, Object target)
```

The return value cannot be modified. It is used for cleanup purposes.

`ThrowsAdvice`: A hook for exceptions of a specified `ThrowableSubclass` type thrown during the intercepted method's execution is provided by

[54]e.g. VTL $\xrightarrow{Velocity}$ TeX $\xrightarrow{\LaTeX}$ PDF

```
void afterThrowing([Method, args, target], ThrowableSubclass)
```

The exception cannot be suppressed. It is used to log exceptions.

An interceptor class implements one or more advice interfaces and is registered for a type to engage methods. The application's Spring/Hibernate architecture skeleton employs interceptors for fundamental concerns in service and DAO layers off-the-shelf:

- Provide a Hibernate `Session` for the thread
- Demarcate (begin and commit or roll back) the database transaction

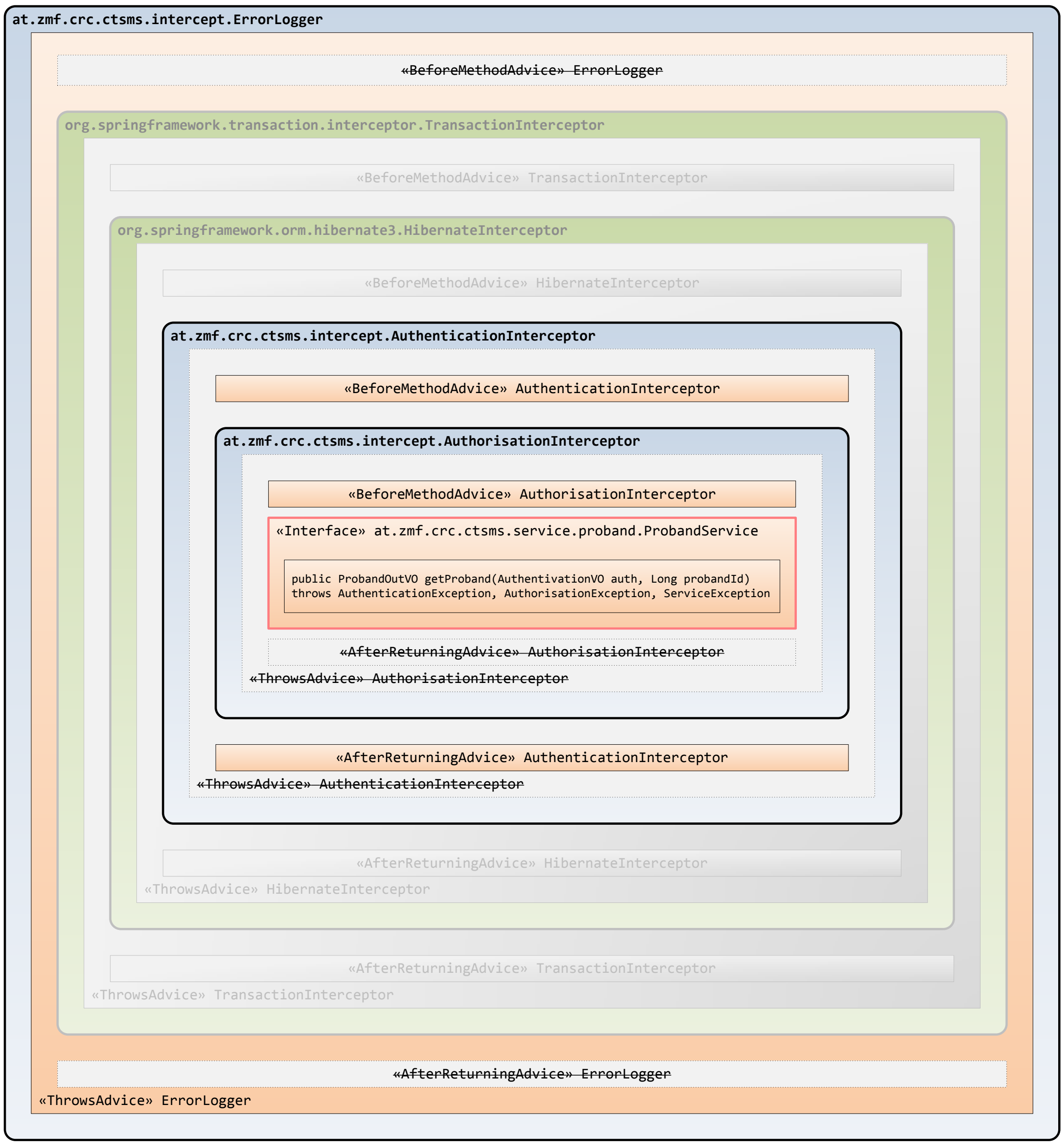

at.zmf.crc.ctsms.intercept.ErrorLogger
~~«BeforeMethodAdvice» ErrorLogger~~
org.springframework.transaction.interceptor.TransactionInterceptor
«BeforeMethodAdvice» TransactionInterceptor
org.springframework.orm.hibernate3.HibernateInterceptor
«BeforeMethodAdvice» HibernateInterceptor
at.zmf.crc.ctsms.intercept.AuthenticationInterceptor
«BeforeMethodAdvice» AuthenticationInterceptor
at.zmf.crc.ctsms.intercept.AuthorisationInterceptor
«BeforeMethodAdvice» AuthorisationInterceptor
«Interface» at.zmf.crc.ctsms.service.proband.ProbandService
public ProbandOutVO getProband(AuthentivationVO auth, Long probandId)
throws AuthenticationException, AuthorisationException, ServiceException
~~«AfterReturningAdvice» AuthorisationInterceptor~~
~~«ThrowsAdvice» AuthorisationInterceptor~~
«AfterReturningAdvice» AuthenticationInterceptor
~~«ThrowsAdvice» AuthenticationInterceptor~~
«AfterReturningAdvice» HibernateInterceptor
«ThrowsAdvice» HibernateInterceptor
«AfterReturningAdvice» TransactionInterceptor
«ThrowsAdvice» TransactionInterceptor
~~«AfterReturningAdvice» ErrorLogger~~
«ThrowsAdvice» ErrorLogger

Figure 4.54.: The "onion skin" structure shown depicts the chain of interceptor invocation. Each interceptor adds its corresponding logic and hands over to the next interceptor. The actual service method implementation represents most inner invocation, shown here for the `ProbandService.getProband` operation.

Three application-specific interceptors will be added to the existing chain of interceptors (figure 4.54):

`AuthenticationInterceptor`: The `AuthenticationInterceptor` described in section 4.5.1 covers the user authentication part of the access control logic and populates the *user context* object (listing 4.22) for subsequent access by service and DAO methods.

`AuthorisationInterceptor`: Authorisation of service method invocations is the second part of the access control logic. The flexible authorisation logic required for the application results in a complex `AuthorisationInterceptor`, which is described in detail in section 4.5.2.

`ErrorLogger`: Exceptions are logged into a separate database table (section 4.5.3).

Spring advices will utilize *dynamic proxies*, which require Java *interface* declarations. Table 4.7 gives a matrix of service interfaces with enabled interceptors each.

service	**description**	**authen-tication**	**author-isation**	**error logging**
`InventoryService`	inventory root and details entities CRUD and retrieval	✓	✓	✓
`StaffService`	person/organisation root and details entities CRUD and retrieval	✓	✓	✓
`CourseService`	course root and details entities CRUD and retrieval	✓	✓	✓
`TrialService`	trial root and details entities CRUD and retrieval	✓	✓	✓
`ProbandService`	proband root and details entities CRUD and retrieval	✓	✓	✓
`UserService`	user root and details entities CRUD and retrieval	✓	✓	✓
`InputFieldService`	input field root and details entities CRUD and retrieval	✓	✓	✓
`JournalService`	`JournalEntry` CRUD and retrieval	✓	✓	✓
`HyperlinkService`	`Hyperlink` CRUD and retrieval	✓	✓	✓
`FileService`	`File` CRUD and retrieval	✓	✓	✓
`SearchService`	`Criteria` and `Criterion` CRUD and retrieval, query execution	✓	✓	✓
`SelectionSetService`	retrieval operations for option items	✓	✗	✓
`ToolsService`	autocompletion suggestion lookup, transactional operations for the CLI tool, utilities (date calculations, session logon, image servlet)	✗	✗	✓

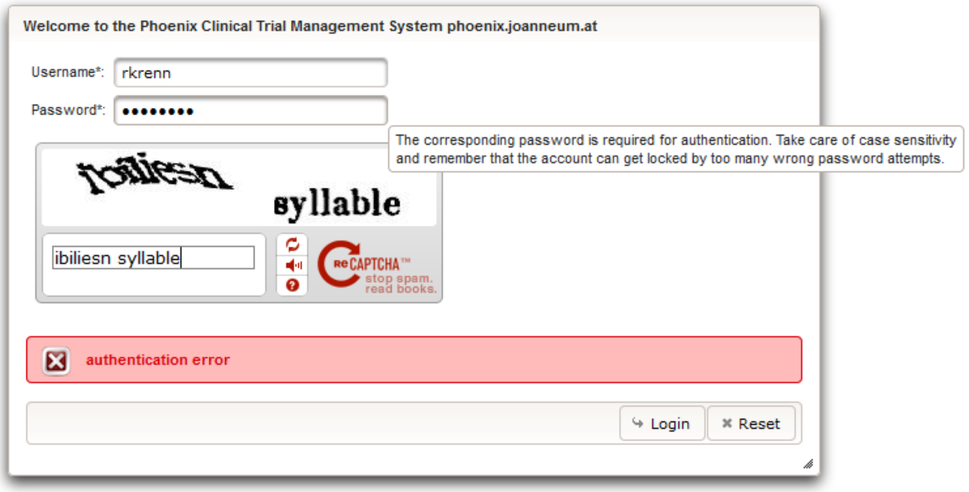
Table 4.7.: Overview of services and their enabled interceptors.

Both `AuthenticationInterceptor` and `AuthorisationInterceptor` execute within the database transaction, which is the ordinary case of interceptor usage considered by AndroMDA templates. AndroMDA directly supports wiring these interceptors by adding interceptor class names to service model elements using the `andromda_spring_service_interceptors` tagged value. The `ErrorLogger` is supposed to be able to intercept exceptions encountered in nested code and log it to a database table. Hence, it has be positioned "on top" of the chain in order to perform a separate database transaction outside the `TransactionInterceptor`, which is responsible to roll back transactions upon any exception occured within its scope. This can be achieved by extending the Spring DI configuration for each service in the `applicationContext.xml` template. A merge point is introduced to place configuration fragments for wiring the `ErrorLogger` interceptor on top of the interceptor chain compiled by AndroMDA.

4.5.1. Authentication

The execution of each service method (except `ToolsService` operations, table 4.7) requires the authentication of the requesting user. Credentials are made up of a username and password string. They can be passed by a parameter of type `AuthenticationVO`. By convention, it is the first parameter of service methods that require authentication.

Authenticating every single service method request could withdraw performance but keep services *stateless*. In contrast to maintaining a session ID table, this robust approach will keep a future API simple when publishing service methods as web service. *Session management* is completely moved into the web tier by utilizing a JSF `@SessionScope` managed bean (`SessionScopedBean`). Until it expires due to user inactivity, a `SessionScopedBean` instance holds the username and plaintext password provided by a user when logging in (figure 4.55) for subsequent service method requests.



Welcome to the Phoenix Clinical Trial Management System phoenix.joanneum.at

Username*: rkrenn

Password*: ••••••••

The corresponding password is required for authentication. Take care of case sensitivity and remember that the account can get locked by too many wrong password attempts.

ibiliesn syllable

authentication error

Figure 4.55.: A UI session is started when entering credentials into the login prompt. A CAPTCHA service like reCAPTCHA (Ahn et al., 2008) can be enabled to prevent automated logon attempts. `AuthenticationException` messages that describe the detailed reason for a failed authentication are opaque by default.

The basic authentication procedure looks up the `User` record for the provided username. Since `User.name` is a unique field, usernames form a natural key for the `User` entity. There is a unique lookup result if the username is valid or no result if there is no matching user account. The associated `Password` entity contains a hash value of the plaintext password string. If this value equals the hashed password credential, the user is successfully authenticated. `createPBEKey` (listing 4.9) is reused to generate secure `PBKDF2WithHmacSHA1` hash values. A particular exception type (`AuthenticationException`) is used to indicate failed authentications and communicate the detailed error.

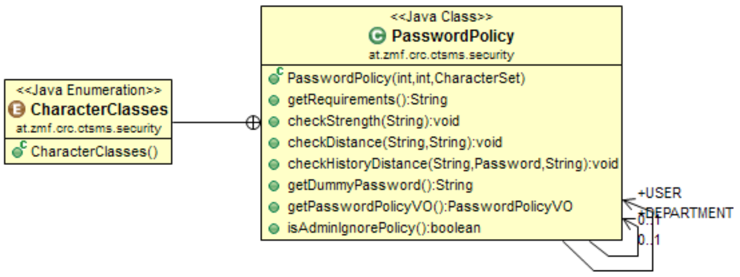

<<Java Class>>
PasswordPolicy
at.zmf.crc.ctsms.security
PasswordPolicy(int,int,CharacterSet)
getRequirements():String
checkStrength(String):void
checkDistance(String,String):void
checkHistoryDistance(String,Password,String):void
getDummyPassword():String
getPasswordPolicyVO():PasswordPolicyVO
isAdminIgnorePolicy():boolean
<<Java Enumeration>>
CharacterClasses
at.zmf.crc.ctsms.security
CharacterClasses()
+USER
+DEPARTMENT
0..1
0..1

Figure 4.56.: A `PasswordPolicy` object holds password requirement definitions and provides methods for testing passwords against them. Definitions of two singleton `PasswordPolicy` instances for user passwords (`USER`) and department passphrases (`DEPARTMENT`) are populated from the `.properties` file of the settings bundle.

Password safety is an important issue nowadays and a central topic of corporate security guidelines. Strong passwords are particularly important for the Phonix CTMS since the database column encryption scheme described in section 4.2.3 relies on PBE encrypted group keys. A password policy is represented by a self-contained `PasswordPolicy` class (figure 4.56) for checking password requirements:

- minimum length
- character ranges with minimum and maximum occurence limits each
- random generation of valid passwords
- human readable summary of password requirements
- minimum *edit distance*[55] to previous passwords

The latter will ensure that password renewals are sufficiently different from previous passwords. Passwords are too similar, if their edit distance falls below a given limit k. To support password edit distance checks, a history of recoverable plaintext passwords has to be stored per user. Therefore the `Password` entity additionally persists the encrypted password string aside the hash value. The `Password` entity's reflexive association maintains the order of previous `Password` instances (figure 4.57). Password strings are encrypted with the user department's passphrase using PBE. This way the edit distance check will work, even if a user sets the password for *another* user belonging to the same department.

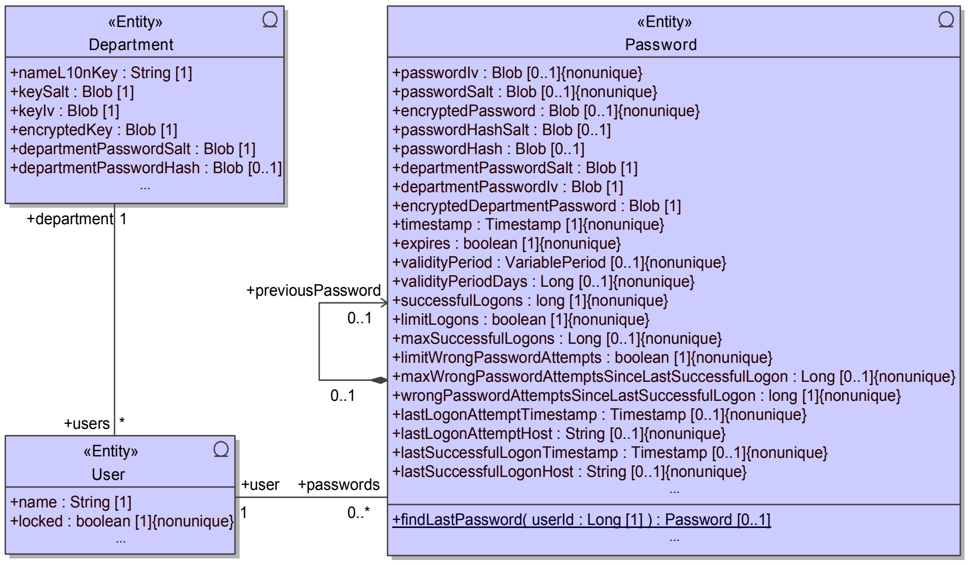

«Entity»
Department
+nameL10nKey : String [1]
+keySalt : Blob [1]
+keyIv : Blob [1]
+encryptedKey : Blob [1]
+departmentPasswordSalt : Blob [1]
+departmentPasswordHash : Blob [0..1]
...
+department 1
+users *
«Entity»
User
+name : String [1]
+locked : boolean [1]{nonunique}
...
+user 1
+passwords 0..*
+previousPassword 0..1
0..1
«Entity»
Password
+passwordIv : Blob [0..1]{nonunique}
+passwordSalt : Blob [0..1]{nonunique}
+encryptedPassword : Blob [0..1]{nonunique}
+passwordHashSalt : Blob [0..1]
+passwordHash : Blob [0..1]
+departmentPasswordSalt : Blob [1]
+departmentPasswordIv : Blob [1]
+encryptedDepartmentPassword : Blob [1]
+timestamp : Timestamp [1]{nonunique}
+expires : boolean [1]{nonunique}
+validityPeriod : VariablePeriod [0..1]{nonunique}
+validityPeriodDays : Long [0..1]{nonunique}
+successfulLogons : long [1]{nonunique}
+limitLogons : boolean [1]{nonunique}
+maxSuccessfulLogons : Long [0..1]{nonunique}
+limitWrongPasswordAttempts : boolean [1]{nonunique}
+maxWrongPasswordAttemptsSinceLastSuccessfulLogon : Long [0..1]{nonunique}
+wrongPasswordAttemptsSinceLastSuccessfulLogon : long [1]{nonunique}
+lastLogonAttemptTimestamp : Timestamp [0..1]{nonunique}
+lastLogonAttemptHost : String [0..1]{nonunique}
+lastSuccessfulLogonTimestamp : Timestamp [0..1]{nonunique}
+lastSuccessfulLogonHost : String [0..1]{nonunique}
...
+findLastPassword(userId : Long [1]) : Password [0..1]
...

Figure 4.57.: Each `User` entity is associated with a history of `Password` entities. The `Password` entity contains 1. the encrypted department passphrase string to decrypt the department key, 2. the encrypted password string for an edit distance check of a new password to be set and 3. the password string hash value for the actual comparison procedure.

[55]also known as *Levenshtein distance* (Levenshtein, 1966)

Cryptographic password creation and user authentication procedures can be summarized:

Create new user password (system setup)

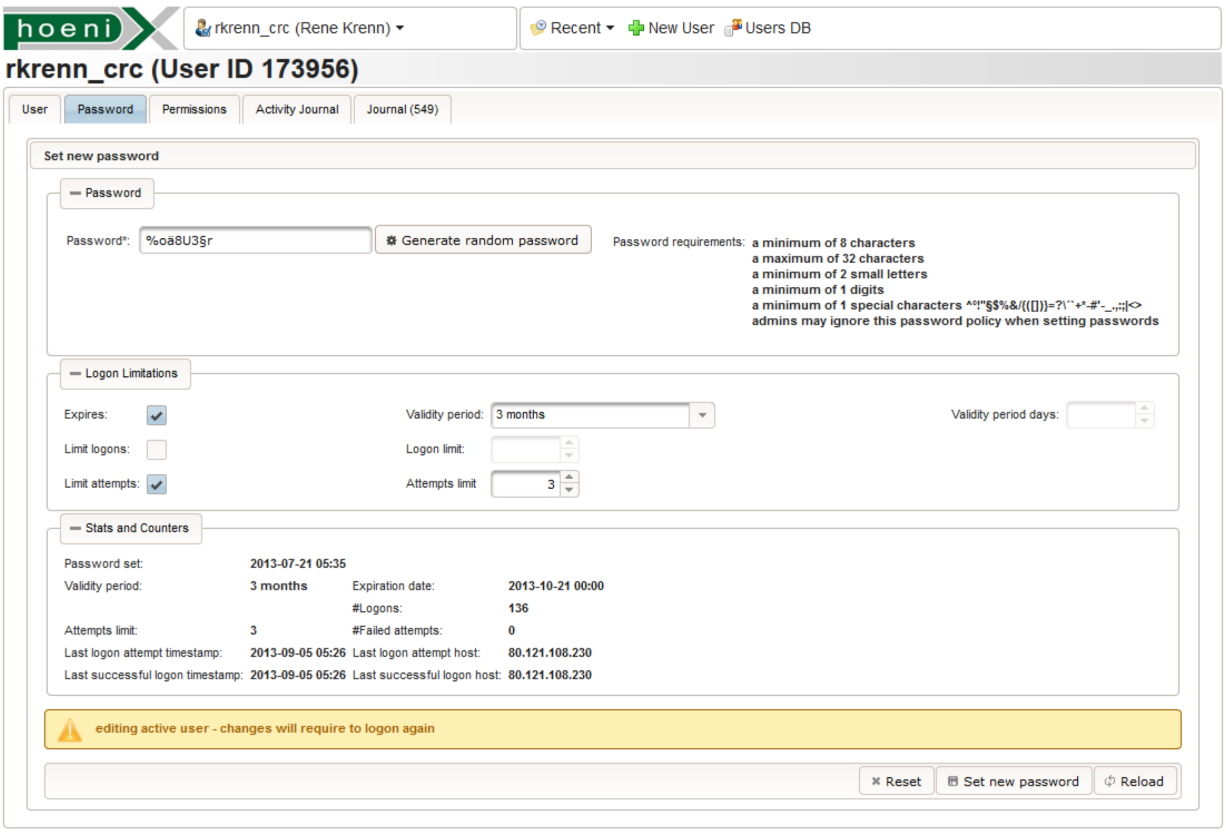
1. department passphrase P_{DP}, desired plaintext user password $P_{UP} \leftarrow$ console input
2. (password edit distance check)
3. user password hash salt $S_{H_{UP}}$ = `createSalt`
4. user password hash H_{UP} = `createPBEKey(`$S_{H_{UP}}$`, `P_{UP}`)`
5. user password encryption salt S_{UP} = `createSalt`
6. encrypted user password C_{UP}, user password encryption initialisation vector IV_{UP}: $(C_{UP}, IV_{UP}) = \mathbf{E}_{PBE:P_{DP},S_{UP}}(P_{UP})$
7. $S_{H_{UP}}, H_{UP}, S_{UP}, C_{UP}, IV_{UP} \rightarrow$ `Password` entity $\rightarrow$ persist

Password edit distance check (system setup)

1. P_{DP}, P_{UP}, user $\leftarrow$ console input
2. encrypted previous passwords $S_{UP_i}, C_{UP_i}, IV_{UP_i} \leftarrow$ load last n `Password` entities of the user
3. decrypted previous passwords $P_{UP_i} = \mathbf{D}_{PBE:P_{DP},S_{UP_i}}(C_{UP_i}, IV_{UP_i})$
4. edit distance minimum $d_n \leftarrow$ `for i = 1 .. n:` d_n `=min(`d_n`, getEditDistance(`P_{UP}`, `P_{UP_i}`)`
5. $d_n \overset{?}{<} k \rightarrow$ new password is too similar to old ones

Authenticate user (regular operation)

1. $S_{H_{UP}}, H_{UP} \leftarrow$ load `Password` $\leftarrow$ user credential parameter
2. credential user password $P^*_{UP} \leftarrow$ password credential parameter
3. credential user password hash H^*_{UP} = `createPBEKey(`$S_{H_{UP}}$`, `P^*_{UP}`)`
4. $H^*_{UP} \overset{?}{=} H_{UP} \rightarrow$ boolean return value



rkrenn_crc (User ID 173956)

Set new password

Password

Password*: %oä8U3§r

Password requirements: a minimum of 8 characters
a maximum of 32 characters
a minimum of 2 small letters
a minimum of 1 digits
a minimum of 1 special characters ^°!"§$%&/{([])}=?\´`+*-#'-_.,:;|<>
admins may ignore this password policy when setting passwords

Logon Limitations

Expires:	✓	Validity period:	3 months	Validity period days:	
Limit logons:		Logon limit:			
Limit attempts:	✓	Attempts limit	3		

Stats and Counters

Password set:	2013-07-21 05:35		
Validity period:	3 months	Expiration date:	2013-10-21 00:00
		#Logons:	136
Attempts limit:	3	#Failed attempts:	0
Last logon attempt timestamp:	2013-09-05 05:26	Last logon attempt host:	80.121.108.230
Last successful logon timestamp:	2013-09-05 05:26	Last successful logon host:	80.121.108.230

editing active user - changes will require to logon again

Figure 4.58.: An opened user's password tab UI allows priviledged users set (other) user's passwords. Whenever a password is set, a new `Password` instance is created.

Password policies are set up for both user passwords and department passphrases, where the latter have more strict requirements. Aside password requirements that concern the actual password string, restrictions regarding their validity can be compulsory. Validity limitations are implemented for user passwords only, since they are used for frequent web UI logins (figure 4.58):

Validity period: A validity period can be specified in terms of the `VariablePeriod` enumeration. It starts with the timestamp a password is created. When the validity period is over, a priviledged user needs to set a new password. A reminder notification informs the user in time.

Account disable: A user account login can be (temporary) disabled by setting the `User.locked` flag. Sinces users cannot be deleted if they already left record versioning footprints, they can therefore be locked permanently.

Limited number of successful logins: The password can only be used for a total of `maxSuccessfulLogons` login actions (`maxSuccessfulLogons=1` for one-time passwords). When the number of logins is exhausted, a priviledged user needs to set a new password. The UI shows a hint, if the number of remaining logins runs out.

Limited number of failed login attempts: If a wrong password is provided `maxWrongPasswordAttemptsSinceLastSuccessfulLogon` times in succession, a priviledged user needs to set a new password.

Both latter features are implemented using counter fields (figure 4.57). The `authenticate` method integrates both the authentication access control logic and optional counter increments in one single method (listing 4.21). Counters are only incremented when the user logs in and `ToolsService.logon` is invoked.

```java
//located in public final class at.zmf.crc.ctsms.util.ServiceUtil
//authenticate a user by provided credentials:
//logon=false: ordinary authentication only, logon=true: perform logon counter increment
public static Password authenticate(AuthenticationVO auth, boolean logon) throws AuthenticationException {
    UserContext userContext = getUserContext();
    Timestamp now = new Timestamp(System.currentTimeMillis()); //login timestamp
    //retrieve the User entitiy specified by the username credential:
    User user = null;
    try {
        user = (User) userDao.searchUniqueName(UserDao.TRANSFORM_NONE, auth.getUsername());
    } catch (Throwable t) { //problems fetching user entity record, e.g. empty or non-unique result:
        AuthenticationException e = L10nUtil.initAuthenticationException(AuthenticationExceptionCodes.UNKNOWN_USER, auth.getUsername());
        e.initCause(t);
        throw e;
    }
    userContext.setUser(user); //populate context object
    if (user == null) { //user not found
        throw L10nUtil.initAuthenticationException(AuthenticationExceptionCodes.UNKNOWN_USER, auth.getUsername());
    }
    if (user.isLocked()) { //account disable
        throw L10nUtil.initAuthenticationException(AuthenticationExceptionCodes.USER_LOCKED, auth.getUsername());
    }
    //retrieve the user's most recent Password entity:
    Password lastPassword = this.getPasswordDao().findLastPassword(user.getId());
    userContext.setLastPassword(lastPassword); //populate context object
    if (lastPassword == null) { //no password persisted yet
        throw L10nUtil.initAuthenticationException(AuthenticationExceptionCodes.NO_PASSWORD_SET, auth.getUsername());
    }
    if (logon) { //log the login attempt:
        lastPassword.setLastLogonAttemptHost(auth.getHost()); //log passed host IP address
        lastPassword.setLastLogonAttemptTimestamp(now); //log timestamp
    }
    //limit number of successful logins:
    if (lastPassword.isLimitLogons()   (lastPassword.getSuccessfulLogons() - (logon ? 0 : 1)) >= lastPassword.getMaxSuccessfulLogons()) {
        throw L10nUtil.initAuthenticationException(AuthenticationExceptionCodes.SUCCESSFUL_LOGON_LIMIT_EXCEEDED, lastPassword.getSuccessfulLogons(),
                lastPassword.getMaxSuccessfulLogons());
    }
    //limit number of failed login attempts:
    if (lastPassword.isLimitWrongPasswordAttempts()   (lastPassword.getWrongPasswordAttemptsSinceLastSuccessfulLogon() - (logon ? 0 : 1)) >= lastPassword.
          getMaxWrongPasswordAttemptsSinceLastSuccessfulLogon()) {
        throw L10nUtil.initAuthenticationException(AuthenticationExceptionCodes.WRONG_PASSWORD_ATTEMPT_LIMIT_EXCEEDED, lastPassword.
                getWrongPasswordAttemptsSinceLastSuccessfulLogon(),lastPassword.getMaxWrongPasswordAttemptsSinceLastSuccessfulLogon());
    }
    if (lastPassword.isExpires()) { //check validity period:
        Date expiration = DateCalc.addInterval(DateCalc.getStartOfDay(lastPassword.getTimestamp()), lastPassword.getValidityPeriod(), lastPassword.
                getValidityPeriodDays());
        if ((new Date()).compareTo(expiration) >= 0) {
            AuthenticationException exception = L10nUtil.initAuthenticationException(AuthenticationExceptionCodes.PASSWORD_EXPIRED);
            exception.setData(expiration);
            throw exception;
        }
    }
    boolean passwordValid = false;
    try { //compare password string hash values:
        passwordValid = Arrays.equals(CryptoUtil.createPBEKey(lastPassword.getPasswordHashSalt(), plainPassword).getEncoded(),lastPassword.
                getPasswordHash());
    } catch (Exception e) { //e.g. empty password
        throw L10nUtil.initAuthenticationException(e.getMessage());
    }
    if (!passwordValid) { //password string hash values do not match
        if (logon) { //increment wrongPasswordAttemptsSinceLastSuccessfulLogon counter
            lastPassword.setWrongPasswordAttemptsSinceLastSuccessfulLogon(lastPassword.getWrongPasswordAttemptsSinceLastSuccessfulLogon() + 1);
        }
        throw L10nUtil.initAuthenticationException(AuthenticationExceptionCodes.WRONG_PASSWORD, auth.getPassword());
    }
    //authentication was successful ...
    userContext.setPlainPassword(auth.getPassword()); //populate context object
    if (logon) {
        lastPassword.setSuccessfulLogons(lastPassword.getSuccessfulLogons() + 1); //increment successfulLogons counter
        lastPassword.setWrongPasswordAttemptsSinceLastSuccessfulLogon(0); //reset wrongPasswordAttemptsSinceLastSuccessfulLogon counter
        lastPassword.setLastSuccessfulLogonHost(auth.getHost()); //log passed host IP address
        lastPassword.setLastSuccessfulLogonTimestamp(now); //log login timestamp
    }
    return lastPassword;
}

//located in public final class at.zmf.crc.ctsms.util.CoreUtil
//retrieve the UserContext instance for the current thread from a ThreadLocal store:
public static UserContext getUserContext() {
    //PrincipalStore is a static artifact generated by AndroMDA:
    UserContext userContext = (UserContext) PrincipalStore.get();
    if (userContext == null) { //create user context object upon first-time access:
        userContext = new UserContext();
        PrincipalStore.set(userContext);
    }
    return userContext;
}
//clear the UserContext:
public static void clearUserContext() {
    PrincipalStore.set(null);
}
```

Listing 4.21: The `authenticate` method integrates authentication control logic and logon counter increments. It creates and populates the `UserContext` (listing 4.22) for the use by imminent service and DAO method code.

The `authenticate` method will essentially look up the user name, compares password hashes and performs remaining checks regarding password validity in a meaningful order. This could also be the place to wire authentication against an external authentication service such as Lightweight Directory Access Protocol (LDAP). When `authenticate` completes without throwing an `AuthenticationException`, the authentication was successfull and the `UserContext` (listing 4.22) is set up for the running thread. The `UserContext` instance is stored in a `ThreadLocal<Principal>` singleton variable, which is prepared by default as a part of the service layer infrastructure. `ThreadLocal` is a synchronized map, which maintains an instance for each active thread accessing the `ThreadLocal` field. `UserContext` provides accessor methods to obtain user dependent settings and cryptographic keys (listing 4.22). They can be recapitulated at this point:

Credentials: The `User` and its associated recent `Password` are populated by the `authenticate` logic. Both entity instances allow deriving all subsequent `UserContext` fields on demand. Furthermore, the plaintext user password credential is stored for deferred decryption logic in remaining accessor methods.

User's i18n settings: The UI language and time zone are user specific settings. DAO transformation methods will access both in order to translate and format VO field values.

User department's plaintext passphrase: To obtain the plaintext department passphrase, it is decrypted using the plaintext user password. It is required to decrypt the department key and for the use case of setting passwords of foreign users. This implies encryption and decryption of these users' old passwords in order to support edit distance checks.

User department's key: The user department's key is required to encrypt and decrypt secret entity field values. To obtain the department key, it is decrypted using the user department's plaintext passphrase.

User's secret key: The user's secret RSA key is required when creating digital signatures. To obtain the secret key, it is decrypted using the plaintext user password.

To increase performance, decryption results are cached and performed on demand when accessing a `UserContext` getter only.

```
public class UserContext implements Principal {
    //context information that will be populated during authentication:
    private User user;
    private Password lastPassword;
    private String plainPassword;
    //derived context fields:
    private Locale locale; private boolean isLocaleSet;
    private TimeZone timeZone; private boolean isTimeZoneSet;
    private String plainDepartmentPassword; private boolean isPlainDepartmentPasswordSet;
    private SecretKey departmentKey; private boolean isDepartmentKeySet;
    private PrivateKey privateKey; private boolean isPrivateKeySet;
    private PublicKey publicKey; private boolean isPublicKeySet;
    //constructors:
    public UserContext() { reset(); }
    public UserContext(User user, Password lastPassword, String plainPassword) {
        reset();
        this.user = user; this.lastPassword = lastPassword; this.plainPassword = plainPassword;
    }
    @Override
    public String getName() { return user == null ? null : user.getName(); } //mandatory override of Principal

    public User getUser() { return user; } //User getter
    public void setUser(User user) { //User setter
        reset(); this.user = user;
    }
    public Password getLastPassword() { return lastPassword; } //Password getter
    public void setLastPassword(Password lastPassword) { //Password setter
        resetDecrypted(); this.lastPassword = lastPassword;
    }
    public void setPlainPassword(String plainPassword) {  //plaintext password string credential (setter only)
        resetDecrypted(); this.plainPassword = plainPassword;
    }

    public String getPlainDepartmentPassword() throws Exception { //chached plaintext department passphrase getter
        if (!isPlainDepartmentPasswordSet  lastPassword != null) {
            plainDepartmentPassword = CryptoUtil.decryptDepartmentPassword(lastPassword, plainPassword);
            isPlainDepartmentPasswordSet = true;
        }
        return plainDepartmentPassword;
    }
    public SecretKey getDepartmentKey() throws Exception { //chached department key getter
        if (!isDepartmentKeySet  user != null) {
            departmentKey = new SecretKeySpec(CryptoUtil.decryptDepartmentKey(user.getDepartment(), getPlainDepartmentPassword
                 ()), CryptoUtil.SYMMETRIC_ALGORITHM);
            isDepartmentKeySet = true;
        }
        return departmentKey;
    }
    public PrivateKey getPrivateKey() throws Exception { //chached secret key getter
        if (!isPrivateKeySet  user != null) {
            KeyPair keyPair = user.getKeyPair();
            privateKey = KeyFactory.getInstance(CryptoUtil.ASYMMETRIC_ALGORITHM).generatePrivate(new PKCS8EncodedKeySpec(
                 CryptoUtil.decrypt(keyPair.getPrivateKeyIv(), keyPair.getPrivateKeySalt(), getPlainDepartmentPassword(),
                 keyPair.getEncryptedPrivateKey())));
            isPrivateKeySet = true;
        }
        return privateKey;
    }
    public Locale getLocale() { //cached user language locale setting getter
        if (!isLocaleSet  user != null) {
            locale = CommonUtil.localeFromString(user.getLocale()); isLocaleSet = true;
        }
        return locale;
    }
    public TimeZone getTimeZone() { //cached user time zone setting getter
        if (!isTimeZoneSet  user != null) {
            timeZone = CommonUtil.timeZoneFromString(user.getTimeZone()); isTimeZoneSet = true;
        }
        return timeZone;
    }

    private void resetDecrypted() { ... } //reset chached decrypted fields
    public void reset() { ... } //reset entire state
}
```

Listing 4.22: The `UserContext` provides relevant user settings and cryptographic keys on demand. Costly decryption operations are cached to increase performance when program logic accesses its fields frequently.

By enabling the `AuthenticationInterceptor`, the `authenticate` method is invoked "before" a service method executes. The interceptor implements `BeforeMethodAdvice` to perform authentication and the user context setup. It additionally implements `AfterReturningAdvice` to clear the user context when the service implementation code completes (listing 4.23).

```
public class AuthenticationInterceptor implements MethodBeforeAdvice, AfterReturningAdvice {

    public AuthenticationInterceptor() { }

    //find the first argument of type AuthenticationVO:
    private static AuthenticationVO getAuthentication(Object[] args) {
        if (args != null) {
            for (int i = 0; i < args.length; i++) { //it will always be the first parameter
                if (args[i] instanceof AuthenticationVO) {
                    return (AuthenticationVO) args[i];
                }
            }
        }
        return null;
    }
    //invoke the authentication access control logic:
    public void before(Method method, Object[] args, Object object) throws Throwable {
        CoreUtil.authenticate(getAuthentication(args), false); //do not increment password counters
    }
    //cleanup to avoid to expose the user context's decrypted fields to remaining interceptors:
    public void afterReturning(Object returnValue, Method method, Object[] args, Object target) throws Throwable {
        CoreUtil.clearUserContext();
    }
}
```

Listing 4.23: The `AuthenticationInterceptor` wires the authentication access control logic and prepares the user context. The `UserContext` object is available to the nested execution path only.

4.5.2. Authorisation

When a user is authenticated, the identity was successfully verified and access is granted in the first place. Now the question of granular access to available resources of a user u from the set of users U arises. Authorisation is the decision process of granting u access to a requested service method s from available service methods S. It can be viewed as table lookup, whereby missing entries map to $false$ by default:

$$(U, S) \rightarrow \{true, false\}$$

A user u has the *privilege*[56] to execute s, if the lookup is positive. For Role-Based Access Control (RBAC), a subset of privileges represents a *role*. Some systems get along with the trivial case of two roles, e.g. "admin" and "user". Roles are *exclusive* if they have no privileges in common. Zero or more roles may be assigned to user accounts to define their privileges. Managing role assignment per user is considered more comfortable than managing discrete privileges, but less flexible due to a coarse granularity. The Phoenix CTMS use cases lack of defined actors, which could be used to denominate roles otherwise. It is therefore preferred to manage privileges more fine-grained. While roles are considered as larger sets of privileges, the Phoenix CTMS introduces permission *profiles* as happy medium. A *profile group* corresponds to a functional part of the application, whose access is meant to be isolable by means of authorisation. Profiles represent meaningful granular roles and are exclusive *among* profile groups but not exclusive *within* a profile group:

- trial profile group:
  - create/edit/delete trial root and detail records
  - create/edit/delete trial details only
  - view trials only
- proband profile group:
  - create/edit/delete proband root and detail records
  - create/edit/delete proband details only
  - view probands only

[56]synonymous with *permission* and *access right*

Regulatory guidelines claim the ability to review concrete access rights of a particular user for any given point of time (*Guidance for Industry - Computerized Systems Used in Clinical Investigations* 2007; *Good Manufacturing Practice - Medicinal Products for Human and Veterinary Use Annex 11: Computerised Systems* 2011). This is provided by the system messages recorded in a user's journal, which are created upon CRUD operations in the course of profile assignments. The journal would be less transparent in the case of coarse roles.

For the Phoenix CTMS, requirements impose a further breakdown of a single service method resource s. Additional conditions C must evaluate to *true* for the invoked service method's *arguments* A.

$$(U, S, C, A) \rightarrow \{true, false\}$$

Argument inspection is required to test conditions of method argument values. The conditions check relationships of arguments and the authenticated user or identity. Three relevant types of conditions with increasing restriction level are distinguished:

Separation of departments Each root entity is associated with a `Department` entity. A department represents a compartment for associated root entity instances. In order to support a multi-client scenario, exclusive access is required. While `Proband` entities cannot be read across departments due to encryption, the authorisation logic has to cover remaining modules and is responsible to check if linked root entity instances belong to the same department of the executing user. An example of the argument condition for a trial detail update operation is given in listing 4.24.

```java
boolean condition = getUserContext().getUser().getDepartment().equals(visitDao.load(visitIn.getId()).getTrial()
    .getDepartment());
```

Listing 4.24: Sample argument condition of privileges for separating departments.

It is desired to have a "super user" profile per system module, whose privileges suppress such checks in order to access items of any department.

Entity hierarchies Even more fine-grained access than provided by department separation is desired for root entity tree hierarchies from section 4.1.1.2. Actually, this is meaningful for `Staff` trees only. An executing `User` can be restricted to modify its `identity Staff`'s child instances only. The argument condition for a staff detail update operation is given in listing 4.25.

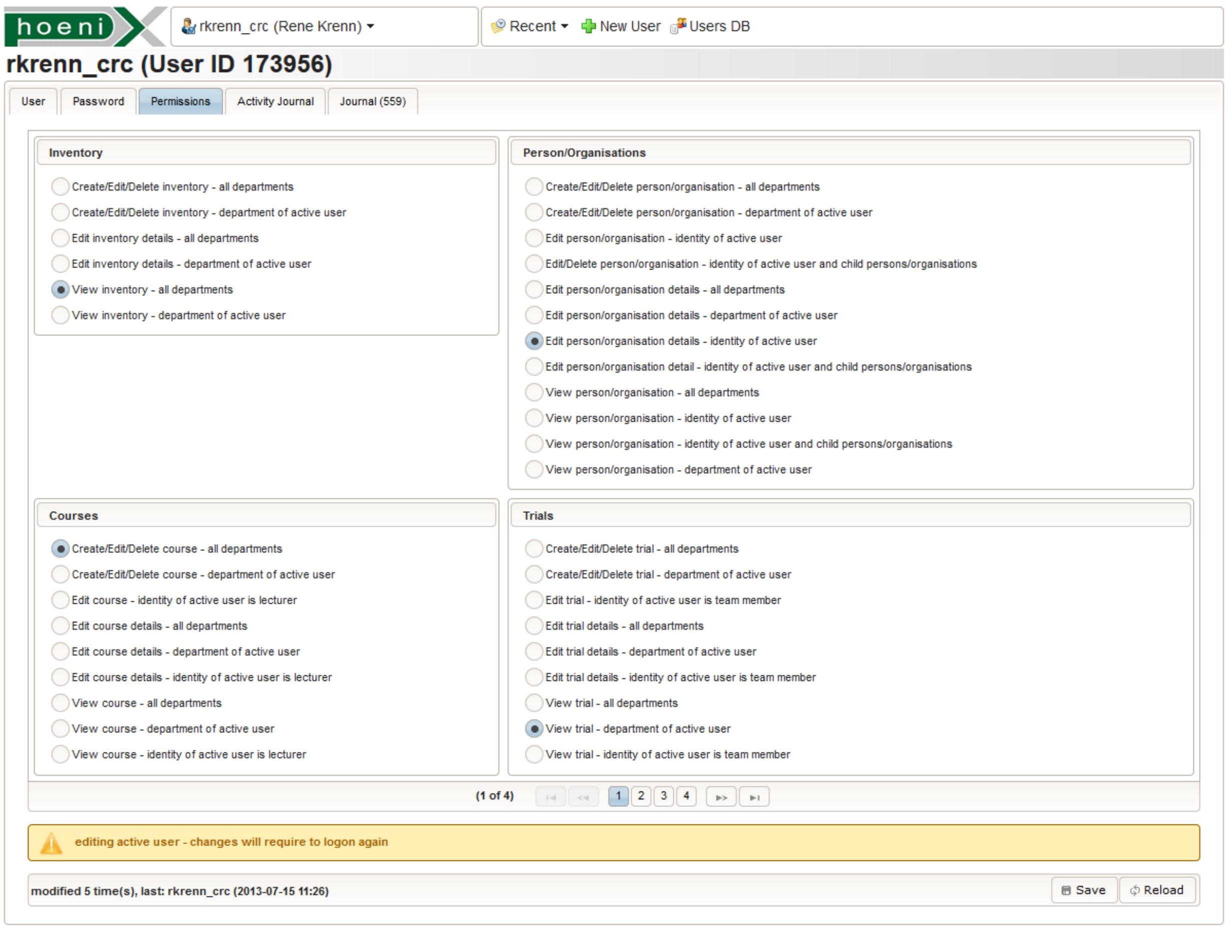
```java
boolean condition = isChild(getUserContext().getUser().getIdentity(), staffDetailDao.load(detailIn.getId()).getStaff()};

private static boolean isChild(Staff identity, Staff staff) {
    if (identity.equals(staff)) { return true; }
    for (Iterator<Staff> it = identity.getChildren().iterator(); i.hasNext();) {
        if (isChild(it.next(),staff)) { return true; }
    }
    return false;
}
```

Listing 4.25: Sample argument condition of privileges for restricting to `Staff` child items.

ACLs The most detailed level of access is to restrict access on a particular root entity instance. As an example, authorisation to modify a trial and its detail records is desired to be grantable on a per-user level. A trial's `TeamMember` detail records list related `Staff` instances. The `TeamMember.access` flag is used to actual grant access to a team member. The argument condition for a trial detail update operation is given in listing 4.26.

```
boolean condition = isTeamMember(getUserContext().getUser().getIdentity(), visitDao.load(visitIn.getId()).getTrial());

private static boolean isTeamMember(Staff identity, Trial trial) {
    for (Iterator<TeamMember> it = trial.getMembers().iterator(); i.hasNext();) {
        TeamMember member = it.next();
        if (member.isAccess()  identity.equals(member.getStaff())) { return true; }
    }
    return false;
}
```

Listing 4.26: Sample privilege argument condition for ACL logic.



rkrenn_crc (User ID 173956)

User | Password | Permissions | Activity Journal | Journal (559)

Inventory

- Create/Edit/Delete inventory - all departments
- Create/Edit/Delete inventory - department of active user
- Edit inventory details - all departments
- Edit inventory details - department of active user
- [✓ View inventory - all departments]
- View inventory - department of active user

Person/Organisations

- Create/Edit/Delete person/organisation - all departments
- Create/Edit/Delete person/organisation - department of active user
- Edit person/organisation - identity of active user
- Edit/Delete person/organisation - identity of active user and child persons/organisations
- Edit person/organisation details - all departments
- Edit person/organisation details - department of active user
- [✓ Edit person/organisation details - identity of active user]
- Edit person/organisation detail - identity of active user and child persons/organisations
- View person/organisation - all departments
- View person/organisation - identity of active user
- View person/organisation - identity of active user and child persons/organisations
- View person/organisation - department of active user

Courses

- [✓ Create/Edit/Delete course - all departments]
- Create/Edit/Delete course - department of active user
- Edit course - identity of active user is lecturer
- Edit course details - all departments
- Edit course details - department of active user
- Edit course details - identity of active user is lecturer
- View course - all departments
- View course - department of active user
- View course - identity of active user is lecturer

Trials

- Create/Edit/Delete trial - all departments
- Create/Edit/Delete trial - department of active user
- Edit trial - identity of active user is team member
- Edit trial details - all departments
- Edit trial details - department of active user
- Edit trial details - identity of active user is team member
- View trial - all departments
- [✓ View trial - department of active user]
- View trial - identity of active user is team member

(1 of 4) 1 2 3 4

editing active user - changes will require to logon again

modified 5 time(s), last: rkrenn_crc (2013-07-15 11:26)

Save | Reload

Figure 4.59.: An opened user's permission tab UI allows priviledged users set their or other user's permission profiles. When saving changes, the `UserPermissionProfile.active` flags are toggled. Journal system messages log granted and revoked profile assignments only to provide a clean privilege history at a glance.

The final design of permission profiles shown in table 4.8 considers argument conditions to implement department and global access levels. Most notably, there is no generic profile to block access to a particular system module absolutely, which is possible for profiles with ACL conditions only. Basically, minimal privileges allow to view items at least. The reason for this profile setup lies in the interdependency of modules described in section 4.1.1.1.[57] However, opening arbitrary items of a system module can be supressed by means of UI navigation however, since privileges for database queries can be granted

[57]Many tab screens of the entity UI view of a granted system module are unusable if retrieval operations of an interdependent module would be restricted totally.

separately. A user with the PROBAND_NO_SEARCH profile will therefore not be able to list any probands to open records subsequentially.

profile group	`PermissionProfile`	**profile name**
Inventory	`INVENTORY_MASTER_ALL_DEPARTMENTS`	Create/Edit/Delete inventory - all departments
	`INVENTORY_MASTER_USER_DEPARTMENT`	Create/Edit/Delete inventory - department of active user
	`INVENTORY_DETAIL_ALL_DEPARTMENTS`	Edit inventory details - all departments
	`INVENTORY_DETAIL_USER_DEPARTMENT`	Edit inventory details - department of active user
	`INVENTORY_VIEW_ALL_DEPARTMENTS`	View inventory - all departments
	`INVENTORY_VIEW_USER_DEPARTMENT`	View inventory - department of active user
Person/Organisations	`STAFF_MASTER_ALL_DEPARTMENTS`	Create/Edit/Delete person/organisation - all departments
	`STAFF_MASTER_USER_DEPARTMENT`	Create/Edit/Delete person/organisation - department of active user
	`STAFF_MASTER_IDENTITY`	Edit person/organisation - identity of active user
	`STAFF_MASTER_IDENTITY_CHILD`	Edit/Delete person/organisation - identity of active user and child persons/organisations
	`STAFF_DETAIL_ALL_DEPARTMENTS`	Edit person/organisation details - all departments
	`STAFF_DETAIL_USER_DEPARTMENT`	Edit person/organisation details - department of active user
	`STAFF_DETAIL_IDENTITY`	Edit person/organisation details - identity of active user
	`STAFF_DETAIL_IDENTITY_CHILD`	Edit person/organisation detail - identity of active user and child persons/organisations
	`STAFF_VIEW_ALL_DEPARTMENTS`	View person/organisation - all departments
	`STAFF_VIEW_USER_DEPARTMENT`	View person/organisation - department of active user
	`STAFF_VIEW_IDENTITY`	View person/organisation - identity of active user
	`STAFF_VIEW_IDENTITY_CHILD`	View person/organisation - identity of active user and child persons/organisations
Courses	`COURSE_MASTER_ALL_DEPARTMENTS`	Create/Edit/Delete course - all departments
	`COURSE_MASTER_USER_DEPARTMENT`	Create/Edit/Delete course - department of active user
	`COURSE_MASTER_LECTURER`	Edit course - identity of active user is lecturer
	`COURSE_DETAIL_ALL_DEPARTMENTS`	Edit course details - all departments
	`COURSE_DETAIL_USER_DEPARTMENT`	Edit course details - department of active user
	`COURSE_DETAIL_LECTURER`	Edit course details - identity of active user is lecturer
	`COURSE_VIEW_ALL_DEPARTMENTS`	View course - all departments
	`COURSE_VIEW_USER_DEPARTMENT`	View course - department of active user
	`COURSE_VIEW_LECTURER`	View course - identity of active user is lecturer
Trials	`TRIAL_MASTER_ALL_DEPARTMENTS`	Create/Edit/Delete trial - all departments
	`TRIAL_MASTER_USER_DEPARTMENT`	Create/Edit/Delete trial - department of active user
	`TRIAL_MASTER_TEAM_MEMBER`	Edit trial - identity of active user is team member
	`TRIAL_DETAIL_ALL_DEPARTMENTS`	Edit trial details - all departments
	`TRIAL_DETAIL_USER_DEPARTMENT`	Edit trial details - department of active user
	`TRIAL_DETAIL_TEAM_MEMBER`	Edit trial details - identity of active user is team member
	`TRIAL_VIEW_ALL_DEPARTMENTS`	View trial - all departments
	`TRIAL_VIEW_USER_DEPARTMENT`	View trial - department of active user
	`TRIAL_VIEW_TEAM_MEMBER`	View trial - identity of active user is team member
Probands	`PROBAND_MASTER_ALL_DEPARTMENTS`	Create/Edit/Delete proband - all departments
	`PROBAND_MASTER_USER_DEPARTMENT`	Create/Edit/Delete proband - department of active user
	`PROBAND_DETAIL_ALL_DEPARTMENTS`	Edit proband details - all departments
	`PROBAND_DETAIL_USER_DEPARTMENT`	Edit proband details - department of active user
	`PROBAND_VIEW_ALL_DEPARTMENTS`	View proband - all departments
	`PROBAND_VIEW_USER_DEPARTMENT`	View proband - department of active user
User	`USER_ALL_DEPARTMENTS`	Create/Edit/Delete user - all departments
	`USER_USER_DEPARTMENT`	Create/Edit/Delete user - department of active user
	`USER_ACTIVE_USER`	Edit user details - active user
Input fields	`INPUT_FIELD_MASTER`	Create/Edit/Delete input field
	`INPUT_FIELD_VIEW`	View input field
Inventory search	`INVENTORY_MASTER_SEARCH`	Create/Edit/Delete/Execute inventory search
	`INVENTORY_SAVED_SEARCH`	Execute stored inventory search
	`INVENTORY_NO_SEARCH`	Inventory search - no access
Person/Organisation search	`STAFF_MASTER_SEARCH`	Create/Edit/Delete/Execute person/organisation search
	`STAFF_SAVED_SEARCH`	Execute stored person/organisation search
	`STAFF_NO_SEARCH`	Person/Organisation search - no access
Course search	`COURSE_MASTER_SEARCH`	Create/Edit/Delete/Execute course search
	`COURSE_SAVED_SEARCH`	Execute stored course search
	`COURSE_NO_SEARCH`	Course search - no access
Trial search	`TRIAL_MASTER_SEARCH`	Create/Edit/Delete/Execute trial search
	`TRIAL_SAVED_SEARCH`	Execute stored trial search
	`TRIAL_NO_SEARCH`	Trial search - no access
Proband search	`PROBAND_MASTER_SEARCH`	Create/Edit/Delete/Execute proband search
	`PROBAND_SAVED_SEARCH`	Execute stored proband search
	`PROBAND_NO_SEARCH`	Proband search - no access
User search	`USER_MASTER_SEARCH`	Create/Edit/Delete/Execute user search
	`USER_SAVED_SEARCH`	Execute stored user search
	`USER_NO_SEARCH`	User search - no access
Input field search	`INPUT_FIELD_MASTER_SEARCH`	Create/Edit/Delete/Execute input field search
	`INPUT_FIELD_SAVED_SEARCH`	Execute stored input field search
	`INPUT_FIELD_NO_SEARCH`	Input field search - no access

Table 4.8.: The complete list of available permission profiles with profile groups. For each user, a permission profile can be chosen per profile group using the UI tab in figure 4.59.

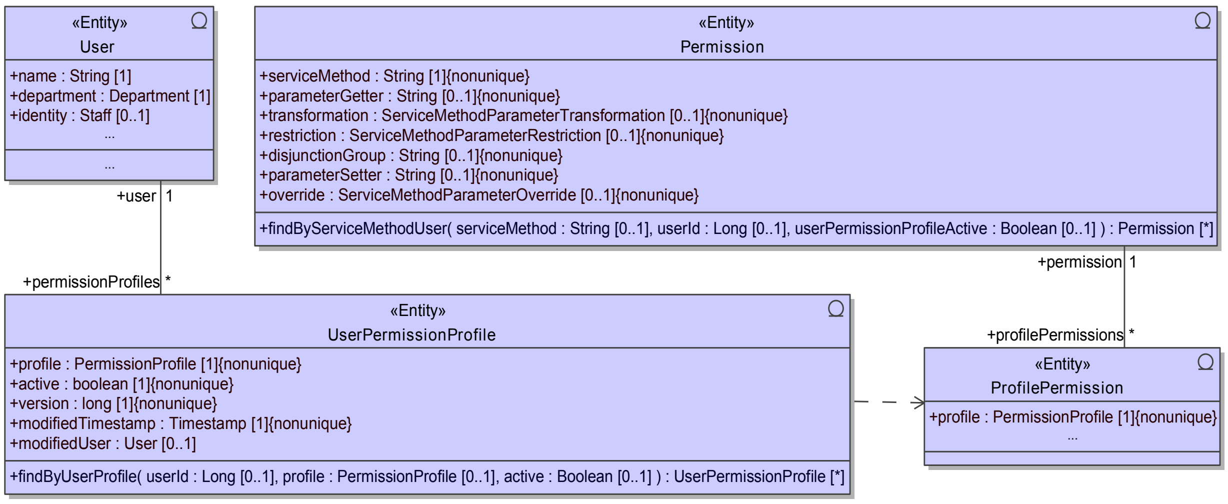

«Entity»
User
+name : String [1]
+department : Department [1]
+identity : Staff [0..1]
...
...
+user 1
+permissionProfiles *
«Entity»
Permission
+serviceMethod : String [1]{nonunique}
+parameterGetter : String [0..1]{nonunique}
+transformation : ServiceMethodParameterTransformation [0..1]{nonunique}
+restriction : ServiceMethodParameterRestriction [0..1]{nonunique}
+disjunctionGroup : String [0..1]{nonunique}
+parameterSetter : String [0..1]{nonunique}
+override : ServiceMethodParameterOverride [0..1]{nonunique}
+findByServiceMethodUser(serviceMethod : String [0..1], userId : Long [0..1], userPermissionProfileActive : Boolean [0..1]) : Permission [*]
+permission 1
+profilePermissions *
«Entity»
UserPermissionProfile
+profile : PermissionProfile [1]{nonunique}
+active : boolean [1]{nonunique}
+version : long [1]{nonunique}
+modifiedTimestamp : Timestamp [1]{nonunique}
+modifiedUser : User [0..1]
+findByUserProfile(userId : Long [0..1], profile : PermissionProfile [0..1], active : Boolean [0..1]) : UserPermissionProfile [*]
«Entity»
ProfilePermission
+profile : PermissionProfile [1]{nonunique}
...

Figure 4.60.: `UserPermissionProfile` items define which profiles are activated for a `User`. Each profile refers a set of `Permission` master data records. Each `Permission` represents the privilege to execute a service method with a specified argument condition.

The data model to support the described authorisation system is shown in figure 4.60. When a `User` is created, a `UserPermissionProfile` detail instance is created for each constant of the `PermissionProfile` enumeration (table 4.8). The `UserPermissionProfile.active` flag is used mark profiles enabled for the user. This way, the revocation of profiles can be well documented in journal system messages. Since profiles are non-exclusive overall, a single `UserPermissionProfile` instance can refer to multiple `ProfilePermission` master data records. Vice versa, a single `ProfilePermission` record is referred by multiple `UserPermissionProfile` from various `User` records. The resulting unconstrained many-to-many association comes into effect by using the `Permission.findByServiceMethodUser` finder method (listing 4.27) to look up a user's privilege for a given service method. `Permission` represents a service method privilege including a *single* argument condition, defined by the entity fields listed below:

`parameterGetter`: Contains a parameter path notation to access an InVO argument field value via reflection for the argument inspection. The first path element will be a method *parameter name*, which specifies a parameter from the service method's parameter list. Since parameter names are not supported by Java reflection[58] directly, this was solved by introducing a user-defined Java *annotation* (`@MethodParameterNames`). AndroMDA's `SpringService.vsl` template was modified to annotate any service method declaration with names of parameters.

`transformation`: Specifies an optional transformation of the argument value retrieved by `parameterGetter`. The `ServiceMethodParameterTransformation` enumeration type catalogues the repertoire of prepared transformation logic. A typical transformation such as `VISIT_ID_TO_TRIAL_ID` derives the related root entity ID from a detail record ID.

`restriction` Selects the actual condition to be satisfied by the transformed argument value. Implemented condition logic such as `TRIAL_USER_DEPARTMENT` is enumerated in the `ServiceMethodParameterRestriction` enumeration type.

`disjunctionGroup`: Allows combining conditions of multiple `Permission` records for a service method using the boolean disjunction. A condition disjunction group provides flexibility when defining privileges of CRUD operations for detail entities associated with multiple root entities.

[58]prior to Java 1.8 (Darcy, 2013)

```
public class PermissionDaoImpl extends PermissionDaoBase {

    //This performance critical query lists a service method's Permission records for a given user:
    //  SELECT * FROM
    //  permission AS p
    //  LEFT JOIN profile_permission AS pp ON p.id = pp.permission_fk
    //  WHERE
    //  p.service_method = ?
    //  AND pp.profile IN
    //      (SELECT profile FROM
    //          user_permission_profile AS upp
    //      WHERE
    //          upp.user_fk = ?
    //          AND upp.active = ?
    //      GROUP BY
    //          upp.profile)
    //The typical usage of this finder method is to set all filter parameters.
    @Override
    protected Collection<Permission> handleFindByServiceMethodUser(String serviceMethod, Long userId, Boolean
         userPermissionProfileActive) throws Exception {
        //create hibernate criteria instance for Permission records:
        org.hibernate.Criteria permissionCritria = this.getSession().createCriteria(Permission.class); //2k records overall,
             360 service methods
        if (serviceMethod != null) { //apply service method name filter if provided
            permissionCritria.add(Restrictions.eq("serviceMethod", serviceMethod)); //narrows to 5.63 records avg, 56 max
        }
        if (userId != null || userPermissionProfileActive != null) {
            //join with ProfilePermission (5k5 records overall, 90 users):
            org.hibernate.Criteria profilePermissionCritria = permissionCritria.createCriteria("profilePermissions",
                 CriteriaSpecification.LEFT_JOIN); //expands to 12.506 records avg, 122 max

            //create a non-correlated subquery criteria for UserPermissionProfile records:
            org.hibernate.criterion.DetachedCriteria subQuery = DetachedCriteria.forClass(UserPermissionProfileImpl.class, "
                 userPermissionProfile"); //5k5 records overall, 90 users
            if (userId != null) { //apply user id filter if provided
                subQuery.add(Restrictions.eq("user.id", userId.longValue())); //narrows to 68 records
            }
            if (userPermissionProfileActive != null) { //apply filter for UserPermissionProfile.active if provided
                subQuery.add(Restrictions.eq("active", userPermissionProfileActive.booleanValue())); //narrows to 14 records
            }
            subQuery.setProjection(Projections.projectionList().add(Projections.property("profile"))); //14 items
            //UserPermissionProfile subquery:
            profilePermissionCritria.add(Subqueries.propertyIn("profile", subQuery)); //narrows to 2.083 records avg, 26 max
        }
        return permissionCritria.list();
    }
    ...
```

Listing 4.27: The `Permission.findByServiceMethodUser` finder method retrieves a service method's `Permission` records for a given user. Since join conditions on explicit columns are not supported by Criteria API, a *subquery* is used as a workaround.

The authorisation logic is entirely encapsulated by the `AuthorisationInterceptor` `BeforeMethodAdvice` implementation. It looks up all `Permission` records for the requested service method of enabled profiles that are assigned to the executing user using the `Permission.findByServiceMethodUser` finder method. Next, the argument conditions of enlisted `Permission` entires are evaluated. An `AuthorisationException` is thrown with a detailed error message upon the first condition found to be not satisfied. This corresponds to the combination of conditions using boolean *conjunction*. If a `disjunctionGroup` string is defined, an exception is thrown only if all conditions with the same `conjunctionGroup` value evaluate to `false`. The authorisation is finally granted, if no exception occured.

In contrast to the clear effect of blocking CRUD operations, restricting service methods for data retrieval tends to confuse users. This is the common case when passing an unauthorized argument value for a parameter such as `departmentId`, which narrows the result by returning items associated with a given department only. Although an error message is generated, some web UI screen designs complicate putting it in touch with the causing user action. As a consequence, summary list views (section 4.6.7) or calendar views (section 4.6.5) would just remain blank until an authorized filter argument is selected. To encounter this issue, *argument instrumentation* is used instead of preventing service method execution. An argument value is overridden with an appropriate value that will not reveal result records the user is basically not authorized to view. The remaining `Permission` entity fields are used to express an argument instrumentation in a way similar to argument conditions:

parameterSetter: Contains a parameter path notation to access an InVO argument field value via reflection for the argument instrumentation.

override: Selects the actual argument value transformation, whose result will replace the original argument value. Predefined program logic for generating the override value is enumerated separately by the `ServiceMethodParameterOverride` enumeration type.

When argument instrumentations was applied, the `AuthorisationInterceptor` hands over to the service method in the service base class, which in turn invokes the handler containing the hand-crafted service method implementation code finally.

`Permission` records and their mapping to profiles represent static master data. They are configurable by means of a Character-Separated Values (CSV) import provided by the CLI tool. The structure of the imported permission definitions is outlined in table 4.9 and refers to two exemplary CRUD service methods. The tabular format of profile master data allows easy reconfigurations, for example to ultimetely restrict delicate root entity delete operations by introducig an even more priviledged super user permission profile. Aside the advantage of implementing change requests easily, the privilege import facility provides a structured way to implement authorisation for additional service methods when extending the application.

```
// Attention: Generated code! Do not modify by hand!
// Generated by: SpringService.vsl in andromda-spring-cartridge.
public interface TrialService {

    ...

    @MethodParameterNames({"auth","modifiedTrial"})
    public TrialOutVO updateTrial(AuthenticationVO auth, TrialInVO modifiedTrial)
        throws AuthenticationException, AuthorisationException, ServiceException;

    @MethodParameterNames({"auth","modifiedVisit"})
    public VisitOutVO updateVisit(AuthenticationVO auth, VisitInVO modifiedVisit)
        throws AuthenticationException, AuthorisationException, ServiceException;

    ...

}
```

Listing 4.28: This excerpt of the generated `TrialService` interface shows a root entity (`updateTrial`) and a detail entity (`updateVisit`) CRUD operation declaration with the custom `@MethodParameterNames` annotations. Relevant argument types are shown in figure 4.61 and 4.62.

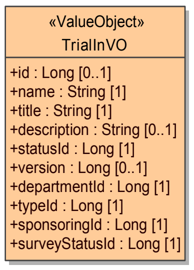


Figure 4.61.: The `TrialInVO` InVO argument type.

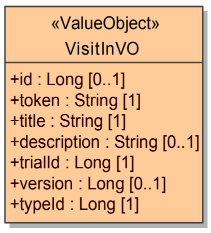


Figure 4.62.: The `VisitInVO` InVO argument type.

service method	`parameterGetter`	`transformation`	`restriction`	belongs of profile
`updateTrial`	`modifiedTrial.getId`		`ANY_TRIAL` (no restriction)	`TRIAL_MASTER_ALL_DEPARTMENTS`
			`TRIAL_USER_DEPARTMENT` (listing 4.24)	`TRIAL_MASTER_USER_DEPARTMENT`
			`TRIAL_IDENTITY_TEAM_MEMBER` (listing 4.26)	`TRIAL_MASTER_TEAM_MEMBER`
	`modifiedTrial.getDepartmentId`		`ANY_DEPARTMENT`	`TRIAL_MASTER_ALL_DEPARTMENTS`
			`USER_DEPARTMENT`	`TRIAL_MASTER_USER_DEPARTMENT`
				`TRIAL_MASTER_TEAM_MEMBER`
`updateVisit`	`modifiedVisit.getId`	`VISIT_ID_TO_TRIAL_ID`	`ANY_TRIAL`	`TRIAL_MASTER_ALL_DEPARTMENTS`
				`TRIAL_DETAIL_ALL_DEPARTMENTS`
			`TRIAL_USER_DEPARTMENT`	`TRIAL_MASTER_USER_DEPARTMENT`
				`TRIAL_DETAIL_USER_DEPARTMENT`
			`TRIAL_IDENTITY_TEAM_MEMBER`	`TRIAL_MASTER_TEAM_MEMBER`
				`TRIAL_DETAIL_TEAM_MEMBER`
	`modifiedVisit.getTrialId`		`ANY_TRIAL`	`TRIAL_MASTER_ALL_DEPARTMENTS`
				`TRIAL_DETAIL_ALL_DEPARTMENTS`
			`TRIAL_USER_DEPARTMENT`	`TRIAL_MASTER_USER_DEPARTMENT`
				`TRIAL_DETAIL_USER_DEPARTMENT`
			`TRIAL_IDENTITY_TEAM_MEMBER`	`TRIAL_MASTER_TEAM_MEMBER`
				`TRIAL_DETAIL_TEAM_MEMBER`

Table 4.9.: Permission definitions and their profile membership configuration are imported using the tabular format shown. Argument conditions are specified by the `parameterGetter` paremeter field path, an optional value `transformation` and selecting a prepared `restriction` program logic.

Like the `AuthenticationInterceptor`, the `AuthorisationInterceptor` advice is basically invoked with every single service method request. Since the authorisation logic is more complex, performance now attracts more attention. Although no obvious bottleneck showed up, the `findByServiceMethodUser` database query (listing 4.27) would be a crucial element here.

The powerful access control concept presented meets requirements and seperates concerns by tying authentication and authorisation to the service layer. The implementation additionally supports *host-based* authorisation at service method level, which is not covered in detail. For example, host-based authorisation can be configured to globally deny retrieving records containing proband's PII when accessing the system from the internet.

Trying to implement the same functional range of authentication and authorisation features using an available framework such as Spring Security will show up obvious difficulties. Basically, the well known and easy-to-use URL-based Spring Security is inappropriate for a fully AJAX-driven JSF UI. In the case of a multi-tiered application with a distinct service layer, it is an design argument not to integrate security features in the web tier at all. Spring Method Security comes closer to the presented solution (Winch, 2013). However, a closer look reveals why taking this approach would not be advantageous:

- Permission configuration is hard-wired using `@PreAuthorize` and `@PostFilter` method annotations or *pointcut* definitions.
- Argument inspection specified by `@PreAuthorize` would require to implement a custom *permission evaluator*, which would effectively contain the current `AuthorisationInterceptor` logic.
- Argument instrumentation is not supported. `@PostFilter` supports filtering service method return value lists however.

4.5.3. Error Logging

AndroMDA templates introduce a default service exception type (e.g. `ProbandServiceException`) for artefacts of each generated service (e.g. `ProbandServiceBase`). It is used it for null checks of service method arguments and to wrap `Throwables` raised in service method implementations, except *application exceptions*. Application exceptions are custom exception types according to explicit UML model elements, which will not be wrapped but re-thrown immediately to retain their type. Both default service exception and application exception types represent *checked* exception types, which are a business logic design element and *expected* to occur frequently and handled by the presentation layer. Templates do not support declaring exceptions for DAO methods, hence *unchecked* exception types[59] are used to wrap *unexpected*

[59] `java.lang.RuntimeException` and subclasses

exceptions such as I/O errors. They are supposed to terminate the request thread after showing a custom error page and are logged separately in Tomcat's `catalina.out` file.

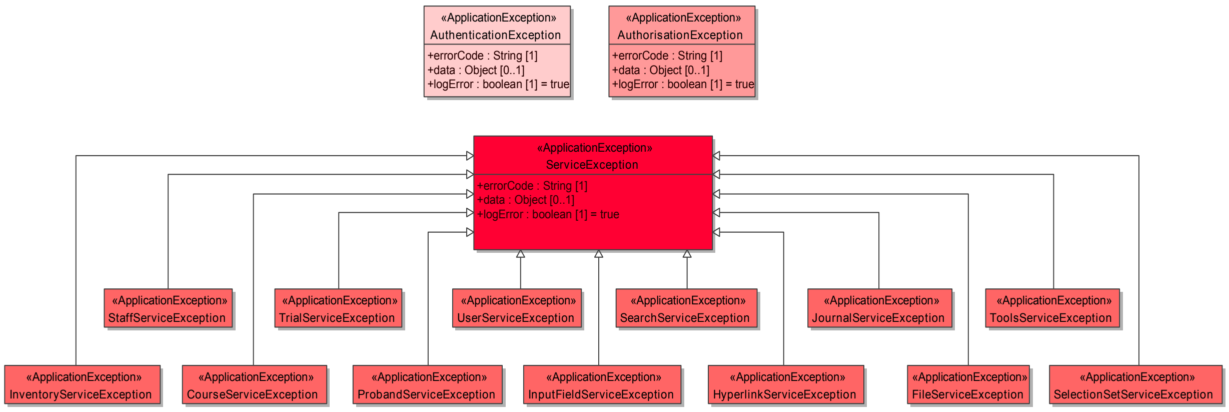

«ApplicationException» AuthenticationException
+errorCode : String [1]
+data : Object [0..1]
+logError : boolean [1] = true
«ApplicationException» AuthorisationException
+errorCode : String [1]
+data : Object [0..1]
+logError : boolean [1] = true
«ApplicationException» ServiceException
+errorCode : String [1]
+data : Object [0..1]
+logError : boolean [1] = true
«ApplicationException» StaffServiceException
«ApplicationException» TrialServiceException
«ApplicationException» UserServiceException
«ApplicationException» SearchServiceException
«ApplicationException» JournalServiceException
«ApplicationException» ToolsServiceException
«ApplicationException» InventoryServiceException
«ApplicationException» CourseServiceException
«ApplicationException» ProbandServiceException
«ApplicationException» InputFieldServiceException
«ApplicationException» HyperlinkServiceException
«ApplicationException» FileServiceException
«ApplicationException» SelectionSetServiceException

Figure 4.63.: Service methods of each service may throw three application exception types. Default service exceptions are generalised by `ServiceException`.

The Phoenix CTMS introduces three application exception types (figure 4.63), which will be linked to all its services interfaces (table 4.7) in order to add corresponding `throws` declarations to service method signatures:

`AuthenticationException` Communicates denial due to authentication failures, detected by the `AuthenticationInterceptor`. Although an exception message with details about the problem is populated, the UI replaces it with an opaque message by default. The web tier can use this type to destroy a session and thus log a user off once it occurs. Revoking access therefore takes effect instantly for a logged-in user.

`AuthorisationException` Communicates denial due to authorisation failures, detected by the `AuthenticationInterceptor`. Their most frequent occurrence is caused by users with view-only privileges, trying to perform CRUD actions. In this case, the `AuthorisationException.message` is displayed by UI response message boxes.

`ServiceException` This application exception generalizes all default service exception types. To accomplish this, the default service exceptions are explicitly created in the UML model, with exactly matching type names. `ServiceException` generalizes each default service exception type, resulting in a corresponding Java type inheritance. Generated parameter input checks and wrapped exceptions of all service interfaces are therefore consolidated by the single `ServiceException` type. The primary purpose of `ServiceException` is to transport response messages for any kind of business logic failures. These will be displayed by UI response message boxes and primarily comprise messages from various kinds of input checks listed below.

- referential integrity: *invalid trial ID '{0}'*
- data model constraints: *proband '{0}' is already on subject list*
- concurrent modifications: *record was modified meanwhile by {0}*
- integrity of encryption domains: *root record (proband '{0}') cannot be decrypted*[60]

[60]The same department key must be used for all encrypted detail records of a root entity record. While the web UI excludes these violations, the checks anticipate interconnected client applications invoking service methods directly.

- value range checks: *'{0}' does not match the format for phone numbers. Example of a correct phone number: +43650123456, +4331612345*
- sanity checks: *end date is before start date*
- business rules: *trial is locked*

`ServiceException` effectively aggregates any occurring exception type occured, which is not related to access control. This will simplify exception processing for the presentation layer as well for a `ThrowsAdvice` interceptor logging exceptions.

Each application exception is equipped with fields to transport additional information related to the exception event:

`errorCode`: This string value can be used to transport canonical identifiers for exception messages, which is useful for assertion conditions in software test cases. By default, the `errorCode` is initialized with the exception message's identifier for l10n bundles (table 4.4).

`data`: The multi-purpose `data` field is of type `Object` to hold a value of any type. A use case is to return multiple exception messages for indexed UI input elements at once, typically required for bulk CRUD operations.

`logError`: This flag is `true` by default and can be set to `false` by the business logic to indicate the exception should not be logged by `ErrorLogger` or re-thrown in a business logic program flow.

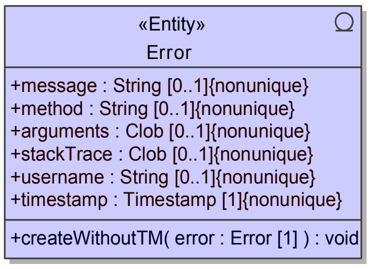

«Entity»
Error
+message : String [0..1]{nonunique}
+method : String [0..1]{nonunique}
+arguments : Clob [0..1]{nonunique}
+stackTrace : Clob [0..1]{nonunique}
+username : String [0..1]{nonunique}
+timestamp : Timestamp [1]{nonunique}
+createWithoutTM(error : Error [1]) : void

Figure 4.64.: The `ErrorLogger` interceptor persists occurred exceptions using the `Error` entity.

The `ErrorLogger` interceptor can be configured to process `AuthenticationExceptions`, `AuthorisationExceptions`, `ServiceExceptions` and remaining `Trowables` by logging them into a database table. The resulting log entries are defined by the `Error` entity (figure 4.64). It provides fields to persist exception details:

`method`: The name of the service method that encountered the exception.

`arguments`: The service method argument array is converted to an XML fragment string using XStream. The dump can contain values of secret fields (table 4.6). Furthermore, the password credential argument as well as arguments of service methods for setting passwords contains secret information that must not be stored in plaintext. Therefore these field values are obfuscated before serializing the argument values.

`stackTrace`: The Java exception stack trace is dumped using `Throwable.printStackTrace`.

Since `ErrorLogger.afterThrowing` executes outside the database transaction context, the `Error.createWithoutTM` DAO method tries to persist the `Error` instance by commiting a separate transaction.

4.6. Web Presentation Layer

Web applications are expected to feature a complex graphical User Interface these days, usable like a desktop application. A state-of-the-art approach was chosen to create a sophisticated AJAX-driven RIAs UI with rigorous level of detail and appealing appearance. The application's *web tier* represents a JSF 2 presentation layer driven by the JSF reference implementation *Mojarra*.

While the application's core tier development started with the Spring/Hibernate skeleton generated by AndroMDA, the web tier was built manually from scratch. This was accomplished by creating a JSF Java project structure described in (Kurz, Müllan, and Marinschek, 2009). The web tier essentially comprises a (large) number of facelets and managed beans as well as supplemental data models for UI components, Js logic and CSS resources. To integrate it into the existing Maven build automation that comes with the use of AndroMDA, the `pom.xml` was set up to define the tier project dependency structure outlined in figure 2.16. The web tier project finally incorporates as if it was another artefact created by AndroMDA.

Deployment model Instead of an application server which integrates support for the complete JEE stack, the lightweight *Apache Tomcat* 6 servlet container is sufficient to run the Phoenix CTMS.

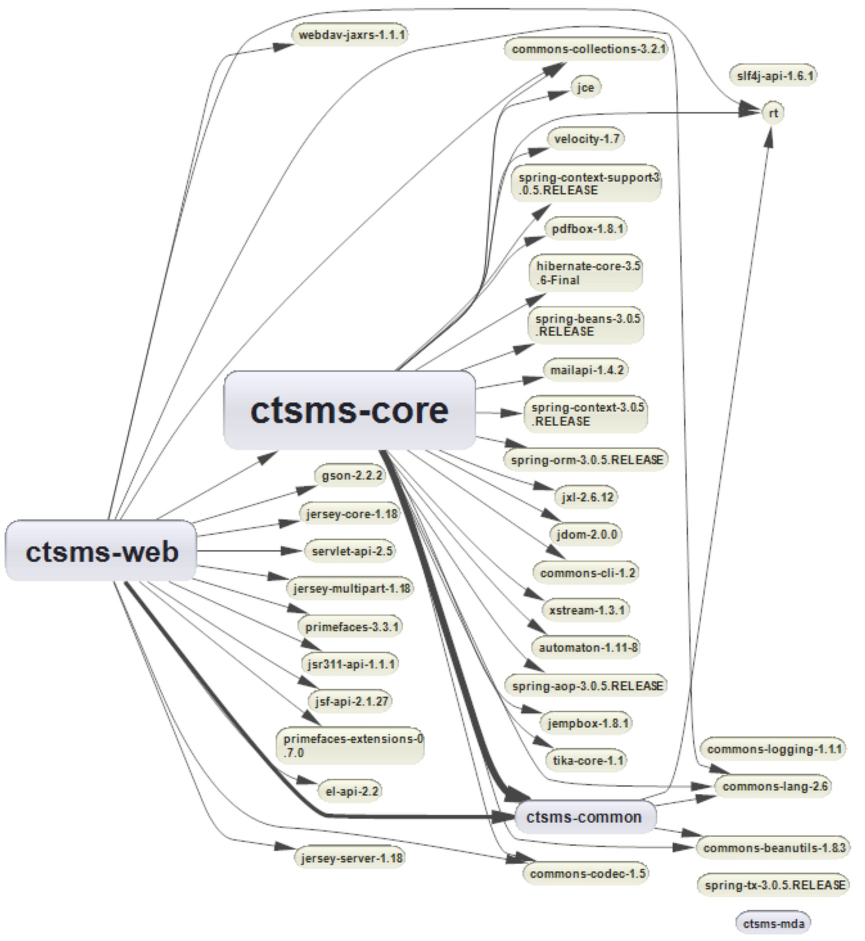


Figure 4.65.: For a monolithic deployment on a single server, the web tier is compiled to a single `.war` file for Tomcat. It links or includes a hierarchy of required libraries (`.jars`), as illustrated in the *dependency graph*. The application's service and DAO layer dependencies are build goals of both the common and core tier project from figure 2.16.

The web application archive (`.war`) deployed in Tomcat includes the backend layer libraries (figure 4.65).

The web tier invokes service methods of the underlying service layer, which are exposed by a *service locator* pattern. The generated default `ServiceLocator` is a Spring factory for service implementation class singleton instances. AndroMDA templates allow to generate an enhanced service locator with Java Remote Method Invocation (RMI) proxies according to declarations in the UML model. This enables (horizontal) *scaling* by distributing the core tier and web tier across different physical server computers (application and frontend servers). An application's RDBMS connectivity traditionally relies on TCP/IP, hence application servers can connect to a database server deployed on separate hardware. The system architecture therefore represents a thorough *distributed system*, which distinguishes the node types below:

Client hosts: web browsers
HTTP server: upstream/fronting web server(s)
Frontend servers: servlet containers running the web tier
Application servers: RMI servers publishing core tier's services
Database servers: RDBMS cluster with multi-master replication[61]

The optional HTTP server in front of Tomcat filters HTTP requests and forwards them using Apache JServ Protocol (AJP). When reachable from internet, this is preferred in order to have more detailed configuration and logging options, serve static content and implement load balancing. For the scope of this work, a monolithic deployment is in focus, which integrates frontend, application and database servers on the same (virtual) machine.

Message queuing The system architecture does not seperate functional components of backend layers, which are supposed to run on different types of application server nodes that could communicate with one another using *message queuing*. Instead, service methods are provided by a monolithic service layer. Service method invocations are performed *synchronously* within servlet request threads, which does not require message queuing for the link between service and presentation layer either. The majority of service methods are designed to have a rather short execution time, which complies with the latency of HTTP request-response cycle expected by users. Uploading large files represent a special case of long-lasting synchronous operations, requiring a continous connection while streaming a file's content to the server. In terms of JEE, *asynchronous* operations can be supported by a messaging infrastructure such as Java Message Service (JMS), which is not part of a standalone servlet container like Tomcat. The message queuing principle is however subject to automated server *jobs* and currently implemented using database tables. For example, notification messages are queued in database tables and sent by email using a separate server job. Queueing of other long operations, which are explicitly triggered by users from the UI and supposed to respond with a result value or completion status are subject to a future version. This especially addresses operations on large entity instance graphs such as deletion or calculating the digital signature value (section 4.3.3).

4.6.1. JSF Web Tier

4.6.1.1. Managed Beans

Managed beans have minimal inderdependency and are declared using JSF annotations (`@ManagedBean`), so there was no need for additional Spring configuration/DI. An overview describes and classifies managed beans according to their instance lifetime (table 2.4):

[61]e.g. pgCluster, Oracle Real Application Clusters (RAC)

Application scoped bean The unique `@ApplicationScoped` managed bean (`ApplicationScopeBean`) can be viewed as singleton. It consolidates various operations independent of a view state, which are frequently used in EL expressions across facelets:

- getter methods for constant values such as tree node types, moving calendar year range, date/datetime input format patterns
- conversion of large integer values into human-readable strings: `byteCountToString`, `durationToString`
- conversion of VO iteration variables with fields of `Color` enumeration type into corresponding CSS style class strings
- string clipping
- string escaping and encoding: `escapeHtml`, `quoteJsString`, `decodeBase64`, `encodeBase64`
- Js parsing: `compressJS`, conversion of POJO graphs from and to JSON
- generating the list of Js constant values to export: browser window names, POST parameter names, ...
- navigation support: view names to URLs mapping methods, `viewAlive` action listener method
- synchronized `TreeSet` to maintain a list of currently "online" users
- utility methods for EL expressions: `mapToList`, `ToolService` service methods

Session scoped bean When a browser accesses the site for the first time, requests lack a *session ID* HTTP cookie header value. The JSF infrastructure adds a new entry into its `map` of sessions by generating an unpredictable session ID value. This entry effectively refers to created `@SessionScoped` managed beans and a growing `Map` of accessed `@ViewScoped` managed beans. A session and related views will be removed from the session map, if the session is *invalidated* explicitly (logout) or if it has *expired* due to exceeding a predefined time-to-live duration. When accessing the web UI, the very first HTTP response contains a `Set-Cookie` directive including the session ID value. It advices the browser to store the `JSESSIONID` value for the scope of the browser session. This cookie value is appended to subsequent HTTP request headers and allows JSF to look up managed bean instances that represent the client session's state.

The web tier uses a single `@SessionScoped` managed bean (`SessionScopeBean`), serving multiple purposes:

Login controller: As described in section 4.5.1, credentials are stored as state of `SessionScopeBean` when a user logs in. The many `@ViewScoped` managed beans can subsequently access these credentials for service method invocations. System logout is implemented by simply invalidating the active session.

Localized constants and option items: Option items for dropdowns have to be localized according to the user's language setting. Frequently required option item sets such as datatable filter options are populated once in order to avoid repeated retrieval operations, which produce load due to evitable `SelectionSetService` service method invocations. These option item sets are therefore fetched and cached right after the successful login operation. The same applies to the costly preparation of constants and static data structures, such as the list of supported time zones (figure 4.31).

Menubar model and controller: `p:menubar` menus of the *menu bar* visible on top of each UI screen consist of sub-menu hierarchies to structure menu items. In order to create dynamic menu structures, PrimeFaces' menu data models require the application code to apply property values and bind action listener methods of menu elements. This takes place in `SessionScopeBean` code in order to build menus depending on the logged-in user:

- user name, identity `Staff` name
- selected language, time zone and theme
- history of recently modified items

Notably, this is the only situation across the entire application, a programatic JSF component tree manipulation takes place.

Image store: `p:graphicImage` wraps the HTML `img` tag and is used to display image data. When used with dynamic content like uploaded CV photos, the browser's separate GET request to download the image requires a technique to manage opened `InputStreams`. The image store is a `SessionScopeBean` field exposing a synchronized `Map` to register and reliably reference any image the user accessed or manipulated during the session.

View scoped managed beans A page is intially requested via HTTP GET method, and displayed in a browser window or tab. `@ViewScoped` managed beans referenced by the page facelet are instantiated and registered using a view ID generated for the GET request. Subsequent POST requests (postbacks) of AJAX interactions contain the view ID parameter, which allows the JSF infrastructure to look up the right `@ViewScoped` managed bean instances again. If the user reloads the page, the view is abandoned in favour of a fresh view coming along with a new ID and cleared state due to new managed bean instances. Thus, a displayed page that exists between two GET requests is denoted as JSF *view*, with a temporal context aside the structural concept of a page.

Request scoped managed beans A fresh `@RequestScoped` managed bean instance is created upon every single request (POST or GET) of a JSF page, whose facelet references the bean. Hence, the bean's state is volatile and cannot be used to keep informations throughout subsequent requests. `ErrorBean` is the sole `@RequestScoped` managed bean implemented. It extracts an unexpected exceptions's message for the system's custom error page.

4.6.1.2. PrimeFaces UI

While native JSF components are limited in their variety and functionality, the *PrimeFaces* component library was choosen to be able to raise the high-end RIA UI within an acceptable time frame. PrimeFaces provides a continually growing set of uniformly skinned, animated components, including replacements for native JSF components. While many of them originate from Yahoo User Interface (YUI) controls (Glass and YUI Contributors, 2014), specific ones represent wrappers for other third-party Js-based controls. PrimeFaces components include a client-side API ("widgets"). The implementation of most widgets relies on *jQuery* (Resig and jQuery Contributors, 2014).

Thousands of PrimeFaces component instances are arranged by facelets to form the Phoenix CTMS UI. The list below shows the PrimeFaces components utilized and their frequency:

- `p:ajax` (284)
- `p:ajaxStatus` (2)
- `p:autoComplete` (35)
- `p:calendar` (40)
- `p:captcha` (1)
- `p:column` (692)
- `p:columnGroup` (1)
- `p:commandButton` (716)
- `p:commandLink` (3)
- `p:confirmDialog` (38)
- `p:dataGrid` (5)
- `p:dataList` (20)
- `p:dataTable` (85)
- `p:dialog` (4)
- `p:draggable` (1)
- `p:droppable` (1)
- `p:fieldset` (8)
- `p:fileDownload` (26)
- `p:fileUpload` (3)
- `p:graphicImage` (1)
- `p:growl` (9)
- `p:inputText` (66)
- `p:inputTextarea` (35)
- `p:lineChart` (1)
- `p:menubar` (15)
- `p:message` (343)
- `p:messages` (73)
- `p:notificationBar` (1)
- `p:panel` (91)
- `p:password` (4)
- `p:photoCam` (1)
- `p:remoteCommand` (154)
- `p:row` (2)
- `p:rowExpansion` (26)
- `p:rowToggler` (26)
- `p:schedule` (2)
- `p:selectBooleanCheckbox` (82)
- `p:selectManyCheckbox` (1)
- `p:selectOneButton` (1)
- `p:selectOneMenu` (94)
- `p:selectOneRadio` (2)
- `p:separator` (133)
- `p:spinner` (25)
- `p:tab` (101)
- `p:tabView` (30)
- `p:toolbar` (75)
- `p:toolbarGroup` (208)
- `p:tooltip` (372)
- `p:tree` (5)
- `p:treeNode` (10)
- `p:treeTable` (1)

AJAX support is inherent to all PrimeFaces components and relies on the underlying PVR concept of JSF. A XMLHttpRequest (XHR) request-response cycle (PPR) is triggered by a client-side event of a component. The HTTP POST request will submit values of components listed by the component tag's `process` attribute. The response contains rendered markup for components to update, as listed by the component tag's `update` attribute. Thus, PVR allows updating page sections by replacing selected parts of the page's DOM tree, instead of reloading the entire page.

PrimeFaces is a living open-source software framework under continuing development. During the lasting UI development, the PrimeFaces version was incremented several times. Some regression failures were managed by adopting Js and/or facelet code only, while others required to adopt Java application code too. Beside new features, PrimeFaces release cycles also address reported issues. Since the Phoenix CTMS settled with PrimeFaces version 3.3.1 for now, some fixes of newer versions were backported manually. It was decided to stay with an unaltered Java codebase of PrimeFaces however.

4.6.1.3. primefaces-extensions Components

Some UI components not available in PrimeFaces can be found in the *primefaces-extensions* add-on:

`pe:codeMirror`: This tag places a primefaces-extensions component, which wraps CodeMirror (Haverbeke and CodeMirror Contributors, n.d.), a Js library to enhance the ordinary HTML `textarea` with features known from Integrated Development Environment (IDE) source code editors. Aside the look-and-feel, `pe:codeMirror` provides configurable syntax highlighting and code completion. It is utilized to facilitate editing Js snippets in the context of input form scripting features.

`pe:javascript`: In contrast to `p:ajax` that binds a PPR cycle to a Js event supported by a component, `pe:javascript` simply binds a client-side Js function instead. It is used with `pe:timeline` and for workarounds in conjunction with `PrimeFaces.ajax.AjaxRequest` wrapper methods. These were required for PrimeFaces input components within nested `p:dataTables`, where `p:ajax` fails to work as usual.

`pe:timeline`: An interactive timeline component is required to visualize trial milestones and phases. Until version 3.0.1, PrimeFaces came with a rudimentary timeline component, which was prototypical and removed finally due to license difficulties. For the Phoenix CTMS, concerns arised about the extensive task to develop another custom JSF component that encapsulates an available timeline Js library. Fortunately, a suitable component was introduced by primefaces-extensions as of 2012, wich was ready for use in productive environments as of May 2013, so it could be integrated alternatively (section 4.6.6).

4.6.1.4. Custom JSF Components

Inspired by the implementation of PrimeFaces components, two custom jsf components were developed:

`ctsms:tagCloud`: An existing PrimeFaces component was modified to get started with creating custom JSF components. The `ctsms:tagCloud` component is a re-implementation of PrimeFaces' `p:tagCloud` component, providing an `onclick` event handler attribute instead of a hyperlink URL attribute for tag cloud items. It is used for tag clouds displayed in the System Modules Overview section of the start page.

`ctsms:sketchPad`: An alternative way for data entry of medical information is to utilize the pen input method. This is supported by a complex custom component that wraps a customized version of Raphael SketchPad (Li, 2013), which in turn uses *Raphaël* (Baranovskiy, 2013), a Js library providing a SVG API for browsers. Beside drawing free sketches, the component's current implementation supports selecting regions by marking them with a cross. This allows producing exactly encoded input similar to `p:selectManyMenu` or `p:selectManyCheckbox`.

4.6.1.5. JSF Templates

While managed beans and models can be structured using class inheritance, JSF provides tags to support *templating* in order to reduce repetitions:

`ui:composition`: This templating tag allows a facelet *F* to *copy* a specified template facelet *T*. *F* will render nothing else but the content of *T*. *T* can define named placeholder nodes using `ui:insert`, which are replaced with specific content defined by *F* by using `ui:define`. Nested compositions give the UI's template hierachy shown in figure 4.66. `ui:composition` without specifying a template can be used used to demarcate content if the facelet is to be included with `ui:include`.

`ui:decorate`: Similar to `ui:composition`, the `ui:decorate` tag *embeds* *T* into *F* instead of copying it. *T* specifies content that is to be placed inside *F* at the position of the `ui:decorate` node. `ui:decorate` is used for facelets of the query editor UI.

`ui:include`: The `ui:include` tag embeds a facelet *T* inside *F* straightforwardly, without `ui:insert`/ `ui:define` decoration. However, the `ui:param` tag allows to pass variable values. Typical applications of `ui:include` are assembling entity views by including their tab contents or rendering boolean values as ✓/⊘ icons.

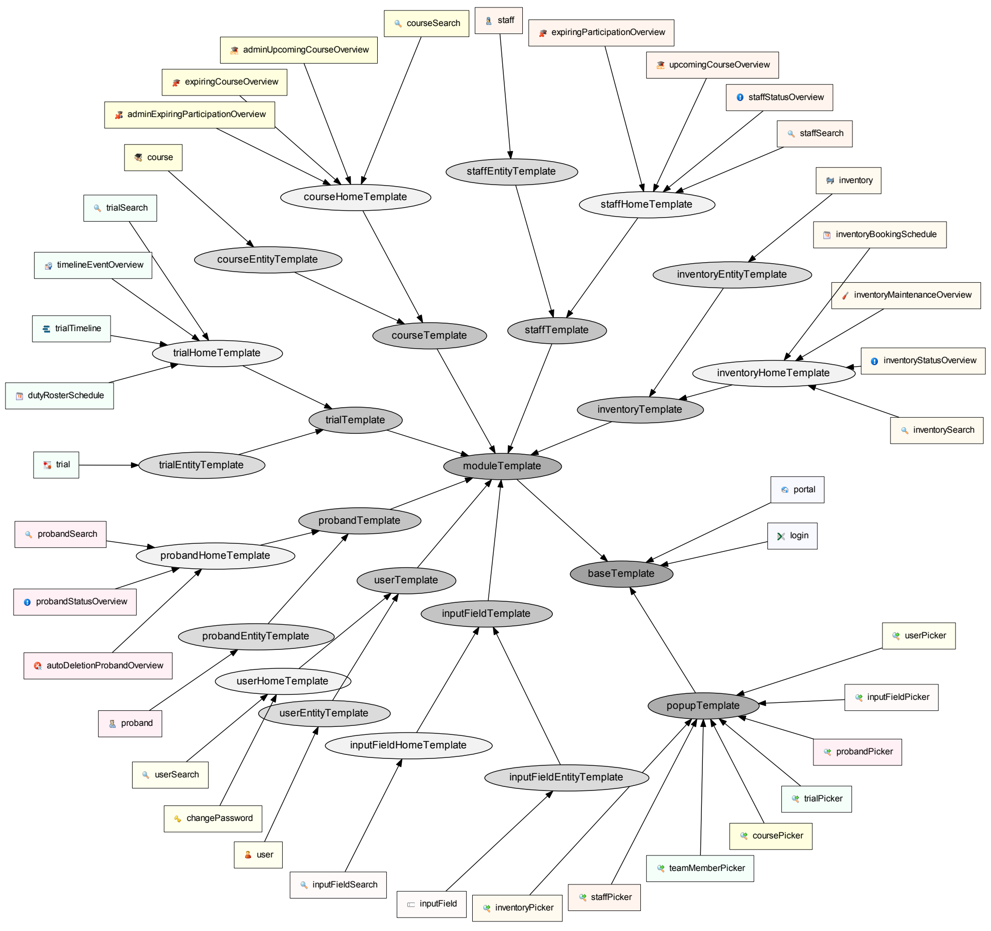


Figure 4.66.: This tree graph shows the template hierarchy of UI page facelets. `baseTemplate.html` represents the root template, which defines the basic layout (head, menubar, headline, content, notificationbar), EL constants, generic CSS and Js resource includes. The facelets represented by leaves correspond to unique UI pages.

4.6.1.6. JSF Servlet Configuration

The JSF core servlet and related infrastructure is configured by two XML files:

Deployment descriptor The deployment descriptor (`web.xml`) registers the JSF request servlet (`javax.faces.webapp.FacesServlet`) as well as custom servlets[62]. To integrate a REST web service, a framework such as the JAX-RS reference implementation "Jersey" requires to register an additional servlet. While the web service is subject to future extensions, an application-specific servlet was introduced as part of the sketchpad JSF component implementation for handling sketch background image requests. Other configuration options allow to bind the user's theme preference (figure 4.67) or the session timeout. Another configuration aspect of the deployment descriptor is the registration of request filters (`javax.servlet.Filter` interface). A custom request filter is used to enforce UTF8 encoding. PrimeFaces requires to register its included upload request filter for handling file uploads, in order to utilize the `p:fileUpload` component. When a page is accessed, another custom filter redirects the GET HTTP requests to the login page (figure 4.55), if the session was invalidated.

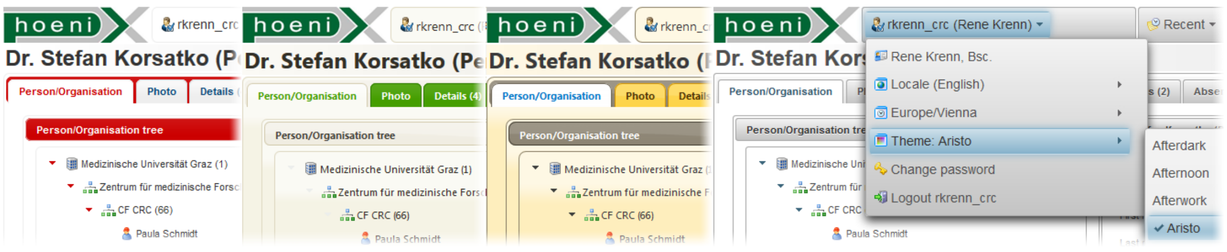


Figure 4.67.: Users can choose from 35+ PrimeFaces UI themes with various colors. Themes can be used by an identity with multiple user accounts to pervasively indicate the account currently being worked with.

Facelet configuration In order to handle `javax.faces.application.ViewExpiredExceptions` that occur with POST requests of AJAX operations in the case of an invalidated session, `faces-config.xml` registers a custom exception handler factory (`javax.faces.context.ExceptionHandlerFactory` decoration). The corresponding `ExceptionHandler` implements a Post/Redirect/Get (PRG) flow to redirect to the login page. Furthermore, l10n message bundles (table 4.5) are specified via `faces-config.xml` for direct use within EL expressions. Finally, `faces-config.xml` allows to setup page navigation by means of navigation rules. These are not useful for AJAX-driven navigation however, which is described more in detail in section 4.6.2.

4.6.1.7. JavaScript Resources

Js artefacts essentially provide the implementation of custom JSF components as well as workarounds and customization of PrimeFaces components. Since Js is a dynamic programming language, the latter can be accomplished without altering PrimeFaces's sources. Js *prototype* overrides and extensions of PrimeFaces's client side widgets include:

[62] `javax.servlet.http.HttpServlet` interface implementations

Client-side component state manipulation A PrimeFaces component manifests in HTML DOM elements and form parameters that represent its *state*. It comprises abstract state information like the value of an input component or the visual enabled/disabled state. Accessing the component's state client-side via widget accessor methods is insufficiently supported by default. Modifying the client-side state of input components programmatically is required in two situations:

- A `p:confirmDialog` prompts the user wether or not to adjust an input's value in order to correct or align it to a preset. If the user agrees, the input value has to be set by a Js function.
- Input form scripting allows calculating dependent values client-side. If a user wants to apply a precalculated value by pressing the corresponding button, the input component's value will be set by a Js function.

After analyzing the jQuery-based implementations of relevant PrimeFaces input component's, widget prototypes were extended to support changing the component's value client-side:

```
AutoComplete.getValue/setValue
InputText.getValue/setValue
InputTextarea.getValue/setValue
SelectBooleanCheckbox.getValue/setValue
SelectManyCheckbox.getValue/setValue
SelectOneMenu.getValue/setValue
Spinner.getValue/setValue
```

Extended access to a component's state was implemented as required:

- `InputText.enable/disable`: enable/disable `p:inputText`
- `SelectBooleanCheckbox.enable/disable`: enable/disable `p:selectBooleanCheckbox`
- `Dialog.getTitle/setTitle`: access a `p:dialog`'s title text
- `TabView.setTabTitle/getTabTitle`: access a `p:tab`'s title text
- `TabView.emphasizeTab`: icons for `p:tab` titles

Custom component rendering Modifications for custom rendering required overriding of entire widget prototypes:

- `LineChart`: jqPlot cartesian charts with x-axis values of date/datetime type
- `Schedule`: display the ISO week number

Patches Prototype method overrides are used to remove minor defects that were found in widget behaviour, e.g.:

- `Dialog.focusFirstInput`: suppress undesired `p:calendar` datetime selector po-pup within `p:dialog`
- `SelectOneMenu.bindKeyEvents`: fix for drop down item selection via keyboard

Js namespace and external libraries At the moment, the Js layer is not yet structured by introducing a global namespaces, since the major part of it are stateless event handler and utility functions, beside PrimeFaces widget prototype re-definitions. An important group of utility functions allows initiating AJAX request cycles from Js code by wrapping the `PrimeFaces.ajax.AjaxRequest` API. While PrimeFaces provides hidden `p:remoteCommand` components for exactly this purpose, these functions were nevertheless introduced to circumvent the dependency on `p:remoteCommand` definitions in facelets. This simplified the implementation of complex update logic part of specific UI features (item pickers, calculated field values) and enabled the workarounds using `pe:javascript` mentioned in section 4.6.1.3.

A number of Js libraries as well as additional plug-ins for libraries used by PrimeFaces were added overall, some of which declaring their own namespaces. As described more in detail in corresponding sections, some of them (marked with "*") were modified to a certain extent:

- jqPlot render plug-ins (Leonello, 2013)
- Really Simple Color Picker in jQuery* (Perera, 2008)
- Base64 encoding/decoding (Albert, 2013)
- JSON2 (Crockford, 2010)
- Datejs (Yoder and McGill, 2010)
- Raphaël*
- Raphael SketchPad*
- Js CommentStripper* (Dunn, 2013)
- `sprintf` for Js (Marasteanu, 2013)

4.6.1.8. CSS Resources

CSS files encapsulate appearance details such as layout issue fixes, a scheme of named colors and icon placement. The color scheme maps the `Color` enumeration constants introduced in section 4.4 to dedicated style classes in order to color e.g. datatable rows according to definitions from master data records. Applying style classes is supported by PrimeFaces components directly in most cases, e.g. by the `rowStyleClass` attribute of `p:dataTable`. Components such as `p:schedule` or `pe:timeline` need style classes tailored to their DOM structure in order to color displayed event items correctly.

Compliance with the various PrimeFaces themes was considered to the extent of layout problems only. There is no hue adjustment of enumerated color values to match the selected theme's colors, which might result in low contrast in some rare situations.

4.6.2. Navigation

JSF introduces facelets to define the layout of a browser page. There is a built-in support for conditional page navigation (`faces-config.xml`), as typically required by non-AJAX JSF applications. For example, after a HTML form submission, the user could be redirected to feedback pages showing either an error or success message. In the case of AJAX JSF applications, PPR allows to refresh components to facilitate the Single-Page Application (SPA) pattern. PPR relies on the component tree as defined in the facelet, hence it is not designed to change the structure of the entire page.[63] An extensive JSF AJAX UI comprising screens with different structure will therefore continue to consist of multiple pages, as is the case for the Phoenix CTMS UI (table 4.10). In this context, (page) navigation is the transition from one page to another, initiated by the user.

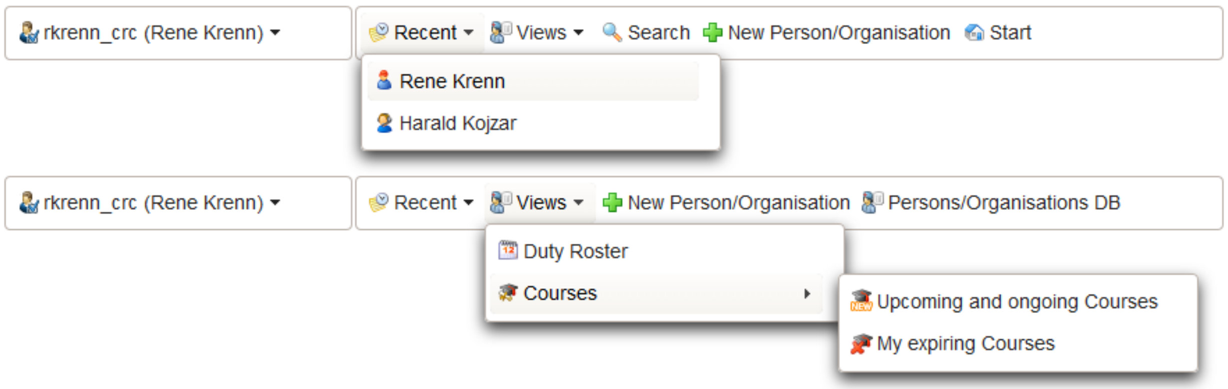


Figure 4.68.: The menu bar provides a navigation scheme characterzied by module home (upper menu bar) and entity view (lower menu bar) pages, as shown for the person/organisation module. A history menu allows to open items recently modified by the user. The view menu shows navigatable module home pages.

[63]Managed beans methods may manipulate the JSF component tree directly. There are issues with this technique and an ongoing specification request for JSF 2.3 (Tijms et al., 2013). A bad practice but a basic alternative is declaring components of any screen within one facelet, and hide (`rendered="false"`) them conditionally.

The login page (figure 4.55) is shown initially and forwards to the *start page* after successful authentication. UI screens of each system module can be divided into two types:

Module home Each system module will comprise a varying number of pages of different structure for consolidated views such as a overview lists, calendars or a query editor. They are referred to as *module home* pages and can be accessed by clicking corresponding menu items in the view menu of the *menu bar*. Configuration option allow to define the module home default pages, which will be shown initially.

Entity view As described in detail in section 4.6.4, a root entity and its associated detail records can be displayed and edited in the corresponding module's *entity view* page. Entity view pages can be accessed by clicking either of the navigation controls below:

1. buttons ("open item") of datatables and *item pickers* in edit areas
2. history menu items of the main menu bar (figure 4.68)
3. `ctsms:tagCloud` items of the start page

Entity view page URLs may contain a parameter for passing the root entity's ID in the query string, allowing to bookmark a specific item's entity view page.

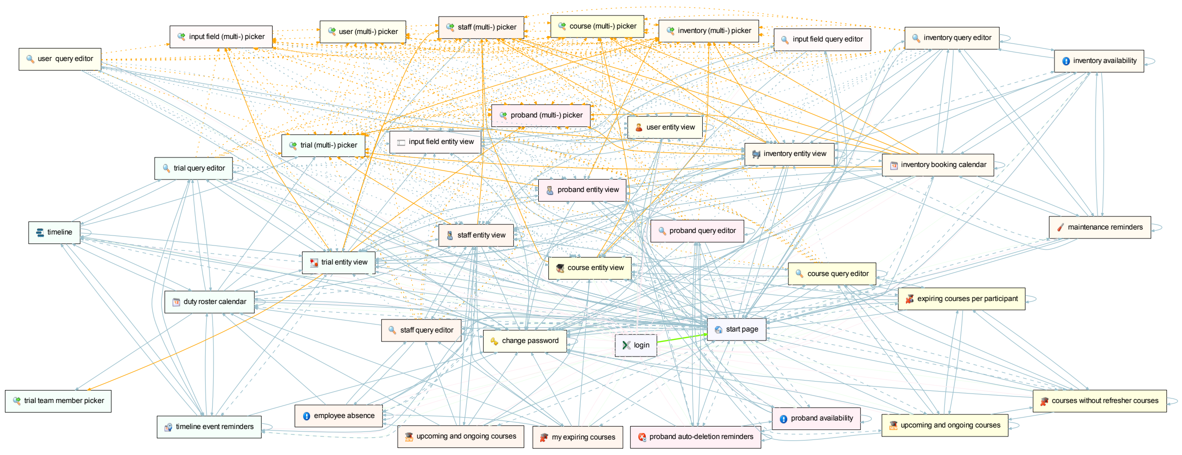


Figure 4.69.: The navigation graph shows possible navigation paths between UI pages. Lightblue lines denote a regular navigation path for opening a page in a permanent window, while orange lines mean opening a transient picker pop-up. Dashed lines indicate configuration options for module home default pages. Dotted lines denote possible pathways originating from a query editor UI, which depend on definitions of entity columns available to criterion terms.

The emerging navigation graph shown in figure 4.69 is a connected graph but does not represent a fully connected mesh[64]. It provides reasonable path lengths, optimized for frequent navigation patterns of users.

[64] n^2 navigation paths

page	window name	URL	parameter
login		`/login.jsf`	
start page	`portal`	`/portal.jsf`	
inventory module:		*/inventory*	
inventory query editor	`inventory_home`	`/inventorySearch.jsf`	`criteriaid`
inventory availability	`inventory_home`	`/inventoryStatusOverview.jsf`	
maintenance reminders	`inventory_home`	`/inventoryMaintenanceOverview.jsf`	
inventory booking calendar	`inventory_home`	`/inventoryBookingSchedule.jsf`	
inventory entity view	`inventory_entity`	`/inventory.jsf`	`inventoryid`
person/organisation module:		*/staff*	
person/organisation query editor	`staff_home`	`/staffSearch.jsf`	`criteriaid`
employee absence	`staff_home`	`/staffStatusOverview.jsf`	
upcoming and ongoing courses	`staff_home`	`/upcomingCourseOverview.jsf`	
my expiring courses	`staff_home`	`/expiringParticipationOverview.jsf`	
person/organisation entity view	`staff_entity`	`/staff.jsf`	`staffid`
course module:		*/course*	
course query editor	`course_home`	`/courseSearch.jsf`	`criteriaid`
upcoming and ongoing courses	`course_home`	`/adminUpcomingCourseOverview.jsf`	
courses without refresher courses	`course_home`	`/expiringCourseOverview.jsf`	
expiring courses per participant	`course_home`	`/adminExpiringParticipationOverview.jsf`	
course entity view	`course_entity`	`/course.jsf`	`courseid`
trial module:		*/trial*	
trial query editor	`trial_home`	`/trialSearch.jsf`	`criteriaid`
timeline event reminders	`trial_home`	`/timelineEventOverview.jsf`	
timeline	`trial_home`	`/trialTimeline.jsf`	
duty roster calendar	`trial_home`	`/dutyRosterSchedule.jsf`	
trial entity view	`trial_entity`	`/trial.jsf`	`trialid`
proband module:		*/proband*	
proband query editor	`proband_home`	`/probandSearch.jsf`	`criteriaid`
proband availability	`proband_home`	`/probandStatusOverview.jsf`	
proband auto-deletion reminders	`proband_home`	`/autoDeletionProbandOverview.jsf`	
proband entity view	`proband_entity`	`/proband.jsf`	`probandid`
input field module:		*/inputfield*	
input field query editor	`input_field_home`	`/inputFieldSearch.jsf`	`criteriaid`
input field entity view	`input_field_entity`	`/inputField.jsf`	`inputfieldid`
user module:		*/user*	
user query editor	`user_home`	`/userSearch.jsf`	`criteriaid`
change password	`user_home`	`/changePassword.jsf`	
user entity view	`user_entity`	`/user.jsf`	`userid`
entity pickers:		*/shared*	
inventory (multi-) picker	`picker_`*n*	`/inventoryPicker.jsf`	`criteriaid`
person/organisation (multi-) picker	`picker_`*n*	`/staffPicker.jsf`	`criteriaid`
trial team member picker	`picker_`*n*	`/teamMemberPicker.jsf`	`trialid`
course (multi-) picker	`picker_`*n*	`/coursePicker.jsf`	`criteriaid`
trial (multi-) picker	`picker_`*n*	`/trialPicker.jsf`	`criteriaid`
proband (multi-) picker	`picker_`*n*	`/probandPicker.jsf`	`criteriaid`
input field (multi-) picker	`picker_`*n*	`/inputFieldPicker.jsf`	`criteriaid`
user (multi-) picker	`picker_`*n*	`/userPicker.jsf`	`criteriaid`

Table 4.10.: This overview shows module home and entity view pages, a user can open. Each page represents a unique UI screen, defined by a particular facelet from figure 4.66. A page is opened in the browser upon the HTTP GET request coming from the `window.open` method. `window.open` takes a URL and window name argument.

Navigation triggered from PrimeFaces UI controls such as `p:commandButton` or `p:menuItem` can be wired using the Js event model. A component's HTML DOM element can fire a Js function (callback) assigned to a supported event listener such as `onclick`. By assigning the `window.open` function, it will cause browsers to request a new page by sending a HTTP GET request when clicking the control. Aside the mandatory page URL parameter, `window.open` can take an optional target argument to specify a browser window name. If a distinct window name is referred for the first time, modern browsers handle opening new windows in two different ways by analyzing remaining `window.open` arguments:

Tabbed browsing Instead of a separate browser window per URL, pages are opened in tabs of the web browser application. This feature is utilized to keep multiple pages open at the same time, thus allowing users to switch between pages instantly, without expicit navigation. Tabs can be arranged on multiple monitors if available. Power users prefer this mode of browsing enterprise web applications, since as much information as possible is displayed at once, avoiding the need of tedious navigation and remembering content. Contrary, the number of tabs might be confusing for users new to the system. Once a tab is opened by a `window.open` call, the specified window name is assigned. A subsequent `window.open` invocation referring to the same window name will load the page in the exsiting tab. This behaviour allows establishing a scheme of tabs to control the number of opened pages. The scheme can be configured (table 4.11), but users preferred to stay with the full-scale tab scheme (figure 4.70) as expected.

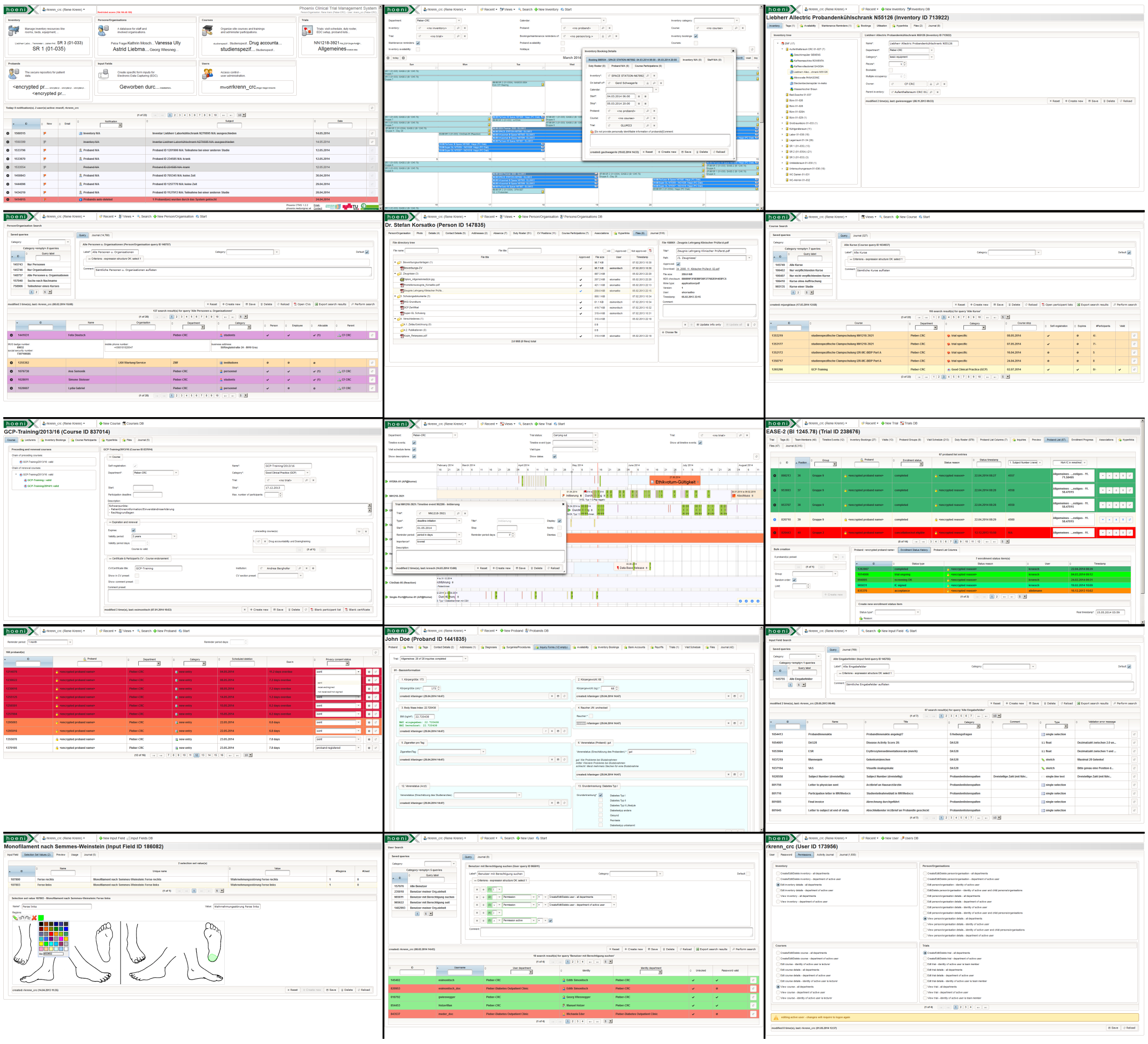

Liebherr Allectric Probandenkühlschrank N55126 (Inventory ID 713922)
Dr. Stefan Korsatko (Person ID 147835)
GCP-Training/2013/16 (Course ID 837014)
EASE-2 (BI 1245.78) (Trial ID 238676)
John Doe (Proband ID 1441835)
Monofilament nach Semmes-Weinstein (Input Field ID 188082)
rkrenn_crc (User ID 173956)

Figure 4.70.: With the full-scale browser tab scheme, a user session comprises up to 15 opened pages.

window name Js variable	configurations for tab schemes		
	single window	**three tabs**	**all possible tabs**
	window names		
`PORTAL_WINDOW_NAME`	“*_self*”	“portal”	“portal”
`INVENTORY_HOME_WINDOW_NAME`	“*_self*”	“home”	“inventory_home”
`INVENTORY_ENTITY_WINDOW_NAME`	“*_self*”	“entity”	“inventory_entity”
`STAFF_HOME_WINDOW_NAME`	“*_self*”	“home”	“staff_home”
`STAFF_ENTITY_WINDOW_NAME`	“*_self*”	“entity”	“staff_entity”
`COURSE_HOME_WINDOW_NAME`	“*_self*”	“home”	“course_home”
`COURSE_ENTITY_WINDOW_NAME`	“*_self*”	“entity”	“course_entity”
`TRIAL_HOME_WINDOW_NAME`	“*_self*”	“home”	“trial_home”
`TRIAL_ENTITY_WINDOW_NAME`	“*_self*”	“entity”	“trial_entity”
`PROBAND_HOME_WINDOW_NAME`	“*_self*”	“home”	“proband_home”
`PROBAND_ENTITY_WINDOW_NAME`	“*_self*”	“entity”	“proband_entity”
`INPUT_FIELD_HOME_WINDOW_NAME`	“*_self*”	“home”	“input_field_home”
`INPUT_FIELD_ENTITY_WINDOW_NAME`	“*_self*”	“entity”	“input_field_entity”
`USER_HOME_WINDOW_NAME`	“*_self*”	“home”	“user_home”
`USER_ENTITY_WINDOW_NAME`	“*_self*”	“entity”	“user_entity”

Table 4.11.: A unique name per distinguished window name Js variable allows displaying all possible tabs side-by-side in the browser window. Because this full scale configuration does not yet allow to compare items, multiple entity view tabs can be opened simultaneously using a window name format pattern. As an example, a unique window name for each proband entity view page to open is derived by inserting the proband record ID for the placeholder in “proband_entity_%s”. The browser tab scheme can be choked, starting from the full set of implemented window name variables. A tab scheme restricted to a single tab only can be configured by using the same window name (“*_self*”) for any window name variable.

Pop-up A pop-up displays an opened page in a separate browser window. While module pages described so far are basically displayed by browser tabs, pop-ups are the decent counterpart for intermediate interactions. After opening, pop-up windows can be closed again programatically using `window.close`. The implemented *item picker* mechanism allows selecting a root entity item by using the query editor (figure 4.71). After instantly executing a query, a desired item is picked from the result set using the relevant result row's ↩ button (“pick item”). While the query editor pop-up is immediately closed again when selecting single items, the popup remains open when picking multiple items in order to be closed manually.

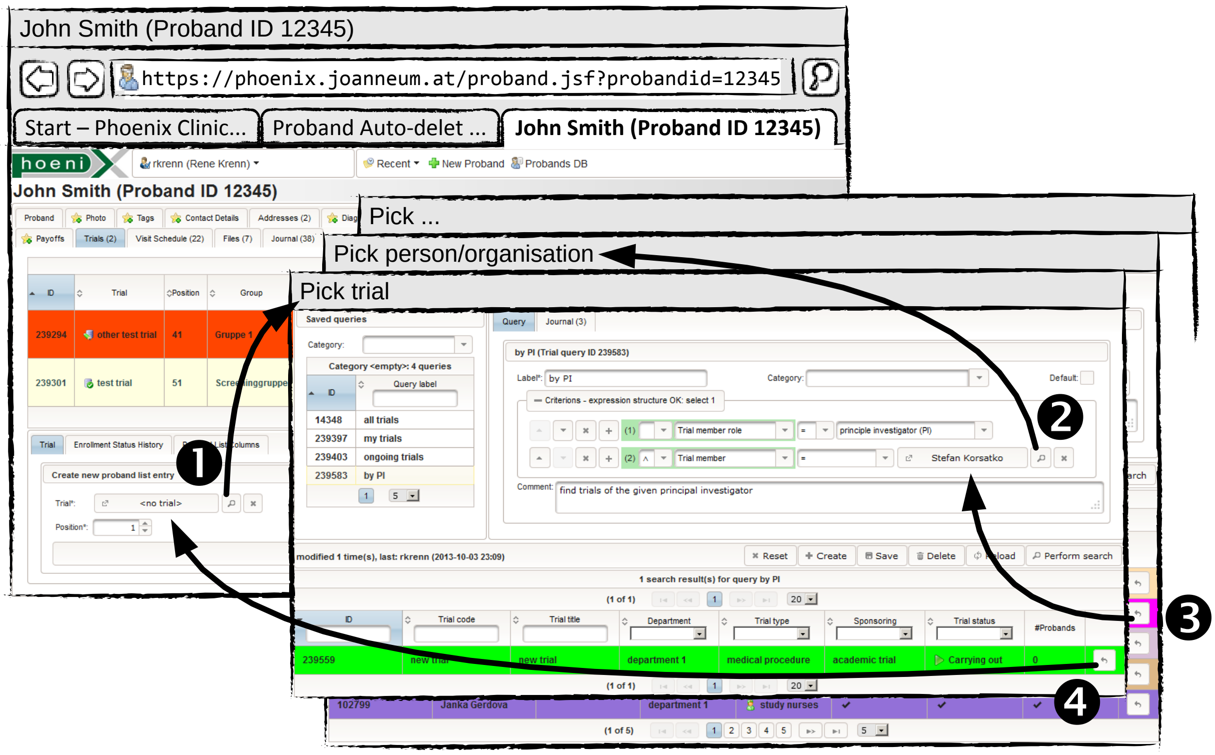

John Smith (Proband ID 12345)
https://phoenix.joanneum.at/proband.jsf?probandid=12345
Start – Phoenix Clinic...
Proband Auto-delet ...
John Smith (Proband ID 12345)
Pick ...
Pick person/organisation
Pick trial
1
2
3
4

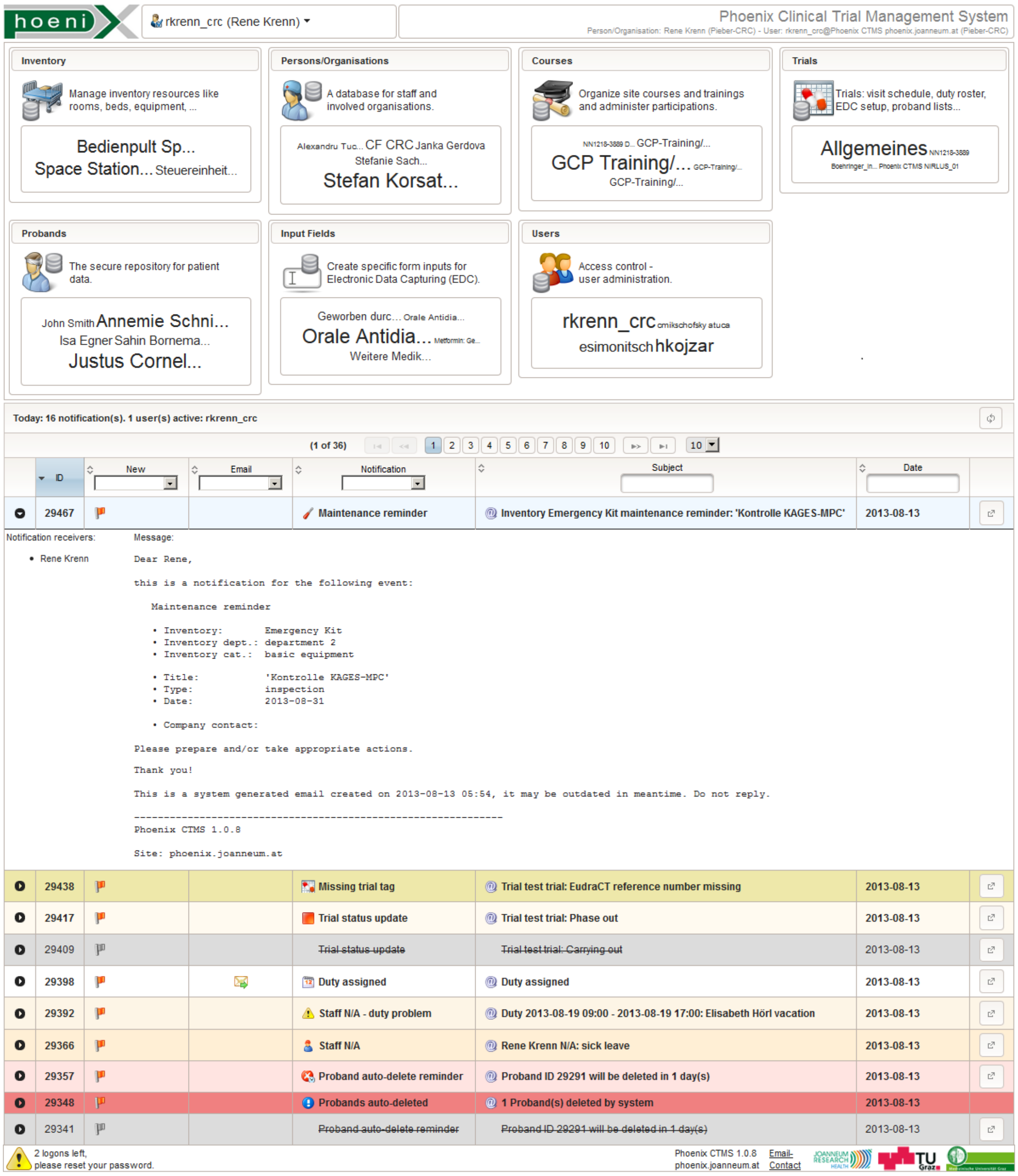
Figure 4.71.: The browser's main window displays UI pages using tabbed browsing. Screens with inputs for root item selection provide a ⌕ button to open the transient picker pop-up (1) to show a query editor UI for the corresponding module. The desired item is selected from the listed result set (4). Since a query criterion may refer to root item itself, a nested picker pop-ups can be opened in this case (2) to select the item designated for a criterion's comparison value (3). This is implemented using special windows names for picker pop-ups, which include the nesting depth n of pop-up windows (table 4.10).

Managing multiple browser windows involves additional issues with handling expired sessions. As stated in the previous section, automatic redirection to the login page after session expiration is prepared for both HTTP POST (AJAX page interactions) and GET requests (`window.open` navigation actions). After a new session is established when the user re-authenticates, a forwarding logic tries to re-open the original page. When a user starts an PPR interaction in one of the remaining windows referring to an obsolete session, the forwarding mechanism obtains the new session ID by re-opening the page. A representative message[65] will inform the user what happened. It is desired to always show this message or the login page in the *originating* browser window. However, this would not be the case when a navigation action opens a page in a new tab. This is why `window.open` invocations are actually encapsulated by an AJAX round-trip. The next section will describe the implementation of this technique in detail, which applies to navigation controls globally.

4.6.3. Start Page

After successful login, the authenticated (active) user is forwarded to a start page (figure 4.72). This portal page shows recent activities at a glance and acts as entry point for navigation. It subdivides into the upper and lower area, whose designs are presented in detail.

[65]"Original action not completed, please retry."



rkrenn_crc (Rene Krenn)

Phoenix Clinical Trial Management System
Person/Organisation: Rene Krenn (Pieber-CRC) - User: rkrenn_crc@Phoenix CTMS phoenix.joanneum.at (Pieber-CRC)

Inventory
Manage inventory resources like rooms, beds, equipment, ...
Bedienpult Sp... Space Station... Steuereinheit...

Persons/Organisations
A database for staff and involved organisations.
Alexandru Tuc... CF CRC Janka Gerdova Stefanie Sach... Stefan Korsat...

Courses
Organize site courses and trainings and administer participations.
NN1218-3889 D... GCP-Training/... GCP Training/... GCP-Training/... GCP-Training/...

Trials
Trials: visit schedule, duty roster, EDC setup, proband lists...
Allgemeines NN1218-3889 Boehringer_In... Phoenix CTMS NIRLUS_01

Probands
The secure repository for patient data.
John Smith Annemie Schni... Isa Egner Sahin Bornema... Justus Cornel...

Input Fields
Create specific form inputs for Electronic Data Capturing (EDC).
Geworben durc... Orale Antidia... Orale Antidia... Metformin: Ge... Weitere Medik...

Users
Access control - user administration.
rkrenn_crc cmikschofsky atuca esimonitsch hkojzar

Today: 16 notification(s). 1 user(s) active: rkrenn_crc

(1 of 36) 1 2 3 4 5 6 7 8 9 10 | 10

ID	New	Email	Notification	Subject	Date
29467	●		Maintenance reminder	Inventory Emergency Kit maintenance reminder: 'Kontrolle KAGES-MPC'	2013-08-13

Notification receivers:

- Rene Krenn

Message:

```
Dear Rene,

this is a notification for the following event:

   Maintenance reminder

   • Inventory:       Emergency Kit
   • Inventory dept.: department 2
   • Inventory cat.:  basic equipment

   • Title:           'Kontrolle KAGES-MPC'
   • Type:            inspection
   • Date:            2013-08-31

   • Company contact:

Please prepare and/or take appropriate actions.

Thank you!

This is a system generated email created on 2013-08-13 05:54, it may be outdated in meantime. Do not reply.

---------------------------------------------------------------
Phoenix CTMS 1.0.8

Site: phoenix.joanneum.at
```

ID	New	Email	Notification	Subject	Date
29438	●		Missing trial tag	Trial test trial: EudraCT reference number missing	2013-08-13
29417	●		Trial status update	Trial test trial: Phase out	2013-08-13
29409			~~Trial status update~~	~~Trial test trial: Carrying out~~	2013-08-13
29398	●	✉	Duty assigned	Duty assigned	2013-08-13
29392	●		Staff N/A - duty problem	Duty 2013-08-19 09:00 - 2013-08-19 17:00: Elisabeth Hörl vacation	2013-08-13
29366	●		Staff N/A	Rene Krenn N/A: sick leave	2013-08-13
29357	●		Proband auto-delete reminder	Proband ID 29291 will be deleted in 1 day(s)	2013-08-13
29348	●		Probands auto-deleted	1 Proband(s) deleted by system	2013-08-13
29341			~~Proband auto-delete reminder~~	~~Proband ID 29291 will be deleted in 1 day(s)~~	2013-08-13

2 logons left, please reset your password.

Phoenix CTMS 1.0.8 phoenix.joanneum.at Email-Contact

Figure 4.72.: The portal page gives a comprehensive overview about recent user activities.

4.6.3.1. System Modules Overview

Each module is depicted by a summary panel containing a link icon to open the module's home tab. With the default configuration, the links will open query editors as a starting point in order to run a query and navigate to items of interest directly. Below the link icon and a short module description, *tag clouds* show the modules' root entity items modified by users recently (figure 4.72).

The implementation of this page section is analyzed in detail to provide a representative insight into the usage of facelets, managed beans and model classes. As a starting point, the facelet fragment in listing 4.29 shows how module summary boxes are realized using a `p:dataGrid`. Each of its columns contains the module summary panel, represented by a `p:panel` component.

```
...
<p:dataGrid <!-- navigation overview of application modules -->
    id="moduleitems"
    value="#{portalBean.portalModuleItems}" <!-- custom model with module descriptions, icons, tag clouds, etc. -->
    var="module" <!-- item iteration variable of type at.zmf.crc.ctsms.web.model.shared.PortalModuleItem -->
    paginator="false"
    columns="4"
    styleClass="ctsms-portalitems-datagrid">
    <!-- the module summary box: -->
    <p:column>
        <p:panel>
            <f:facet name="header">
                <h:outputText value="#{module.label}" />
            </f:facet>
            <h:panelGrid columns="2" cellpadding="0" styleClass="ctsms-input-panelgrid"
                columnClasses="ctsms-portalitemicon-column,ctsms-portalitemdescription-column">
                <!-- A Js function like openProbandHome that prepares and invokes window.open is assigned to the the icon
                    onclick event handler: -->
                <h:panelGroup><div class="ctsms-portalitem-icon #{module.icon}" onclick="#{module.onClick};"></div></
                    h:panelGroup>
                <!-- A short module description: -->
                <ui:include src="/META-INF/includes/shared/multiLineOutputText.xhtml">
                    <ui:param name="multiLineText" value="#{module.description}"/>
                    <ui:param name="rendered" value="true"/>
                    <ui:param name="multiLineTextClass" value=""/>
                </ui:include>
            </h:panelGrid>
            <!-- The custom tag cloud component displays recently edited items and allows to open them. An entity view is
                opened by a corresponding Js functions like openProband. -->
            <ctsms:tagCloud styleClass="ctsms-tagcloud" model="#{module.tagModel}" />
        </p:panel>
    </p:column>
</p:dataGrid>
...
<p:remoteCommand <!-- a remote command to perform window.open invocations (GET request) by callback of a successful AJAX
     response (POST request) -->
    process="@this"
    name="viewAliveRC"
    actionListener="#{applicationScopeBean.viewAliveEcho}"
    oncomplete="handleViewAliveEcho(xhr, status, args)" />
...
```

Listing 4.29: The start page's upper area shows a PrimeFaces `p:dataGrid` component, which arranges module summary panels. Each `p:panel` component contains a plain HTML `div` for the module home tab link icon, a multiline text for a short module description and a `ctsms:tagCloud` component showing recently edited root entities.

Inside a summary panel, the link icon and module description text are arranged side-by-side using a native table component (`h:panelGrid`). Since JSF 1.2 (JSR-252), facelets may contain plain HTML markup aside JSF components without precautions. This improvement comes into effect for the link icon, which is a plain `div` element. It displays the icon image using a CSS style class that refers to a corresponding background image. The `onclick` event handler binds an applicable Js function to open the module home tab. By example, `openProbandHome` shown in listing 4.30 performs navigation by employing a PPR to invoke `window.open` for opening the proband module's default home tab.

```
//constants are exported upon initial page load:
var PROBAND_URL = '/proband/proband.jsf';
var PROBAND_ID = 'probandid';
var PROBAND_ENTITY_WINDOW_NAME = 'proband_entity_view_%d';
var PROBAND_START_URL = '/proband/probandSearch.jsf';
var PROBAND_HOME_WINDOW_NAME = 'proband_home';

var AJAX_VIEW_ALIVE_JS_CALLBACK = 'viewAliveJsCallback';
...

//common.js:
function openProband(probandId) { //open the proband entity view:
    if (typeof probandId !== 'undefined'  probandId) {
        _openEntity(PROBAND_URL + '?' + PROBAND_ID + '=' + encodeURIComponent(probandId), sprintf(PROBAND_ENTITY_WINDOW_NAME,
              probandId));
    }
}
function _openEntity(url, windowName) {
    _viewAlive('_openEntityCallback', {'url':encodeBase64(url, false), 'entityWindowName':windowName});
}
function _openEntityCallback(args) { //entity view window.open callback:
    window.open(decodeBase64(args.url), args.entityWindowName).focus();
}

function openProbandHome() { //open the initial proband home view:
    _openHome(PROBAND_START_URL, PROBAND_HOME_WINDOW_NAME);
}
function _openHome(url,windowName) {
    _viewAlive('_openHomeCallback', {'url':encodeBase64(url,false), 'homeWindowName':windowName});
}
function _openHomeCallback(args) { //home view window.open callback:
    window.open(decodeBase64(args.url), args.homeWindowName).focus();
}

function _viewAlive(functionName, args) { //invoke the viewAliveRC p:remoteCommand component:
    if (_testFunction(window[functionName])) {
        var viewAliveCallbackParams = {};
        viewAliveCallbackParams[AJAX_VIEW_ALIVE_JS_CALLBACK] = functionName;
        if (args) { //convert (arbitrary) argument data structure to a JSON string:
            viewAliveCallbackParams[AJAX_VIEW_ALIVE_JS_CALLBACK_ARGS] = JSON.stringify(args);
        }
        viewAliveRC(prepareRemoteCommandParameters(viewAliveCallbackParams));
    }
}
function handleViewAliveEcho(xhr, status, args) { //handle ApplicationScopeBean.viewAliveEcho AJAX response:
    if (_testFlag(args,AJAX_OPERATION_SUCCESS)   _testPropertyExists(args,AJAX_VIEW_ALIVE_JS_CALLBACK)) {
        var ajaxViewAliveJsCallback = args[AJAX_VIEW_ALIVE_JS_CALLBACK];
        if (_testFunction(window[ajaxViewAliveJsCallback])) { //check if the callback function is a valid function
            if (_testPropertyExists(args,AJAX_VIEW_ALIVE_JS_CALLBACK_ARGS)) {
                //parse the reflected argument JSON string and invoke the callback:
                window[ajaxViewAliveJsCallback](JSON.parse(args[AJAX_VIEW_ALIVE_JS_CALLBACK_ARGS]));
            } else {
                window[ajaxViewAliveJsCallback]();
            }
        }
    }
}
```

Listing 4.30: The callback mechanism for the UI's browser tab navigation is powered by Js utility methods.

PrimeFaces' `p:remoteCommand` is an invisible component to provide a convenient Js API for PPR. As with any other component, an optional Js callback can be bound using the `oncomplete` attribute, which will be executed when the AJAX request completed successfully. The `viewAliveRC` `p:remoteCommand` component's action listener attribute binds to the stateless `ApplicationScopeBean.viewAliveEcho` method (listing 4.31). It relays `window.open` parameters (page URL, window name) and the Js callback method name that finally invokes `window.open` to effectively perform the page navigation. The reason for this callback mechanism is to cause an AJAX roundtrip, which will trigger JSF's `ViewExpiredException` in the case the session was invalidated since the page was loaded. As desired, the expiry redirection (section 4.6.1.6) will take place in the originating window just as any other AJAX operation.

```
@ManagedBean(eager=true)
@ApplicationScoped
public class ApplicationScopeBean {

    //action listener method for the viewAliveRC p:remoteCommand component:
    public void viewAliveEcho() {
        FacesContext context = FacesContext.getCurrentInstance();
        Map parameterMap = context.getExternalContext().getRequestParameterMap();
        //get expected parameters posted by the Js _viewAlive method:
        String ajaxViewAliveJsCallback = (String) parameterMap.get(JSValues.AJAX_VIEW_ALIVE_JS_CALLBACK.toString());
        String ajaxViewAliveJsCallbackArgs = (String) parameterMap.get(JSValues.AJAX_VIEW_ALIVE_JS_CALLBACK_ARGS.toString());
        //org.primefaces.context.RequestContext is an API to tweak the ongoing JSF AJAX request cycle:
        RequestContext requestContext = RequestContext.getCurrentInstance();
        if (requestContext != null) {
            requestContext.addCallbackParam(JSValues.AJAX_OPERATION_SUCCESS.toString(), true);
            //Retrieved parameter values are appended to the AJAX response without any modifications. They comprise items of
                 the args array of the Js handleViewAliveEcho(xhr, status, args) callback.
            requestContext.addCallbackParam(JSValues.AJAX_VIEW_ALIVE_JS_CALLBACK.toString(), ajaxViewAliveJsCallback);
            if (ajaxViewAliveJsCallbackArgs != null) {
                requestContext.addCallbackParam(JSValues.AJAX_VIEW_ALIVE_JS_CALLBACK_ARGS.toString(),
                     ajaxViewAliveJsCallbackArgs);
            }
        }
    }

    ...
```

Listing 4.31: `ApplicationScopeBean`'s methods such as `viewAliveEcho` are available to any session- or view-scoped managed bean.

```
@ManagedBean
@ViewScoped
public class PortalBean extends ManagedBeanBase {
    //model fields:
    private ArrayList<PortalModuleItem> portalModuleItems; //models for module summary boxes
    private NotificationLazyModel notificationModel; //datatable lazy model for the notification list

    ...

    public PortalBean() {
        super();
        notificationModel = new NotificationLazyModel(); //todo: Spring DI
    }

    @PostConstruct
    private void init() { //initialize models:
        updateModels();
    }

    //action listener method overrride to support an optional refresh button:
    @Override protected boolean reloadImpl() {
        updateModels();
        return true;
    }

    private void updateModels() {
        if (portalModuleItems == null) { //initialize a static list of PortalModuleItems:
            portalModuleItems = new ArrayList<PortalModuleItem>();
            portalModuleItems.add(new PortalModuleItem(MessageCodes.INVENTORY_PORTAL_ITEM_LABEL), ...
            portalModuleItems.add(new PortalModuleItem(MessageCodes.STAFF_PORTAL_ITEM_LABEL), ...
            portalModuleItems.add(new PortalModuleItem(MessageCodes.COURSE_PORTAL_ITEM_LABEL), ...
            portalModuleItems.add(new PortalModuleItem(MessageCodes.TRIAL_PORTAL_ITEM_LABEL), ...
            portalModuleItems.add(new PortalModuleItem(MessageCodes.PROBAND_PORTAL_ITEM_LABEL,"ctsms-largeicon-probandhome",
                 MessageCodes.PROBAND_PORTAL_ITEM_DESCRIPTION,"openProbandHome()",JournalModule.PROBAND_JOURNAL));
            portalModuleItems.add(new PortalModuleItem(MessageCodes.INPUT_FIELD_PORTAL_ITEM_LABEL, ...
            portalModuleItems.add(new PortalModuleItem(MessageCodes.USER_PORTAL_ITEM_LABEL, ...
        }
        //refresh the PortalModuleItem models' state:
        for (Iterator<PortalModuleItem> it = portalModuleItems.iterator(); it.hasNext();) {
            it.next().updateTagModel();
        }
        //refresh the notification list LazyDataModel's state:
        notificationModel.updateRowCount();
        ...
    }
    ...
```

Listing 4.32: The portal page facelet is backed by the `PortalBean` view-scoped managed bean.

The portal page's managed bean (PortalBean, listing 4.32)) provides accessors for fields holding tailored model beans as well as an action listener method for refreshing[66] data provided by these model beans.

Typically, model beans represent POJO graphs, whose fields are bound by EL expressions of component attribute values. The p:panelGrid's data model comprises a static list of PortalModuleItem instances (listing 4.33). Each PortalModuleItem POJO provides fields with dynamic data for a module summary panel:

- localized panel title label
- icon style class
- localized description text
- Js method for the icon div's onlick event handler to open the module's initial home tab
- DefaultTagCloudModel containing a list of DefaultTagCloudItems, each made up of:
    - clipped tag cloud item label
    - Js method for the tag cloud item's onlick event handler to open the corresponding entity view tab
    - tag cloud item strength (size)

According to PrimeFaces' p:tagCloud component implementation, ctsms:tagCloud is backed by the DefaultTagCloudModel data model, which represents a list of DefaultTagCloudItems with details of each tag cloud item to be displayed. When clicking a tag cloud item, the denoted root entity is supposed to be opened. Hence, the tag cloud item's onclick event handler is bound to a respective Js callback such as openProband for navigating to the corresponding entity view. DefaultTagCloudItems are populated using a suitable service method prepared for this purpose (JournalService.getActivityTags). Each DefaultTagCloudItem's strength field reflects the item's size in order to illustrate frequency of CRUD operations of the root entity record or its detail records.

[66]A corresponding refresh button is however not yet designed for this particular page part at the moment.

```
//the model for a module's summary box:
public class PortalModuleItem {
    //getters and setters for:
    private String label;
    private String icon;
    private String description;
    private String onClick;

    private JournalModule module;
    private DefaultTagCloudModel tagModel; //a list of DefaultTagCloudItems

    ...
    //method to refresh the model state:
    public void updateTagModel() {
        tagModel.clear();
        Collection<ActivityTagVO> activityTags = null;
        try { //service method request to retrieve the first-n root entities of latest journal system messages for the given
              module.
            activityTags = WebUtil.getServiceLocator().getJournalService().getActivityTags(WebUtil.getAuthentication(), module,
                    null, null, WebUtil.getUser().getDepartment().getId(), Settings.getIntNullable(SettingCodes.
                   MAX_TAG_CLOUD_ITEMS, Bundle.SETTINGS, DefaultSettings.MAX_TAG_CLOUD_ITEMS));
        } catch (AuthenticationException e) { //logout, if the authentication fails:
            WebUtil.publishException(e); //taken up by ExceptionHandler (the custom javax.faces.context.ExceptionHandlerWrapper
                 ) to trigger PRG
        } catch (Exception e) { //e.g. no permission
            //no tag cloud items ...
        }
        if (activityTags != null) { //populate the DefaultTagCloudModel:
            Long maxCount = null;
            for (Iterator<ActivityTagVO> it = activityTags.iterator(); it.hasNext()) {
                //an ActivityTagVO contains a JournalEntry referring the root entity and the total count of JournalEntries for
                     this root entity:
                ActivityTagVO activityTag = it.next();
                DefaultTagCloudItem tagCloudItem = new DefaultTagCloudItem();
                switch (module) {
                    case PROBAND_JOURNAL: //populate the DefaultTagCloudItem:
                        if (count != null  count > 0) {
                            tagCloudItem.setLabel(CommonUtil.clipString(activityTag.getJournalEntry().getProband().getName()));
                            tagCloudItem.setOnclick(MessageFormat.format("openProband({0})", Long.toString(activityTag.
                                 getJournalEntry().getProband().getId())));
                            tagCloudItem.setStrength(activityTag.getCount());
                            tagModel.addTag(tagCloudItem);
                        }
                        break;
                    case ...
                }

            }
            //normalize strength values ...
        }
    }
    ...

//the tag cloud item model:
public class DefaultTagCloudItem implements TagCloudItem {

    private String label;
    private String onclick;
    private int strength = 1;

    ...
```

Listing 4.33: Data models for JSF components such as `PortalModuleItem` and `DefaultTagCloudItem` represent simple POJOs. As shown in the example, they typically integrate methods for populating their fields from the database.

4.6.3.2. Notification messages

The portal page's lower section shows the active user identity's *notification messages* (figure 4.72). The whole history of notifications is presented by a datatable with familiar PSF capabilities. Besides being displayed in the summary list, notifications will be sent to enabled email addresses of recipients. Sending notifications is implemented by a *consumer-producer* pattern. `Notification` and `NotificationRecipient` entities (figure 4.73) form the underlying *queue* table.

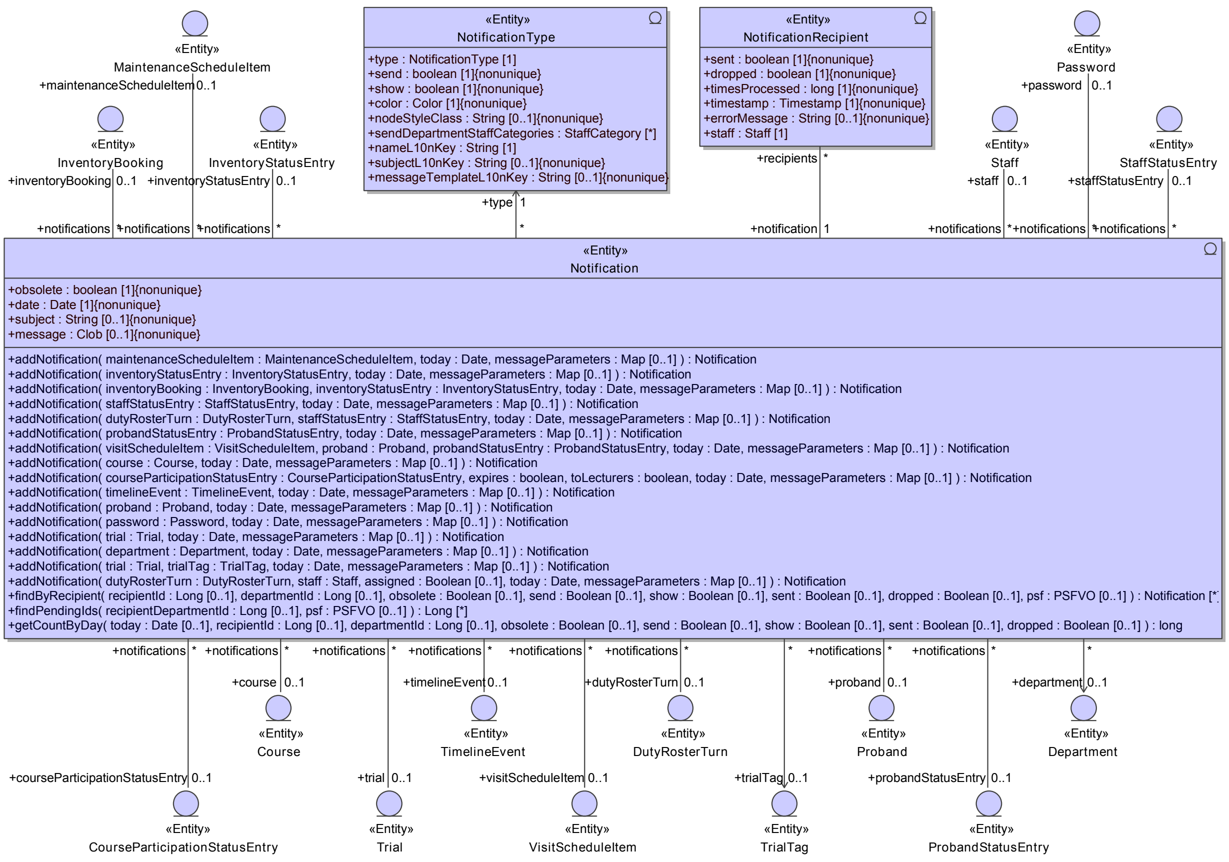

«Entity»
NotificationType
+type : NotificationType [1]
+send : boolean [1]{nonunique}
+show : boolean [1]{nonunique}
+color : Color [1]{nonunique}
+nodeStyleClass : String [0..1]{nonunique}
+sendDepartmentStaffCategories : StaffCategory [*]
+nameL10nKey : String [1]
+subjectL10nKey : String [0..1]{nonunique}
+messageTemplateL10nKey : String [0..1]{nonunique}
«Entity»
NotificationRecipient
+sent : boolean [1]{nonunique}
+dropped : boolean [1]{nonunique}
+timesProcessed : long [1]{nonunique}
+timestamp : Timestamp [1]{nonunique}
+errorMessage : String [0..1]{nonunique}
+staff : Staff [1]
«Entity»
Notification
+obsolete : boolean [1]{nonunique}
+date : Date [1]{nonunique}
+subject : String [0..1]{nonunique}
+message : Clob [0..1]{nonunique}
+addNotification(maintenanceScheduleItem : MaintenanceScheduleItem, today : Date, messageParameters : Map [0..1]) : Notification
+addNotification(inventoryStatusEntry : InventoryStatusEntry, today : Date, messageParameters : Map [0..1]) : Notification
+addNotification(inventoryBooking : InventoryBooking, inventoryStatusEntry : InventoryStatusEntry, today : Date, messageParameters : Map [0..1]) : Notification
+addNotification(staffStatusEntry : StaffStatusEntry, today : Date, messageParameters : Map [0..1]) : Notification
+addNotification(dutyRosterTurn : DutyRosterTurn, staffStatusEntry : StaffStatusEntry, today : Date, messageParameters : Map [0..1]) : Notification
+addNotification(probandStatusEntry : ProbandStatusEntry, today : Date, messageParameters : Map [0..1]) : Notification
+addNotification(visitScheduleItem : VisitScheduleItem, proband : Proband, probandStatusEntry : ProbandStatusEntry, today : Date, messageParameters : Map [0..1]) : Notification
+addNotification(course : Course, today : Date, messageParameters : Map [0..1]) : Notification
+addNotification(courseParticipationStatusEntry : CourseParticipationStatusEntry, expires : boolean, toLecturers : boolean, today : Date, messageParameters : Map [0..1]) : Notification
+addNotification(timelineEvent : TimelineEvent, today : Date, messageParameters : Map [0..1]) : Notification
+addNotification(proband : Proband, today : Date, messageParameters : Map [0..1]) : Notification
+addNotification(password : Password, today : Date, messageParameters : Map [0..1]) : Notification
+addNotification(trial : Trial, today : Date, messageParameters : Map [0..1]) : Notification
+addNotification(department : Department, today : Date, messageParameters : Map [0..1]) : Notification
+addNotification(trial : Trial, trialTag : TrialTag, today : Date, messageParameters : Map [0..1]) : Notification
+addNotification(dutyRosterTurn : DutyRosterTurn, staff : Staff, assigned : Boolean [0..1], today : Date, messageParameters : Map [0..1]) : Notification
+findByRecipient(recipientId : Long [0..1], departmentId : Long [0..1], obsolete : Boolean [0..1], send : Boolean [0..1], show : Boolean [0..1], sent : Boolean [0..1], dropped : Boolean [0..1], psf : PSFVO [0..1]) : Notification [*]
+findPendingIds(recipientDepartmentId : Long [0..1], psf : PSFVO [0..1]) : Long [*]
+getCountByDay(today : Date [0..1], recipientId : Long [0..1], departmentId : Long [0..1], obsolete : Boolean [0..1], send : Boolean [0..1], show : Boolean [0..1], sent : Boolean [0..1], dropped : Boolean [0..1]) : long
«Entity» MaintenanceScheduleItem
«Entity» InventoryBooking
«Entity» InventoryStatusEntry
«Entity» Password
«Entity» Staff
«Entity» StaffStatusEntry
«Entity» Course
«Entity» TimelineEvent
«Entity» DutyRosterTurn
«Entity» Proband
«Entity» Department
«Entity» CourseParticipationStatusEntry
«Entity» Trial
«Entity» VisitScheduleItem
«Entity» TrialTag
«Entity» ProbandStatusEntry
+type 1
+recipients *
+notification 1
+notifications *

Figure 4.73.: When joined, `Notification` and `NotificationRecipient` represent a queue table for sending notification emails asynchronously.

Producers At the moment, 20 different types of notifications (table 4.12) are generated by two types of sources:

CRUD operations: Modifying item containing scheduling information that would lead to conflicts like temporal overlapping is restricted by input validation in obvious cases only, such as booking an inventory for the same period of time. If conflicts arise, the UI tolerates collisions as a consistent state in general and just indicates them. Users decide whether to react or ignore collisions in the end. To promote this less restrictive paradigm, *notification messages* are sent to relevant recipients in order to draw attention. This is supposed to help resolving situations like duty assignments overlapping employee's absence entries. The anticipated behaviour is a timely correction of a relevant record, which renders the notification obsolete. If it was not sent already, the `Notification.obsolete` flag indicates to ignore the message when processing notifications for sending. The start page indicates obsolete (revoked) notifications in the notification datatable using rows with ~~strikethrough~~ text decoration (figure 4.72). Another type of update actions that produce notification messages include changing an item's status, such as the course participation status (`CourseParticipationStatusEntry.status`) or a trial's status (`Trial.status`). Subsequent status transitions need to revoke previous notifications regarding the same entity. To implement the logic for updating the `obsolete` flags of preceding notifications, it is required to persist a relationship with the root or detail entity each `Notification` refers to.

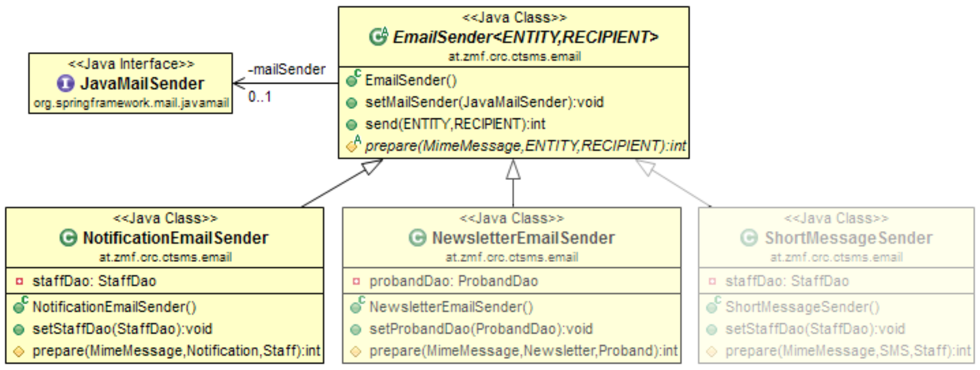
Daily job: The CLI tool provides a job option for creating notifications, scheduled to run once a day. During execution, it creates `Notifications` for detected events (table 4.12). If the actual date enters the time span (such as a reminder period) defined by an item (such as an inventory maintenance reminder), the responsible persons are to be notified. This implies to *suppress* generating another notification record on the next day. It is therefore again required to keep track of notifications that were created for a particular item already.

As outlined, notifications regarding the same entity are considered (and updated) when creating new notifications. The `Notification` model therefore shows many-to-one associations between `Notification` and referenced entities. To capture the associated entities' current state enclosed in message texts, a notification's message content is generated using *Velocity templates* when persisting the notification instance. A notification's recipients are mapped by the `NotificationRecipient` relationship. As a consequence, the message's language applys to all recipients, irrespective of their specific language settings.

notification	**email**	**primary recipient set**	**job**	**reason/triggering**
Maintenance reminder	✓	`MaintenanceScheduleItem.responsiblePerson`	✓	on the first day of the reminder period prior the reminder's due date recurrence
Inventory N/A	✗	`Staffs` of `Inventory.department`	✓	on the first day of the inventory's non-availability
Inventory N/A - booking problem	✓	`InventoryBooking.onBehalfOf`	✗	whenever an inventory's non-availability entry (`InventoryStatusEntry`) is created or updated, that overlaps with an existing inventory booking
Staff N/A	✓	`StaffStatusEntry.staff.parent`	✓	on the first day of the person's absence
Staff N/A - duty problem	✓	`DutyRosterTurn.trial.members.staff`	✗	whenever a person's absence entry (`StaffStatusEntry`) is created or updated, that overlaps with an assigned duty
Proband N/A	✗	`Staffs` of `ProbandStatusEntry.proband.department`	✓	on the first day of the proband's non-availability
Proband N/A - visit problem	✗	`VisitScheduleItem.trial.members.staff`	✗	whenever a proband's non-availability entry (`ProbandStatusEntry`) is created or updated, that overlaps with an `VisitScheduleItem` of the participating proband's group
Course expiration reminder	✓	`Course.lecturers.staff`	✓	e.g. three months prior to the course's expiration date
Course expiration reminder (participant)	✓	`Course.participations.staff`	✓	e.g. one month prior to the course's expiration date
Course participation status update	✓	`CourseParticipationStatusEntry.staff` if originated by a lecturer, `CourseParticipationStatusEntry.course.lecturers.staff` if originated by a participant	✗	whenever a `CourseParticipationStatusEntry` is created or updated by changing its status to a status relevant for notification
Trial timeline event reminder	✓	`TimelineEvent.trial.members.staff`[67]	✓	on the first day of the reminder period prior the timeline event's start date, if activated
Proband auto-delete reminder	✗	`Staffs` of `Proband.department`	✓	e.g. 10 days before the final deletion of the proband by the auto-delete job
Password expiration reminder	✓	`Password.user.identity`	✓	e.g. 7 days before the user password expires
Trial status update	✓	`Trial.members.staff`[67]	✗	whenever a `Trial` is created or updated by changing its status to a status relevant for notification
Probands auto-deleted	✗	`Staffs` of departments, whose probands were deleted	✓	after probands were effectively deleted by the auto-delete job
Missing trial tag	✓	`Trial.members.staff`[67]	✗	whenever a `Trial` is updated by changing its status to a status relevant for notification, while `TrialTagValues` for `TrialTags` configured as required are still missing
Duty update	✓	`DutyRosterTurn.staff`	✗	whenever an existing duty is updated, without altering the assigned employee
Duty assigned	✓	new `DutyRosterTurn.staff`	✗	whenever a duty is created or an existing duty is updated by altering the assigned employee
Duty unassigned	✓	original `DutyRosterTurn.staff`	✗	whenever an existing duty is updated by altering the assigned employee
New course	✗	`Staffs` of `Course.department`	✗	whenever a new course is created for self-registration

Table 4.12.: Overview of notification types.

Consumer The CLI tool provides another job for sending notifications as emails, scheduled to run every 5 minutes. During execution it retrieves a list of pending notifications. To equalize load, the number of notifications per cycle can be limited. When iterating pending notifications, a single email is sent to all recipient `Staff`'s email addresses marked by the `StaffContactDetailValue.notifiy` flag. This logic is encapsulated by the `NotificationEmailSender` (figure 4.74), which utilizes `org.springframework.mail.javamail.JavaMailSender` to effectively send emails using a configurable server for outgoing emails (listing 4.34). The notification's original set of recipients is filtered using a configurable matrix[68]. For example, frequent notifications like the "Proband auto-delete reminder" are prevented from being sent to recipients of category "management" and marked by the `NotificationRecipient.dropped` flag for subsequent job cycles.



<<Java Interface>>
JavaMailSender
org.springframework.mail.javamail
-mailSender
0..1
<<Java Class>>
EmailSender<ENTITY,RECIPIENT>
at.zmf.crc.ctsms.email
EmailSender()
setMailSender(JavaMailSender):void
send(ENTITY,RECIPIENT):int
prepare(MimeMessage,ENTITY,RECIPIENT):int
<<Java Class>>
NotificationEmailSender
at.zmf.crc.ctsms.email
staffDao: StaffDao
NotificationEmailSender()
setStaffDao(StaffDao):void
prepare(MimeMessage,Notification,Staff):int
<<Java Class>>
NewsletterEmailSender
at.zmf.crc.ctsms.email
probandDao: ProbandDao
NewsletterEmailSender()
setProbandDao(ProbandDao):void
prepare(MimeMessage,Newsletter,Proband):int
<<Java Class>>
ShortMessageSender
at.zmf.crc.ctsms.email
staffDao: StaffDao
ShortMessageSender()
setStaffDao(StaffDao):void
prepare(MimeMessage,SMS,Staff):int

Figure 4.74.: The `EmailSender` abstract base class is a `JavaMailSender` API adapter. In a future version, alternative mail senders could be implemented (semitransparent implementation classes). For example, `NewsletterEmailSender` will process a `Newsletter` and send it to email addresses (`ProbandContactDetailValue`) of a `Proband`. Another option would be sending Short Messages using an external email-to-SMS gateway.

```xml
<bean id="notificationMailSender" class="org.springframework.mail.javamail.JavaMailSenderImpl">
    <property name="host" value="smtp.joanneum.at"/>
    <property name="port" value="25"/>
    <!-- <property name="username" value="username" />
    <property name="password" value="password" /> -->
    <property name="javaMailProperties">
        <props>
            <prop key="mail.smtp.starttls.enable">false</prop>
        </props>
    </property>
</bean>
<bean id="notificationEmailSender" class="at.zmf.crc.ctsms.email.NotificationEmailSender">
    <property name="staffDao" ref="staffDao" />
    <property name="mailSender" ref="notificationMailSender"/>
</bean>
```

Listing 4.34: The outgoing mail server used for sending emails via `JavaMailSender` is set up by means of Spring configuration.

Future features such as sending participant information material or newsletters to probands can therefore employ an alternative mail server configuration. This allows to isolate the risk of a blacklisted mail server due to probands reporting the emails as spam. Another supported option is sending Short Messages, which is typically required for Mobile Transaction Authentication Numbers (mTANs) when implementing *two-way authentication*.

[67]If activated for the particular trial team membership only.

[68]A many-to-many association between `NotificationType` and `StaffCategory` contains master data that defines enabled notification types per staff category of recipients. It is accessed with the `NotificationType.sendDepartmentStaffCategories` field.

4.6.4. Entity Views

A module's root entity and its detail records can be viewed and edited using *entity views*. Entity views expose most CRUD and retrieval service methods and represent the UI's core components. Each module's entity view follows a uniform design: a `p:tabView` arranges the root entity fields in its first `p:tab` (master tab), while additional tabs are provided per detail entity (detail tabs). Each tab's content is defined by a separate facelet file (listing 4.37). The tab facelets are combined by the entity view facelet using `ui:include` tags to construct an entity view page (listing 4.35).

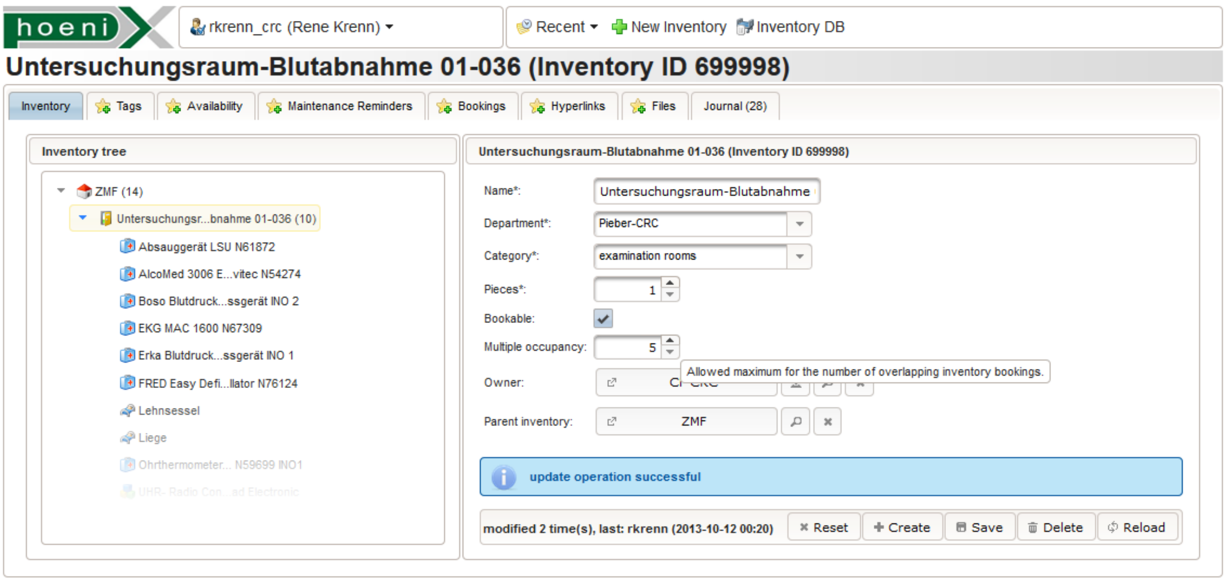
```
<?xml version="1.0" encoding="UTF-8" ?>
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:p="http://primefaces.org/ui"
    xmlns:h="http://java.sun.com/jsf/html">
    <h:head><title>trial tab</title></h:head>
    <h:body><ui:composition template="/META-INF/templates/trialEntityTemplate.xhtml"> <!-- The rendered page is a result of
        inserting a fragment into the single "content" placeholder of trialEntityTemplate.xhtml -->
        <ui:param name="title" value="#{trialBean.title}"/> <!-- this template variable is applied to the HTML <title> tag in
            baseTemplate.xhtml -->
        <ui:param name="windowName" value="#{trialBean.windowName}"/> <!-- this template variable is applied to the Js constant
            export in moduleTemplate.xhtml -->
        <!-- fill in template "content" placeholder: -->
        <ui:define name="content">
            <h:panelGrid columns="1" cellpadding="0" styleClass="ctsms-content-panelgrid">
                <p:tabView styleClass="ctsms-tabview" id="tabView"
                    widgetVar="trialTabView" <!-- a Js widget object variable is exposed for client-side updates of tab titles
                        and icons -->
                    onTabChange="handleTrialTabSwitch(index)" > <!-- update a tab's content when switching to it -->
                    <!-- the master tab is always enabled: -->
                    <p:tab id="trialmain" disabled="false" titleStyleClass="#{!trialBean.created ? 'ctsms-tabtitle-emphasized'
                        : 'ctsms-tabtitle'}" title="#{triallabels.trial_main_tab_label}">
                        <ui:include src="/META-INF/includes/trial/trialMaster.xhtml"/>
                    </p:tab>
                    <!-- the trial tag tab is enabled upon the inital page load only if an existing trial is opened: -->
                    <p:tab id="trialtags" disabled="#{!trialBean.created}" titleStyleClass="#{trialBean.isTabEmphasized(
                        applicationScopeBean.jsValue('AJAX_TRIAL_TAG_VALUE_COUNT')) ? 'ctsms-tabtitle-emphasized' : 'ctsms-
                        tabtitle'}" title="#{trialBean.getTabTitle(applicationScopeBean.jsValue('AJAX_TRIAL_TAG_VALUE_COUNT'))
                        }">
                        <ui:include src="/META-INF/includes/trial/trialTag.xhtml"/> <!-- embed the tab content -->
                    </p:tab>
                    <!-- the next trial detail entity tab: -->
                    <p:tab id="teammembers" disabled="#{!trialBean.created}" ...
                    ...
                </p:tabView>
            </h:panelGrid>
        </ui:define>
        </ui:composition></h:body>
</html>
```

Listing 4.35: The representative trial entity view facelet shown consists of a `p:tabView` in the first place. Each tab page's content is outsourced into its own facelet file and included using `ui:insert`.

A tab view's control and update logic relies on the `handle*TabSwitch` Js handler (listing 4.36), which invokes `p:remoteCommands` for refreshing a tab's content when switching to it. The master tab is the only tab enabled by default (listing 4.35). Until a new root entity item is created, remaining (detail) tabs are disabled. An existing item is opened by navigating to its entity view (section 4.6.2) or by clicking an item in the item hierarchy tree (figure 4.75, 4.76 and 4.77), if available in the master tab. If opening an item is prohibited due to failed decryption or restricted privileges, the remaining tabs are disabled. The browser tab's `document.title` and a leading headline display a unique title of the root entity instance opened.

```
//A tab's content is initialized lazily when switching to it by invoking a tab's p:remoteCommand. It will cause a reset of the
      tab's datatable and item edit area.
function handleTrialTabSwitch(index) {
    switch (index) {
        case 0: break;
        case 1: swithToTrialTagTabRC(); break; //invoke remote command
        case ...
        default: break;
    }
    return true;
}
//Root entity CRUD and (re-)load operations provide org.primefaces.context.RequestContext parameters for this Js callback
      method:
function handleTrialChanged(xhr, status, args) {
    if (_testFlag(args,AJAX_OPERATION_SUCCESS)   _testPropertyExists(args,AJAX_WINDOW_TITLE_BASE64)   _testPropertyExists(args,
         AJAX_ROOT_ENTITY_CREATED)  _testPropertyExists(args,AJAX_WINDOW_NAME)) {
        //when a root entity was sucessfully opened or modified, titles and tabs are set up initially:
        window.name = args[AJAX_WINDOW_NAME]; //update the browser tab's window name
        enableTabs(trialTabView,1,args[AJAX_ROOT_ENTITY_CREATED]); //enable detail entity tabs
        handleUpdateTrialTabTitles(xhr, status, args); //set tab titles and icons
        var title = decodeBase64(args[AJAX_WINDOW_TITLE_BASE64]);
        document.title = title; //set the browser tab's title
        setText(HEADLINE_ID, title); //set the headline text
        updateTrialEntityMenuBar(); //the menu bar's item history menu is likely to be outdated after a CRUD operation
    }
}
//Updating the tab titles and icons utilizes the emphasizeTab and setTabTitle extensions of the PrimeFaces.widget.TabView
      prototype:
function handleUpdateTrialTabTitles(xhr, status, args) {
    var tabIndex = 0;
    if (_testPropertyExists(args,AJAX_OPERATION_SUCCESS)   _testPropertyExists(args,AJAX_ROOT_ENTITY_CREATED)) { //set the trial
          master tab title/icon
        trialTabView.emphasizeTab(tabIndex,_testFlag(args,AJAX_OPERATION_SUCCESS)   !args[AJAX_ROOT_ENTITY_CREATED]);
    }
    tabIndex++;
    if (_testPropertyExists(args,AJAX_TRIAL_TAG_TAB_TITLE_BASE64)   _testPropertyExists(args,AJAX_TRIAL_TAG_VALUE_COUNT)) { //
          set the trial tag tab title/icon
        trialTabView.setTabTitle(tabIndex,decodeBase64(args[AJAX_TRIAL_TAG_TAB_TITLE_BASE64]));
        trialTabView.emphasizeTab(tabIndex,args[AJAX_TRIAL_TAG_VALUE_COUNT] == 0);
    }
    ...
}
```

Listing 4.36: An entity view's Js resource file like `trial.js` essentailly consists of loose event handler and AJAX callback methods. In general, the browser is adviced to load required Js resource files upon page load by rendering script inclusion markup (`<script src="...">`) in the page's HTML `<head>` section. Including required Js resource files is covered by facelet templates like `trialTemplate.xhtml`.



Untersuchungsraum-Blutabnahme 01-036 (Inventory ID 699998)

Inventory tree

ZMF (14)
Untersuchungsr...bnahme 01-036 (10)
Absauggerät LSU N61872
AlcoMed 3006 E...vitec N54274
Boso Blutdruck...ssgerät INO 2
EKG MAC 1600 N67309
Erka Blutdruck...ssgerät INO 1
FRED Easy Defi...llator N76124
Lehnsessel
Liege
Ohrthermometer... N59699 INO1
UHR- Radio Con...ad Electronic

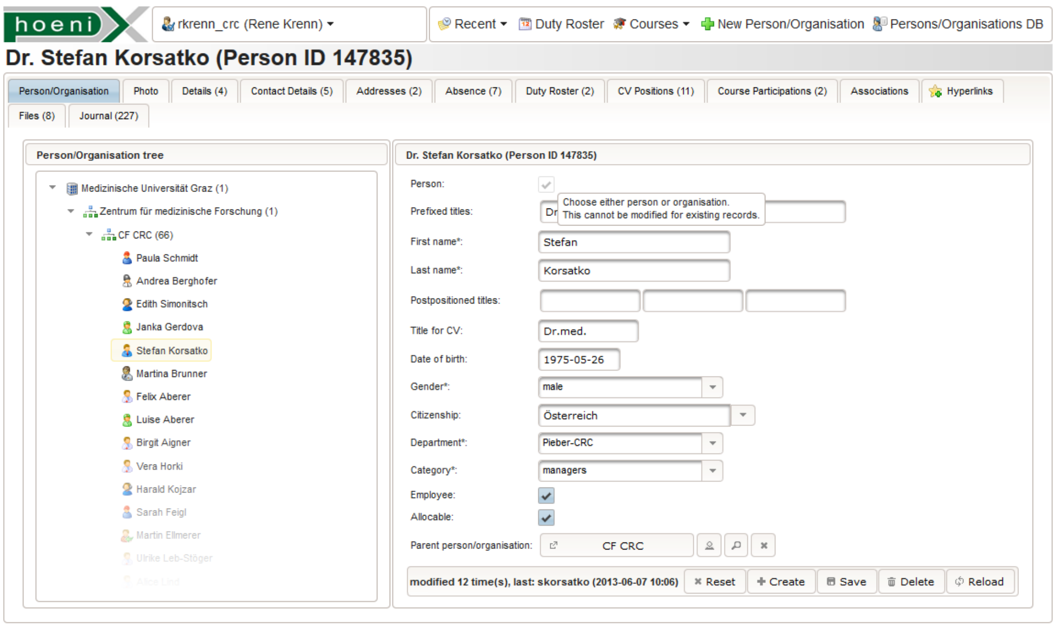
Untersuchungsraum-Blutabnahme 01-036 (Inventory ID 699998)

Name*: Untersuchungsraum-Blutabnahme
Department*: Pieber-CRC
Category*: examination rooms
Pieces*: 1
Bookable: ✓
Multiple occupancy: 5
Allowed maximum for the number of overlapping inventory bookings.
Owner: CRC
Parent inventory: ZMF

update operation successful

modified 2 time(s), last: rkrenn (2013-10-12 00:20)

Figure 4.75.: The inventory entity view's master tab. A successful CRUD operation is always signaled by a light blue response message.



Dr. Stefan Korsatko (Person ID 147835)

Person/Organisation tree

- Medizinische Universität Graz (1)
  - Zentrum für medizinische Forschung (1)
    - CF CRC (66)
      - Paula Schmidt
      - Andrea Berghofer
      - Edith Simonitsch
      - Janka Gerdova
      - Stefan Korsatko
      - Martina Brunner
      - Felix Aberer
      - Luise Aberer
      - Birgit Aigner
      - Vera Horki
      - Harald Kojzar
      - Sarah Feigl
      - Martin Ellmerer
      - Ulrike Leb-Stöger
      - Alice Lind

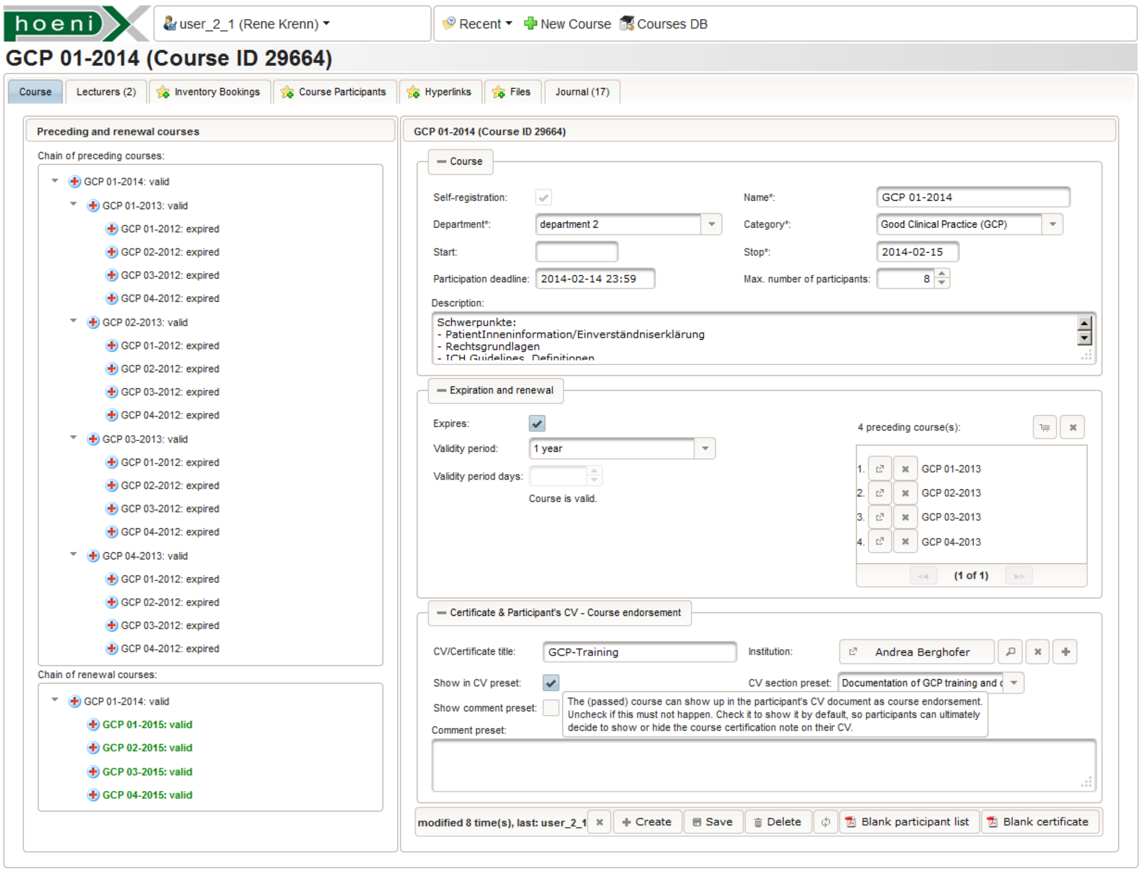
Dr. Stefan Korsatko (Person ID 147835)

Person:	✓
Prefixed titles:	Dr
First name*:	Stefan
Last name*:	Korsatko
Postpositioned titles:	
Title for CV:	Dr.med.
Date of birth:	1975-05-26
Gender*:	male
Citizenship:	Österreich
Department*:	Pieber-CRC
Category*:	managers
Employee:	✓
Allocable:	✓
Parent person/organisation:	CF CRC

Choose either person or organisation.
This cannot be modified for existing records.

modified 12 time(s), last: skorsatko (2013-06-07 10:06)

Figure 4.76.: The person/organisation entity view's master tab.



GCP 01-2014 (Course ID 29664)

Preceding and renewal courses

Chain of preceding courses:

- GCP 01-2014: valid
  - GCP 01-2013: valid
    - GCP 01-2012: expired
    - GCP 02-2012: expired
    - GCP 03-2012: expired
    - GCP 04-2012: expired
  - GCP 02-2013: valid
    - GCP 01-2012: expired
    - GCP 02-2012: expired
    - GCP 03-2012: expired
    - GCP 04-2012: expired
  - GCP 03-2013: valid
    - GCP 01-2012: expired
    - GCP 02-2012: expired
    - GCP 03-2012: expired
    - GCP 04-2012: expired
  - GCP 04-2013: valid
    - GCP 01-2012: expired
    - GCP 02-2012: expired
    - GCP 03-2012: expired
    - GCP 04-2012: expired

Chain of renewal courses:

- GCP 01-2014: valid
  - GCP 01-2015: valid
  - GCP 02-2015: valid
  - GCP 03-2015: valid
  - GCP 04-2015: valid

GCP 01-2014 (Course ID 29664)

Course

Self-registration:	✓	Name*:	GCP 01-2014
Department*:	department 2	Category*:	Good Clinical Practice (GCP)
Start:		Stop*:	2014-02-15
Participation deadline:	2014-02-14 23:59	Max. number of participants:	8

Description:

Schwerpunkte:
- PatientInneninformation/Einverständniserklärung
- Rechtsgrundlagen
- ICH Guidelines, Definitionen

Expiration and renewal

Expires:	✓
Validity period:	1 year
Validity period days:	

Course is valid.

4 preceding course(s):

1. GCP 01-2013
2. GCP 02-2013
3. GCP 03-2013
4. GCP 04-2013

(1 of 1)

Certificate & Participant's CV - Course endorsement

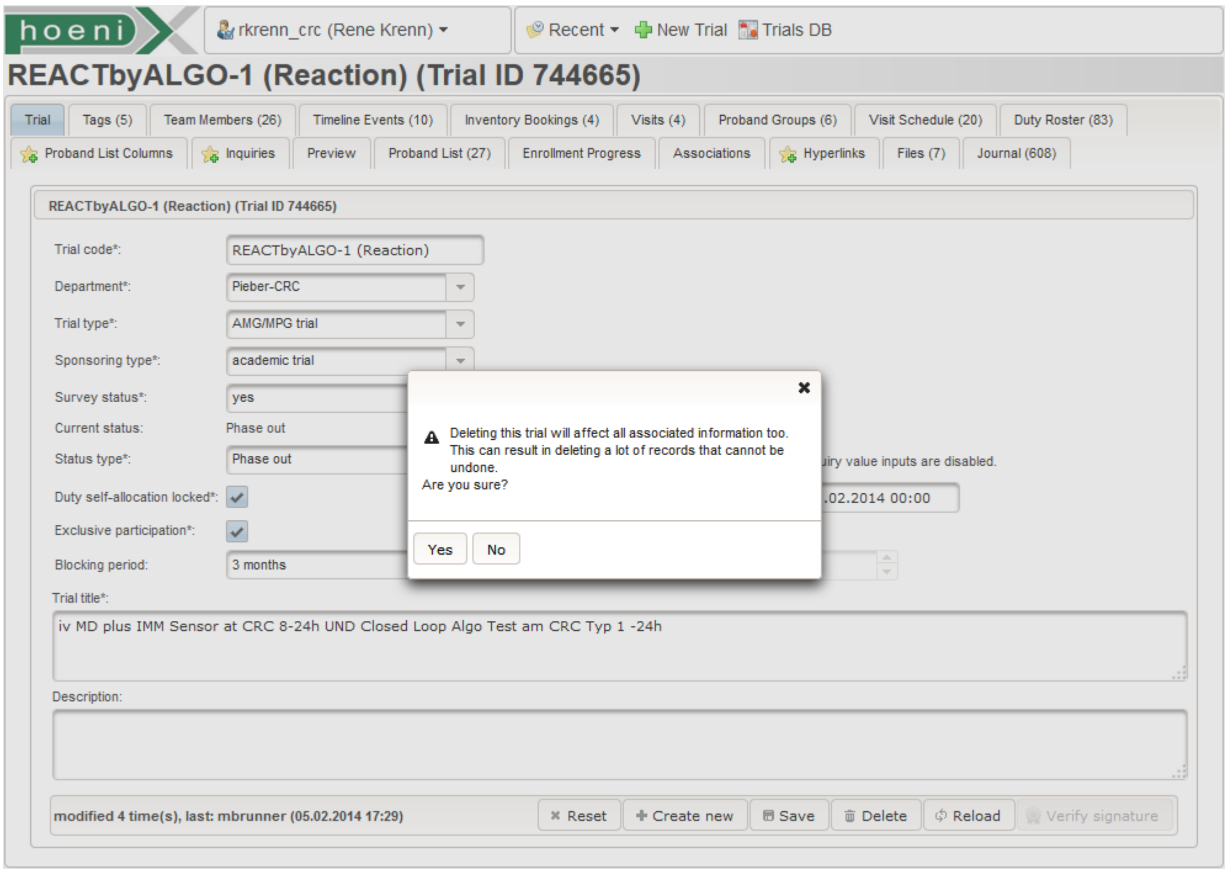
CV/Certificate title:	GCP-Training	Institution:	Andrea Berghofer
Show in CV preset:	✓	CV section preset:	Documentation of GCP training and c
Show comment preset:			
Comment preset:			

The (passed) course can show up in the participant's CV document as course endorsement. Uncheck if this must not happen. Check it to show it by default, so participants can ultimately decide to show or hide the course certification note on their CV.

modified 8 time(s), last: user_2_1

Figure 4.77.: The course entity view's master tab.

The master tab exposes a module's root entity CRUD operations via `p:commandButtons` of a toolbar container component at the bottom. After creating a root entity, empty detail entity tabs are indicated by ✵ icons in detail tab titles. Detail tab titles are updated upon any relevant PPR cycle and show the number of existing items for an instant overview without the need to view datatables by switching to the tabs. Service method error messages such as validation failures, concurrent modification or insufficient privileges are displayed as well as messages to confirm successful execution (figure 4.75). In case of an error, the input component values are restored. Since create operations are not idempotent by design, pressing the "create" button allows creating record copies quickly. Delete operations of simple detail records can basically be undone by simply recreating the record, since form values remain populated. However, removing a master record with associated detail records requires dropping the entire entity instance graph. This implies the cascaded removal of a magnitude of records, which *cannot* be undone. Therefore, a modal `p:confirmDialog` asks the user to approve the requested operation (figure 4.78). Toolbar buttons are disabled according to the existence of a record, e.g. an item cannot be updated if it has not been created yet. However, the lack of user privileges is not reflected by means of disabled UI controls in general.



REACTbyALGO-1 (Reaction) (Trial ID 744665)

REACTbyALGO-1 (Reaction) (Trial ID 744665)

Trial code*:	REACTbyALGO-1 (Reaction)
Department*:	Pieber-CRC
Trial type*:	AMG/MPG trial
Sponsoring type*:	academic trial
Survey status*:	yes
Current status:	Phase out
Status type*:	Phase out
Duty self-allocation locked*:	✓
Exclusive participation*:	✓
Blocking period:	3 months

iry value inputs are disabled.

.02.2014 00:00

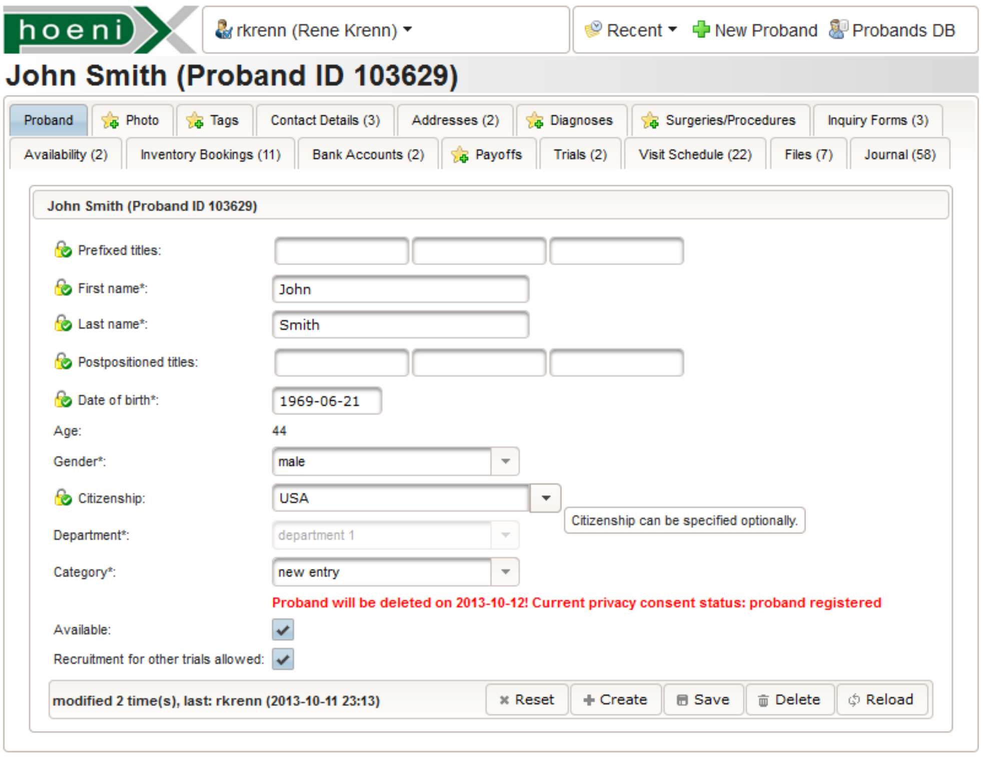
Deleting this trial will affect all associated information too. This can result in deleting a lot of records that cannot be undone.
Are you sure?

Trial title*:
iv MD plus IMM Sensor at CRC 8-24h UND Closed Loop Algo Test am CRC Typ 1 -24h

Description:

modified 4 time(s), last: mbrunner (05.02.2014 17:29)

Figure 4.78.: The trial entity view's master tab. This screen shows the attempt to delete the opened `Trial` entity instance graph. Extensive delete operations like this require a confirmation.



John Smith (Proband ID 103629)

John Smith (Proband ID 103629)

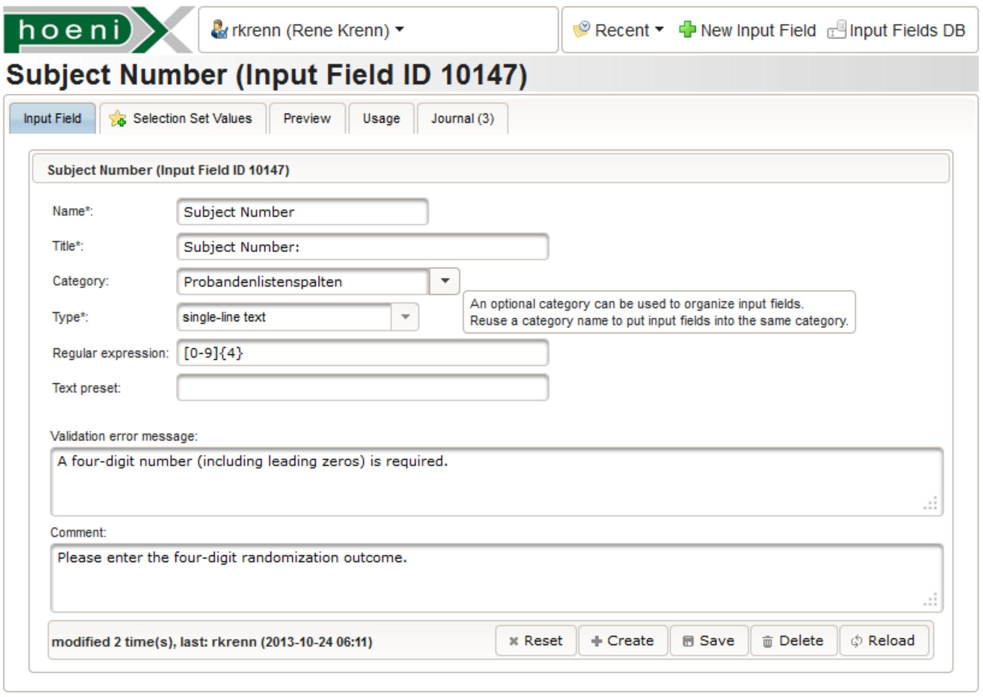
Prefixed titles:	
First name*:	John
Last name*:	Smith
Postpositioned titles:	
Date of birth*:	1969-06-21
Age:	44
Gender*:	male
Citizenship:	USA
Department*:	department 1
Category*:	new entry
Available:	✓
Recruitment for other trials allowed:	✓

Citizenship can be specified optionally.

Proband will be deleted on 2013-10-12! Current privacy consent status: proband registered

modified 2 time(s), last: rkrenn (2013-10-11 23:13)

Figure 4.79.: The proband entity view's master tab.



Subject Number (Input Field ID 10147)

Subject Number (Input Field ID 10147)

Name*:	Subject Number
Title*:	Subject Number:
Category:	Probandenlistenspalten
Type*:	single-line text
Regular expression:	[0-9]{4}
Text preset:	

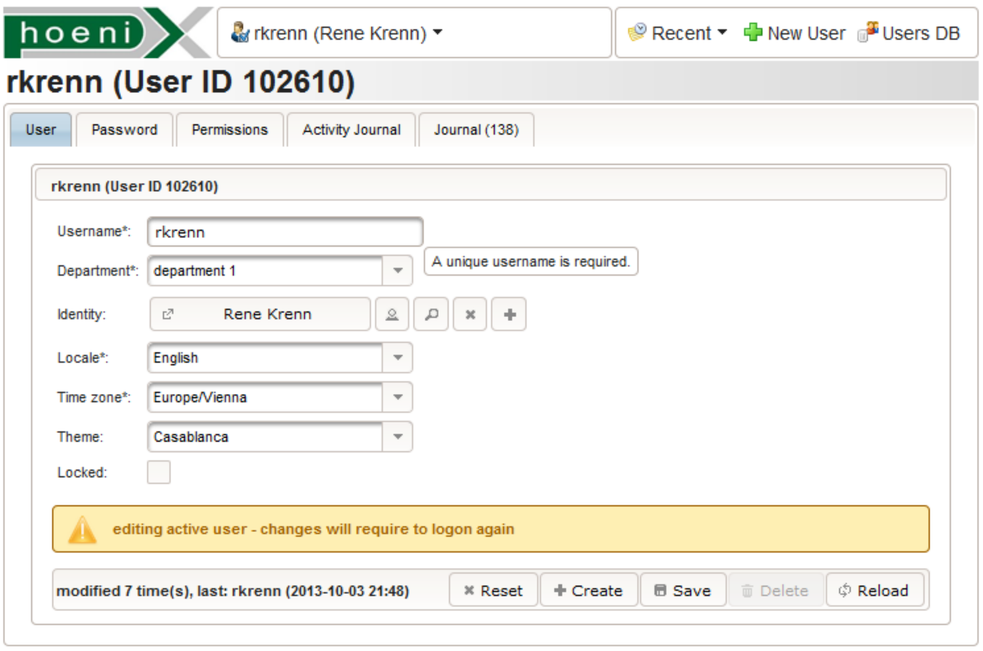
An optional category can be used to organize input fields. Reuse a category name to put input fields into the same category.

Validation error message:
A four-digit number (including leading zeros) is required.

Comment:
Please enter the four-digit randomization outcome.

modified 2 time(s), last: rkrenn (2013-10-24 06:11)

Figure 4.80.: The input field entity view's master tab.


hoeni | rkrenn (Rene Krenn) | Recent | New User | Users DB

rkrenn (User ID 102610)

User | Password | Permissions | Activity Journal | Journal (138)

rkrenn (User ID 102610)

Username*:	rkrenn
Department*:	department 1
Identity:	Rene Krenn
Locale*:	English
Time zone*:	Europe/Vienna
Theme:	Casablanca
Locked:	☐

A unique username is required.

editing active user - changes will require to logon again

modified 7 time(s), last: rkrenn (2013-10-03 21:48) | Reset | Create | Save | Delete | Reload

Figure 4.81.: The user entity view's master tab.

A detail tab exposes CRUD operations for a particular detail detail entity of the opened root entity. In general, they follow the uniform design outlined in figure 4.82. Above a compact edit area, a datatable displays existing detail entity records. `p:tooltip` texts are provided per input component but can be disabled globally in the underlying `baseTemplate.xhtml` template. It provides additional configuration options for consistent component appearance and behaviour aspects such as the global jQuery animation effect setting, `p:autoComplete` attributes or `p:dataTable` defaults like paginator bar styles.

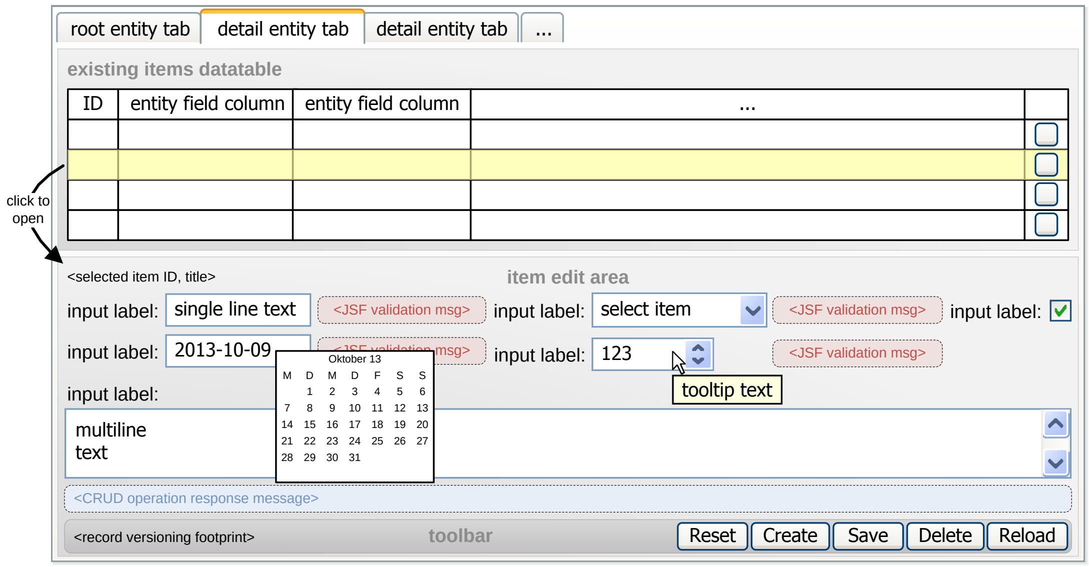
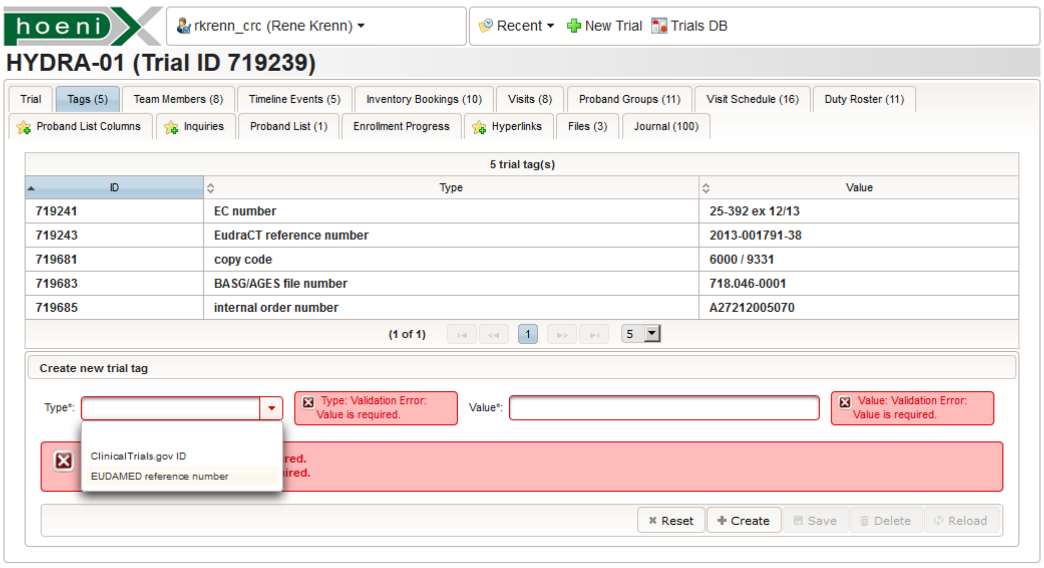


Figure 4.82.: The consequent design scheme of a detail entity tab suggests the idea of automated generation. In fact, an attempt for an AndroMDA cartridge is proposed in (Collin, 2010).



rkrenn_crc (Rene Krenn) | Recent | New Trial | Trials DB

HYDRA-01 (Trial ID 719239)

Trial | Tags (5) | Team Members (8) | Timeline Events (5) | Inventory Bookings (10) | Visits (8) | Proband Groups (11) | Visit Schedule (16) | Duty Roster (11)
Proband List Columns | Inquiries | Proband List (1) | Enrollment Progress | Hyperlinks | Files (3) | Journal (100)

5 trial tag(s)

ID	Type	Value
719241	EC number	25-392 ex 12/13
719243	EudraCT reference number	2013-001791-38
719681	copy code	6000 / 9331
719683	BASG/AGES file number	718.046-0001
719685	internal order number	A27212005070

(1 of 1) 1 5

Create new trial tag

Type*: [ ] Type: Validation Error: Value is required. Value*: [ ] Value: Validation Error: Value is required.

ClinicalTrials.gov ID
EUDAMED reference number

red.
ired.

Reset | Create | Save | Delete | Reload

Figure 4.83.: As an example, the trial tags detail tab is shown as incarnation of the scheme from figure 4.82. Empty input checks are performed for required fields (marked with "*") by enabling the component's `required` flag.

Since input validation is part of the service layer, JSF validators are not utilized at all. However, the component's `required` attribute is used to display error messages for mandatory fields directly next to the corresponding input component (figure 4.83). Listing 4.37 gives the full implementation of the trial tag detail tab content facelet as a most simple but representative example. The corresponding view-scoped managed bean (`TrialTagBean`) is given in listing 4.38.

```
<?xml version="1.0" encoding="UTF-8" ?>
<html xmlns="http://www.w3.org/1999/xhtml"
     xmlns:ui="http://java.sun.com/jsf/facelets"
     xmlns:f="http://java.sun.com/jsf/core"
     xmlns:p="http://primefaces.org/ui"
     xmlns:h="http://java.sun.com/jsf/html">
    <h:head><title>trial tags</title></h:head>
    <h:body><ui:composition> <!-- the detail tab facelet is to be included by the trial entity view facelet -->
        <h:form id="trialtag_form">
            <p:remoteCommand process="@this" name="swithToTrialTagTabRC" actionListener="#{trialTagBean.tabSwitch(trialBean.in.id)}"
                update="trialtag_list,trialtag_input" /> <!-- this p:remoteCommand provides an AJAX cycle to refresh the entire tab content -->
            <h:panelGrid columns="1" cellpadding="0" styleClass="ctsms-content-panelgrid" >
                <p:dataTable styleClass="ctsms-datatable" lazy="true" <!-- datatable for existing records: -->
                     rows="#{dataTablesRowsDefault}" <!-- default settings from baseTemplate.xhtml -->
                     paginator="#{dataTablesPaginatorDefault}"
                     paginatorTemplate="#{dataTablesPaginatorTemplateDefault}"
                     ...
                     sortBy="#{tagValue.vo.id}" <!-- default sorting -->
                     emptyMessage="#{labels.datatable_permission_empty_message}"
                     var="tagValue" <!-- row iteration variable of type TrialTagValueOutVO -->
                     value="#{trialTagBean.tagValueModel}" <!-- result rows model of the backing managed bean -->
                     id="trialtag_list"
                     selectionMode="single" <!-- enable selecting single rows -->
                     selection="#{trialTagBean.selectedTrialTagValue}">
                    <p:ajax event="rowSelect" update=":tabView:trialtag_form:trialtag_input" /> <!-- PVR when selecting a row -->
                    <f:facet name="header"><h:outputText value="#{trialTagBean.tagValueModel.rowCount} #{triallabels.trial_tag_trialtag_list_header}" /><
                        /f:facet>
                    <p:column sortBy="#{tagValue.vo.id}"> <!-- id column -->
                        <f:facet name="header"><h:outputText value="#{labels.id_column}" /></f:facet>
                        <h:outputText value="#{tagValue.vo.id}" />
                    </p:column>
                    <p:column sortBy="#{tagValue.vo.tag}"> <!-- trial tag type column -->
                        <f:facet name="header"><h:outputText value="#{triallabels.trial_tag_trialtag_list_tag_name_column}" /></f:facet>
                        <h:outputText value="#{tagValue.vo.tag.name}" />
                    </p:column>
                    <p:column sortBy="#{tagValue.vo.value}"> <!-- tag value column -->
                        <f:facet name="header"><h:outputText value="#{triallabels.trial_tag_trialtag_list_value_column}" /></f:facet>
                        <h:outputText value="#{tagValue.vo.value}" />
                    </p:column>
                </p:dataTable>
                <p:panel id="trialtag_input"> <!-- item edit area: -->
                    <f:facet name="header"><h:outputText value="#{trialTagBean.title}" /></f:facet>
                    <h:panelGrid columns="1" cellpadding="0" styleClass="ctsms-input-panelgrid" rowClasses="ctsms-input-row,ctsms-message-row,ctsms-
                        toolbar-row">
                        <h:panelGrid columns="6" cellpadding="2" columnClasses="ctsms-label-for-column,ctsms-input-column,ctsms-message-for-column,ctsms-
                            label-for-column,ctsms-input-column,ctsms-message-for-column">
                            <!-- the trial tag type dropdown: -->
                            <h:outputLabel for="trialTag" value="#{triallabels.trial_tag_label}" />
                            <h:panelGroup><p:selectOneMenu id="trialTag" value="#{trialTagBean.in.tagId}" required="true"
                                 label="#{triallabels.trial_tag}" styleClass="ctsms-control">
                                <f:converter converterId="at.zmf.ID" />
                                <f:selectItem itemLabel="#{labels.no_selection_label}"
                                     itemValue="#{applicationScopeBean.noSelectionValue}" />
                                <f:selectItems value="#{trialTagBean.availableTags}"  var="selectItem"
                                     itemLabel="#{selectItem.label}" itemValue="#{selectItem.value}" />
                            </p:selectOneMenu>
                            <p:tooltip rendered="#{enableTooltips}" for="trialTag" ><h:outputText value="#{triallabels.trial_tag_tooltip}" escape="false"
                                 /></p:tooltip></h:panelGroup>
                            <p:message for="trialTag" />
                            <!-- the tag value input element: -->
                            <h:outputLabel for="tagValue" value="#{triallabels.trial_tag_value_label}" />
                            <h:panelGroup><p:inputText id="tagValue" value="#{trialTagBean.in.value}" required="true"
                                 label="#{triallabels.trial_tag_value}" styleClass="ctsms-control-larger" />
                            <p:tooltip rendered="#{enableTooltips}" for="tagValue" ><h:outputText value="#{triallabels.trial_tag_value_tooltip}" escape="
                                 false"/></p:tooltip></h:panelGroup>
                            <p:message for="tagValue" />
                        </h:panelGrid>
                        <p:messages /> <!-- CRUD operation response message -->
                        <p:toolbar> <!-- record versioning footprint: -->
                            <p:toolbarGroup align="left"> <!-- record versioning footprint: -->
                                <h:outputText styleClass="ctsms-toolbar-text" value="#{trialTagBean.created ? trialTagBean.modifiedAnnotation : ''}" />
                            </p:toolbarGroup>
                            <p:separator />
                            <p:toolbarGroup align="right"> <!-- CRUD p:commandButtons: -->
                                <p:commandButton process="@this" value="#{labels.reset_button_label}" actionListener="#{trialTagBean.reset}"
                                     oncomplete="handleUpdateTrialTabTitles(xhr, status, args)" icon="ui-icon ui-icon-close"
                                     ajax="true" disabled="false" update="trialtag_list,trialtag_input" />
                                <p:commandButton process="@this,trialtag_input" value="#{labels.add_button_label}" actionListener="#{trialTagBean.add}"
                                     oncomplete="handleUpdateTrialTabTitles(xhr, status, args)" icon="ui-icon ui-icon-plusthick"
                                     ajax="true" disabled="#{!trialTagBean.createable}" update="trialtag_list,trialtag_input" />
                                <p:commandButton process="@this,trialtag_input" value="#{labels.update_button_label}" actionListener="#{trialTagBean.
                                     update}"
                                     oncomplete="handleUpdateTrialTabTitles(xhr, status, args)" icon="ui-icon ui-icon-disk"
                                     ajax="true" disabled="#{!trialTagBean.editable}" update="trialtag_list,trialtag_input" />
                                <p:commandButton process="@this" value="#{labels.delete_button_label}" actionListener="#{trialTagBean.delete}"
                                     oncomplete="handleUpdateTrialTabTitles(xhr, status, args)" icon="ui-icon ui-icon-trash"
                                     ajax="true" disabled="#{!trialTagBean.removable}" update="trialtag_list,trialtag_input" />
                                <p:commandButton process="@this" value="#{labels.load_button_label}" actionListener="#{trialTagBean.load}"
                                     oncomplete="handleUpdateTrialTabTitles(xhr, status, args)" icon="ui-icon-refresh"
                                     ajax="true" disabled="#{!trialTagBean.created}" update="trialtag_list,trialtag_input" />
                            </p:toolbarGroup>
                            <p:separator />
                            <p:toolbarGroup align="right"> <!-- animated icon to indicate an ongoing AJAX request: -->
                                <ui:include src="../shared/toolbarAjaxStatus.xhtml"/>
                            </p:toolbarGroup>
                        </p:toolbar>
                    </h:panelGrid>
                </p:panel>
            </h:panelGrid>
        </h:form>
    </ui:composition></h:body>
</html>
```

Listing 4.37: The complete `trialTag.xhtml` tab content facelet.

```
@ManagedBean
@ViewScoped
public class TrialTagBean extends ManagedBeanBase {
    //model fields:
    private TrialTagValueInVO in; //backing InVO fields
    private TrialTagValueOutVO out;
    private Long trialId; //the ID of the opened root entity
    private TrialTagValueLazyModel tagValueModel; //datatable lazy model (existing TrialTagValue items)
    private ArrayList<SelectItem> availableTags; //TrialTag type dropdown selection set (p:selectOneMenu)

    public TrialTagBean() {
        super(); tagValueModel = new TrialTagValueLazyModel(); //todo: Spring DI
    }
    @PostConstruct
    private void init() { //initialize models:
        //An alternative entity view implementation could use separate JSF pages per detail entity instead of the p:tabView.
        //The URL to access this page can contain a parameter like ?trialtagvalueid=12345 to open a detail entity item directly:
        Long id = WebUtil.getLongParamValue(GetParamNames.TRIAL_TAG_VALUE_ID);
        if (id != null) { tabSwitchImpl(id); }
        else { updateModels(); }
    }

    private void updateModels() {
        if (in == null) { in = new TrialTagValueInVO(); } //instantiate the InVO
        //initialize the InVO:
        if (out != null) {  copyTagValueOutToIn(in, out); trialId = in.getTrialId(); } //by copying OutVO fields
        else { initTagValueDefaultValues(in, trialId); } //by applying default/preset values
        //refresh the Notification List LazyDataModels state:
        tagValueModel.setTrialId(in.getTrialId()); //configure the trial tag value list LazyDataModel to show existing items of the same (opened) trial
        tagValueModel.updateRowCount(); //refresh the trial tag value list LazyDataModel's state
        //populate availableTags (SelectionSetService.getAvailAbleTrialTags) ...
    }
    public static void copyTagValueOutToIn(TrialTagValueInVO in, TrialTagValueOutVO out) { ... }
    public static void initTagValueDefaultValues(TrialTagValueInVO in, Long trialId) { ... }

    //The ManagedBeanBase base class provides an implementation scheme for CRUD action listener methods.
    //It encapsulates appending custom AJAX parameter values using org.primefaces.context.RequestContext:
    //  public void create() { appendRequestContextCallbackArgs(createImpl()); }
    //  protected abstract boolean createImpl();
    //By default, no parameter key-value pairs are added:
    //  protected void appendRequestContextCallbackArgs(boolean operationSuccess) {}
    @Override protected boolean createImpl() { //the create button action listener method implementation
        //when creating a new item from an existing one, record versioning field have to be cleared temporarily for the service method invocation:
        Long idBackup = in.getId(); in.setId(null);
        Long versionBackup = in.getId(); in.setVersion(null);
        try { //service method invocation:
            out = WebUtil.getServiceLocator().getTrialService().addTrialTagValue(WebUtil.getAuthentication(), in);
            updateModels();
            Messages.addLocalizedMessage(FacesMessage.SEVERITY_INFO, MessageCodes.CREATE_OPERATION_SUCCESSFUL);
            return true; //operation completed sucessfully
        } catch (ServiceException e) { //input validation error:
            in.setId(idBackup); in.setVersion(versionBackup); //restore record versioning fields
            Messages.addMessage(FacesMessage.SEVERITY_ERROR, e.getMessage());
        } catch (AuthenticationException e) { //logout, if the authentication fails
            in.setId(idBackup); in.setVersion(versionBackup);
            Messages.addMessage(FacesMessage.SEVERITY_ERROR, e.getMessage());
            WebUtil.publishException(e);
        } catch (AuthorisationException e) { //user is not authorized
            in.setId(idBackup); in.setVersion(versionBackup);
            Messages.addMessage(FacesMessage.SEVERITY_ERROR, e.getMessage());
        }
        return false;
    }
    @Override protected void appendRequestContextCallbackArgs(boolean operationSuccess) { //auxiliary AJAX parameter values for dynamic p:tab titles:
        RequestContext requestContext = WebUtil.appendTabTitleArgs(null,JSValues.AJAX_TRIAL_TAG_TAB_TITLE_BASE64,JSValues.AJAX_TRIAL_TAG_VALUE_COUNT,
              MessageCodes.TRIAL_TAGS_TAB_TITLE,MessageCodes.TRIAL_TAGS_TAB_TITLE_WITH_COUNT,new Long(tagValueModel.getRowCount())); //provide the
              updated item count for the detail entity's tab title
        if (operationSuccess  in.getTrialId() != null) {
            WebUtil.appendTabTitleArgs(requestContext,JSValues.AJAX_TRIAL_JOURNAL_TAB_TITLE_BASE64,JSValues.AJAX_TRIAL_JOURNAL_ENTRY_COUNT,MessageCodes.
                   TRIAL_JOURNAL_TAB_TITLE,MessageCodes.TRIAL_JOURNAL_TAB_TITLE_WITH_COUNT,WebUtil.getJournalCount(JournalModule.TRIAL_JOURNAL, in.
                   getTrialId())); //since JournalEntries are created by CRUD service methods, the journal tab's title is to be updated as well
        }
    }
    //remaining action listener implementations:
    @Override protected boolean updateImpl() { ... }
    @Override protected boolean deleteImpl() { ... }
    @Override protected boolean reloadImpl() { ... }
    @Override protected boolean resetImpl() { out = null; updateModels(); return true; }
    @Override protected boolean tabSwitchImpl(Long trialId) { //when switching to this detail entity tab, the entire content is reset and initialized
        out = null; this.trialId = trialId; updateModels(); return true;
    }
    //model getters:
    public TrialTagValueInVO getIn() { return in; }
    public TrialTagValueOutVO getOut() { return out; }
    public ArrayList<SelectItem> getAvailableTags() { return availableTags; }
    public TrialTagValueLazyModel getTagValueModel() { return tagValueModel; }
    //row selection for the datatable:
    public TrialTagValueOutVO getSelectedTrialTagValue() { return out; }
    public void setSelectedTrialTagValue(TrialTagValueOutVO trialTagValue) {
        out = trialTagValue; updateModels();
    }
    //dynamic UI labels:
    @Override public String getEditAreaTitle() { ... }
    @Override public String getRecordVersioningFootprint() { ... }
    //fields for p:commandButton disabled attributes:
    @Override public boolean isCreated() { return out != null; }
    @Override public boolean isCreateable() { return in.getTrialId() == null; }
    @Override public boolean isRemovable() { return isCreated(); }
    @Override public boolean isEditable() { return isCreated(); }
}
```

Listing 4.38: The complete `TrialTagBean` managed bean class.

4.6.5. Calendar Views

Full page *calendar views* utilize PrimeFace's `p:schedule` component, specifically designed to display *schedule events*. Any entity containing temporal interval information characterized by `start` and `stop` fields of either date or datetime type qualify as a compatible schedule event type. To distinguish event types visually, CSS style classes are used which apply custom colors and icons. Users can switch between a month, week and day agenda view that arrange schedule events of the corresponding time span. For the week and day mode, records with date type are displayed in form of bars in the upmost area (*all day* events), while datetime interval items are represented by boxes at the corresponding position in the actual schedule grid. The underlying jQuery FullCalendar Js implementation (Arshaw, 2013) provides lazy-loading of items and interactive edit operations (event move and resize) known from web calendars like Google Calendar. Lazy-loading is supported by specific finder methods according to listing 4.3. Interactive editing invokes applicable update operations, whose success or failure response messages are displayed by a fading `p:growl` box in the upper right screen corner. To edit event details, PrimeFaces promotes the use of its `p:dialog` component to show an overlaid edit area when an editable item is selected by clicking. When clicking in an unused region of the schedule grid, the edit area dialog can be opened to create a new event.

The Phoenix CTMS UI provides two calendar views which can be opened in separate browser tabs:

Inventory booking calendar To manage reservations of bookable inventory items, related events below can be displayed and filtered by. However, the inventory booking calendar view supports editing of inventory bookings only at the moment.

All day schedule events:

- Holidays
- Maintenance reminders (`MaintenanceScheduleItem`)
- Courses (`Course`)

Datetime interval schedule events:

- Inventory bookings (`InventoryBooking`)
- Inventory availability (`InventoryStatusEntry`
- Proband availability (`ProbandStatusEntry`

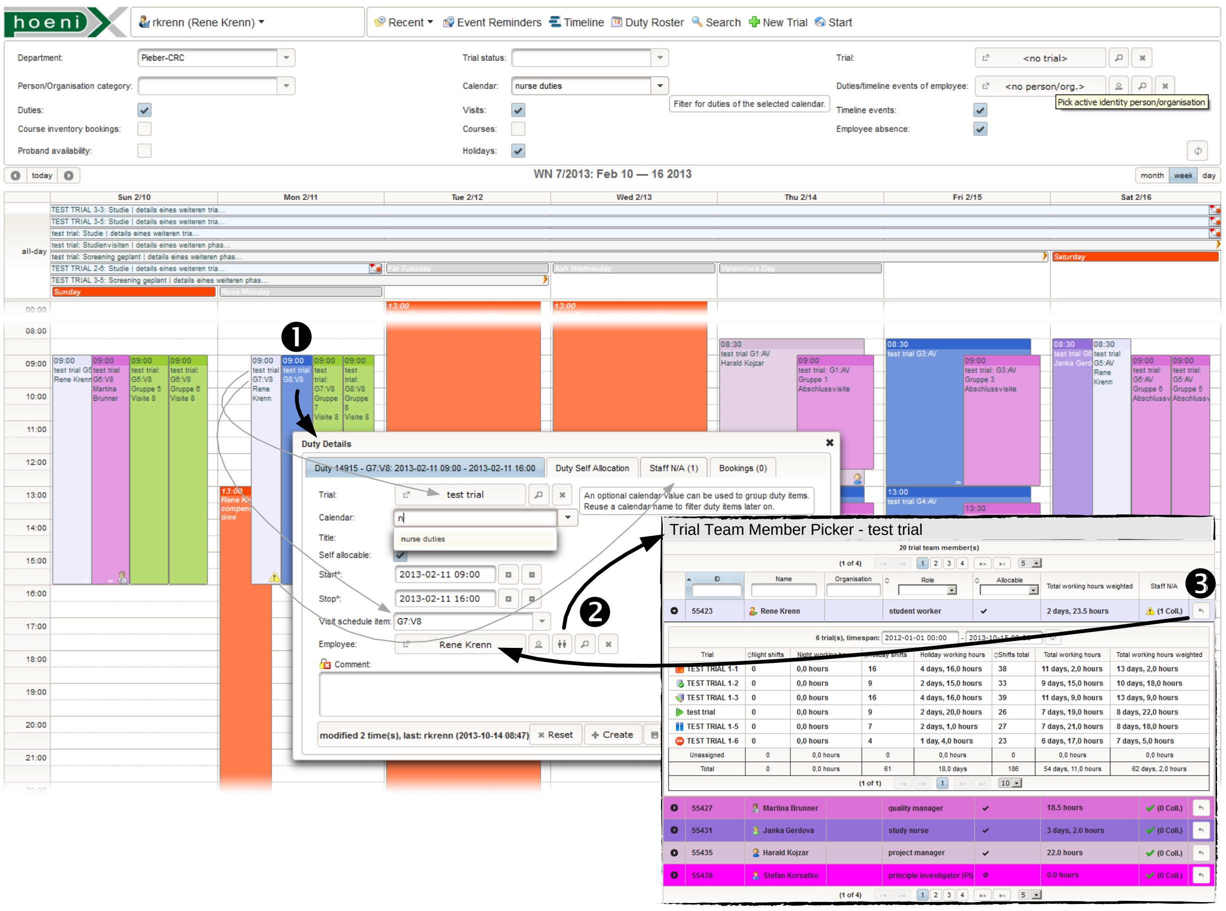
Duty roster calendar To manage the duty roster of allocable employees, related events below can be displayed and filtered by. However, the duty roster calendar view supports editing of duties only at the moment.

All day schedule events:

- Holidays
- Trial timeline events (`TimelineEvent`)
- Courses (`Course`)

Datetime interval schedule events:

- Duties (`DutyRosterTurn`)
- Trial visits (`VisitScheduleItem`)
- Employee availability (`StaffStatusEntry`)
- Proband availability (`ProbandStatusEntry`)
- Course inventory bookings (`InventoryBooking`)



Department:	Pieber-CRC	Trial status:		Trial:	<no trial>
Person/Organisation category:		Calendar:	nurse duties	Duties/timeline events of employee:	<no person/org.>
Duties:	✓	Visits:	✓	Timeline events:	✓
Course inventory bookings:		Courses:		Employee absence:	✓
Proband availability:		Holidays:	✓		

Filter for duties of the selected calendar.

Pick active identity person/organisation

WN 7/2013: Feb 10 — 16 2013

Sun 2/10 | Mon 2/11 | Tue 2/12 | Wed 2/13 | Thu 2/14 | Fri 2/15 | Sat 2/16

TEST TRIAL 3-3: Studie | details eines weiteren tria...
TEST TRIAL 3-5: Studie | details eines weiteren tria...
test trial: Studie | details eines weiteren tria...
test trial: Studienvisiten | details eines weiteren phas...
test trial: Screening geplant | details eines weiteren phas...
TEST TRIAL 2-6: Studie | details eines weiteren tria...
TEST TRIAL 3-5: Screening geplant | details eines weiteren phas...
Sunday
Rose Monday
Fat Tuesday
Ash Wednesday
Valentine's Day
Saturday

Duty Details

Duty 14915 - G7:V8: 2013-02-11 09:00 - 2013-02-11 16:00 | Duty Self Allocation | Staff N/A (1) | Bookings (0)

Trial:	test trial
Calendar:	n
	nurse duties
Title:	
Self allocable:	✓
Start*:	2013-02-11 09:00
Stop*:	2013-02-11 16:00
Visit schedule item:	G7:V8
Employee:	Rene Krenn
Comment:	

An optional calendar value can be used to group duty items. Reuse a calendar name to filter duty items later on.

modified 2 time(s), last: rkrenn (2013-10-14 08:47) Reset Create

Trial Team Member Picker - test trial

20 trial team member(s)

(1 of 4) 1 2 3 4 5

ID	Name	Organisation	Role	Allocable	Total working hours weighted	Staff N/A
55423	Rene Krenn		student worker	✓	2 days, 23.5 hours	⚠ (1 Coll.)

6 trial(s), timespan: 2012-01-01 00:00 - 2013-10-15 00:00

Trial	Night shifts	Night working hours	Holiday shifts	Holiday working hours	Shifts total	Total working hours	Total working hours weighted
TEST TRIAL 1-1	0	0,0 hours	16	4 days, 16,0 hours	38	11 days, 2,0 hours	13 days, 2,0 hours
TEST TRIAL 1-2	0	0,0 hours	9	2 days, 15,0 hours	33	9 days, 15,0 hours	10 days, 18,0 hours
TEST TRIAL 1-3	0	0,0 hours	16	4 days, 16,0 hours	39	11 days, 9,0 hours	13 days, 9,0 hours
test trial	0	0,0 hours	9	2 days, 20,0 hours	26	7 days, 19,0 hours	8 days, 22,0 hours
TEST TRIAL 1-5	0	0,0 hours	7	2 days, 1,0 hours	27	7 days, 21,0 hours	8 days, 18,0 hours
TEST TRIAL 1-6	0	0,0 hours	4	1 day, 4,0 hours	23	6 days, 17,0 hours	7 days, 5,0 hours
Unassigned	0	0,0 hours	0	0,0 hours	0	0,0 hours	0,0 hours
Total	0	0,0 hours	61	18,0 days	186	54 days, 11,0 hours	62 days, 2,0 hours

(1 of 1) 1 10

ID	Name	Organisation	Role	Allocable	Total working hours weighted	Staff N/A
55427	Martina Brunner		quality manager	✓	18.5 hours	✓ (0 Coll.)
55431	Janka Gerdova		study nurse	✓	3 days, 2.0 hours	✓ (0 Coll.)
55435	Harald Kojzar		project manager	✓	22.0 hours	✓ (0 Coll.)
55439	Stefan Korsatko		principle investigator (PI)	⊘	0.0 hours	✓ (0 Coll.)

(1 of 4) 1 2 3 4 5

Figure 4.84.: The duty roster calendar view is shown in the trial home browser tab. Existing shifts can be edited by clicking the corresponding schedule event (1) to show `p:dialog` for modifying event details. Collisions with other event types relevant for the allocated person are indicated by the ⚠ icon. A new duty is created by clicking in a blank region of the schedule grid. If the click position is covered by the time span of a visit appointment, corresponding fields are set by default (grey lines). To choose an employee from the set of trial team members, the transient trial team member picker pop-up can be opened (2). By expanding a listed team member's `p:rowExpansion`, a detailed shift summary table is shown. Administrative users can adjust the summary date range and compare totals. After selecting a desired employee from the list (3), the picker browser window closes and modifications can be saved.

Figure 4.84 shows how the duty roster calendar is used to edit a duty schedule event, e.g. in order to (re-)assign an employee. If the duty schedule event is marked to allow *self-allocation*, the dialog's second tab allows users with less privileges to "grab" or release the selected duty with a single click. An administrative user may assign an employee when creating or modifying the duty. The person/organization picker (figure 4.71) can be used to select an arbitrary employee. As an alternative, the trial team member picker shows a list of trial team members, if a trial is set in the dialog. Its datatable's row `p:rowExpansions` can display the employee's *shift summary* for a given date range[69]. This feature should facilitate a fair allocation. Since proband care includes night shifts as well as shifts on weekends and public holidays, shift summaries take *shift weighting* into account (table 4.13).

[69]The shift summary for a predefined time window around the duty date is shown by default.

configuration parameter	**value**	**description**
`night_interval_start_hour`	22	A night interval is defined by [`night_interval_start_hour`: `night_interval_start_minute`:00.000, next day `night_interval_stop_hour`: `night_interval_stop_minute`:00.000)
`night_interval_start_minute`	0	
`night_interval_stop_hour`	6	
`night_interval_stop_minute`	0	
`night_shift_threshold_secs`	14400	If the overlapping of the duty's time span [`start`, `stop`) and a night interval is $\geq$ `night_shift_threshold_secs`, the duty counts as *night shift*.
`holiday_shift_threshold_secs`	21600	If the overlapping of the duty's time span and a weekend interval [Saturday, 00:00:00.000 - Monday 00:00:00.000) or a public holiday interval [00:00:00.000, next day 00:00:00.000) is $\geq$ `holiday_shift_threshold_secs`, the duty counts as *holiday shift*.
`shift_extra_secs`	10800	Night and holiday shifts are awarded by adding `shift_extra_secs` to their duration.

Table 4.13.: Implemented configuration parameters for shift weighting.

The consideration of shift weighting when calculating shift totals implies the computation of public holiday dates, unless they are maintained manually. In addition, holidays will show up in the calendar views in form of all day events. The date H of a holiday or commemoration day can be expressed by a tuple consisting of one of five *base date functions* $t_B(y, month, dayofmonth, weekday, n)$ (table 4.14) and corresponding argument values:

$$H := (t_{B_H}, month_H, day\,of\,month_H, weekday_H, n_H, offset_H)$$

base date calculation t_B	**input parameters**	**description**	**example**
`STATIC_DATE`	year, month, day of month	a simple static date	New Year's Day
`EASTER_DATE`	year	the Easter Sunday's date for the given year (The Episcopal Church, 2000)	Whit Monday
`NTH_WEEKDAY_AFTER_DATE`	year, month, day of month, weekday, n	the date of the *n*-th weekday $\geq$ the given date	Mother's Day is the second Sunday after May, 1st.
`NTH_WEEKDAY_BEFORE_DATE`	year, month, day of month, weekday, n	the date of the *n*-th weekday $\leq$ the given date	Begin of DST is the last Sunday in March (the first Sunday before or equal March, 31st).
`WEEKDAYS`	year, weekday	a list of all dates of a specified weekday for a given year	all Saturdays of the year

Table 4.14.: Identified base dates for calculating dates of holidays and commemoration days of a given year.

The concrete holiday date $t_H(y)$ for a given year y is denoted by:

$$t_H(y) = t_{B_H}(y, month_H, day\,of\,month_H, weekday_H, n_H) + offset_H[day]$$

Relevant holiday date definitions H represent master data (table 4.15), which is stored by the `Holiday` entity (figure 4.85).

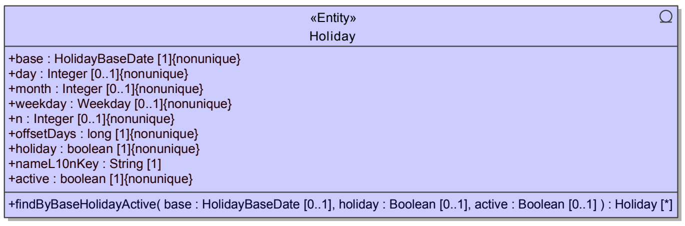

«Entity»
Holiday
+base : HolidayBaseDate [1]{nonunique}
+day : Integer [0..1]{nonunique}
+month : Integer [0..1]{nonunique}
+weekday : Weekday [0..1]{nonunique}
+n : Integer [0..1]{nonunique}
+offsetDays : long [1]{nonunique}
+holiday : boolean [1]{nonunique}
+nameL10nKey : String [1]
+active : boolean [1]{nonunique}
+findByBaseHolidayActive(base : HolidayBaseDate [0..1], holiday : Boolean [0..1], active : Boolean [0..1]) : Holiday [*]

Figure 4.85.: The `Holiday` entity provides fields to store holiday date definitions H.

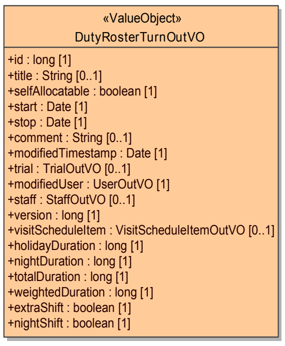
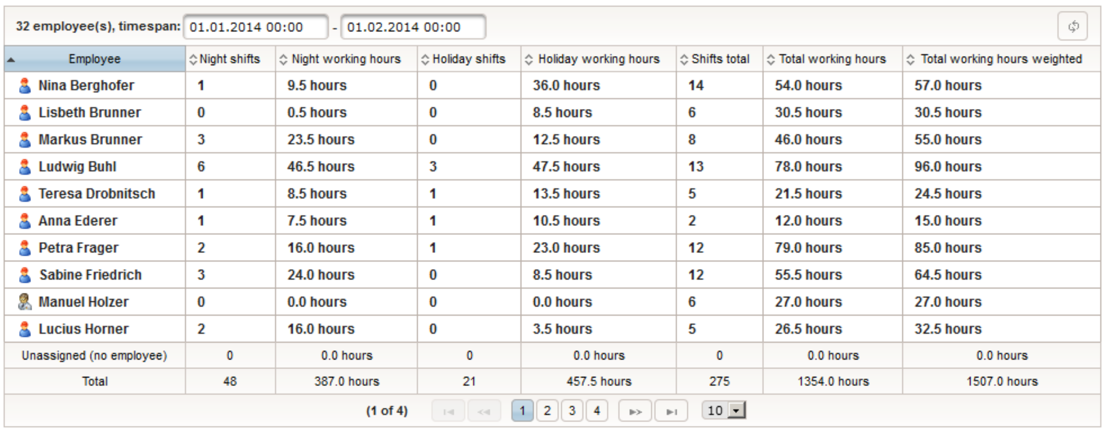

«ValueObject»
DutyRosterTurnOutVO
+id : long [1]
+title : String [0..1]
+selfAllocatable : boolean [1]
+start : Date [1]
+stop : Date [1]
+comment : String [0..1]
+modifiedTimestamp : Date [1]
+trial : TrialOutVO [0..1]
+modifiedUser : UserOutVO [1]
+staff : StaffOutVO [0..1]
+version : long [1]
+visitScheduleItem : VisitScheduleItemOutVO [0..1]
+holidayDuration : long [1]
+nightDuration : long [1]
+totalDuration : long [1]
+weightedDuration : long [1]
+extraShift : boolean [1]
+nightShift : boolean [1]

Figure 4.86.: The `DutyRosterTurnOutVO` OutVO's `*Duration` fields and `*Shift` flags are populated by the DAO transformation method.

holiday name	**base date calculation**	**month**	**day of month**	**weekday**	**n**	**offset**	**holiday**
Saturdays	`WEEKDAYS`			SATURDAY		0	✓
Sundays	`WEEKDAYS`			SUNDAY		0	✓
New Year's Day	`STATIC_DATE`	1	1			0	✓
Twelfth Day	`STATIC_DATE`	1	6			0	✓
Easter Sunday	`EASTER_DATE`					0	✓
Easter Monday	`EASTER_DATE`					1	✓
National Holiday	`STATIC_DATE`	5	1			0	✓
Mother's Day	`NTH_WEEKDAY_AFTER_DATE`	5	1	SUNDAY	2	0	✓
Ascension Day	`EASTER_DATE`					39	✓
Whit Sunday	`EASTER_DATE`					49	✓
Whit Monday	`EASTER_DATE`					50	✓
Corpus Christi	`EASTER_DATE`					60	✓
Assumption Day	`STATIC_DATE`	8	15			0	✓
Austrian National Holiday	`STATIC_DATE`	10	26			0	✓
Allhallows	`STATIC_DATE`	11	1			0	✓
1st Sunday in Advent	`NTH_WEEKDAY_BEFORE_DATE`	12	24	SUNDAY	1	-21	✓
2nd Sunday in Advent	`NTH_WEEKDAY_BEFORE_DATE`	12	24	SUNDAY	1	-14	✓
Immaculate Conception	`STATIC_DATE`	12	8			0	✓
3rd Sunday in Advent	`NTH_WEEKDAY_BEFORE_DATE`	12	24	SUNDAY	1	-7	✓
4th Sunday in Advent	`NTH_WEEKDAY_BEFORE_DATE`	12	24	SUNDAY	1	0	✓
Christmas Eve	`STATIC_DATE`	12	24			0	✓
Christmas Day	`STATIC_DATE`	12	25			0	✓
Boxing Day	`STATIC_DATE`	12	26			0	✓
New Year's Eve	`STATIC_DATE`	12	31			0	✓
Women's Carnival Day	`EASTER_DATE`					-52	✗
Rose Monday	`EASTER_DATE`					-48	✗
Fat Tuesday	`EASTER_DATE`					-47	✗
...							✗

Table 4.15.: Holiday date master data currently contains a list of 22 public holidays and 91 (international) commemoration days. The latter are not required by calculations and thus displayed only. Definitions were gathered from internet search and former projects.



32 employee(s), timespan: 01.01.2014 00:00 - 01.02.2014 00:00

Employee	Night shifts	Night working hours	Holiday shifts	Holiday working hours	Shifts total	Total working hours	Total working hours weighted
Nina Berghofer	1	9.5 hours	0	36.0 hours	14	54.0 hours	57.0 hours
Lisbeth Brunner	0	0.5 hours	0	8.5 hours	6	30.5 hours	30.5 hours
Markus Brunner	3	23.5 hours	0	12.5 hours	8	46.0 hours	55.0 hours
Ludwig Buhl	6	46.5 hours	3	47.5 hours	13	78.0 hours	96.0 hours
Teresa Drobnitsch	1	8.5 hours	1	13.5 hours	5	21.5 hours	24.5 hours
Anna Ederer	1	7.5 hours	1	10.5 hours	2	12.0 hours	15.0 hours
Petra Frager	2	16.0 hours	1	23.0 hours	12	79.0 hours	85.0 hours
Sabine Friedrich	3	24.0 hours	0	8.5 hours	12	55.5 hours	64.5 hours
Manuel Holzer	0	0.0 hours	0	0.0 hours	6	27.0 hours	27.0 hours
Lucius Horner	2	16.0 hours	0	3.5 hours	5	26.5 hours	32.5 hours
Unassigned (no employee)	0	0.0 hours	0	0.0 hours	0	0.0 hours	0.0 hours
Total	48	387.0 hours	21	457.5 hours	275	1354.0 hours	1507.0 hours

(1 of 4) 1 2 3 4 10

Figure 4.87.: This shift summary view example shows totals of employee shift hours spent for a specific trial during the given date range.

The `isHoliday` utility method is capable of deciding whether a given date is a public holiday or not. It is required to populate fields of the relevant `DutyRosterTurnOutVO` (figure 4.86). Totals as displayed in shift summary views (figure 4.87) are calculated by adding up shift durations, which requires to iterate a considerable number n of `DutyRosterTurn` records and days m as specified by the date range.[70] The computation takes $O(nm)$ time and entails a high frequency of `isHoliday` invocations, which can be significantly optimized by means of *memoization*. A static `Map` singleton of pre-calculated holiday dates limits the frequency of database read operations and thus speeds up summation. The adverse question about the extent of the initial population of the holiday dates map arises. The actual implementation is optimized to addresses these concerns by organizing holiday dates maps per *year* in a growing, superior map (`HashMap<Integer, HashMap<Date, HolidayVO>>`). Computed dates of an entire year are appended to the holiday date map *on demand*, whenever a year value of `isHoliday`'s date argument debuts. Concurrent servlet thread contexts therefore require synchronized access in order to ensure thread-safety.

4.6.6. Timeline View

The UI provides a single, full-page *timeline view* to display scheduled events in the trial home browser tab. The `pe:timeline` component is used to present a Gantt chart showing *timeline events* related to trials. Basically, any entity containing temporal interval information characterized by `start` and `stop` fields of either date or datetime type qualifies as a compatible timeline event type. To distinguish event types visually, CSS style classes are used which apply custom colors and icons. The timeline component's resolution can be interactively adjusted within a predefinable range ("zooming"). For event entities providing date and time information, the resolution of the time (x-) axis can therefore increased from years, months or weeks up hours and minutes. At the moment, two event types are supported:

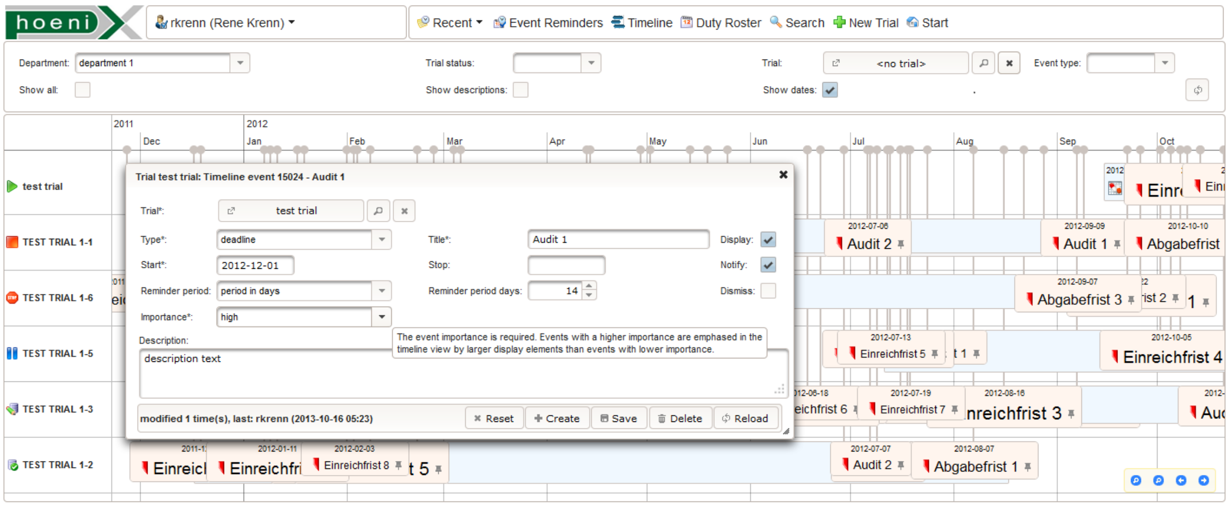
trial timeline events: `TimelineEvent` records with date fields to map a trial's timeline events such as deadlines, milestones and phases.

trial visit events: A trial's visit schedule is defined by visit appointments for each proband group, mapped by `VisitScheduleItem` records with datetime fields.

Events with non-null values for both `start` and `stop` fields represent a time interval (e.g. project phase, visit appointment), while events with a start value only are displayed as single points in time (e.g.

[70]For simplification, a single duty does not span across multiple days. The implemented algorithm is capable of handling this case however.

deadlines, milestones). Time intervals are depicted by a bar of corresponding size, single points in time by a box and a marker line perpendicular to the x-axis. Interactive edit actions (event move and resize) trigger applicable update operations to persist changes. The service methods' success or failure response messages are displayed by a fading `p:growl` box in the upper right screen corner. A `p:dialog` component overlays an edit area to modify trial timeline event details, when an item is clicked. Likewise, the edit area dialog shows input elements with blank/preset values to create a new trial timeline event, upon clicking in an empty region.



Department: department 1
Trial status:
Trial: <no trial>
Event type:
Show all:
Show descriptions:
Show dates: ✓

Trial test trial: Timeline event 15024 - Audit 1

Trial*: test trial
Type*: deadline
Title*: Audit 1
Display: ✓
Start*: 2012-12-01
Stop:
Notify: ✓
Reminder period: period in days
Reminder period days: 14
Dismiss:
Importance*: high
Description: description text

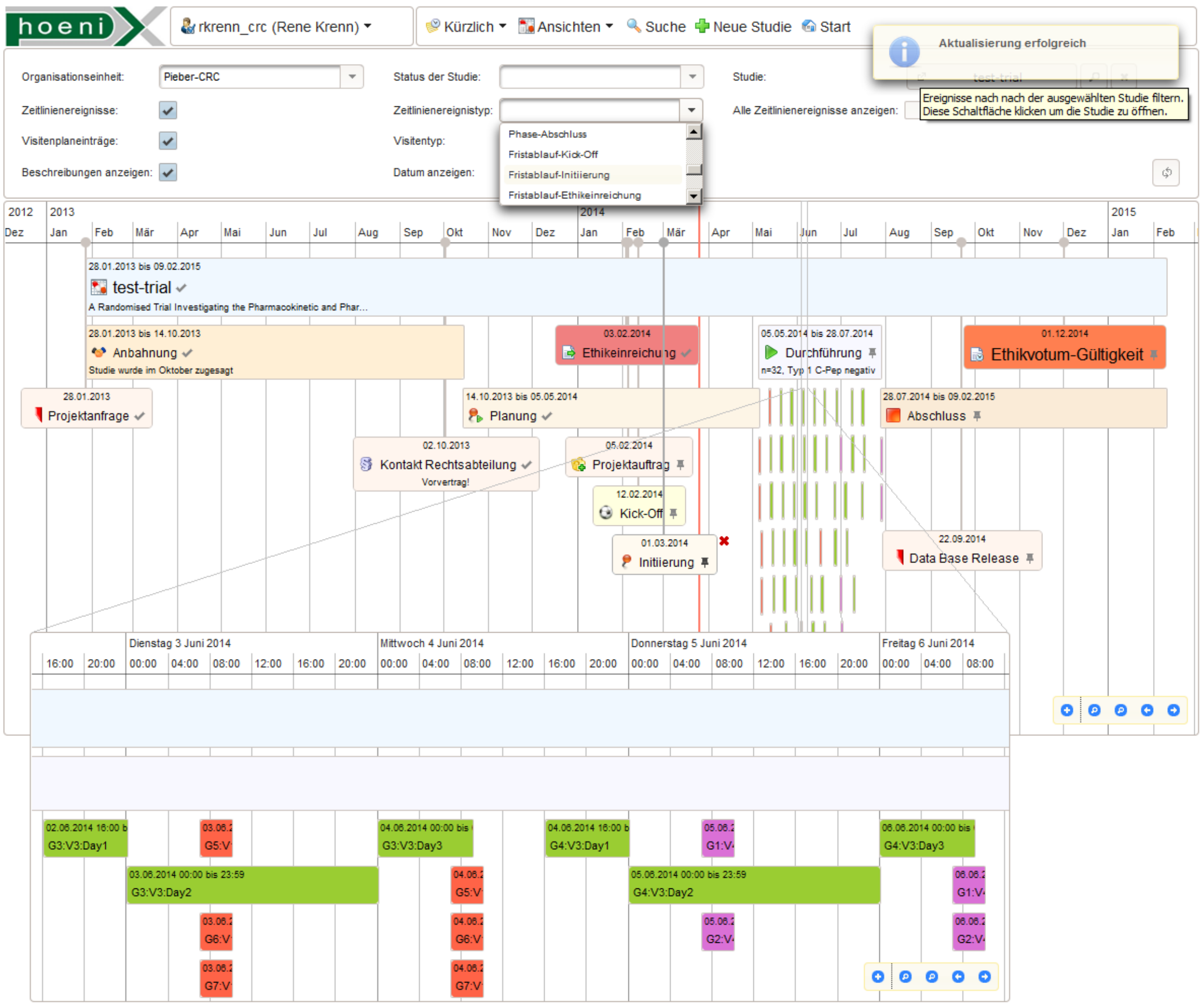
The event importance is required. Events with a higher importance are emphased in the timeline view by larger display elements than events with lower importance.

modified 1 time(s), last: rkrenn (2013-10-16 05:23)

Figure 4.88.: In *group mode*, the timeline view visualizes events records of multiple trials. A `p:dialog` is used to display and modify event details, similar to calendar views. The event importance property defines the font scaling of the event's title. The "Display" flag can be used to hide the event.

primefaces-extension's `pe:timeline` is a wrapper component for the timeline.js library of the Common Hybrid Agent Platform (CHAP) Link Library (Jong, 2013), capable of rendering hundreds of events with acceptable performance. In order to support a large number of event records, the version of primefaces-extensions used required a workaround to enable lazy-loading. Event records for the displayed time span are retrieved using finder methods according to listing 4.3.

The client-side rendering provided by timeline.js supports two modes for the layout of events (group and default mode). Group mode requires a group label (e.g. `TimelineEvent.trial.id`) per event item in order to organize events into display groups. Multiple groups are arranged one below the other, while a group's event bars and boxes are rendered in one line and may overlap. Grouping is used to display events of multiple trials, e.g. if no event filters are set (figure 4.88). If filters narrow the `TimelineEvent` result set to consist of a single trial only, the default display mode without grouping (figure 4.89) is applied. Events are arranged in several lines as required using a reflow algorithm in order to prevent overlapping in this case.


Organisationseinheit: Pieber-CRC
Status der Studie:
Studie:
Zeitlinienereignisse: ✓
Zeitlinienereignistyp:
Alle Zeitlinienereignisse anzeigen:
Visitenplaneinträge: ✓
Visitentyp:
Beschreibungen anzeigen: ✓
Datum anzeigen:

Aktualisierung erfolgreich

Ereignisse nach nach der ausgewählten Studie filtern. Diese Schaltfläche klicken um die Studie zu öffnen.

Figure 4.89.: The timeline component's default rendering mode (without event grouping) is used to display events of a single trial. Without item overlapping, it is easier to use interactive actions for manipulating events. After selecting, an event gets the focus and appears in the foreground (highest z-order). A focused event can be moved, resized or hidden (✖ icon). These modifications are persisted instantly using applicable service methods and confirmed by transient `p:growl` messages. Dismissed timeline events are indicated by the ✓ icon, while events not dismissed yet are indicated by the 📌 icon. A high density of events is typical for the trial visit schedule and can be mastered with the timeline view's fluent zooming (mouse wheel, pinch-to-zoom) and panning capabilities.

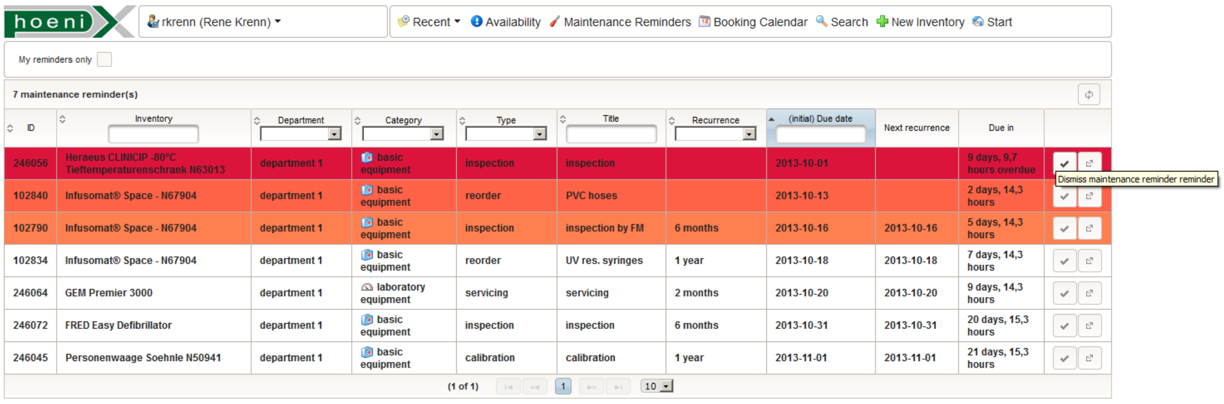
4.6.7. Overview Lists

Beside query editors, timeline and calendar views, users can open *overview list* pages when navigating a module's home tab. Overview lists are basically full-page `p:dataTables` without an explicit item edit area. However, instant update operations without the need to enter record details can be triggered by components placed per table row, such as `p:commandButton` or `p:selectOneMenu`. A short description is given per overview list page:

Inventory module

Inventory availability: This summary list shows inventory items which are unavailable at the moment. For each inventory listed, at least a single `InventoryStatusEntry` item encloses the current date.

Maintenance reminders: A maintenance reminder item shows up in the list, if the current date is within its reminder period. Maintenance reminders are presented in the style of a checklist (figure 4.90). Users can *dismiss* resolved maintenance tasks.



7 maintenance reminder(s)

ID	Inventory	Department	Category	Type	Title	Recurrence	(initial) Due date	Next recurrence	Due in
246056	Heraeus CLINICIP -80°C Tieftemperaturenschrank N63013	department 1	basic equipment	inspection	inspection		2013-10-01		9 days, 9,7 hours overdue
102840	Infusomat® Space - N67904	department 1	basic equipment	reorder	PVC hoses		2013-10-13		2 days, 14,3 hours
102790	Infusomat® Space - N67904	department 1	basic equipment	inspection	inspection by FM	6 months	2013-10-16	2013-10-16	5 days, 14,3 hours
102834	Infusomat® Space - N67904	department 1	basic equipment	reorder	UV res. syringes	1 year	2013-10-18	2013-10-18	7 days, 14,3 hours
246064	GEM Premier 3000	department 1	laboratory equipment	servicing	servicing	2 months	2013-10-20	2013-10-20	9 days, 14,3 hours
246072	FRED Easy Defibrillator	department 1	basic equipment	inspection	inspection	6 months	2013-10-31	2013-10-31	20 days, 15,3 hours
246045	Personenwaage Soehnle N50941	department 1	basic equipment	calibration	calibration	1 year	2013-11-01	2013-11-01	21 days, 15,3 hours

Figure 4.90.: Maintenance reminder items are hidden when dismissed. Recurring reminders will show up again when the reminder period of the next recurrence begins.

Utilization: A summary view breaks down totals of inventory booking durations related to a given date range.

Person/organisation module

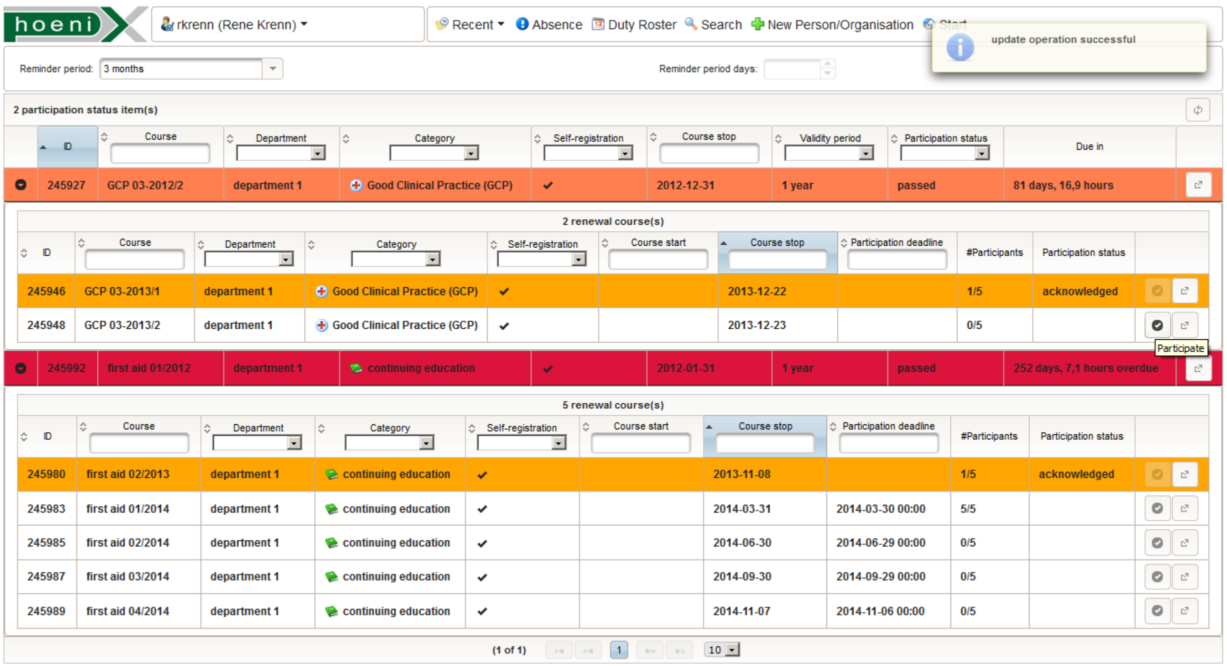
Employee absence: This overview list shows employees which are absent at the moment. For each employee listed, at least a single `StaffStatusEntry` item encloses the current date.

Shift summary: A shift summary view breaks down totals of shift durations *per employee*.

Upcoming and ongoing courses: Courses with a participation deadline[71] in the future are displayed including the participation status of the active user's identity `Staff`. A ✔ button ("Participate") allows self-registration for corresponding courses. Courses configured for self-registration are hidden, if the limit for the number of participants is exceeded already.

My expiring courses: Courses passed by the active user's identity that are about to expire are listed at-a-glance (figure 4.91). An expired course requires the attendance of one of the available refresher courses, which are displayed by a nested datatable. The list of refresher courses can be shown/hidden by expanding/collapsing the expiring course row's `p:rowExpansion` area.

[71]The course's stop date is used if a participation deadline is not specified.



update operation successful

Reminder period: 3 months

Reminder period days:

2 participation status item(s)

	ID	Course	Department	Category	Self-registration	Course stop	Validity period	Participation status	Due in
	245927	GCP 03-2012/2	department 1	Good Clinical Practice (GCP)	✓	2012-12-31	1 year	passed	81 days, 16,9 hours

2 renewal course(s)

ID	Course	Department	Category	Self-registration	Course start	Course stop	Participation deadline	#Participants	Participation status
245946	GCP 03-2013/1	department 1	Good Clinical Practice (GCP)	✓		2013-12-22		1/5	acknowledged
245948	GCP 03-2013/2	department 1	Good Clinical Practice (GCP)	✓		2013-12-23		0/5	

	ID	Course	Department	Category	Self-registration	Course stop	Validity period	Participation status	Due in
	245992	first aid 01/2012	department 1	continuing education	✓	2012-01-31	1 year	passed	252 days, 7,1 hours overdue

5 renewal course(s)

ID	Course	Department	Category	Self-registration	Course start	Course stop	Participation deadline	#Participants	Participation status
245980	first aid 02/2013	department 1	continuing education	✓		2013-11-08		1/5	acknowledged
245983	first aid 01/2014	department 1	continuing education	✓		2014-03-31	2014-03-30 00:00	5/5	
245985	first aid 02/2014	department 1	continuing education	✓		2014-06-30	2014-06-29 00:00	0/5	
245987	first aid 03/2014	department 1	continuing education	✓		2014-09-30	2014-09-29 00:00	0/5	
245989	first aid 04/2014	department 1	continuing education	✓		2014-11-07	2014-11-06 00:00	0/5	

(1 of 1) 1 10

Figure 4.91.: Users can register to refresher courses of their expiring courses, if available. Clicking the corresponding button invokes a specific update operation to perform the registration. A success or error message is displayed by the `p:growl` box in the upper right corner.

Course module

Upcoming and ongoing courses: In contrast to the person/organization module's list of upcoming and ongoing courses, courses with expired participation deadline or exceeded number of participants are not hidden. A custom row color coding reflects the progress of participations:

yellow: The course allows self-registration and has just been created. Participants might have been invited by a course administrator by creating participations with the initial "invited" state, but no participant responded with the "acknowledged" state yet. Otherwise, there are no participations yet at all.

orange: The course is not yet over, thus there is no "passed" participation state at all. However, some participants have set their participation state to "acknowledged" already.

lime: The state of any participation is either set to "acknowledged" or "passed".

red: At least one participant of a course *without self-registration* set his/her participation state to "cancelled".

dark orange: Remaining participation constellations of both courses with self-registration and registration by administrator can include:

- participations with both "passed" and "failed" state
- participations are still set to their initial state ("invited"/"registered")
- participations of a self-registration course are set to "cancelled"

Courses without refresher courses: Course administrators need to overview courses that are about to expire in order to plan and schedule refresher courses.

Expiring courses per participant: While the "My expiring courses" overview lists expiring courses of the individual user, this overview list shows expiring courses attended by any person. Each row represents a course participation of a particular course and person and can be expanded to display available refresher courses. Course administrators can invite the person directly to one of these refresher courses.

Trial module

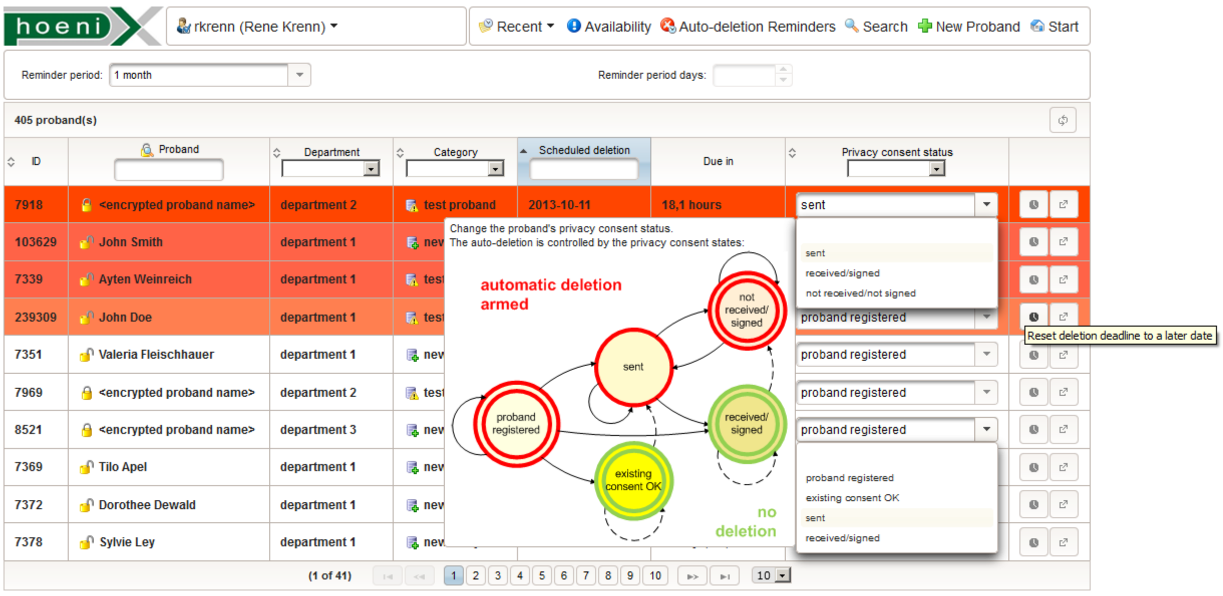
Timeline event reminders: Timeline events always define a start date, wether they represent trial milestones or phases. A timeline event shows up in this overview list, if the current date reaches the timeline event's reminder period relating to its start date. Timeline events are presented in the style of a checklist. Users can dismiss resolved/achieved milestones.

Shift summary: A shift summary view breaks down totals of shift durations *per trial*.

Proband module

Proband availability: This overview list shows probands which are unavailable at the moment. For each proband listed, at least a single `ProbandStatusEntry` item encloses the current date.

Proband auto-deletion reminders: Probands entered during recruitment are pending for automatic deletion initially. They appear on this list which allows tracking and modifying the state of their data privacy consent (figure 4.92).



Reminder period: 1 month

Reminder period days:

405 proband(s)

Change the proband's privacy consent status.
The auto-deletion is controlled by the privacy consent states:

ID	Proband	Department	Category	Scheduled deletion	Due in	Privacy consent status
7918	<encrypted proband name>	department 2	test proband	2013-10-11	18,1 hours	sent
103629	John Smith	department 1	nev			
7339	Ayten Weinreich	department 1	tes			
239309	John Doe	department 1	tes			proband registered
7351	Valeria Fleischhauer	department 1	nev			proband registered
7969	<encrypted proband name>	department 2	tes			proband registered
8521	<encrypted proband name>	department 3	nev			proband registered
7369	Tilo Apel	department 1	nev			
7372	Dorothee Dewald	department 1	nev			
7378	Sylvie Ley	department 1	nev			

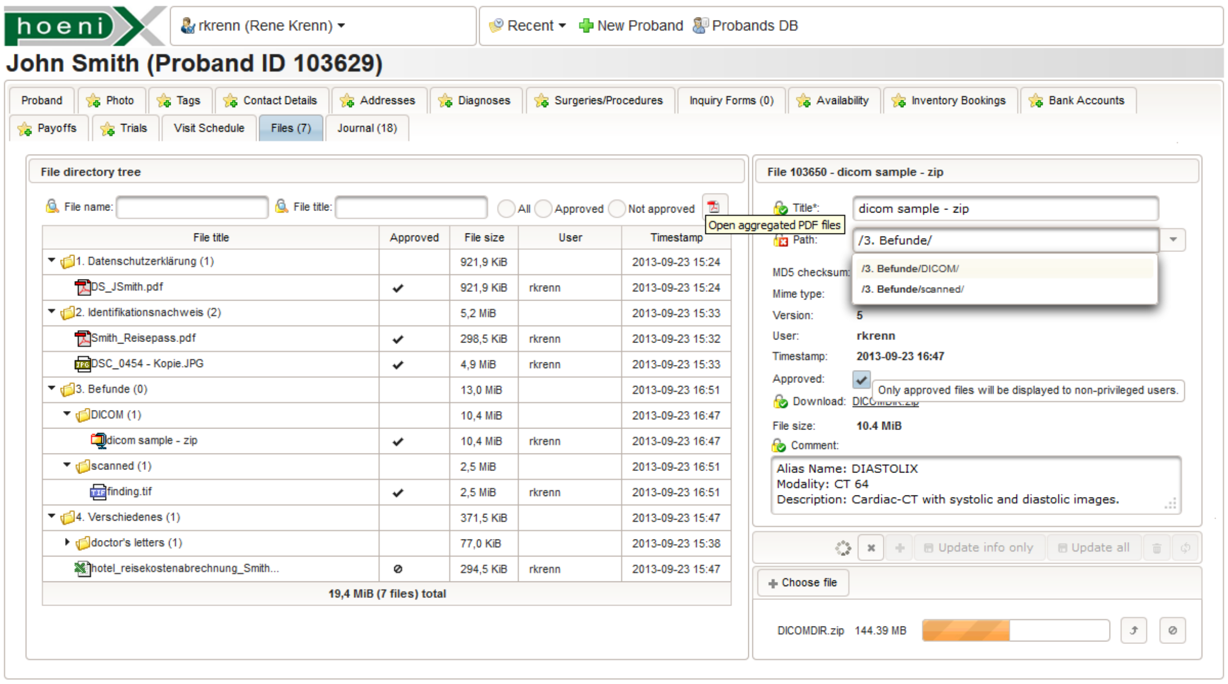
sent
received/signed
not received/not signed

proband registered
existing consent OK
sent
received/signed

Figure 4.92.: The workflow to obtain the probands' signed data privacy consent relies on this reminder overview list, showing pending deletions of newly created proband records.

4.7. Document Management

Phoenix CTMS database modules integrate competitive document management capabilities. Each root entity instance gets its own virtual file system. The file repository is presented in form of a complete UI for file browsing (figure 4.93) in the "Files" tab of entity views. Since intensive use is expected, the implementation challenges multiple topics regarding performance and usability.



John Smith (Proband ID 103629)

File directory tree

File name: File title: All Approved Not approved

File title	Approved	File size	User	Timestamp
1. Datenschutzerklärung (1)		921,9 KiB		2013-09-23 15:24
DS_JSmith.pdf	✓	921,9 KiB	rkrenn	2013-09-23 15:24
2. Identifikationsnachweis (2)		5,2 MiB		2013-09-23 15:33
Smith_Reisepass.pdf	✓	298,5 KiB	rkrenn	2013-09-23 15:32
DSC_0454 - Kopie.JPG	✓	4,9 MiB	rkrenn	2013-09-23 15:33
3. Befunde (0)		13,0 MiB		2013-09-23 16:51
DICOM (1)		10,4 MiB		2013-09-23 16:47
dicom sample - zip	✓	10,4 MiB	rkrenn	2013-09-23 16:47
scanned (1)		2,5 MiB		2013-09-23 16:51
finding.tif	✓	2,5 MiB	rkrenn	2013-09-23 16:51
4. Verschiedenes (1)		371,5 KiB		2013-09-23 15:47
doctor's letters (1)		77,0 KiB		2013-09-23 15:38
hotel_reisekostenabrechnung_Smith...	⊘	294,5 KiB	rkrenn	2013-09-23 15:47
19,4 MiB (7 files) total				

File 103650 - dicom sample - zip

Title*:	dicom sample - zip
Path:	/3. Befunde/
	/3. Befunde/DICOM/
	/3. Befunde/scanned/
MD5 checksum:	
Mime type:	
Version:	5
User:	rkrenn
Timestamp:	2013-09-23 16:47
Approved:	✓ Only approved files will be displayed to non-privileged users.
Download:	DICOMDIR.zip
File size:	10.4 MiB
Comment:	

Alias Name: DIASTOLIX
Modality: CT 64
Description: Cardiac-CT with systolic and diastolic images.

DICOMDIR.zip 144.39 MB

Figure 4.93.: This sample UI shows the *file manager* tab of an opened proband. An existing file record is updated by a uploading another large file.

4.7.1. File Upload and Storage

To store a file, it is transferred from the user workstation's local file system source to the system by means of *HTTP upload*[72]. The UI uses PrimeFaces's `p:fileUpload` component that supports drag-and-drop file uploads[73] by means of the HTML5 File API. A progress bar indicates the transfer progress, relevant for uploading large files[74]. A queueing mechanism (batch upload) supports sequential uploading of multiple files selected or dragged at once. `p:fileUpload` employs Apache Commons FileUpload to *stream* incoming file data into a temporary file in order to preserve memory, if it exceeds a configurable threshold. The same technique of handling large files is pursued throughout the related core tier's service and DAO methods.

[72]A HTTP POST request containing encoded binary data as request body.
[73]Available on selected browsers only, excluding Internet Explorer (IE).
[74]The maximum file size is actually limited by `Integer.MAX_VALUE` (2147483647 bytes).

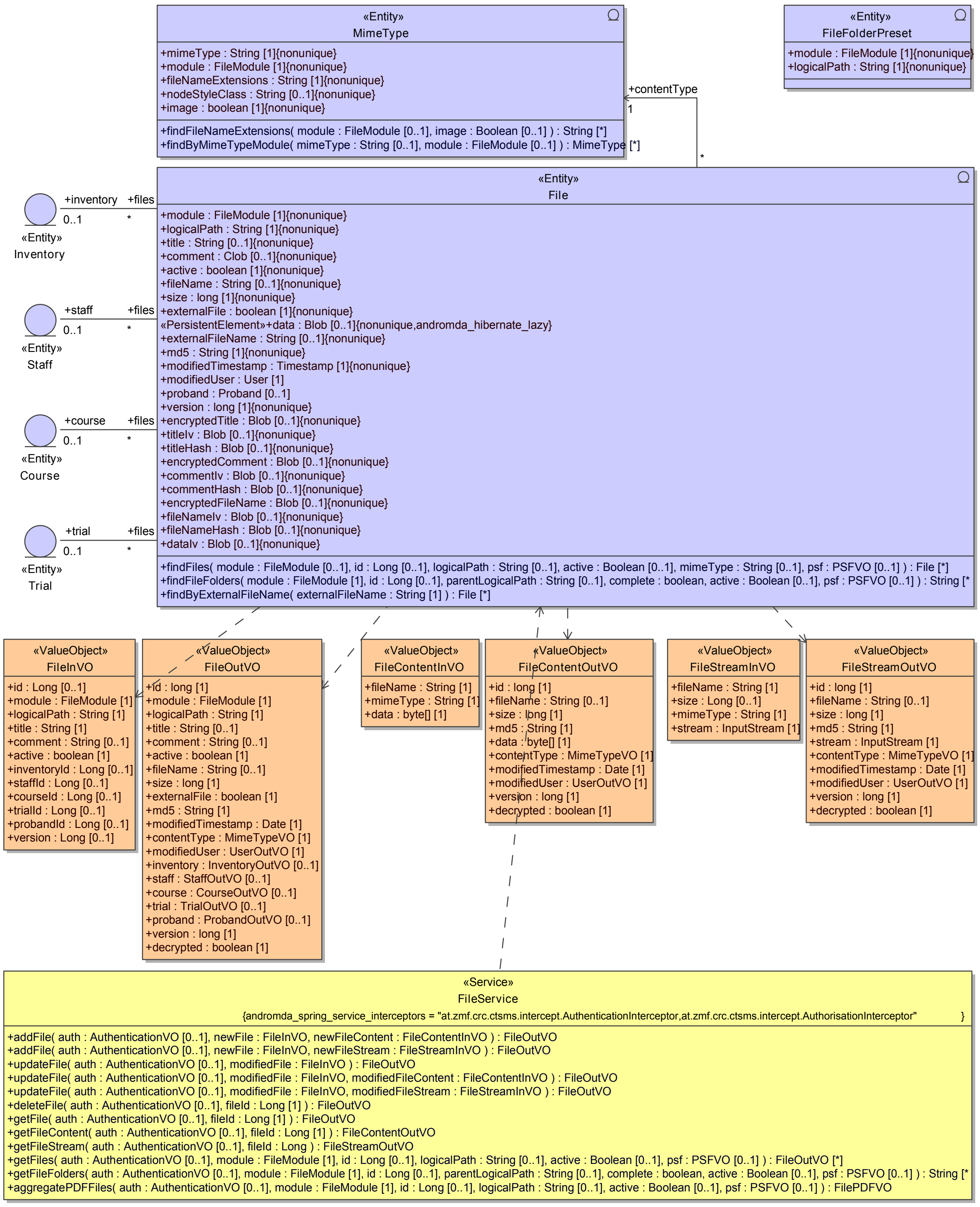

«Entity»
MimeType
+mimeType : String [1]{nonunique}
+module : FileModule [1]{nonunique}
+fileNameExtensions : String [1]{nonunique}
+nodeStyleClass : String [0..1]{nonunique}
+image : boolean [1]{nonunique}
+findFileNameExtensions(module : FileModule [0..1], image : Boolean [0..1]) : String [*]
+findByMimeTypeModule(mimeType : String [0..1], module : FileModule [0..1]) : MimeType [*]
«Entity»
FileFolderPreset
+module : FileModule [1]{nonunique}
+logicalPath : String [1]{nonunique}
+contentType
1
*
«Entity»
File
+module : FileModule [1]{nonunique}
+logicalPath : String [1]{nonunique}
+title : String [0..1]{nonunique}
+comment : Clob [0..1]{nonunique}
+active : boolean [1]{nonunique}
+fileName : String [0..1]{nonunique}
+size : long [1]{nonunique}
+externalFile : boolean [1]{nonunique}
«PersistentElement»+data : Blob [0..1]{nonunique,andromda_hibernate_lazy}
+externalFileName : String [0..1]{nonunique}
+md5 : String [1]{nonunique}
+modifiedTimestamp : Timestamp [1]{nonunique}
+modifiedUser : User [1]
+proband : Proband [0..1]
+version : long [1]{nonunique}
+encryptedTitle : Blob [0..1]{nonunique}
+titleIv : Blob [0..1]{nonunique}
+titleHash : Blob [0..1]{nonunique}
+encryptedComment : Blob [0..1]{nonunique}
+commentIv : Blob [0..1]{nonunique}
+commentHash : Blob [0..1]{nonunique}
+encryptedFileName : Blob [0..1]{nonunique}
+fileNameIv : Blob [0..1]{nonunique}
+fileNameHash : Blob [0..1]{nonunique}
+dataIv : Blob [0..1]{nonunique}
+findFiles(module : FileModule [0..1], id : Long [0..1], logicalPath : String [0..1], active : Boolean [0..1], mimeType : String [0..1], psf : PSFVO [0..1]) : File [*]
+findFileFolders(module : FileModule [1], id : Long [0..1], parentLogicalPath : String [0..1], complete : boolean, active : Boolean [0..1], psf : PSFVO [0..1]) : String [*
+findByExternalFileName(externalFileName : String [1]) : File [*]
+inventory
+files
0..1
*
«Entity»
Inventory
+staff
+files
0..1
*
«Entity»
Staff
+course
+files
0..1
*
«Entity»
Course
+trial
+files
0..1
*
«Entity»
Trial
«ValueObject»
FileInVO
+id : Long [0..1]
+module : FileModule [1]
+logicalPath : String [1]
+title : String [1]
+comment : String [0..1]
+active : boolean [1]
+inventoryId : Long [0..1]
+staffId : Long [0..1]
+courseId : Long [0..1]
+trialId : Long [0..1]
+probandId : Long [0..1]
+version : Long [0..1]
«ValueObject»
FileOutVO
+id : long [1]
+module : FileModule [1]
+logicalPath : String [1]
+title : String [0..1]
+comment : String [0..1]
+active : boolean [1]
+fileName : String [0..1]
+size : long [1]
+externalFile : boolean [1]
+md5 : String [1]
+modifiedTimestamp : Date [1]
+contentType : MimeTypeVO [1]
+modifiedUser : UserOutVO [1]
+inventory : InventoryOutVO [0..1]
+staff : StaffOutVO [0..1]
+course : CourseOutVO [0..1]
+trial : TrialOutVO [0..1]
+proband : ProbandOutVO [0..1]
+version : long [1]
+decrypted : boolean [1]
«ValueObject»
FileContentInVO
+fileName : String [1]
+mimeType : String [1]
+data : byte[] [1]
«ValueObject»
FileContentOutVO
+id : long [1]
+fileName : String [0..1]
+size : long [1]
+md5 : String [1]
+data : byte[] [1]
+contentType : MimeTypeVO [1]
+modifiedTimestamp : Date [1]
+modifiedUser : UserOutVO [1]
+version : long [1]
+decrypted : boolean [1]
«ValueObject»
FileStreamInVO
+fileName : String [1]
+size : Long [0..1]
+mimeType : String [1]
+stream : InputStream [1]
«ValueObject»
FileStreamOutVO
+id : long [1]
+fileName : String [0..1]
+size : long [1]
+md5 : String [1]
+stream : InputStream [1]
+contentType : MimeTypeVO [1]
+modifiedTimestamp : Date [1]
+modifiedUser : UserOutVO [1]
+version : long [1]
+decrypted : boolean [1]
«Service»
FileService
{andromda_spring_service_interceptors = "at.zmf.crc.ctsms.intercept.AuthenticationInterceptor,at.zmf.crc.ctsms.intercept.AuthorisationInterceptor" }
+addFile(auth : AuthenticationVO [0..1], newFile : FileInVO, newFileContent : FileContentInVO) : FileOutVO
+addFile(auth : AuthenticationVO [0..1], newFile : FileInVO, newFileStream : FileStreamInVO) : FileOutVO
+updateFile(auth : AuthenticationVO [0..1], modifiedFile : FileInVO) : FileOutVO
+updateFile(auth : AuthenticationVO [0..1], modifiedFile : FileInVO, modifiedFileContent : FileContentInVO) : FileOutVO
+updateFile(auth : AuthenticationVO [0..1], modifiedFile : FileInVO, modifiedFileStream : FileStreamInVO) : FileOutVO
+deleteFile(auth : AuthenticationVO [0..1], fileId : Long [1]) : FileOutVO
+getFile(auth : AuthenticationVO [0..1], fileId : Long [1]) : FileOutVO
+getFileContent(auth : AuthenticationVO [0..1], fileId : Long [1]) : FileContentOutVO
+getFileStream(auth : AuthenticationVO [0..1], fileId : Long) : FileStreamOutVO
+getFiles(auth : AuthenticationVO [0..1], module : FileModule [1], id : Long [0..1], logicalPath : String [0..1], active : Boolean [0..1], psf : PSFVO [0..1]) : FileOutVO [*]
+getFileFolders(auth : AuthenticationVO [0..1], module : FileModule [1], id : Long [0..1], parentLogicalPath : String [0..1], complete : boolean, active : Boolean [0..1], psf : PSFVO [0..1]) : String [*
+aggregatePDFFiles(auth : AuthenticationVO [0..1], module : FileModule [1], id : Long [0..1], logicalPath : String [0..1], active : Boolean [0..1], psf : PSFVO [0..1]) : FilePDFVO

Figure 4.94.: The `FileService` provides CRUD operations to manage `File` records, including binary file data.

An uploaded file is persisted using the `File` entity that aggregates all required attributes (figure 4.94). It provides two options to persist file contents:

Database storage Binary data can be stored in database tables directly using either a byte array or a BLOB column. A RDBMS typically handles byte array columns like the majority of its supported column value types, for example in the way how operators of a SQL statement are applied. Since these operations are performed in memory, the RDBMS tend to have performance issues if the size of stored values exceeds a certain range. A RDBMS can provide BLOB as alternative column type with extraordinary handling, which supports streaming for database interface drivers like JDBC. By default, AndroMDA templates map both ≪byte[]≫ and ≪BLOB≫ UML data types types to byte array table columns and the Java `byte[]` type. Hence, the option of BLOB streaming is not available to application code without template modification, although it is supported by Hibernate. File contents need to be kept in memory by handling `byte[]` variables. The data model provides a byte array column (`File.data`) to store file content. However, service methods ensure a size limit for `byte[]` values in this case. If the size of a file to be stored is too large, the alternative of file system persistence described next is to be used.

File system storage The uploaded file is passed as stream to write directly to a corresponding file located in a configurable directory within a file system mounted by the application server. To avoid collisions, `java.util.UUID.randomUUID` is used to generate a Universally Unique Identifier (UUID) as random file name.[75] The `File` record references the physical file by saving the file name (`File.externalFileName`). Since updating the `File` record containing metadata only requires minimal time compared to writing the large binary content into a file, VO types separate file metadata and content. While transaction isolation is directly supported by a RDBMS for byte array/BLOB storage, the combination of database and file system access results in a *distributed transaction* scenario. It is encountered by multiple measures that cope with concurrent update, delete and retrieval operations:

1. *Row locking:* A database transaction covers the entire execution duration of a service method's logic, which now includes file system access. At the beginning of update and delete operations, a write lock is acquired for the designated `File` record to prevent lost updates for concurrent updates. In the context of long-lasting transactions it is desired to abort and throw an exception rather than to block the program flow (and thus the UI) until a lock can be obtained. This behaviour can be achieved by using the `org.hibernate.LockMode.UPGRADE_NOWAIT` parameter value for Hibernate's `Session.load` operations, which causes to emit `SELECT ... FOR UPDATE` *`NOWAIT`* SQL statements. Instead of waiting, the RDBMS will raise an error in this case. The exception is propagated to immediately inform the attempting user about the concurrent modification in progress.
2. *Write file before persisting record:* A file is always existent before it is referenced by a database record. Input validation is performed right after row locking and would result in premature interruption without modifications to the file system or database. If the subsequent file write process is interrupted due to an error (e.g. `java.io.IOException`), an *orphaned file* is left over but the database content is not touched and remains consistent. The same situation arises, if the database transaction fails to commit in the end. To minimize the number of orphaned files, service methods attempt to remove a file immediately, if the database transaction is going to be cancelled. Create operations can guarantee to have valid references at any given time. This even holds for uncommitted `File` items visible to a parallel transaction due to a phantom read.
3. *Update file by creating another file:* Updating file content is implemented by creating a new file instead of overwriting the existing one. During a long-lasting update operation, concurrent retrieval operations consistently stream the old file, as long as the database transaction is not committed yet.

[75]Files are saved in a single file system directory per system module, resulting in a large number of files per directory over time. This could be refactored in a future version but will not cause problems if the storage partition uses a modern file system like Fourth Extended Filesystem (ext4), which scale well with large directories for file open, create, and delete operations. Listing directory contents ($O(n)$) is never performed by the application logic.

A concurrent update or delete operation is immediately aborted by row locking otherwise (see 1. Row locking).

3. *Ignore deleting files:* Delete operations will remove database records only, while associated files are left over in the file system. This intentionally results in additional orphaned files (see 2. Write file before persisting record) that subject to a deferred cleanup, but let concurrent retrieval operations complete without interference. While concurrent retrieval operation can continue to stream a requested file, a simultaneous update or delete operation is immediately aborted by row locking otherwise (see 1. Row locking). The CLI tool provides a task option to schedule cleanup of orphaned files from time to time.

The current web tier implementation sticks with the file system persistence mode to uniformly store managed files in the file system only. A reason for still supporting to store file contents in a table column is to support a configuration option to disable storing any data outside the database. This alternative mode of operation requires to enable the size limitation for uploaded files. Overloads of `FileService` methods (figure 4.94) provide `byte[]` parameters (`FileContentInVO`, `FileContentOutVO`) instead of `java.io.InputStream` parameters (`FileStreamInVO`, `FileStreamOutVO`). Method argument and return value serialization may cause difficulties with stream objects when trying to generate Simple Object Access Protocol (SOAP) web services using AndroMDA. In this case, the service methods with `byte[]` parameter are supposed to be published instead.

File encryption and decryption Proband documents such as signed proband letters and diagnostic findings will contain PII, even their file names can allude the proband's identity. This also applies to selected documents of the trial master file. Hence, proband and trial file repositories must support encryption of concerned `File` record fields:

- file content (`File.data`)
- file name (`File.fileName`)
- title (`File.title`)
- comment (`File.comment`)

Note that the document path information (`File.logicalPath`) is *not* encrypted. This would conflict with the implementation of a finder method for listing virtual directory contents, which is based on querying path substrings. While column encryption is used to protect `File` fields including the `data` byte array column, document content stored as files in the file system requires a separate encryption. Files are written to disk using `java.io.FileOutputStream` and read from disk using `FileInputStream`. To wire encryption for output streams, `javax.crypto.CipherOutputStream` combines a given output stream and a `Cipher` block cipher object (listing 4.9) configured to `ENCRYPT_MODE`. Decryption of input streams is provided by `CipherInputStream` (listing 4.39), which combines a given input stream and a `Cipher` configured to `DECRYPT_MODE`. Both cipher stream classes will apply `Cipher.update` to processed `byte[]` chunks.

```
public class FileDaoImpl extends FileDaoBase {

    ...

    @Override
    public void toFileStreamOutVO(File source, FileStreamOutVO target) {

        ...

        if (CommonUtil.getUseFileEncryption(source.getModule())) { //encrypted files are enabled for proband and trial files
            only
            try {
                //nest input streams:
                if (source.isExternalFile()) { //file content is stored in a physical file:
                    target.setStream(new VerifyMD5InputStream(CryptoUtil.createDecryptionStream(source.getDataIv(), new
                        FileInputStream(CoreUtil.prependDirectory(source.getExternalFileName()))), source.getMd5()));
                } else if (source.getData() != null) { //file content is stored in a byte array table column:
                    target.setStream(new VerifyMD5InputStream(CryptoUtil.createDecryptionStream(source.getDataIv(), new
                        ByteArrayInputStream(source.getData())), source.getMd5()));
                }
                //decrypt reamining fields exposed by FileStreamOutVO:
                target.setFileName((String) CryptoUtil.decrypt(source.getFileNameIv(), source.getEncryptedFileName()));
                target.setDecrypted(true);
            } catch (FileNotFoundException e) { //bypass file access error types as unchecked exception ...
                throw new RuntimeException(e);
            } catch (IOException e) {
                throw new RuntimeException(e);
            } catch (Exception e) { //... but swallow remaining problems, caused by decryption:
                //reset all decrypted VO fields:
                if (target.getStream() != null) {
                    try { target.getStream().close(); } catch (IOException e1) {}
                }
                target.setStream(null);
                target.setFileName(null);
                target.setDecrypted(false); //at least one decryption failed
            }
        } else { //remaining modules do not use encryption
            //...
        }
    }

    ...
```

Listing 4.39: The `FileStreamOutVO` transformation method sets up the decryption stream for retrieving a `File`.

Support for digital signatures Digital signatures for entity graphs presented in section 4.3.3 allow to prove integrity of relevant database records. While file content persisted in byte array table columns is covered by the entity graph traversal when calculating a signature value, files stored in the file system have to be considered by including their file checksum. For a large and/or numerous included files, creating a signature value will definitely take too long if file checksums have to be calculated. Instead, a checksum is calculated and persisted with the `File` record when a file is uploaded. The entity graph traversal will just include the existing `File.md5` checksum field value. If the `md5` value was adjusted to match the content of a manipulated file, an existing signature would fail to verify. Of course this is useless, unless the physical file is bound by some checksum comparison. This is implemented by verifying the stored checksum value whenever a file is downloaded. When streaming a file, the DAO transformation method (listing 4.39) employs `VerifyMD5Stream` (a `FilterInputStream` *decorator*) to re-calculate the checksum value. It compares the stored checksum value with the recent checksum that is available after the last stream chunk was read. If they differ, a `java.io.IOException` is generated and wrapped into the unchecked `java.lang.RuntimeException` type. This unexpected exception propagates and causes the file download to abort, thus compromised files are never successfully served to users.

4.7.2. File Browsing

The directory structure and contained files of a repository is visualized with a PrimeFaces `p:tree` or `p:treeTable` UI component. Repositories of most root entity types are likely to contain only tens of files, whereas documents of a trial can comprise thousands of files. The UI has to be capable of displaying this magnitude of files and folders with acceptable performance. This is accomplished by implementing lazy-loading using `p:tree`'s `expand` AJAX event. Whenever a folder tree node is expanded by the user, the node's first-level child elements are retrieved from the database in order to update the UI via PPR. In contrast to `p:tree`, the appearance of `p:treeTable` is closer to familiar OS file browsers and shows additionally details including file size, record modification timestamp, user, approval state and a summary bar with totals. Since the `p:treeTable` implementation of the PrimeFaces version used lacks support for lazy-loading, the file manager UI can be configured to use either `p:tree` or `p:treeTable` per system module, depending on the expected number of files in repositories.

Folder structure and preset folders To simplify usage, a design decision was taken in favour of avoiding explicit operations for directory manipulation. The user is not required to create folders and sub-folders prior to uploading files into them. When persisting an uploaded file, a file path instantly specifies the desired location. Folders are derived from the entered file path and a directory structure emerges. As a consequence, empty folders are not possible however. To compensate, a preset folder tree can be provided for each system module (figure 4.95). As a result, the preset folder tree will overlap directory structures originating from uploaded files. The tree merge operation behind is based on database queries, implemented by the `File.findFileFolders` finder method.

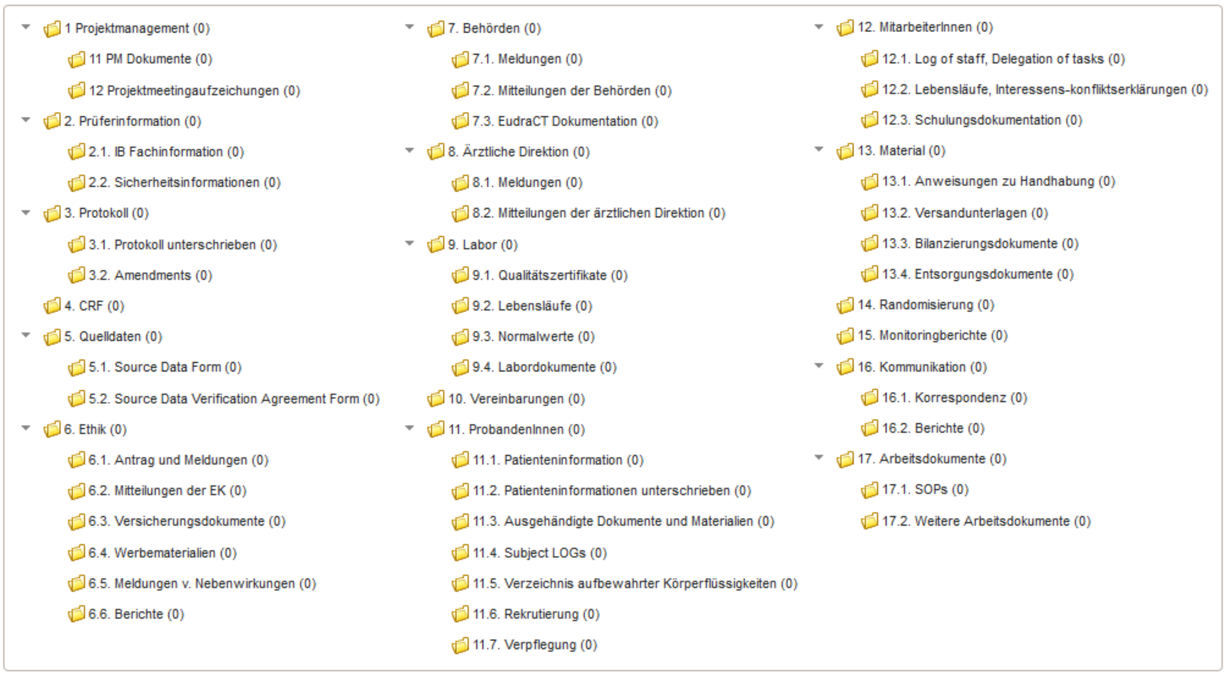

- 1 Projektmanagement (0)
  - 11 PM Dokumente (0)
  - 12 Projektmeetingaufzeichungen (0)
- 2. Prüferinformation (0)
  - 2.1. IB Fachinformation (0)
  - 2.2. Sicherheitsinformationen (0)
- 3. Protokoll (0)
  - 3.1. Protokoll unterschrieben (0)
  - 3.2. Amendments (0)
- 4. CRF (0)
- 5. Quelldaten (0)
  - 5.1. Source Data Form (0)
  - 5.2. Source Data Verification Agreement Form (0)
- 6. Ethik (0)
  - 6.1. Antrag und Meldungen (0)
  - 6.2. Mitteilungen der EK (0)
  - 6.3. Versicherungsdokumente (0)
  - 6.4. Werbematerialien (0)
  - 6.5. Meldungen v. Nebenwirkungen (0)
  - 6.6. Berichte (0)
- 7. Behörden (0)
  - 7.1. Meldungen (0)
  - 7.2. Mitteilungen der Behörden (0)
  - 7.3. EudraCT Dokumentation (0)
- 8. Ärztliche Direktion (0)
  - 8.1. Meldungen (0)
  - 8.2. Mitteilungen der ärztlichen Direktion (0)
- 9. Labor (0)
  - 9.1. Qualitätszertifikate (0)
  - 9.2. Lebensläufe (0)
  - 9.3. Normalwerte (0)
  - 9.4. Labordokumente (0)
- 10. Vereinbarungen (0)
- 11. ProbandenInnen (0)
  - 11.1. Patienteninformation (0)
  - 11.2. Patienteninformationen unterschrieben (0)
  - 11.3. Ausgehändigte Dokumente und Materialien (0)
  - 11.4. Subject LOGs (0)
  - 11.5. Verzeichnis aufbewahrter Körperflüssigkeiten (0)
  - 11.6. Rekrutierung (0)
  - 11.7. Verpflegung (0)
- 12. MitarbeiterInnen (0)
  - 12.1. Log of staff, Delegation of tasks (0)
  - 12.2. Lebensläufe, Interessens-konfliktserklärungen (0)
  - 12.3. Schulungsdokumentation (0)
- 13. Material (0)
  - 13.1. Anweisungen zu Handhabung (0)
  - 13.2. Versandunterlagen (0)
  - 13.3. Bilanzierungsdokumente (0)
  - 13.4. Entsorgungsdokumente (0)
- 14. Randomisierung (0)
- 15. Monitoringberichte (0)
- 16. Kommunikation (0)
  - 16.1. Korrespondenz (0)
  - 16.2. Berichte (0)
- 17. Arbeitsdokumente (0)
  - 17.1. SOPs (0)
  - 17.2. Weitere Arbeitsdokumente (0)

Figure 4.95.: The file folder structure shown is the preset for a trial's file repository. The folder structure can be defined by `FileFolderPreset` master data records. Folder names and hierarchy was elaborated by CRC staff during the system rollout.

Filing documents into predefined folders is the appreciated use case in contrast to creating inconsistent folder structures among root entity items. As an aid when typing the file's path, the file manager UI provides a `p:autoComplete` input with suggestions to quickly catch a full path within the existing directory structure of a repository.

File download After selecting a `File` record from the directory tree of the file manager UI, the file can be transferred to the user's workstation. The file download is started by clicking a link and backed by a separate PrimeFaces servlet. For a stable memory consumption, file streams are passed across the application's layers. This enables to withstand a large number of simultaneous downloads and downloads of large files. The `Content-Disposition: attachment` HTTP header value ensures the file download is managed and monitored by the browser. Finally, the user can decide to save the file to the local file system with a file name suggested by the application or open[76] it directly.

File types To provide the file *content type*[77] required for the `Content-type` header value of a download HTTP response, it is stored by the `File` record when a file is uploaded. Since PrimeFaces's `p:fileUpload` component is bound to Apache Commons FileUpload, identification of the MIME-type of an file relies on the `Content-type` HTTP header value provided by the browser[76] when uploading.

Known MIME-types are registered during system setup by importing master data using the the CLI tool. This supports an optional restriction of file types accepted by the system. The file system tree can indicate the files' content type using icons by refererring to style classes defined in a separate CSS file (`mimetypes.css`). As a start, it was decided to prepare style classes with icon images for about 40 relevant MIME-types of more than 1000 registered overall. They include well-known file types to cover e.g. image files and document files of common office software suites. Encrypted files are depicted by the uniform 🔒 icon.

Search filter A file repository can be filtered by `File.fileName` or `fileTitle` to search for file items, in a way similar to filering by column values known from PSF-enabled datatables (section 4.2.4). Corresponding filter text input elements are processed per keypress event, hence search results are updated as-you-type. RDBMS *indexes*[78] for both columns accelerate finder method queries. This makes file searching fast compared to a physical file system, especially in the case of large file repositories of trials.

Document approval The `File.active` flag is used to support a default filtering of displayed files for users with restricted privileges. When invoking the `FileService.getFiles` and `getFileFolders` retrieval operations, argument instrumentation (section 4.5.2) will override the corresponding service method filter parameter for users with view-only permission profiles. This allows users with edit permission profiles for a module's root entity details to disable the `active` flag in order to hide (preliminary) documents from remaining users.

Aggregation of PDF files PDF files are expected to represent a major part of uploaded files, if enterprise document scanner devices will be used to support *scan-and-store* workflows. Field observation revealed a tendency of users to download all available files (e.g. application papers) of an entity (e.g. `Staff`) in a row, which requires opening and downloading files on-by-one. Therefore, a separate button was introduced to facilitate the retrieval of multiple files with `application/pdf` MIME-type at once. The aggreagted PDF file is created on-the-fly and contains all decryptable PDF document files that match optional search filters entered.

4.8. Electronic Data Capture (EDC)

Subject recruitment is a critical factor to the success of clinical trials (Reynolds, 2011). Determining a subject's eligibility can require to record a multitude of medical parameters, which may vary from trial

[76]Depending on the OS, the installation of software capable to handle the file's content type is required.
[77]also known as *file type* or *MIME-type*
[78]Indexed columns allow a RDBMS to choose a fast tree search over slow linear table scans when planning a query execution.

to trial. Extending a trial's CRF or reusing information from CRFs for a recruiment strategy aside the (commissioned) trial is usually restricted. Therefore, the Phoenix CTMS is designed to support capturing medical information for recruitment purposes, e.g. during the initial registration of a proband. In addition, individual subject parameters not covered by a CRF can be tracked for organisational purposes during an ongoing trial. From the implementation point of view, both capabilities can be compared to the EDC functionality seen in current CDMSs and should be as powerful.

4.8.1. Input Fields

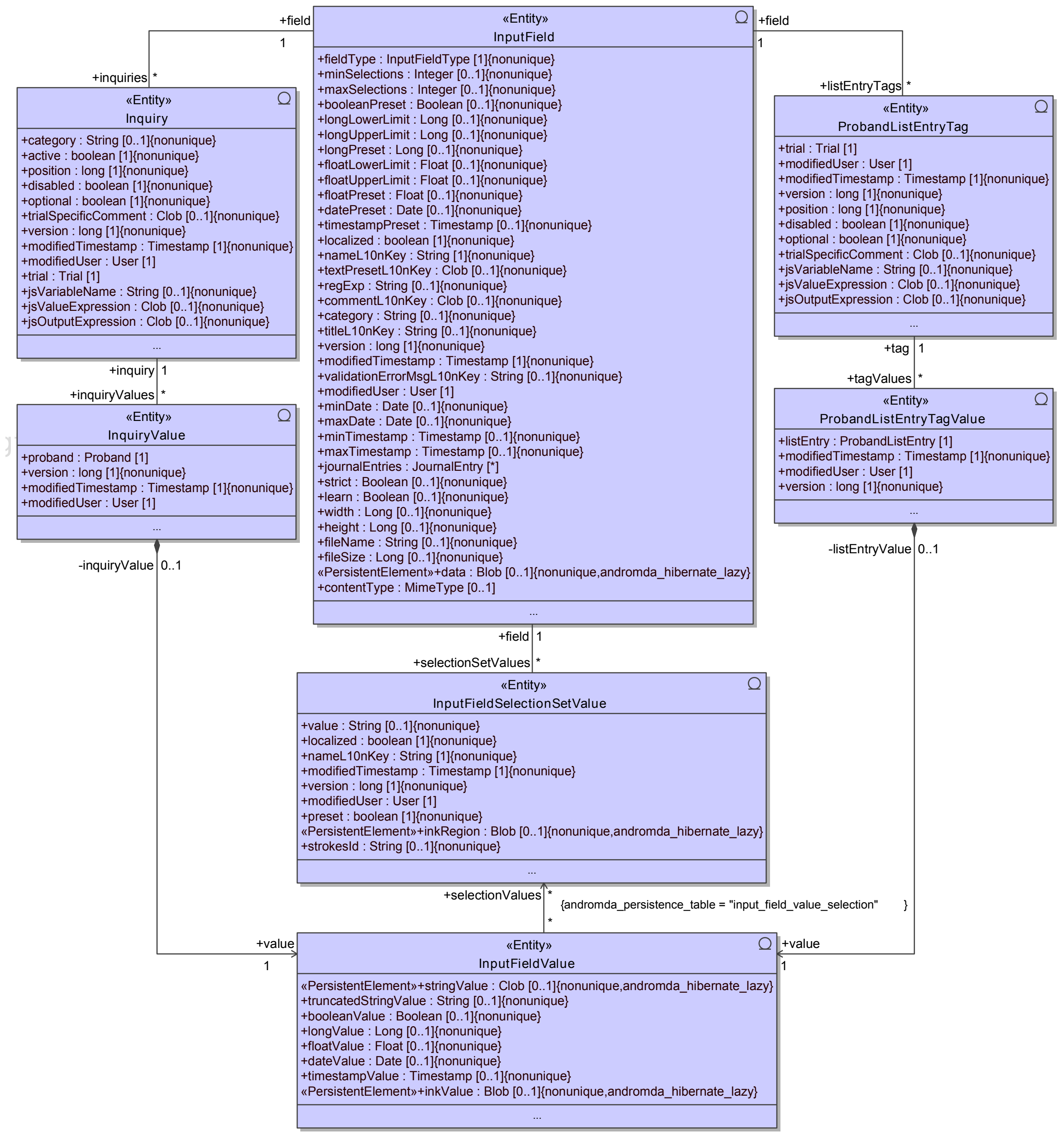

«Entity»
InputField
+fieldType : InputFieldType [1]{nonunique}
+minSelections : Integer [0..1]{nonunique}
+maxSelections : Integer [0..1]{nonunique}
+booleanPreset : Boolean [0..1]{nonunique}
+longLowerLimit : Long [0..1]{nonunique}
+longUpperLimit : Long [0..1]{nonunique}
+longPreset : Long [0..1]{nonunique}
+floatLowerLimit : Float [0..1]{nonunique}
+floatUpperLimit : Float [0..1]{nonunique}
+floatPreset : Float [0..1]{nonunique}
+datePreset : Date [0..1]{nonunique}
+timestampPreset : Timestamp [0..1]{nonunique}
+localized : boolean [1]{nonunique}
+nameL10nKey : String [1]{nonunique}
+textPresetL10nKey : Clob [0..1]{nonunique}
+regExp : String [0..1]{nonunique}
+commentL10nKey : Clob [0..1]{nonunique}
+category : String [0..1]{nonunique}
+titleL10nKey : String [0..1]{nonunique}
+version : long [1]{nonunique}
+modifiedTimestamp : Timestamp [1]{nonunique}
+validationErrorMsgL10nKey : String [0..1]{nonunique}
+modifiedUser : User [1]
+minDate : Date [0..1]{nonunique}
+maxDate : Date [0..1]{nonunique}
+minTimestamp : Timestamp [0..1]{nonunique}
+maxTimestamp : Timestamp [0..1]{nonunique}
+journalEntries : JournalEntry [*]
+strict : Boolean [0..1]{nonunique}
+learn : Boolean [0..1]{nonunique}
+width : Long [0..1]{nonunique}
+height : Long [0..1]{nonunique}
+fileName : String [0..1]{nonunique}
+fileSize : Long [0..1]{nonunique}
«PersistentElement»+data : Blob [0..1]{nonunique,andromda_hibernate_lazy}
+contentType : MimeType [0..1]
...
+field
1
+inquiries
*
«Entity»
Inquiry
+category : String [0..1]{nonunique}
+active : boolean [1]{nonunique}
+position : long [1]{nonunique}
+disabled : boolean [1]{nonunique}
+optional : boolean [1]{nonunique}
+trialSpecificComment : Clob [0..1]{nonunique}
+version : long [1]{nonunique}
+modifiedTimestamp : Timestamp [1]{nonunique}
+modifiedUser : User [1]
+trial : Trial [1]
+jsVariableName : String [0..1]{nonunique}
+jsValueExpression : Clob [0..1]{nonunique}
+jsOutputExpression : Clob [0..1]{nonunique}
...
+inquiry
1
+inquiryValues
*
«Entity»
InquiryValue
+proband : Proband [1]
+version : long [1]{nonunique}
+modifiedTimestamp : Timestamp [1]{nonunique}
+modifiedUser : User [1]
...
-inquiryValue
0..1
+field
1
+listEntryTags
*
«Entity»
ProbandListEntryTag
+trial : Trial [1]
+modifiedUser : User [1]
+modifiedTimestamp : Timestamp [1]{nonunique}
+version : long [1]{nonunique}
+position : long [1]{nonunique}
+disabled : boolean [1]{nonunique}
+optional : boolean [1]{nonunique}
+trialSpecificComment : Clob [0..1]{nonunique}
+jsVariableName : String [0..1]{nonunique}
+jsValueExpression : Clob [0..1]{nonunique}
+jsOutputExpression : Clob [0..1]{nonunique}
...
+tag
1
+tagValues
*
«Entity»
ProbandListEntryTagValue
+listEntry : ProbandListEntry [1]
+modifiedTimestamp : Timestamp [1]{nonunique}
+modifiedUser : User [1]
+version : long [1]{nonunique}
...
-listEntryValue
0..1
+field
1
+selectionSetValues
*
«Entity»
InputFieldSelectionSetValue
+value : String [0..1]{nonunique}
+localized : boolean [1]{nonunique}
+nameL10nKey : String [1]{nonunique}
+modifiedTimestamp : Timestamp [1]{nonunique}
+version : long [1]{nonunique}
+modifiedUser : User [1]
+preset : boolean [1]{nonunique}
«PersistentElement»+inkRegion : Blob [0..1]{nonunique,andromda_hibernate_lazy}
+strokesId : String [0..1]{nonunique}
...
+selectionValues
*
{andromda_persistence_table = "input_field_value_selection" }
*
+value
1
+value
1
«Entity»
InputFieldValue
«PersistentElement»+stringValue : Clob [0..1]{nonunique,andromda_hibernate_lazy}
+truncatedStringValue : String [0..1]{nonunique}
+booleanValue : Boolean [0..1]{nonunique}
+longValue : Long [0..1]{nonunique}
+floatValue : Float [0..1]{nonunique}
+dateValue : Date [0..1]{nonunique}
+timestampValue : Timestamp [0..1]{nonunique}
«PersistentElement»+inkValue : Blob [0..1]{nonunique,andromda_hibernate_lazy}
...

Figure 4.96.: This detailed view of the EAV data models from figure 4.16 puts the `InputField` root entity into focus.

A proband's clinical parameters or answers to inquiry questions are organized by means of an EAV data model (section 4.1.1.3). Parameter types (attributes) are represented by the `InputField` root entity (figure 4.96), which can be managed by the system's input field module (figure 4.80). The `InputField` entity fields are designed to hold definitions of multiple types of UI input elements. Aside name, title (prompt) and a category[79] to organize input fields, a preset value of appropriate data type can be defined. To support user-defined input validation such as range checks of entered values, entity fields for range limits and a validation error message are required. Some types of input elements allow to select items rather than entering a literal value, which implies to define a set of option values, mapped by the `InputFieldSelectionSetValue` detail entity. The `InputFieldType` enumeration type lists eleven input field types, which are currently supported:

Single-line text Aside short free-text values, alphanumeric tokens such as subject IDs according to a scheme given by the sponsor qualify for using a single-line text input field. To enforce a certain string format, a RegEx can be specified for input validation. The entered value is persisted by the `InputFieldValue.stringValue` field.

Multi-line text Longer answers or comments can be entered by a multi-line text input. Since values are not encrypted, the icon reminds users to omit entering a subject's PII. The entered value is persisted by the `InputFieldValue.stringValue` field. Since `stringValue` is a ≪CLOB≫ entity field allowing to store potentially long strings[80], a truncated copy of the text's first line is separately saved in `InputFieldValue.truncatedStringValue`. In general, this is supplied for any string input type (single-line text, multi-line text, autocomplete text) in order to populate text suggestions with acceptable performance. The suggestions are a helpful feature for the query editor when creating a query to search for data originating from these free-text input types.

Single selection This input is presented by a `p:selectOneMenu` dropdown. A single item can be chosen from a set of prepared `InputFieldSelectionSetValue` items (figure 4.111). A dedicated `InputFieldSelectionSetValue` item can be marked as preset value. The item selection is persisted by the `InputFieldValue.selectionSetValues` association.

Autocomplete text The popular `p:autoComplete` input is supported as advanced alternative to either the single-line text or the single selection input elements. Suggestions backed by the input field's `InputFieldSelectionSetValues` are displayed as-you-type. An item must be selected from these suggestions in order to save the value if the `InputField.strict` flag is set. While this corresponds to the single selection input type, setting the `strict` flag to `false` allows users to save an arbitrary string, which corresponds to the behaviour of the single-line text input type. In the latter case, the `learn` flag can be set in order to automatically append yet unknown items to suggestions.

Multiple selection In contrast to the dropdown input element of the single selection input field type, a multiple selection is presented by a group of checkboxes (`p:selectManyCheckbox`). Input validation covers a minimum and maximum limit of checked items. Multiple dedicated `InputFieldSelectionSetValue` items can be marked as preset value. Selected items are persisted by the `InputFieldValue.selectionSetValues` association.

[79] The `category` field can be used to arrange input field in "libraries".

[80] By default, conventional entity fields of UML ≪string≫ type are mapped to table columns of a string data type with a fixed size limit of 1024 characters supported by the RDBMS. ≪Character Large Object (CLOB)≫ is mapped to a TEXT table column supporting large strings.

Yes/no A traditional stand-alone checkbox is rendered by PrimeFace's `p:selectOneCheckbox` input component. The checkbox value is persisted by the `InputFieldValue.booleanValue` field. The preset value can be set to either `false` or `true`. To create an input field for ternary values, the single selection input type is supposed to be used instead.

Integer Integer numbers can be entered and adjusted by up- and down-buttons of the `p:spinner` control. The entered value is persisted by the `InputFieldValue.longValue` field. The `inputValidationErrorMessage` shows up, if specified range limits are exceeded. Similar to any other input type that supports input validation logic, the error message text is supposed to contain a hint about the range limits can be specified by users themselves. As always, the input validation is performed server-side, e.g. by the corresponding service method for saving an entered value. The validation message is additionally appended to the input field's tooltip text.

Decimal Decimal numbers can be entered by a standard `p:textInput`. The default JSF converter in use accepts string input values which can be processed by `Float.parseFloat`. The entered value is persisted by the `InputFieldValue.floatValue` entity field. Its ≪Float≫ data type could be extended to ≪Double≫ or even ≪BigDecimal≫ in a future version. A capable Js data type is required for any input field type however.

Date A date selector is provided by `p:calendar` for date entries. Throughout the system, dates can alternatively be entered directly in a string date format. The format pattern can be configured for example to match the ISO 8601 format. The date value is persisted by the corresponding `InputFieldValue.dateValue` entity field. A typical use case would be capturing the *IC date*. A lower date value limit makes sense in this case to avoid typos.

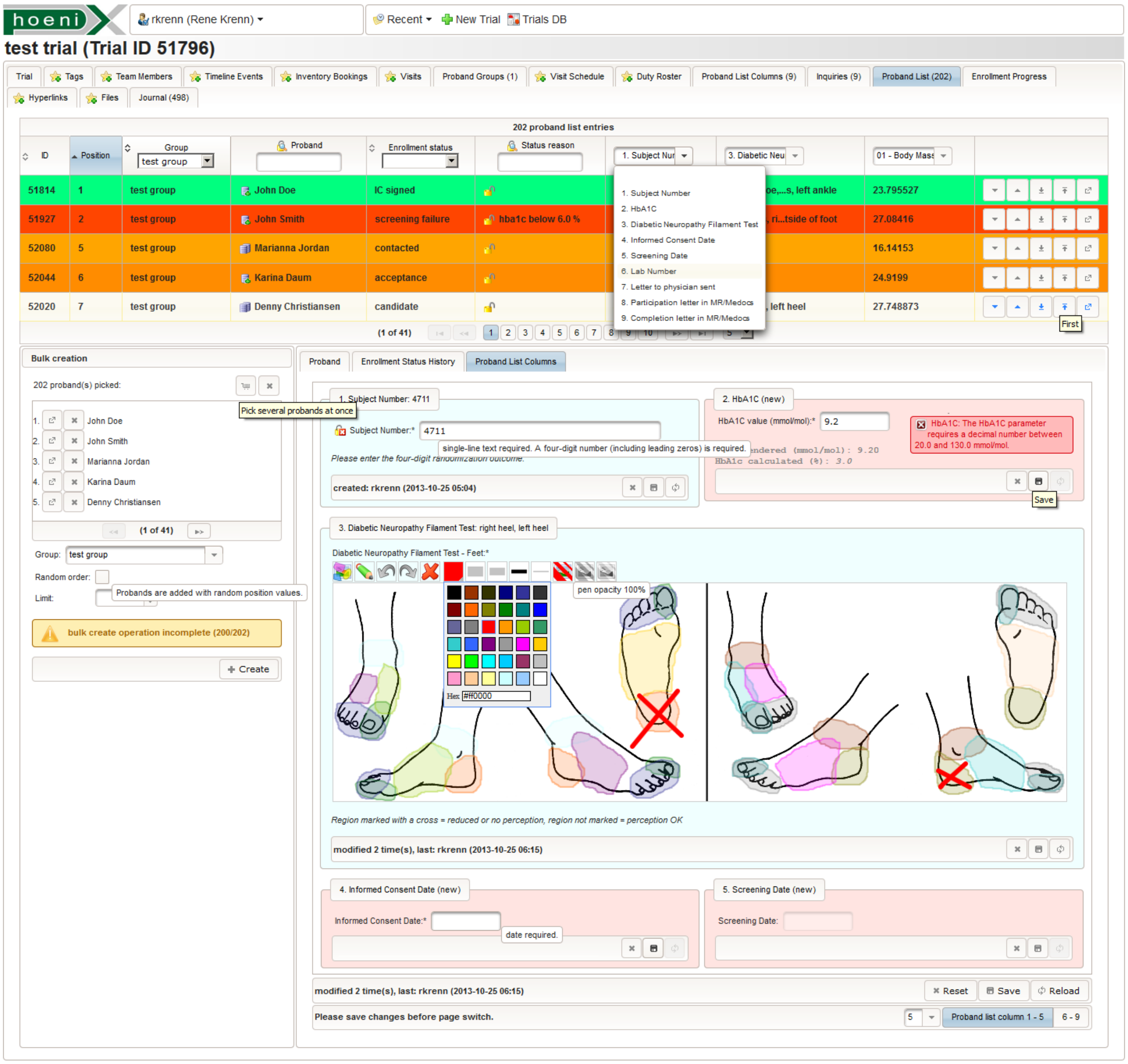
Datetime The datetime input extends the date input type and allows to specify a timestamp by selecting date *and* time. `p:calendar` displays slider controls to adjust the time of day part. The datetime value is persisted by the corresponding `InputFieldValue.timestampValue` entity field.

Sketch The sketch input type uses a custom JSF component to display and create sketches. The `InputFieldValue.inkValue` entity field is used to store a sketch in SVG format. Since sketches would be difficult to process by means of SQL queries, the sketch input type additionally integrates a multiple selection input logic. Therefore, each prepared `InputFieldSelectionSetValue` item defines a selectable region. The sketch input is outlined in detail in section 4.8.4.

CRUD service methods for `InputField` and `InputFieldSelectionSetValue` entities are encapsulated by the `InputFieldService`. Delete operations such as removing a `InputFieldSelectionSetValue` or the entire input field are fully supported but may alter captured data and cause the creation of a large number of journal entries, if lots of entered value exists already. An input field's type cannot be altered once it is created, although it would be possible for some constellations[81] The input field module's entity view provides a tab to preview and test data entry for a created input field before using it (figure 4.112). Another tab shows datatables to overview the usage of the particular input field in trials including totals of entered values.

[81]For example, changing a single-line text input to a multi-line text input will not cause a loss of information.

4.8.2. Form Composition



test trial (Trial ID 51796)

202 proband list entries

ID	Position	Group	Proband	Enrollment status	Status reason	1. Subject Nur	3. Diabetic Neu	01 - Body Mass
		test group						
51814	1	test group	John Doe	IC signed			oe,...s, left ankle	23.795527
51927	2	test group	John Smith	screening failure	hba1c below 6.0 %		ri...tside of foot	27.08416
52080	5	test group	Marianna Jordan	contacted				16.14153
52044	6	test group	Karina Daum	acceptance				24.9199
52020	7	test group	Denny Christiansen	candidate			, left heel	27.748873

(1 of 41) 1 2 3 4 5 6 7 8 9 10 5

1. Subject Number
2. HbA1C
3. Diabetic Neuropathy Filament Test
4. Informed Consent Date
5. Screening Date
6. Lab Number
7. Letter to physician sent
8. Participation letter in MR/Medocs
9. Completion letter in MR/Medocs

202 proband(s) picked:

Pick several probands at once

1. John Doe
2. John Smith
3. Marianna Jordan
4. Karina Daum
5. Denny Christiansen

(1 of 41)

Group: test group

Random order:

Limit:

Probands are added with random position values.

bulk create operation incomplete (200/202)

1. Subject Number: 4711

Subject Number:* 4711

single-line text required. A four-digit number (including leading zeros) is required.

Please enter the four-digit randomization outcome.

created: rkrenn (2013-10-25 05:04)

2. HbA1C (new)

HbA1C value (mmol/mol):* 9.2

HbA1C: The HbA1C parameter requires a decimal number between 20.0 and 130.0 mmol/mol.

```
...ndered (mmol/mol): 9.20
HbA1c calculated (%): 3.0
```

3. Diabetic Neuropathy Filament Test: right heel, left heel

Diabetic Neuropathy Filament Test - Feet:*

Region marked with a cross = reduced or no perception, region not marked = perception OK

modified 2 time(s), last: rkrenn (2013-10-25 06:15)

4. Informed Consent Date (new)

Informed Consent Date:*

date required.

5. Screening Date (new)

Screening Date:

modified 2 time(s), last: rkrenn (2013-10-25 06:15)

Please save changes before page switch. 5 Proband list column 1 - 5 6 - 9

Figure 4.97.: The proband list is the primary screen for managing a trial's participants. The trial's eligibility criteria can be expressed by a proband query, whose listed query results can be picked page-wise using the multi-picker mechanism. Selected participants are finally added to the proband list by transactional bulk creation, which optionally limits and shuffles[82]created proband list entries (lower left area of the screen). Each proband can be assigned to a treatment group ("Proband" tab of the lower right area). When conducting the trial, a participant's lifecycle is described by its enrollment state ("Enrollment Status History" tab of the lower right area, figure 4.43). The active tab of the lower right area shows the input form for data entry of user-defined proband list columns, which can be defined in the screen shown in figure 4.98. Presented input fields with no value saved yet are displayed with rose background color, while fields with an existing value have a cyan background. The datatable in the upper area represents the actual list of participants. Users may choose any two of availale proband list columns to view entered values at a glance. The third column allows displaying entered values of an inquiry form field respectively.

[82]The set of picked probands is shuffled using `Colllections.shuffle`. For reproducible shuffling, the `java.util.Random` Pseudo-Random Number Generator (PRNG) is initialized with a seed value from a true entropy source (`java.security.SecureRandom`). The seed value as well as input and output sets are logged in a journal entry record.

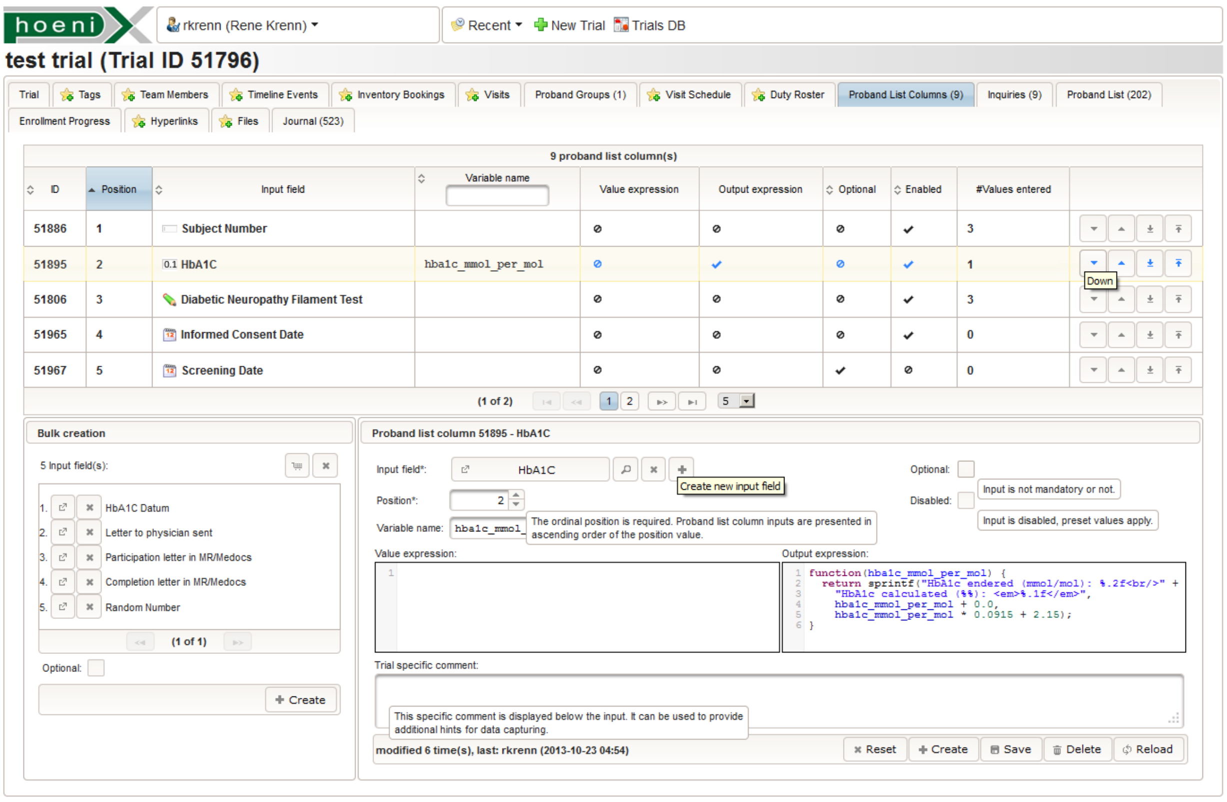
The usage of a prepared input field comes down to two scenarios. Individual input forms for both *inquiry questions* (figure 4.106) and *proband list columns* (figure 4.97) can be composed for each trial by arranging input fields.

Inquiry form During recruitment, baseline information is gathered for a coarse identification of eligible candidates (pre-screening). Qualified candidates are enrolled by adding them to the trial's proband list in the next step. Since questions to candidates depend on a trial's inclusion/exclusion criteria, a questionnaire can be composed by arranging individual inquiry questions (figure 4.105). The definition of a inquiry question is mapped by the `Inquiry` entity. The composed input form may show a large number of `Inquiry` items, which requires a lazy-loading technique. The inquiry input form can be previewed and tested. When the trial state changes to "ongoing", the prepared input form is released for data entry and can be selected from a list of available trial inquiry forms in the proband view's "Inquiry Forms" tab (figure 4.106).

Proband list columns In the past, users maintained check list spreadsheets to track and document the completion of specific tasks per participant. While this could be mapped using Boolean variables, additional participant attributes of various data types were identified, e.g.

Subject-related identifiers: subject ID, randomisation outcome token, laboratory number, ...
Documentary dates: informed consent date, screening date, ...
Medical information: Supporting medical information regarding pre-screening and preparation of trial visits or relevant information that might not be covered by the CRF such as condition of veins, fasting state, alcohol level or other recent readings, ...

To avoid future shortcomings, it was decided to utilize input fields once again for these use cases. An additional input form allows entering values per proband list entry. Captured values of an attribute can be surveyed by displaying them in string format in data cells of a particular column of the proband list datatable. The set up of a proband list column is mapped by the `ProbandListEntryTag` entity. Similar to inquiry questions, a trial's proband list columns can be managed using a separate UI screen (figure 4.98).



test trial (Trial ID 51796)

9 proband list column(s)

ID	Position	Input field	Variable name	Value expression	Output expression	Optional	Enabled	#Values entered
51886	1	Subject Number		⊘	⊘	⊘	✓	3
51895	2	0.1 HbA1C	hba1c_mmol_per_mol	⊘	✓	⊘	✓	1
51806	3	Diabetic Neuropathy Filament Test		⊘	⊘	⊘	✓	3
51965	4	Informed Consent Date		⊘	⊘	⊘	✓	0
51967	5	Screening Date		⊘	⊘	✓	⊘	0

(1 of 2) 1 2 5

Bulk creation

5 Input field(s):

1. HbA1C Datum
2. Letter to physician sent
3. Participation letter in MR/Medocs
4. Completion letter in MR/Medocs
5. Random Number

(1 of 1)

Optional:

Create

Proband list column 51895 - HbA1C

Input field*:	HbA1C
Position*:	2
Variable name:	hba1c_mmol_
Optional:	
Disabled:	

The ordinal position is required. Proband list column inputs are presented in ascending order of the position value.

Input is not mandatory or not.

Input is disabled, preset values apply.

Value expression:

Output expression:

```
function(hba1c_mmol_per_mol) {
  return sprintf("HbA1c endered (mmol/mol): %.2f<br/>" +
    "HbA1c calculated (%%): <em>%.1f</em>",
    hba1c_mmol_per_mol + 0.0,
    hba1c_mmol_per_mol * 0.0915 + 2.15);
}
```

Trial specific comment:

This specific comment is displayed below the input. It can be used to provide additional hints for data capturing.

modified 6 time(s), last: rkrenn (2013-10-23 04:54)

Figure 4.98.: The trial entity view's tab shown allows defining proband list columns. A proband list column represents an attribute of proband list entries which are managed using the proband list (figure 4.97). A proband list column is defined by selecting an input field and specifying options ("Optional", "Disabled"). The order of the input form for proband list column values is specified by a unique position value per item. A trial specific comment can be used for a unique identification, in the case the same input field is used for multiple columns. The example shown uses form scripting (listing 4.41) to display the measurement unit conversion of an entered HbA_{1c} reading field.

An input form presents input fields of corresponding Inquiry or ProbandListEntryTag items for data entry in a two-column layout.[83] Inquiry and ProbandListEntryTag entities from figure 4.96 extend their associated InputField with settings for detailed control:

Input field: An input field is picked using the input field picker pop-up. The same input field field can be used for multiple Inquiry/ProbandListEntryTag items building an input form. The input field cannot be exchanged, once entered values have been saved.

Category: Input fields of inquiry input forms can be grouped by a collapsable p:fieldSet frame (figure 4.106). This *form section* groups Inquiry items with the same category label. When setting up inquiry questions, existing Inquiry.category labels are suggested by a p:autoComplete input (figure 4.105). Form sections are not implemented for proband list columns at the moment.

Position: The position integer value defines the order of input fields. Therefore, a unique position value is required among the trial's ProbandListEntryTags. For Inquiry items, uniqueness is enforced within the category section only. Datatables displaying Inquiry and ProbandListEntryTag entities provide buttons to swap items. A future version could implement drag-and-drop for this purpose.

Optional: This flag indicates if the data entry for the input field is mandatory or optional. For example, an error is displayed when trying to save a mandatory input field of multiple selection type with no

[83] Sketch input fields may span both columns, if their width exceeds a limit.

option item selected, irrespective of limits of checked items specified for the input field. The optional flag has no effect for input fields of checkbox type.

Disabled: An input field can be disabled to prohibit manual data entry. The input value persisted is constrained to the unaltered preset value.

Display: Inquiry input fields can be (temporarily) hidden from the input form. In contrast to a disabled input field, a hidden input field's value will not persisted either when saving the whole input form page.

Trial specific comment: If an input field (e.g. "dose") is used multiple times in the same input form, they can be distinguished by this (multi-line) comment string (e.g. "morning", "midday", "evenings"). The `trialSpecificComment` is displayed in input forms below the (optional) input field's comment. The trial specific comment can also be used to give specific hints for data entry in the context of a particular trial.

The remaining `Inquiry.js*` and `ProbandListEntryTag.js*` entity fields are required for the Js-based form scripting feature described in the next section.

To support forms with a large number of input fields, a custom pagination implementation allows switching between lazy-loaded pages. Since saving every single input field value is tedious when entering data into a blank form initially, data entry of a complete page can be stored by transactional bulk operations. Users can update particular values from that point on. Instead of explicit create and update service methods, *save operations* manage to create or update corresponding `InquiryValue` (listing 4.42) or `ProbandListEntryTagValue` (listing 4.43) instances including the actual `InputFieldValue` records in order to persist a given number of entered form field values. This is examined in detail in section 4.8.3.2.

The current implementation of input forms is lightweight but suffices EDC processes as required for recruitment and subject management purposes. Simple input field types do not support neither longitudinal nor aggregated data but will keep dynamic database query expressions manageable with the built-in query editor. The Phoenix CTMS lacks EDC feature details below, which may be available in related software systems (section 2.2.3):

Branching logic: The branching logic feature allows hiding subsequent input fields by evaluating user-definable expressions, which depend on values entered. For example, specific detail questions may appear according to the answer to former "base" questions. This is extensively supported for example by recent versions of REDCap and related systems or services for electronic surveys in general, such as SurveyMonkey (SurveyMonkey, 2013). The Phoenix CTMS supports a related configuration option to collapse form sections and individual input fields by default, if values have already been entered or not. This was however rejected by users.

Query-based preset values: The ArchiMed system supports dynamic preset values, populated by database queries (Duftschmid and Wrba, 2004), for example against data already entered for probands before. The `InputField.learn` mode for autocomplete input fields is a related, but limited option supported in the Phoenix CTMS.

Input form layout: The implemented tabular input form layout is less sophisticated than EDC form screens of systems with full-featured form designers like ArchiMed, which allow absolute positioning of input fields on a blank form canvas.

Value aggregates: To support entering longitudinal data, some EDC or CDMS systems provide input field types for single- or multi-dimensional arrays of a data type or heterogeneous arrays. These complex data structures could be implemented in the Phoenix CTMS by introducing a tree hierarchy of input fields.

Alternative input field types: Existing systems may provide a wider variety of input field types for the same purpose, e.g. a single selection could alternatively be presented by multiple checkboxes instead of a dropdown only.

4.8.3. Calculated Field Values

Selected parameters relevant to eligibility criteria (such as the BMI value) or CRFs are derived from basic parameters which can be actually measured (such as body weight and height). To save time and avoid errors when capturing data, it is desired to automate calculations. User-defineable computation of dependent form field values is a specific form scripting feature common to sophisticated EDC software systems. Since derived parameters are definitely required by real-life inclusion/exclusion criteria, the Phoenix CTMS input forms were enhanced to support calculated field values as well.

4.8.3.1. Variable and Computation Model

An input form consisting of n input fields provides a set of entered values $E = \{e_i\} = \{e_1, e_2, e_3, \ldots\}$. A separate *symbol table* $V = \{v_i\}$ is introduced, containing a variable v_i for each input field ($|E| = |V| = n$). When computing field values, each variable's value is populated by either the entered value e_i or the result of a function f_i, which depends on a set of parameters $P_i \subset V$:

$$v_i = \begin{cases} e_i \\ f_i(P_i) \end{cases}$$

Therefore, an input form's field value computation is effectively defined by the set of functions $F = \{f_i\}$ ($|F| < n$). Variables $v_r \notin \bigcup_{P_i}$ represent root nodes R of *n-ary expression trees* (figure 4.99).

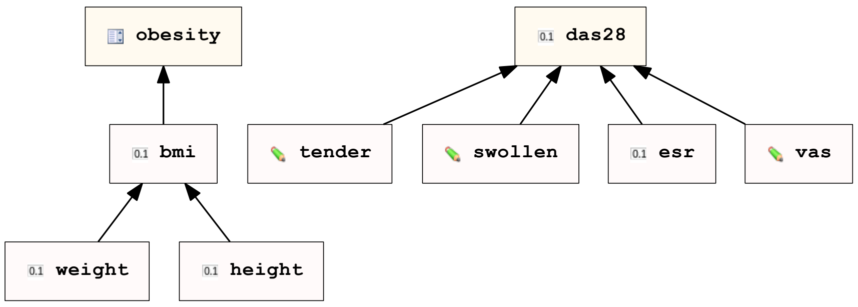


Figure 4.99.: An input form's fields represent expression tree nodes. Values of input fields with children (`bmi`, `obesity`, `das28`) depend on leaf nodes and can be calculated automatically. While leaf nodes correspond to values entered directly, users can finally accept a calculated variable's value by copying it using a dedicated button.

$V := \{obesity, bmi, weight, height, das28, tender, swollen, esr, vas\}$

$$
\begin{aligned}
v_1 &: obesity = f_1(bmi)\\
v_2 &: bmi = f_2(weight, height)\\
v_3 &: weight = e_3\\
v_4 &: height = e_4\\
v_5 &: das28 = f_3(tender, swollen, esr, vas)\\
v_6 &: tender = e_6\\
v_7 &: swollen = e_7\\
v_8 &: esr = e_8\\
v_9 &: vas = e_9
\end{aligned}
$$

$F := \{f_1, f_2, f_5\}$

$f_1 : \mathbb{R} \rightarrow \{\texttt{shortweight}, \texttt{normal weight}, \texttt{overweight}, \texttt{adiposity degree I}, \texttt{adiposity degree II}, \texttt{adiposity degree III}\} : bmi \mapsto obesity$

$f_2 : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R} : weight, height \mapsto bmi$

$f_5 : \mathbb{N} \times \mathbb{N} \times \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R} : tender, swollen, esr, vas \mapsto das28$

$$R := V \setminus \bigcup_{P_i} = \{obesity, das28\}$$

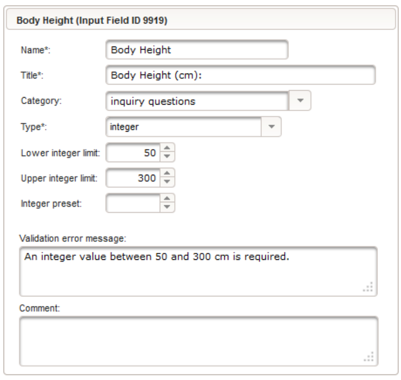
$$
\begin{aligned}
P_1 &= \{bmi\}\\
P_2 &= \{weight, height\}\\
P_5 &= \{tender, swollen, esr, vas\}
\end{aligned}
$$

While expression trees can be disjoint or share leaf nodes, cycles must be forbidden for a successful computation. To express and evaluate a function f_i, JavaScript was chosen over developing a custom DSL. Expressions can be entered directly when composing the input form (figure 4.105) using the Js notation for creating anonymous functions, which is shown by example in listing 4.40 for the Disease Activity Score (DAS)28 (Prevoo et al., 1995):

$$f_5 : das28 = 0.56\sqrt[2]{tender} + 0.28\sqrt[2]{swollen} + 0.70\ln(esr) + 0.014vas$$

```
function (tender, swollen, esr, vas) { //DAS (28 joints) computation:
  return 0.56 * Math.sqrt(tender.ids.length) + //tender joint count (0-28)
    0.28 * Math.sqrt(swollen.ids.length) + //swollen joint count (0-28)
    0.7 * Math.log(esr) + //erythrocyte sedimentation rate reading (5-30 mm/h)
    0.014 * getInputFieldSelectionSetValue("vas",vas.ids[0]).value; //the proband's subjective assessment of recent disease
          activity (visual analog scale, 0-100 mm)
}
```

Listing 4.40: This value expression defines the function f_5 of the expression tree example. It is used by the input form example shown in figure 4.115.



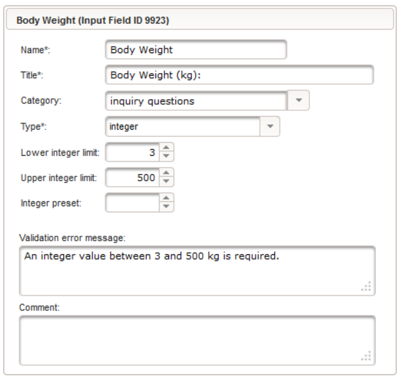
Body Height (Input Field ID 9919)

Name*:	Body Height
Title*:	Body Height (cm):
Category:	inquiry questions
Type*:	integer
Lower integer limit:	50
Upper integer limit:	300
Integer preset:	

Validation error message:
An integer value between 50 and 300 cm is required.

Comment:

Figure 4.100.: The setup of a "Body Height" input field.



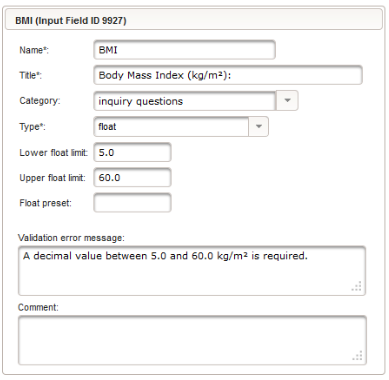
Body Weight (Input Field ID 9923)

Name*:	Body Weight
Title*:	Body Weight (kg):
Category:	inquiry questions
Type*:	integer
Lower integer limit:	3
Upper integer limit:	500
Integer preset:	

Validation error message:
An integer value between 3 and 500 kg is required.

Comment:

Figure 4.101.: The setup of a "Body Weight" input field.



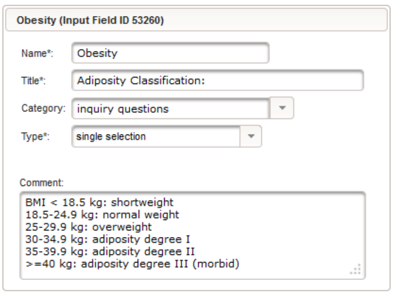
BMI (Input Field ID 9927)

Name*:	BMI
Title*:	Body Mass Index (kg/m²):
Category:	inquiry questions
Type*:	float
Lower float limit:	5.0
Upper float limit:	60.0
Float preset:	

Validation error message:
A decimal value between 5.0 and 60.0 kg/m² is required.

Comment:

Figure 4.102.: The input field setup of a decimal number entry for the BMI value.



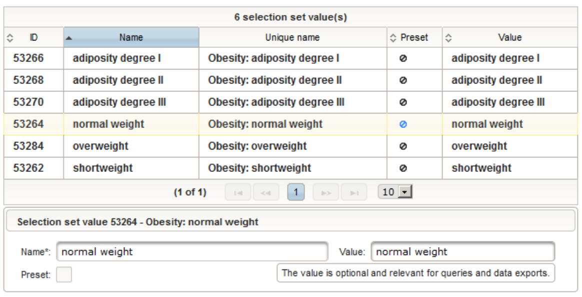
Obesity (Input Field ID 53260)

Name*:	Obesity
Title*:	Adiposity Classification:
Category:	inquiry questions
Type*:	single selection

Comment:

```
BMI < 18.5 kg: shortweight
18.5-24.9 kg: normal weight
25-29.9 kg: overweight
30-34.9 kg: adiposity degree I
35-39.9 kg: adiposity degree II
>=40 kg: adiposity degree III (morbid)
```

Figure 4.103.: The single selection type input is suitable for obesity classification data entry. Option items can be managed by the separate UI shwon in figure 4.104.



6 selection set value(s)

ID	Name	Unique name	Preset	Value
53266	adiposity degree I	Obesity: adiposity degree I	⊘	adiposity degree I
53268	adiposity degree II	Obesity: adiposity degree II	⊘	adiposity degree II
53270	adiposity degree III	Obesity: adiposity degree III	⊘	adiposity degree III
53264	normal weight	Obesity: normal weight	⊘	normal weight
53284	overweight	Obesity: overweight	⊘	overweight
53262	shortweight	Obesity: shortweight	⊘	shortweight

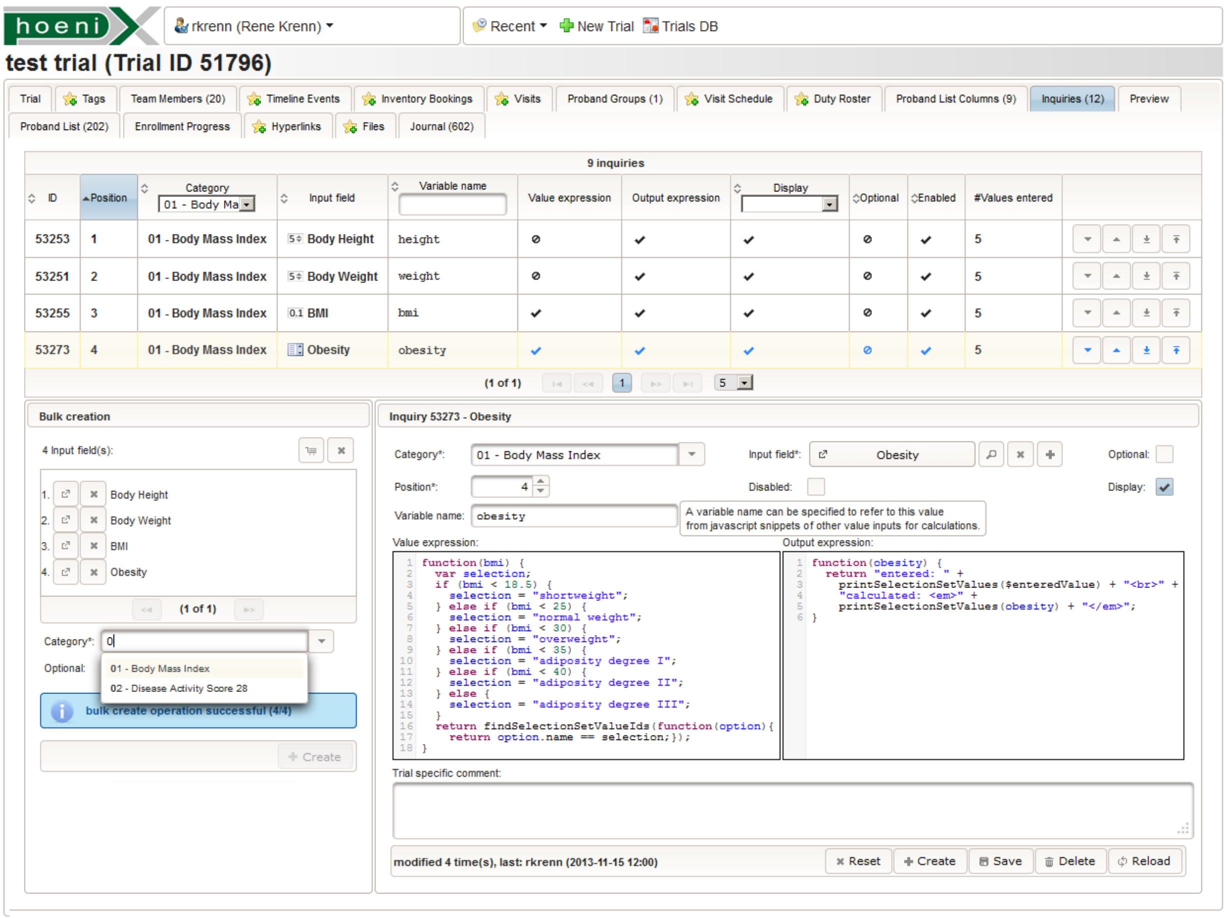
(1 of 1) 1 10

Selection set value 53264 - Obesity: normal weight

Name*: normal weight Value: normal weight

Preset: The value is optional and relevant for queries and data exports.

Figure 4.104.: Option items are represented by selection set value records. Basically, an option item value has a unique name and an optional (string) value.



test trial (Trial ID 51796)

9 inquiries

ID	Position	Category (01 - Body Ma)	Input field	Variable name	Value expression	Output expression	Display	Optional	Enabled	#Values entered
53253	1	01 - Body Mass Index	Body Height	height	⊘	✓	✓	⊘	✓	5
53251	2	01 - Body Mass Index	Body Weight	weight	⊘	✓	✓	⊘	✓	5
53255	3	01 - Body Mass Index	BMI	bmi	✓	✓	✓	⊘	✓	5
53273	4	01 - Body Mass Index	Obesity	obesity	✓	✓	✓	⊘	✓	5

(1 of 1) 1 5

Bulk creation

4 Input field(s):

1. Body Height
2. Body Weight
3. BMI
4. Obesity

(1 of 1)

Category*: 0

Optional:

01 - Body Mass Index
02 - Disease Activity Score 28

bulk create operation successful (4/4)

Inquiry 53273 - Obesity

Category*:	01 - Body Mass Index	Input field*:	Obesity	Optional:	☐
Position*:	4	Disabled:	☐	Display:	✓

Variable name: obesity

A variable name can be specified to refer to this value from javascript snippets of other value inputs for calculations.

Value expression:

```javascript
function(bmi) {
  var selection;
  if (bmi < 18.5) {
    selection = "shortweight";
  } else if (bmi < 25) {
    selection = "normal weight";
  } else if (bmi < 30) {
    selection = "overweight";
  } else if (bmi < 35) {
    selection = "adiposity degree I";
  } else if (bmi < 40) {
    selection = "adiposity degree II";
  } else {
    selection = "adiposity degree III";
  }
  return findSelectionSetValueIds(function(option){
    return option.name == selection;});
}
```

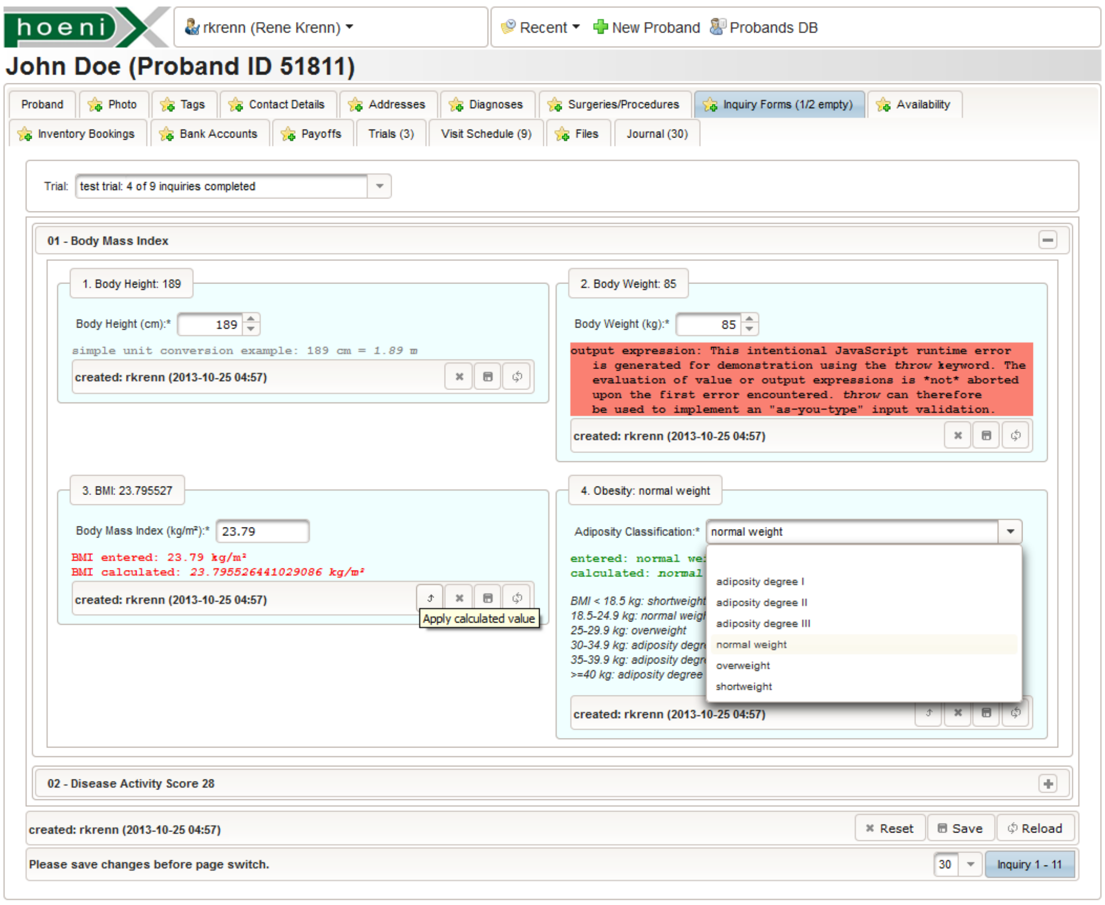
Output expression:

```javascript
function(obesity) {
  return "entered: " +
    printSelectionSetValues($enteredValue) + "<br>" +
    "calculated: <em>" +
    printSelectionSetValues(obesity) + "</em>";
}
```

Trial specific comment:

modified 4 time(s), last: rkrenn (2013-11-15 12:00)

Figure 4.105.: The trial entity view's inquiry tab is used to compose proband inquiry input forms. The datatable shows the setup of the BMI/obesity expression tree example. Prepared input fields (figure 4.100, 4.100, 4.102 and 4.103) are arranged to compose a section of the input form shown in figure 4.106. The section is formed by specifying the same category label ("01 - Body Mass Index") for inquiry input fields which are supposed to appear in the same `p:fieldSet` frame. When a variable name is specified, value and output expressions can be entered in corresponding `pe:codeMirror` UI input components. These are configured to provide syntax highlighting for the JavaScript language.



John Doe (Proband ID 51811)

Trial: test trial: 4 of 9 inquiries completed

01 - Body Mass Index

1. Body Height: 189

Body Height (cm):* 189

```
simple unit conversion example: 189 cm = 1.89 m
```

created: rkrenn (2013-10-25 04:57)

2. Body Weight: 85

Body Weight (kg):* 85

```
output expression: This intentional JavaScript runtime error
   is generated for demonstration using the throw keyword. The
   evaluation of value or output expressions is *not* aborted
   upon the first error encountered. throw can therefore
   be used to implement an "as-you-type" input validation.
```

created: rkrenn (2013-10-25 04:57)

3. BMI: 23.795527

Body Mass Index (kg/m²):* 23.79

```
BMI entered: 23.79 kg/m²
BMI calculated: 23.795526441029086 kg/m²
```

created: rkrenn (2013-10-25 04:57)

4. Obesity: normal weight

Adiposity Classification:* normal weight

```
entered: normal we
calculated: normal
```

BMI < 18.5 kg: shortweight
18.5-24.9 kg: normal weig
25-29.9 kg: overweight
30-34.9 kg: adiposity degr
35-39.9 kg: adiposity degr
>=40 kg: adiposity degree

adiposity degree I
adiposity degree II
adiposity degree III
normal weight
overweight
shortweight

created: rkrenn (2013-10-25 04:57)

02 - Disease Activity Score 28

created: rkrenn (2013-10-25 04:57)

Please save changes before page switch. 30 Inquiry 1 - 11

Figure 4.106.: A trial's inquiry form can be selected for data entry in the proband entity view. An input form comprises pages of input fields for capturing inquiry question answers or other parameter values, which are layouted and grouped by their category label. Entered values can be saved separately or page-wise. The output expression results are displayed below the input fields' input components. The first inquiry question ("Body Height") uses an output expression for unit conversions. The output expression of "Body Weight" generates a Js runtime error to demonstrate their visual appearance. The "BMI" inquiry uses a value expression to compute the field value ($f_2 : bmi = \frac{weight[kg]}{height[m]^2}$). The value expression of "Obesity" is shown in figure 4.105. Both "BMI" and "Obesity" inquiry questions show the ⤴ button, which adjusts the value entered to the recent calculation result.

Aside value expressions f_i, independent Js output expressions o_j can be specified per input field. o_j takes a set of parameters $Q_j \subseteq V$ and returns a *string value*, which is displayed by the UI (figure 4.106). In contrast to value expressions, function compositions of output expressions cannot be constructed and therefore neither expression trees nor potential cycles exist. An output expression can be used to reflect the corresponding v_i value or results of optional processing like unit conversion (listing 4.41) below the input component.

```
function (hba1c_mmol_per_mol) { //print the original and converted HbA1c reading:
  return sprintf("HbA1c endered (mmol/mol): %.2f<br/>" +
    "HbA1c calculated (%%): <em>%.1f</em>",
    hba1c_mmol_per_mol,
    hba1c_mmol_per_mol * 0.0915 + 2.15);
}
```

Listing 4.41: This output expression will print a IFCC regime HbA_{1c} input field value an its converted NGSP regime value (Day and Bailey, 2009).

An output expression allows to build strings in a user-defined format including HTML markup, e.g. for individual highlighting. By default, output strings are displayed using a monospaced typeface with dynamic coloring:

`green text`: The text is printed in green color if the entered value equals the calculated value. The implementation considers a tolerance $\varepsilon = 10^{-6}$ for float equality checks.
`red text`: The entered value differs from the calculated value. By applying the calculated value v_i, users can copy the calculated value into the corresponding entered value e_i. This will update the displayed input field value and the red output text turns green.
`grey text`: If an output expression is defined but no value expression, the string is displayed in grey color. There is no ↥ button ("Apply calculated value") available in this case.
`salmon background`: If either the output expression or value expression contains syntax or runtime errors, the Js interpreter error message is forwarded. A predefined error message is shown, if a value expression cycle was detected.

4.8.3.2. Implementation

The desired behaviour can be outlined as follows: a clinet-side *computation pass* updates variable values v_i and output strings, whenever an entered value e_i is modified by the user or a PPR cycle updates input field UI components (figure 4.107).

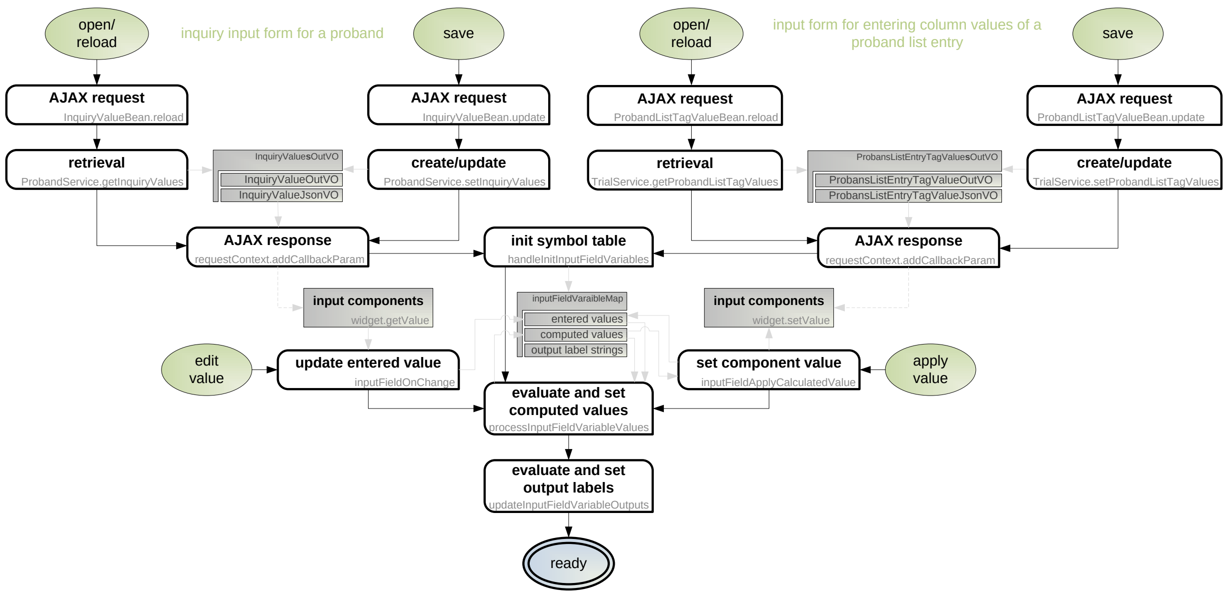

open/ reload
inquiry input form for a proband
save
open/ reload
input form for entering column values of a proband list entry
save
AJAX request
InquiryValueBean.reload
AJAX request
InquiryValueBean.update
AJAX request
ProbandListTagValueBean.reload
AJAX request
ProbandListTagValueBean.update
retrieval
ProbandService.getInquiryValues
InquiryValuesOutVO
InquiryValueOutVO
InquiryValueJsonVO
create/update
ProbandService.setInquiryValues
retrieval
TrialService.getProbandListTagValues
ProbansListEntryTagValuesOutVO
ProbansListEntryTagValueOutVO
ProbansListEntryTagValueJsonVO
create/update
TrialService.setProbandListTagValues
AJAX response
requestContext.addCallbackParam
init symbol table
handleInitInputFieldVariables
AJAX response
requestContext.addCallbackParam
input components
widget.getValue
inputFieldVaraibleMap
entered values
computed values
output label strings
input components
widget.setValue
edit value
update entered value
inputFieldOnChange
set component value
inputFieldApplyCalculatedValue
apply value
evaluate and set computed values
processInputFieldVariableValues
evaluate and set output labels
updateInputFieldVariableOutputs
ready

Figure 4.107.: This flow diagram gives an overview of the field value calculation logic described in this section. Staring with an AJAX request to load an input form initially, it superimposes the regular PPR cycle (dashed lines). Black arrows denote execution order, while gray arrows indicate (parts of) the data flow.

To inspect the implementation, a PPR cycle is considered as starting point. The cycle is performed when the input form is loaded initially, reloaded or saved.

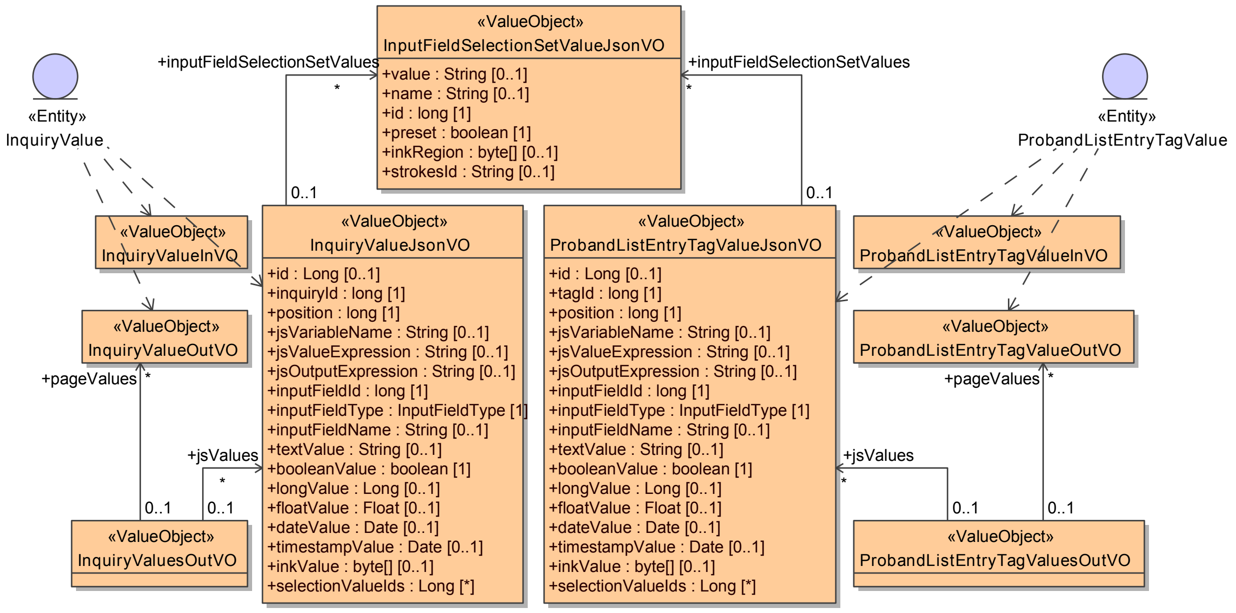
```
// Attention: Generated code! Do not modify by hand!
// Generated by: SpringService.vsl in andromda-spring-
     cartridge.
public interface ProbandService {

    ...

    @MethodParameterNames({"auth","inquiryValuesIns"})
    public InquiryValuesOutVO setInquiryValues(
         AuthenticationVO auth, Set<InquiryValueInVO>
         inquiryValuesIns) throws AuthenticationException,
         AuthorisationException, ServiceException;

    @MethodParameterNames({"auth","trialId","category","
         active","probandId","sort","psf"})
    public InquiryValuesOutVO getInquiryValues(
         AuthenticationVO auth, Long trialId, String
         category, Boolean active, Long probandId, boolean
         sort, PSFVO psf) throws AuthenticationException,
         AuthorisationException, ServiceException;

    ...
```

Listing 4.42: Multiple `InquiryValue` records of an input form page can be persisted or retrieved transactionally with a single service method invocation.

```
// Attention: Generated code! Do not modify by hand!
// Generated by: SpringService.vsl in andromda-spring-
     cartridge.
public interface TrialService {

    ...

    @MethodParameterNames({"auth","
         probandListEntryTagValuesIns"})
    public ProbandListEntryTagValuesOutVO
         setProbandListEntryTagValues(AuthenticationVO auth,
          Set<ProbandListEntryTagValueInVO>
         probandListEntryTagValuesIns) throws
         AuthenticationException, AuthorisationException,
         ServiceException;

    @MethodParameterNames({"auth","probandListEntryId","sort"
         ,"psf"})
    public ProbandListEntryTagValuesOutVO
         getProbandListEntryTagValues(AuthenticationVO auth,
          Long probandListEntryId, boolean sort, PSFVO psf)
         throws AuthenticationException,
         AuthorisationException, ServiceException;

    ...
```

Listing 4.43: Multiple `ProbandListEntryTagValue` records of an input form page can be persisted or retrieved transactionally with a single service method invocation.

An input form UI page PPR cycle will invoke a save (`set*Values`) or retrieval (`get*Values`) service method for `InquiryValue` (listing 4.42) or `ProbandListEntryTagValue` (listing 4.43) entities to load or persist values of input fields displayed by the selected input form page. `set*Values` operations allow rolling back transactions, if the same (page of) input fields of the same proband are edited by users at the same time (section 4.1.3.2).


«ValueObject»
InputFieldSelectionSetValueJsonVO
+value : String [0..1]
+name : String [0..1]
+id : long [1]
+preset : boolean [1]
+inkRegion : byte[] [0..1]
+strokesId : String [0..1]
+inputFieldSelectionSetValues
*
0..1
+inputFieldSelectionSetValues
*
0..1
«Entity»
InquiryValue
«Entity»
ProbandListEntryTagValue
«ValueObject»
InquiryValueInVO
«ValueObject»
InquiryValueOutVO
+pageValues
*
0..1
+jsValues
*
0..1
«ValueObject»
InquiryValuesOutVO
«ValueObject»
InquiryValueJsonVO
+id : Long [0..1]
+inquiryId : long [1]
+position : long [1]
+jsVariableName : String [0..1]
+jsValueExpression : String [0..1]
+jsOutputExpression : String [0..1]
+inputFieldId : long [1]
+inputFieldType : InputFieldType [1]
+inputFieldName : String [0..1]
+textValue : String [0..1]
+booleanValue : boolean [1]
+longValue : Long [0..1]
+floatValue : Float [0..1]
+dateValue : Date [0..1]
+timestampValue : Date [0..1]
+inkValue : byte[] [0..1]
+selectionValueIds : Long [*]
«ValueObject»
ProbandListEntryTagValueJsonVO
+id : Long [0..1]
+tagId : long [1]
+position : long [1]
+jsVariableName : String [0..1]
+jsValueExpression : String [0..1]
+jsOutputExpression : String [0..1]
+inputFieldId : long [1]
+inputFieldType : InputFieldType [1]
+inputFieldName : String [0..1]
+textValue : String [0..1]
+booleanValue : boolean [1]
+longValue : Long [0..1]
+floatValue : Float [0..1]
+dateValue : Date [0..1]
+timestampValue : Date [0..1]
+inkValue : byte[] [0..1]
+selectionValueIds : Long [*]
«ValueObject»
ProbandListEntryTagValueInVO
«ValueObject»
ProbandListEntryTagValueOutVO
+pageValues
*
+jsValues
*
0..1
0..1
«ValueObject»
ProbandListEntryTagValuesOutVO
Figure 4.108.: Beside InVOs and OutVOs, additional DTOs (`*JsonVO`) are modelled for `InquiryValue` and `ProbandListEntryTagValue` entities. `*JsonVO` instance are designed to be serialized in JSON format directly. `*ValuesOutVO` VOs are used by service methods for page-wise operations (listing 4.42, 4.43).

However, these service methods need special precautions to support field value calculation in combination with pagination. Extensive expression trees may span multiple input form pages. Additional values required by expressions of input fields visible on the selected input form page may reside on other pages. Therefore the `InquiryValuesOutVO` or `ProbandListEntryTagValuesOutVO` container VOs (figure 4.108) returned by concerned service methods are each populated with:

- Field values visible on the selected input form page (`pageValues`)
- Field values, the visible values' expressions depend on (`jsValues`)[84]

With the current implementation, field value calculation is basically performed *client-side*, meaning the value and output Js expressions are evaluated by the browser (only). This implies the server-side business logic is not aware of expression trees. At the moment, `jsValues` is therefore simply populated with *any* value of an associated `Inquiry` or `ProbandListEntryTag` item which defines a non-empty Js value expression or output expression.[85]

```java
@ManagedBean
@ViewScoped
public class InquiryValueBean extends ManagedBeanBase { //managed bean for the proband's inquiry input form

    private ArrayList<InquiryValueInVO> inquiryValuesIn; //the current input form page's InVOs
    private Collection<InquiryValueOutVO> inquiryValuesOut;
    private Collection<InquiryValueJsonVO> jsInquiryValuesOut; //holds any v_i value with non-empty f_i or o_i

    private Long probandId; //the ID of the selected root entity
    private Long trialId; //display the inquiry input form of the selected trial
    private Paginator paginator; //a custom paginator model for input form pagination

    private final static com.google.gson.Gson GSON = new com.google.gson.GsonBuilder() //JSON converter singleton
        .setExclusionStrategies(new GsonExclusionStrategy(UserOutVO.class)) //exclude OutVO graph cycles
        .serializeNulls()
        .setDateFormat(CommonUtil.INPUT_DATETIME_PATTERN)
        .create();

    ...

    //The ManagedBeanBase base class provides an implementation scheme for action listener methods.
    //It encapsulates appending custom AJAX parameter values using org.primefaces.context.RequestContext:
    //  public void reload() { appendRequestContextCallbackArgs(reloadImpl()); }
    //  protected abstract boolean reloadImpl();
    //By default, no parameter key-value pairs are added:
    //  protected void appendRequestContextCallbackArgs(boolean operationSuccess) {}
    @Override protected boolean reloadImpl() { //the "Reload" button action listener method implementation
        ...
        try { //service method invocation:
            InquiryValuesOutVO result = WebUtil.getServiceLocator().getProbandService().getInquiryValues(WebUtil.
                  getAuthentication(),trialId,true,probandId,true,paginator.getPsf());
            inquiryValuesOut = result.getPageValues();
            jsInquiryValuesOut = result.getJsValues();
            ...
            return true; //operation completed sucessfully
        } catch (ServiceException e) { ...
        } catch (AuthenticationException e) { ...
        ...
        }
        return false;
    }
    @Override protected void appendRequestContextCallbackArgs(boolean operationSuccess) { //auxiliary AJAX parameter values:
        ...
        if (operationSuccess) {
            //AJAX_INPUT_FIELD_VARIABLE_VALUES_BASE64 transports jsInquiryValuesOut to initialize the client-side field value
                  calculation:
            requestContext.addCallbackParam(JSValues.AJAX_INPUT_FIELD_VARIABLE_VALUES_BASE64.toString(),WebUtil.encodeBase64(
                 GSON.toJson(jsInquiryValuesOut)));
            //pass additional records in the same way, e.g. current proband (ProbandOutVO) and trial (TrialOutVO) ...
        }
    }

    ...
```

Listing 4.44: The inquiry input forms tab's facelet is backed by the `InquiryValueBean` managed bean. Its action listener methods propagate serialized `*JsonVO` DTOs, containing any required data to setup and execute the computation of field values client-side.

[84]In order to successfully evaluate f_i and o_i of a visible field value i, values $\bigcup_{P_i} \cup \bigcup_{O_i}$ are required.

[85]Since Js snippets entered by users may contain comments and blank lines, the server-side logic requires a simple Js parser to determine empty expressions however.

The list of `*JsonVO` objects carried by the `jsValues` field of service method results is converted into a JSON string using *Gson*. This string is transferred to the browser by piggybacking payload on the HTTP response using PrimeFace's `RequestContext` API (listing 4.44). To wire client-side processing of the JSON string, relevant UI components capable of initiatiating PPRs are configured to invoke a Js callback after the AJAX response was received (listing 4.45).

```
<p:commandButton
    process="@this"
    value="#{labels.reload_button_label}"
    actionListener="#{inquiryValueBean.reload}"
    oncomplete="handleUpdateProbandTabTitles(xhr, status, args);handleInitInputFieldVariables(xhr, status, args)" <!--
         oncomplete handlers for updating the entity view tab titles and initilaizing field value computation -->
    icon="ui-icon-refresh"
    ajax="true"
    disabled="#{!inquiryValueBean.created}"
    update="inquiriestrial,inquiry_input_grid" /> <!-- PPR target: update the entire input field layout container and the trial
          dropdown for selecting the inquiry input form -->
```

Listing 4.45: To process `*JsonVO` objects and trigger the calculation of field values, an AJAX `oncomplete` callback is registered. This is shown by this facelet fragment for the inquiry input form's page "Reload" button.

```
var inputFieldVariableMap = {}; //the symbol table hash map, a global object variable
function handleInitInputFieldVariables(xhr, status, args) { //AJAX oncomplete callback for initializing field value computation
    if (_testFlag(args,AJAX_OPERATION_SUCCESS)) {
        inputFieldVariableMap = {}; //clear the symbol table
        _initInputFieldVariableValues(args); //initialize the symbol table
        //_initAdditionalVars(args); store records passed additionally, e.g. the current proband and trial
        _processInputFieldVariableValues(); //run the field value computation algorithm to update the symbol table initially
        _updateInputFieldVariableOutputs(); //evaluate output expressions and set corresponding UI output label <div>s
    }
}
function _initInputFieldVariableValues(args) { //process the AJAX_INPUT_FIELD_VARIABLE_VALUES_BASE64 AJAX parameter containing
      serialized InquiryValueJsonVO or ProbandListEntryTagValueJsonVO lists:
    if (_testPropertyExists(args,AJAX_INPUT_FIELD_VARIABLE_VALUES_BASE64)) {
        var inputFieldVariableValues = JSON.parse(decodeBase64(args[AJAX_INPUT_FIELD_VARIABLE_VALUES_BASE64])); //assemble Js
             array of *ValueJsonVO objects passed in JSON format
        var cs = new CommentStripper(); //a simple Js parser to remove comments and blank lines (Dunn, 2013)
        for (i = 0; i < inputFieldVariableValues.length; i++) {
            var inputFieldVariableValue = inputFieldVariableValues[i];
            //convert fields, e.g. date string to date objects:
            if (inputFieldVariableValue.timestampValue != null  inputFieldVariableValue.timestampValue.length > 0) {
                inputFieldVariableValue.timestampValue = Date.parseExact(inputFieldVariableValue.timestampValue,
                     INPUT_DATETIME_PATTERN);
            }
            ...
            //strip comments and blank lines from Js expressions to ensure the first line contains the Js function declaration:
            inputFieldVariableValue.jsValueExpression = cs.strip(inputFieldVariableValue.jsValueExpression); //value expression
                 Js snippet
            inputFieldVariableValue.jsOutputExpression = cs.strip(inputFieldVariableValue.jsOutputExpression); //output
                 expression Js snippet
            ...
            var inputFieldVariable = {};
            if (inputFieldVariableValue.jsOutputExpression != null  inputFieldVariableValue.jsOutputExpression.length > 0) { //
                 derive the DOM element ID of the output label <div>
                if (_testPropertyExists(inputFieldVariableValue, "inquiryId")) {
                    inputFieldVariable.outputId = INPUT_FIELD_OUTPUT_ID_PREFIX + inputFieldVariableValue.inquiryId;
                } else if (_testPropertyExists(inputFieldVariableValue, "tagId")) {
                    inputFieldVariable.outputId = INPUT_FIELD_OUTPUT_ID_PREFIX + inputFieldVariableValue.tagId;
                }
            } else {
                inputFieldVariable.outputId = null; //no output expression (and label)
            }

            inputFieldVariable.processed = false; //the variable's state is "not processed" initially
            inputFieldVariable.valueErrorMessage = null; //Js evaluation error of value expression f_i
            inputFieldVariable.outputErrorMessage = null; //Js evaluation error of output expression o_i
            inputFieldVariable.output = null; //output expression o_i result
            inputFieldVariable.delta = false; //false: green text color, true: red text color
            inputFieldVariable.value = inputFieldVariableValue; //variable value v_i
            inputFieldVariableMap[inputFieldVariableValue.jsVariableName] = inputFieldVariable;
            inputFieldVariable.enteredValue = jQuery.extend(true, {}, inputFieldVariableValue); //entered value: deep copy of
                 the variable value v_i
        }
    }
}
```

Listing 4.46: The client-side symbol table is a map of variable objects. Each variable name references a variable object data structure, which is initialized by processing the corresponding `*JsonVO` item. The setup completes with an initial computation pass.

On the client side, the `handleInitInputFieldVariables` callback initializes the symbol table by processing the serialized list of `InquiryValueJsonVO` or `ProbandListEntryTagValueJsonVO` items, which are therefore modeled to provide identical fields (figure 4.108).

```
var sampleVariableObject = {
   "outputId":"inputfield_output_53273", //DOM element ID of output label <div>
   "processed":true, //value expression f_i is ready and up-to-date
   "valueErrorMessage":null, //Js evaluation error of value expression f_i
   "outputErrorMessage":null, //Js evaluation error of output expression o_i
   "output":null, //output expression o_i result
   "delta":false, //false: green text color, true: red text color
   "value":{ //variable value v_i
      "id":53489, //InquiryValue.id
      "inquiryId":53273, //InquiryValue.inquiry.id
      "position":4, //InquiryValue.inquiry.position
      "jsVariableName":"obesity", //InquiryValue.inquiry.jsVariableName
      "jsValueExpression":"function(bmi) {\r\n  var selection;\r\n  if (bmi < ...", //InquiryValue.inquiry.jsValueExpression f_i
      "jsOutputExpression":"function(obesity) {\r\n  return \"entered: \" + ...", //InquiryValue.inquiry.jsOutputExpression o_i
      "inputFieldId":53260, //InquiryValue.inquiry.field.id
      "inputFieldType":"SELECT_ONE", //InquiryValue.inquiry.field.fieldType
      "inputFieldName":"Obesity", //InquiryValue.inquiry.field.name
      "textValue":null, //InquiryValue.value.textValue
      "booleanValue":false, //InquiryValue.value.booleanValue
      "longValue":null, //InquiryValue.value.longValue
      "floatValue":null, //InquiryValue.value.floatValue
      "dateValue":null, //InquiryValue.value.dateValue
      "timestampValue":null, //InquiryValue.value.timestampValue
      "inkValues":null, //InquiryValue.value.inkValue
      "selectionValueIds":[ //InquiryValue.value.selectionValueIds
         53264
      ],
      "inputFieldSelectionSetValues":[ //InquiryValue.inquiry.field.selectionSetValues
         { "value":"normal weight","name":"normal weight","id":53264,"preset":false,"inkRegions":null,"strokesId":null },
         { "value":"overweight","name":"overweight","id":53284,"preset":false,"inkRegions":null,"strokesId":null },
         { "value":"shortweight","name":"shortweight","id":53262,"preset":false,"inkRegions":null,"strokesId":null },
         { "value":"adiposity degree I",... },
         ...
      ]
   },
   "enteredValue":{ //entered value e_i
      ...
   }
};
```

Listing 4.47: This listing shows the JSON representation of a variable object sample instance (v_1 : *obesity*) after initialisation.

The symbol table is represented by a Js object variable (`inputFieldVariableMap`), which is used as a map for accessing variable objects using their variable name (listing 4.46). A variable object combines the variable value v_i and entered value e_i (listing 4.47). When passing values to/from expressions or UI components, the variable object's value data structures (`sampleVariableObject.value` and `sampleVariableObject.enteredValue`) are marshalled into a suitable Js object type (e.g. `Date`, array) or primitive type (boolean, numeric, string) that corresponds the input field's type. Additional fields of variable objects support the computation algorithm, which evaluates value expressions by traversing the expression tree in depth-first order (listing 4.48). The entered value is copied for variables without value expression. When a computation pass completes, any variable object's `processed` field is set to `true`. Output expressions are processed in the final step to set the text of output label `<div>` elements. An explicit Js function context is prepared, which masks the global variable scope (sandboxing). This should reduce the risk of crashing the UI's Js infrastructure by (accidentially) overriding existing functions in the value or output expression code.

```
function _processInputFieldVariableValues() { //a complete computation pass:
    for (var variableName in inputFieldVariableMap) { //iterate variable objects (in arbitrary order)
        _processInputFieldVariableValue(variableName,{}); //each variable is considered as potential expression tree root
    }
}
function _processInputFieldVariableValue(variableName,cycleCheckMap) { //compute a single variable object of the symbol table:
    var inputFieldVariable = inputFieldVariableMap[variableName];
    if (inputFieldVariable) {
        if (inputFieldVariable.processed) { //the calculated value is already present
            return inputFieldVariable.value;
        }
        //evaluate jsValueExpression:
        var evaluation = _evalInputFieldVariableExpression(variableName,inputFieldVariable.value.jsValueExpression,"value
             expression",cycleCheckMap,inputFieldVariable,true);
        if (evaluation != null) { //update the variable value, unless evaluation was aborted:
            inputFieldVariable.valueErrorMessage = evaluation.errorMessage;
            _setInputFieldVariableValue(inputFieldVariable.value,evaluation.returnValue); //set the type-dependent value field,
                 e.g. inputFieldVariable.value.floatValue
        }
        //reset output expression related fields:
        inputFieldVariable.outputErrorMessage = null;
        inputFieldVariable.output = null;
        inputFieldVariable.processed = true;
        inputFieldVariable.delta = !_equalInputFieldVariable(inputFieldVariable);
        return inputFieldVariable.value;
    } else { return null; }
}
function _evalInputFieldVariableExpression(variableName,expression,errorMessagePrefix,cycleCheckMap,inputFieldVariable,
      copyEnteredValue) { //evaluate value expressions (or output expressions):
    var errorMessage;
    var returnValue;
    //a simple function f_i could look like: expression = "function (weight, height) {\n\r return weight / (height * height);\n\r
       }"
    if (expression != null  expression.length > 0  expression.indexOf("{") > 0) { //test if an expression is present
        if (variableName in cycleCheckMap) {
            errorMessage = errorMessagePrefix + ": circular dependency for variable " + variableName + " detected";
            returnValue = null;
        } else {
            cycleCheckMap[variableName] = true;
            var definition = expression.substring(0,expression.indexOf("{")).replace(/\s+/gm,''); //remove whitespaces:
                 definition = "function(weight,height)"
            var expressionDeclarationRegExp = /^function\((([a-zA-Z0-9_]+,?)*)\)/; //RegEx to match a Js anonymous function
                 declaration, including groups for parameters
            if (expressionDeclarationRegExp.test(definition)) { //the function declaration is syntactically correct:
                var matches = expressionDeclarationRegExp.exec(definition);
                var argNames = matches[1].split(","); //P_i: argNames = ["weight", "height"]
                var mask = {}; //a context for the expression evaluation is constructed:
                for (var p in this) { //sandbox: mask global properties
                    mask[p] = undefined;
                }
                var argValue;
                for (var i = 0; i < argNames.length; i++) { //prepare P_i variable values:
                    if (argNames[i] != null  argNames[i].length > 0) {
                        //expression tree depth-first traversal:
                        argValue = _processInputFieldVariableValue(argNames[i],cycleCheckMap);
                        if (inputFieldVariable.valueErrorMessage  variableName == argNames[i]) {
                            return null; //abort to prevent overriding the cycle error message which was set already
                        } else {
                            mask[argNames[i]] = _getInputFieldVariableValue(argValue); //get the type-dependent value, e.g.
                                 inputFieldVariable.value.floatValue
                        }
                    }
                }
                //these "system variables" will allow users writing more complex Js expression snippets:
                mask["$enteredValue"] = _getInputFieldVariableValue(inputFieldVariable.enteredValue); //the entered value
                mask["$oldValue"] = _getInputFieldVariableValue(inputFieldVariable.value); //the variable value prior its
                     update
                _exportExpressionUtils(mask,inputFieldVariable); //export Js libraries, additional system variables and helper
                     methods
                returnValue = null;
                try { //try to evaluate the expression:
                    returnValue = (new Function( "with(this) {\nreturn (" +  expression + ")(" + argNames.join(",") + ");\n}"))
                         .call(mask);
                    errorMessage = null;
                } catch (e) {
                    errorMessage = errorMessagePrefix + ": " + e.toString();
                }
            } else { //the function declaration cannot be parsed:
                errorMessage = errorMessagePrefix + ": value expression declaration for variable " + variableName + " invalid";
                returnValue = null;
            }
        }
        return {'errorMessage':errorMessage,'returnValue':returnValue}; //return the evaluation result
    } else if (copyEnteredValue) { //deep copy e_i to v_i:
        inputFieldVariable.value = jQuery.extend(true, {}, inputFieldVariable.enteredValue);
        return null;
    } else { return null; }
}
```

Listing 4.48: The implementation of the field value computation algorithm.

When developing expressions, users can rely on predefined helper methods, variables and Js libraries, which are exported to the function context as an aid:

`$enteredValue` The entered value can be referred explicitly, which is for example useful for output expressions in order to face the computed and entered value.

`$oldValue` The variable's previous value (before it was updated by the ongoing computation pass) can be referred in expressions.

`$selectionSetValues` An item selection value is provided in form of a Js array of `InputFieldSelectionSetValue.ids`. `$selectionSetValues` is a predefined map for `sampleVariableObject.value.inputFieldSelectionSetValues` array elements, using ID values as keys. Expression code can therefore look up an `InputFieldSelectionSetValueJsonVO` object to access its fields like the `name` or `value` strings.

`$proband, $trial, ...` The implementation can be extended by passing additional context information, e.g. in form of related records such as the root entity. For example, the `$proband` system variable provides the `ProbandOutVO`[86] object of the currently opened proband, containing `age` and `gender` fields. As an example, the value expression in listing 4.49 implements the Modification of Diet in Renal Disease (MDRD) formula for calculating the Estimated Glomerular Filtration Rate (eGFR), which depends on age and gender values (Levey, 1999):

$$\text{eGFR}[ml/min] = 175\ S_{cr}[mg/dl]^{-1.154} age^{-0.203} \cdot \begin{cases} 1.210\ if\ black \\ 1\ else \end{cases} \cdot \begin{cases} 0.742\ if\ female \\ 1\ else \end{cases}$$

```
function(s_cr, skin_color) { // s_cr: serum creatinin concentration (decimal), skin_color (single selection)
    var result = 175.0 * Math.pow(s_cr, -0.203) * Math.pow($proband.age, -0.203);
    if (getInputFieldSelectionSetValue('skin_color', skin_color[0]).value == 'black') {
        result *= 1.210;
    }
    if ($proband.gender.sex == 'FEMALE') {
        result *= 0.742;
    }
    return result;
}
```

Listing 4.49: This value expression implements the MDRD formula for calculating the eGFR.

`getInputFieldSelectionSetValue` While `$selectionSetValues` provides a lookup map for option items of the expression's variable only, this helper methods allows to look up an `InputFieldSelectionSetValueJsonVO` object of other variables.

`printSelectionSetValues` This helper method simplifies the conversion of the field's item selection value array into a (custom) string format for output, which could be required frequently in output expressions.

[86]While `*JsonVO` DTOs are designed for a fast and hassle-free conversion into JSON format using Gson, circular references of general OutVOs (section 4.2.1) have to be cut off by implementing a custom `com.google.gson.ExclusionStrategy`.

`findSelectionSetValueIds` As shown by the value expression of the `obesity` variable in figure 4.105, the reverse lookup of a selection set value is required to produce a valid expression result. In the example, the `findSelectionSetValueIds` helper method's anonymous function argument defines the match condition for finding the ID value of an `InputFieldSelectionSetValue` record with a given `name` field value.

`sprintf` This function allows to format and convert numeric Js value types into strings.

`Date` The Datejs library remains available to the sandbox in order to provide comprehensive date value calculation and formatting capabilities.

`JSON` Since no specific helper methods for a programmatic manipulation of sketches are implemented at the moment, the JSON2 library namespace is provided to support handling the JSON format of sketch values at least.

After creating the symbol table and an initial computation pass, all variable objects are initialized at this point and users start entering data. Field value computation results should be updated with every single keypress. Therefore, each input field has a `onchange` event handler assigned that fits its input type (listing 4.50). The changed input field's UI component value is copied to the variable object's entered value field. Variable objects and output labels can now be refreshed by repeating the computation pass. To schedule refreshing a variable's calculation result, its `processed` flag is simply set to `false`. Instead of refreshing dependent variables only, the current implementation updates all variable objects, since a computation pass is fast enough for the expected complexity of value and output expressions.

```
//input components' onchange event handlers:
function singleLineTextOnChange(variableName,widget,outputId) { ... }
function selectOneOnChange(variableName,widget,outputId) {
    _inputFieldOnChange(variableName,[ widget.getValue() ]);
}
function selectManyOnChange(variableName,widget,outputId) { ... }
...

function _inputFieldOnChange(variableName,newValue) {
    var inputFieldVariable = inputFieldVariableMap[variableName];
    if (inputFieldVariable) {
        _setInputFieldVariableValue(inputFieldVariable.enteredValue,newValue); //set the entered value
        _refreshInputFieldVariables(); //refresh all calculated values and output labels
    }
}
function _refreshInputFieldVariables(exclude) {
    _invalidateInputFieldVariableValues(exclude);
    _processInputFieldVariableValues();
    _updateInputFieldVariableOutputs();
}
function _invalidateInputFieldVariableValues(exclude) {
    for (var variableName in inputFieldVariableMap) {
        if (!exclude || exclude != variableName) {
            inputFieldVariableMap[variableName].processed = false; //schedule computation
        }
    }
}
```

Listing 4.50: `onchange` event handlers capture the input component's value and trigger a computation pass whenever the input changed by the user.

While typing in data, users see output labels adjusting as they type. Users have to press an input field's ⤴ button ("Apply calculated value") explicitly in order to accept a calculated value. When applying the calculated value, the UI component's value is set to the computed value (listing 4.51). Since this amounts to changing the entered value, a computation pass is triggered to update dependent variables again. When saving the form, the input field's UI components value is persisted, disregarding the actual computation result. Differing entered and calculated values are therefore emphasized by the `red` output label text color.

```
function singleLineTextApplyCalculatedValue(variableName,widget,sourceId,rowId) { ... }
function selectOneApplyCalculatedValue(variableName,widget,sourceId,rowId) {
    var newValue = _inputFieldApplyCalculatedValue(variableName);
    widget.setValue(newValue ? newValue[0] : null); //update the UI component's value
    ajaxRequest(sourceId,sourceId,null,null); //optional AJAX POST request to synchronize the components value with managed
          bean/model
}
function selectManyApplyCalculatedValue(variableName,widget,sourceId,rowId) { ... }
...

function _inputFieldApplyCalculatedValue(variableName) {
    var inputFieldVariable = inputFieldVariableMap[variableName];
    if (inputFieldVariable) {
        inputFieldVariable.enteredValue = jQuery.extend(true, {}, inputFieldVariable.value);
        inputFieldVariable.delta = false;
        _refreshInputFieldVariables(variableName);
        return _getInputFieldVariableValue(inputFieldVariable.value);
    }
    return null;
}
```

Listing 4.51: The `onclick` event handler callback of each “Apply calculated value” button copies `sampleVariableObject.value` into `sampleVariableObject.enteredValue` and triggers a computation pass to adjust the remaining variable values and output labels.

In order to set a UI component’s value programmatically via Js, relevant PrimeFaces client-side widget prototypes were extended with value getter and setter methods. PrimeFaces does not provide accessor methods for most input compononents out-of-the-box, as outlined in section 4.6.1.7. An example is given for the `p:selectOneMenu` input component used by the single selection input field (listing 4.52).

```
//p:selectOneMenu value getter:
PrimeFaces.widget.SelectOneMenu.prototype.getValue = function() {
    var selectedOption = this.options.filter(':selected');
    return selectedOption.attr('value'); //e.g. a InputFieldSelectionSetValue.id
}
//p:selectOneMenu value setter:
PrimeFaces.widget.SelectOneMenu.prototype.setValue = function(value) { //e.g. the InputFieldSelectionSetValue.id
    var _self = this;
    var index = -1;
    for (var i = 0; i < _self.options.length; i++) { //lookup the given option value's element index by searching the options
          array:
        if (_self.options[i].value == value) {
            index = _self.options[i].index;
            break;
        }
    }
    if (index > 0) { //select the option item if found:
        _self.selectItem(_self.items.eq(index)); //the selectItem method is provided by PrimeFaces however
    }
}
```

Listing 4.52: This `getValue` and `getValue` functions allow setting and capturing a dropdown component’s value. Widget prototype extensions like this are required for any PrimeFaces UI input component used for input field types.

Summarizing, the approach for the field value computation feature presented relies on the lightweight, but yet powerful client-side evaluation of Js expressions. It provides sufficient performance and optimization options for scaling up to large expression trees of even larger input forms, which consider loading or updating dependent variables only. To support client applications without Js interpreter in a future version, Rhino (Boyd and Rhino Contributors, 2012) could be used for executing Js code server-side without the need to port existing expressions.

Phoenix CTMS users will need basic Js skills in order to develop their own expression snippets. This premise also holds e.g. for users of REDCap, but could nevertheless be discouraging. However, the concept provides a way to set up individual field value calculations using the UI, without the need to touch any kind of master data or even application code.

The development of a DSL for calculating values in clinical input forms using the parser generator framework Another Tool for Language Recognition (ANTLR) (Parr, 2007) is described in detail in (Lemmé, 2012). This approach was integrated in the SPICS project.

4.8.4. Sketch Input

Paper-based forms allow capturing certain information more intuitive, accurate or precisely in some situations. In order to adopt this advantage, the idea was to provide a *sketch* input field type, allowing users to draw *strokes* on UI screens using a pen-enabled device. Aside creating and persisting free-form sketches, applications like below were considered for the scope of the implementation, which require an elementary analysis of the stroke data:

- Visual Analog Scale (VAS) (figure 4.112)
- specifying an anatomical location
- scans of existing (multiple choice) questionnaires in paper form
- images for answer choices

4.8.4.1. Sketch Model

A vector-based representation of a single pen stroke is an ordered list of points $p_i = (x_i, y_i)$ with Cartesian float coordinates, generated by sampling the pointing device while it was moved.[87] The list is referred as *path* $P := (p_1, p_2, p_3, \dots, p_n)$ with arbitrary length $n > 1$. A sketch is represented by a list of stroke paths. Stroke z-order and undo/redo operations can be managed if the list is ordered by the time of creation of a stroke. A stroke can be rendered by a spline with knots p_i. For simplification, first-order splines are assumed, giving $n-1$ straight line segments $\overline{p_i, p_{i+1}}$ between points. A *region* is considered as a *simple* polygon, described by a *closed* path ($p_1 = p_n$) without intersections of its line segments. A region represents an area which is usually marked or selected by specifying an arbitrary point inside it. This concept suffices the desired functionality and gets along with two simple algorithms:

Line segment intersection Calculation of the intersection point u of two line segments a and b:

$$a := \overline{(x_{1_a}, y_{1_a}), (x_{2_a}, y_{2_a})} = \{A_a x + B_a y = C_a\}\,; A_a = y_{2_a} - y_{1_a}; B_a = x_{1_a} - x_{2_a}; C_a = A_a x_{1_a} + B_a y_{1_a}$$

$$b := \overline{(x_{1_b}, y_{1_b}), (x_{2_b}, y_{2_b})} = \{A_b x + B_b y = C_b\}\,; A_b = y_{2_b} - y_{1_b}; B_b = x_{1_b} - x_{2_b}; C_b = A_b x_{1_b} + B_b y_{1_b}$$

$$u = (x_u, y_u) = \texttt{intersection}(a,b) = \left(\frac{B_bC_a - B_aC_b}{d}, \frac{A_aC_b - A_bC_a}{d}\right); d = A_aB_b - A_bB_a$$

$d \neq 0$[88]; $x \geq min(x_{1_a}, x_{2_a}); x \leq max(x_{1_a}, x_{2_a}); y \geq min(y_{1_a}, y_{2_a}); y \leq max(y_{1_a}, y_{2_a}); x \geq min(x_{1_b}, x_{2_b}); x \leq max(x_{1_b}, x_{2_b}); y \geq min(y_{1_b}, y_{2_b}); y \leq max(y_{1_b}, y_{2_b})$

Point-in-Polygon (PiP) test The *crossing number*[89] algorithm (O'Rourke, 1998) can be used to decide wether a point v is enclosed by a region path C:

$$v = (x_v, y_v); C := (c_1, c_2, c_3, \dots, c_n); c_i = (x_i, y_i)$$

```
1. exterior point $q = (q_x, q_y) \leftarrow (min(x_i) - \epsilon, min(y_i) - \epsilon)$[90]
2. intersection count $r \leftarrow 0$
3. for i = 1 .. $n-1$:
       if exists intersection($\overline{c_i, c_{i+1}}$, $\overline{q, v}$): $r = r + 1$
4. $r \bmod 2 \overset{?}{=} 1 \Leftrightarrow v$ is inside $C \rightarrow$ boolean return value
```

[87]For example, coordinates can be captured whenever a Js `onmousemove` event is fired while a mouse button is pressed. The corresponding `ontouchmove` event can be used for devices with touchscreen.

[88]Lines must not be parallel.

[89]also known as *even-odd rule*

[90]Point q is an arbitrary point *outside* C.

4.8.4.2. Client-side Sketch Control Implementation

A sketch `InputField` is presented by a sketch UI control showing an user-defined *background image*, which is overlaid with rendered stroke paths (figure 4.109).

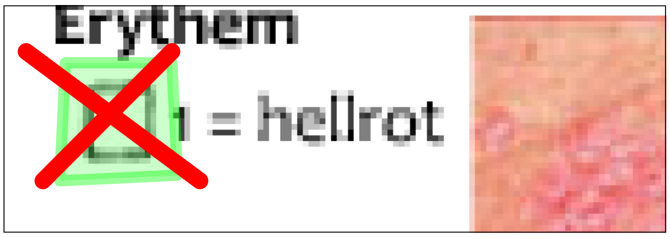
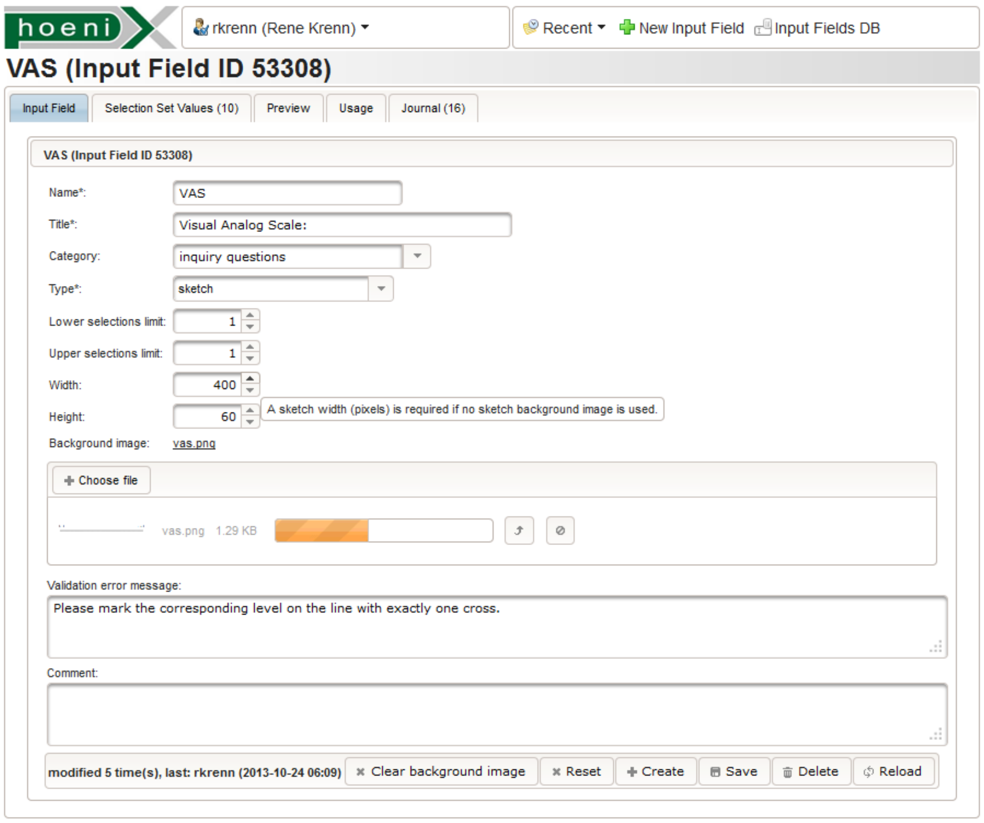


Figure 4.109.: A sketch contains two types of paths. Regions are closed and filled paths (in lime color in the example), which are read-only and can be hidden in input forms. Pen strokes are open paths (red), whose intersections are used to identify selected regions. A background image can be traced when preparing the semitransparent region polygons.



VAS (Input Field ID 53308)

VAS (Input Field ID 53308)

Name*:	VAS
Title*:	Visual Analog Scale:
Category:	inquiry questions
Type*:	sketch
Lower selections limit:	1
Upper selections limit:	1
Width:	400
Height:	60
Background image:	vas.png

A sketch width (pixels) is required if no sketch background image is used.

vas.png 1.29 KB

Validation error message:
Please mark the corresponding level on the line with exactly one cross.

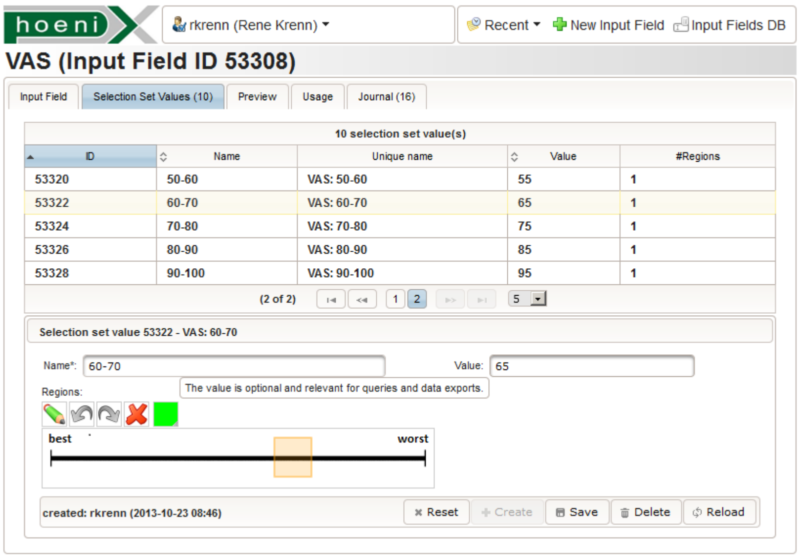
Comment:

modified 5 time(s), last: rkrenn (2013-10-24 06:09)

Figure 4.110.: This master tab shows the setup of a VAS sketch input field. In contrast to an input field for multiple selection, sketch dimensions and a background image can be specified. For the VAS, a prepared image showing a bare scale (line with “best” and “worst” end point labels) is uploaded.

Two modes of viewing and editing paths can be distinguished:

Edit regions Users may define region areas by drawing them and save them in a `InputFieldSelectionSetValue` record[91] associated with the input field (figure 4.111). Region paths are automatically closed and semi-transparently filled with a color of choice.



VAS (Input Field ID 53308)

10 selection set value(s)

ID	Name	Unique name	Value	#Regions
53320	50-60	VAS: 50-60	55	1
53322	60-70	VAS: 60-70	65	1
53324	70-80	VAS: 70-80	75	1
53326	80-90	VAS: 80-90	85	1
53328	90-100	VAS: 90-100	95	1

(2 of 2) 1 2 5

Selection set value 53322 - VAS: 60-70

Name*: 60-70 Value: 65

The value is optional and relevant for queries and data exports.

created: rkrenn (2013-10-23 08:46)

Figure 4.111.: Each option item defined contains one or more region paths. For the VAS input field example, squares are used to quantize the scale. Drawing exact geometry like rectangles is not supported at the moment. The region shown was therefore created by editing the related `InputFieldSelectionSetValue.inkRegion` value directly in the database. Selection presets are not supported for sketches.

Sketching When capturing data by drawing on a sketch control displayed by an input form, users can select a region by drawing two crossing lines, whose intersection point has to be enclosed by the desired region (figure 4.112). Visibility of region areas can be turned on and off when sketching. Pen strokes represent the sketch input field's primary data value, which is persisted without region paths when saving[91]. Selected region items are persisted by the `InputFieldValue.selectionSetValues` association to facilitate database queries.

[91]For flexibility when choosing a data format for stroke paths, the data model (figure 4.96) provides byte array table columns for the `InputFieldValue.inkValue` and `InputFieldSelectionSetValue.inkRegion` entity fields.

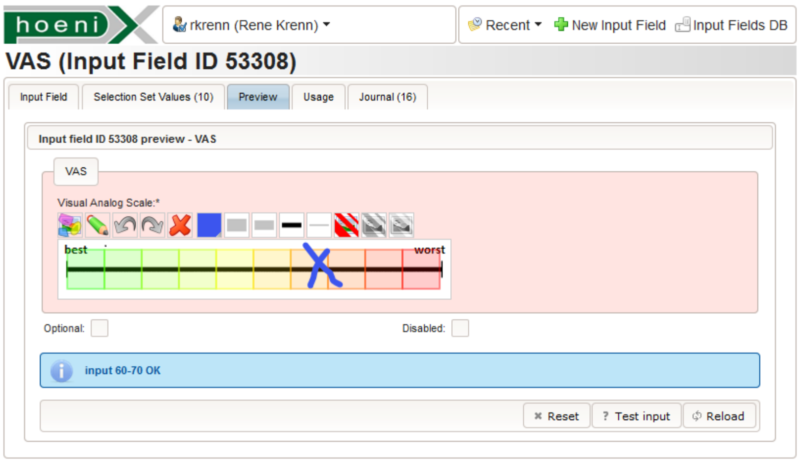


Figure 4.112.: Users can preview and test a created input field before using it for input forms. This is particularly helpful to check the correct behaviour when marking sketch regions, right after they have been defined.

Aside the drawing area (canvas) capable of capturing pen strokes, the sketch control requires a toolbox with buttons and switches for basic control and style options:

- region visibility toggle
- pencil/eraser mode toggle
- clear canvas button
- undo/redo buttons
- color picker
- pen thickness selector
- pen opacity selector

The functioniality behind is known from existing software with a sketch pad interface such as Windows Journal or Jarnal (Levine, Teege, and Hagerer, 2013). An existing browser-based sketch pad UI component for an integration in feasible time is a prerequisite preferred over a bare graphics API. To avoid introducing a dependency on a browser plug-in[92], it is desired to utilize native support of modern browsers for rendering strokes/regions to visualize a sketch. Relevant standards for this purpose below introduce a DOM element, which represents an output area for rendering graphics primitives. The graphics primitives (paths, images, . . .) can be created using a basic API for the browser's supported scripting language, usually superseded by optional (Js) libraries.

HTML5 Canvas The HTML5 `<canvas>` comes with a frame buffer Js API for drawing primitives like curves, images or video frames and even 3D vertex arrays (*WebGL Specification Version 1.0.2* 2013). Multiple available Js libraries provide drawing data models (scene graphs) including event handling for interactive processing. PaintWeb (Sucan, 2013) is an example of an integrated Js component for image editing. It would be a candidate for pixel-based sketches.

SVG The HTML5 `<svg>` tag allows including SVG markup within HTML files directly. Its vector drawing primitives are reflected by the DOM, providing an inherent drawing model for Js. The Raphaël Js library is an enhanced API, which is utilized for example by the Raphael SketchPad Js component. It provides essential vector-based sketch editing capabilities, but lacks functionality for managing closed paths and (background) images.

[92]e.g. Microsoft Silverlight, Adobe Flash, Java Applet . . .

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<svg height="40" version="1.1" width="120" xmlns="http://www.w3.org/2000/svg"> <!-- the SVG canvas represents the sketch area
    -->
  <desc>a sample SVG sketch</desc>
  <!-- a closed path represents a region polygon: -->
  <path style="stroke-opacity: 0.4; stroke-linecap: round; stroke-linejoin: round; fill-opacity: 0.2;" fill="#ff6900" stroke="#
      ff6900" d="M11,10L30,10L31,30L10,31Z" stroke-opacity="0.4" stroke-width="2" stroke-linecap="round" stroke-linejoin="
      round" fill-opacity="0.2"/>
  <!-- each open path represents a single pen stroke: -->
  <path style="stroke-opacity: 1; stroke-linecap: round; stroke-linejoin: round;" fill="none" stroke="#3c55ee" d="M4,8L35,31"
      stroke-opacity="1" stroke-width="3" stroke-linecap="round" stroke-linejoin="round"/>
  <path style="stroke-opacity: 1; stroke-linecap: round; stroke-linejoin: round;" fill="none" stroke="#3c55ee" d="M30,8L7,31"
      stroke-opacity="1" stroke-width="3" stroke-linecap="round" stroke-linejoin="round"/>
</svg>
```

Listing 4.53: The SVG format represents a XML document with node elements denoting drawing primitive instances. This SVG document sample source produces the sketch shown in figure 4.109 when rendered, without the background image.

The established SVG format is basically human readable and supported by many software packages for vector-based image editing, such as Inkscape (Inkscape Contributors, 2013). Since sketches are to be persisted in the database, the lightweight Raphael SketchPad was chosen because it relies on the SVG format standard to represent stroke data (listing 4.53). Its implementation was extended to support region paths by utilizing the "Z" (close path) token for SVG `<path>` definitions. Whenever a new pen stroke is captured, the fired `change` event invokes the implemented region selection algorithm. This algorithm works in two steps, which both include optimizations to reach acceptable performance:

1. Stroke path intersections At first, intersections between all m strokes of the sketch are determined. Since the naive approach would have an $O(m^2)$ duration, the intersection list of the previous `change` event is updated to increase performance (memoization). A fast bounding-box-overlap test shortcuts non-intersecting paths prior the more costly $O(k, l)$ calculation of two respective stroke paths with k and l line segments.

2. Point-in-region tests In order to populate a set of `InputFieldSelectionSetValue.strokesId` string values[93] corresponding the selected regions, each intersection point is tested against all region paths defined. If a region `<path>`'s `strokeId` was already found to be selected, the region PiP test is omitted. A faster point-in-bounding-box test shortcuts the more costly PiP test. The `isPointInsidePath` method provided by the version of the underlying Raphaël library used is capable of handling Bézier splines but produces incorrect PiP results unfortunately (figure 4.113). Therefore, a correct PiP implementation optimized for linear line segments was added.

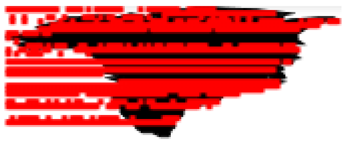


Figure 4.113.: Raphaël's original `isPointInsidePath` utility method does not work as expected, as shown in this test case, taken from (Martins, 2013). It illustrates `isPointInsidePath` results (red squares) when sampling a closed path (black shape).

[93]Multiple region paths may be defined for each `InputFieldSelectionSetValue`. When creating a region using the UI, a UUID is generated for the `InputFieldSelectionSetValue.strokesId` entity field. The sketch control initializes each `<path>` element's non-standard `strokeId` attribute with the UUID value.

Summarizing, the client-side implementation of the custom sketch control consists of:

- modified Raphaël, Raphael SketchPad and Really Simple Color Picker Js libraries
- `sketch.js` containing methods for initialisation, toolbar control and the `change` event callback to wire the region selection algorithm
- a `table` containing `<div>`s of toolbar elements
- a `<div>` element for the `<svg>` container element
- a hidden `<input>` element for holding a stringified JSON data structure, consisting of an array of SVG stroke paths and an array of stroke IDs of selected regions
- CSS style classes for frames and background colors, enabled and disabled state of toolbar buttons and icons

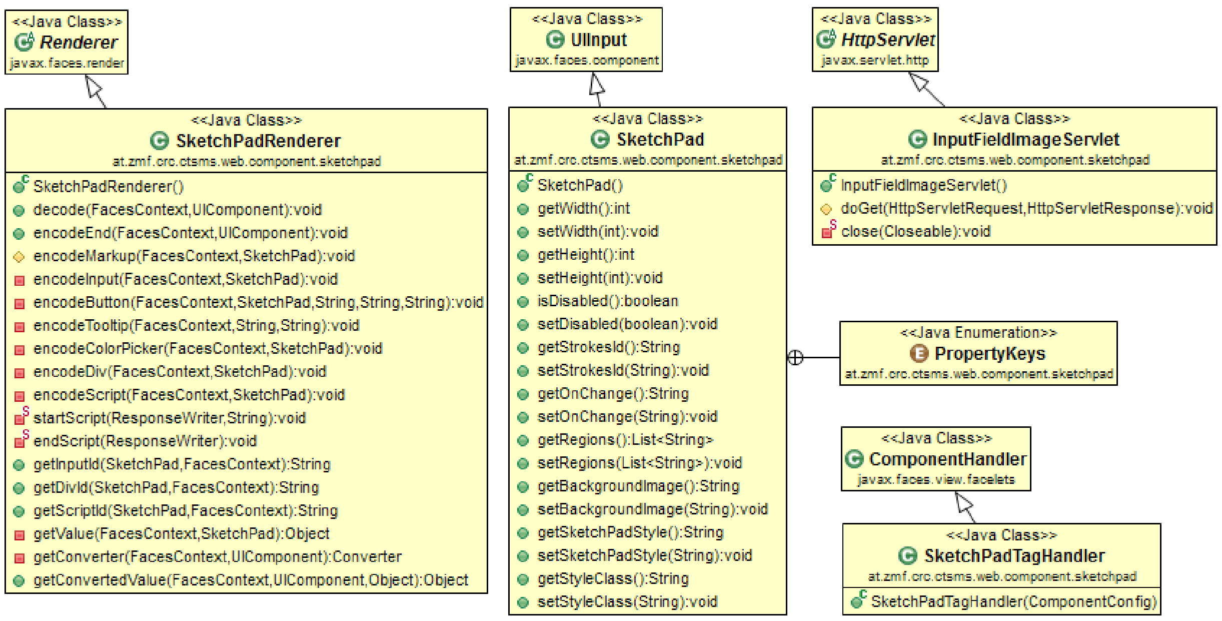
```html
<span id="tabView:inputfielddummy_form:sketchpaddummy" class="sketch_container"> <!-- enclosing span element -->
    <!-- for AJAX POST requests, the hidden form field to holds sketch data and region selection in myacjson format: -->
    <input id="tabView:inputfielddummy_form:sketchpaddummy_input" type="hidden" value="[...]">
    <!-- toolbar table: -->
    <table class="sketch-toolbar-table">
        <tbody><tr> <!-- button event handler callbacks are attached during widget initialisation using jQuery: -->
            <!-- 1. region toggle button: -->
            <td><div id="tabView:inputfielddummy_form:sketchpaddummy_div_regionToggler" class="sketch-region-toggler-on"></div>
                 </td>
            <!-- 2. draw mode switch button: pencil (capture new strokes)/eraser (remove existing strokes by clicking) -->
            ...
        </tr></tbody>
    </table>
    <!-- svg container div: -->
    <div id="tabView:inputfielddummy_form:sketchpaddummy_div" class="sketchpad" style="width: 120px; height: 40px; cursor:
          crosshair; background-image: url("/inputfieldimage?inputfieldid=12345");">
        <svg height="40" version="1.1" width="120" xmlns="http://www.w3.org/2000/svg" ...>
            <!-- reflected svg markup -->
        </svg>
    </div>
</span>
```

Listing 4.54: This HTML fragment shows the markup rendered to display a sketch control in the browser. The toolbar consists of a table of `<div>` elements representing toolbar controls like buttons. Raphael SketchPad manipulates the container `<div id="..._div>"` element's `<svg>` child element, causing the browser to update the rendered output instantly.

The sketch background image is *not* embedded using the SVG `<image>` raster graphics element. Instead, an uploaded image file (figure 4.110) is persisted separately for the input field in the `InputField.data` entity field. The `<svg>` container's surrounding `<div>` (listing 4.54) gets a dynamic CSS `background-image` `style` attribute to load the sketch background image using the `InputField`'s ID. The browsers' resulting GET HTTP requests for retrieving the images are served by a separate `HttpServlet` (`InputFieldImageServlet`).

4.8.4.3. Sketch Control JSF Component

For an easy placement of the sketch controls in facelets, a custom JSF component (`ctsms:sketchPad`) was implemented. In order to integrate with the JSF infrastructure including the PPR lifecycle, a custom JSF component implementation requires to follow a strict structure, which was adopted from PrimeFaces components. According to this structure, the sketch control component implementation consists of several elements (figure 4.114). `SketchPadRenderer` encapsulates rendering of the HTML markup for displaying the sketch control on a page, as described in the previous section. The HTML fragment will additionally contain required *inline* Js code for initializing the sketch control, which is executed by the browser when a PPR cycle completes.



<<Java Class>>
Renderer
javax.faces.render
<<Java Class>>
UIInput
javax.faces.component
<<Java Class>>
HttpServlet
javax.servlet.http
<<Java Class>>
SketchPadRenderer
at.zmf.crc.ctsms.web.component.sketchpad
SketchPadRenderer()
decode(FacesContext,UIComponent):void
encodeEnd(FacesContext,UIComponent):void
encodeMarkup(FacesContext,SketchPad):void
encodeInput(FacesContext,SketchPad):void
encodeButton(FacesContext,SketchPad,String,String,String):void
encodeTooltip(FacesContext,String,String):void
encodeColorPicker(FacesContext,SketchPad):void
encodeDiv(FacesContext,SketchPad):void
encodeScript(FacesContext,SketchPad):void
startScript(ResponseWriter,String):void
endScript(ResponseWriter):void
getInputId(SketchPad,FacesContext):String
getDivId(SketchPad,FacesContext):String
getScriptId(SketchPad,FacesContext):String
getValue(FacesContext,SketchPad):Object
getConverter(FacesContext,UIComponent):Converter
getConvertedValue(FacesContext,UIComponent,Object):Object
<<Java Class>>
SketchPad
at.zmf.crc.ctsms.web.component.sketchpad
SketchPad()
getWidth():int
setWidth(int):void
getHeight():int
setHeight(int):void
isDisabled():boolean
setDisabled(boolean):void
getStrokesId():String
setStrokesId(String):void
getOnChange():String
setOnChange(String):void
getRegions():List<String>
setRegions(List<String>):void
getBackgroundImage():String
setBackgroundImage(String):void
getSketchPadStyle():String
setSketchPadStyle(String):void
getStyleClass():String
setStyleClass(String):void
<<Java Class>>
InputFieldImageServlet
at.zmf.crc.ctsms.web.component.sketchpad
InputFieldImageServlet()
doGet(HttpServletRequest,HttpServletResponse):void
close(Closeable):void
<<Java Enumeration>>
PropertyKeys
at.zmf.crc.ctsms.web.component.sketchpad
<<Java Class>>
ComponentHandler
javax.faces.view.facelets
<<Java Class>>
SketchPadTagHandler
at.zmf.crc.ctsms.web.component.sketchpad
SketchPadTagHandler(ComponentConfig)

Figure 4.114.: Classes of the Sketch JSF component implementation.

Facelets can place a sketch component instance using the `ctsms:sketchPad` tag (listing 4.55). As with any other JSF component, EL can be used to bind tag attribute values to managed bean fields.

```
<ctsms:sketchPad id="someId"
    value="#{someSketchBean.inkValueWithSelections}" <!-- string value from listing 4.57 -->
    <!-- If a sketch background image is defined, sketch width and height should match the background image's dimensions.
         Otherwise the background's extent is clipped or blank at the bottom/right. -->
    width="#{someSketchBean.width}"
    height="#{someSketchBean.height}"
    regions="#{someSketchBean.inkRegions}" <!-- string value from listing 4.56 -->
    disabled="#{someSketchBean.disabled}" <!-- a disabled sketch control is greyed out and prohibits drawing strokes -->
    <!-- The <svg> container's surrounding <div> CSS background-image style attribute is can be set to show a user-defined
         sketch background. To use it for displaying the background image specified for the input field, the URL of the
         background image serlvet is specified: -->
    backgroundImage="#{request.contextPath}/inputfieldimage?#{applicationScopeBean.parameterName('INPUT_FIELD_ID')}=#{
         someSketchBean.inputField.id}"
    onChange="someJSCallbackMethod" /> <!-- A custom onChange callback can be registered, as known from other JSF components.
         For example, it is used to update field value calculations instantly with every single stroke drawn. -->
```

Listing 4.55: This facelet fragment demonstrates how to place a sketch control with the `ctsms:sketchPad` tag.

The `ctsms:sketchPad` tag element's attributes are defined by the `SketchPad` class. This state value holder (`java.faces.component.UIInput`) is the server-side representation of a sketch component instance, which will be registered in the JSF component tree. The component is configured for the region edit mode if the `strokeId` attribute is set. For regular sketching mode, region SVG paths can be specified in JSON format (listing 4.56) using the tag element's `region` attribute.

```
[
   {
      "fill":"#ff6900",
      "stroke":"#ff6900",
      "path":"M11,10L30,10L31,30L10,31Z",
      "stroke-opacity":0.4,
      "stroke-width":2,
      "stroke-linecap":"round",
      "stroke-linejoin":"round",
      "type":"path",
      "fill-opacity":0.2,
      "strokes-id":"bf5bd8e7-27f8-4d49-9ca5-ac0bf8098ee5"
   }
]
```

Listing 4.56: Region SVG paths are stored in the JSON format shown.

```
[
   {
      "stroke":"#3c55ee",
      "fill":"none",
      "stroke-linecap":"round",
      "path":[
         [ "M",4,8 ],
         [ "L",35,31 ]
      ],
      "stroke-linejoin":"round",
      "stroke-opacity":1,
      "type":"path",
      "stroke-width":3
   },
   {
      "stroke":"#3c55ee",
      "fill":"none",
      "stroke-linecap":"round",
      "path":[
         [ "M",30,8 ],
         [ "L",7, 31 ]
      ],
      "stroke-linejoin":"round",
      "stroke-opacity":1,
      "type":"path",
      "stroke-width":3
   },
   "bf5bd8e7-27f8-4d49-9ca5-ac0bf8098ee6,..."
]
```

Listing 4.57: Region selection and stroke SVG paths of a sketch are stored in the JSON format shown.

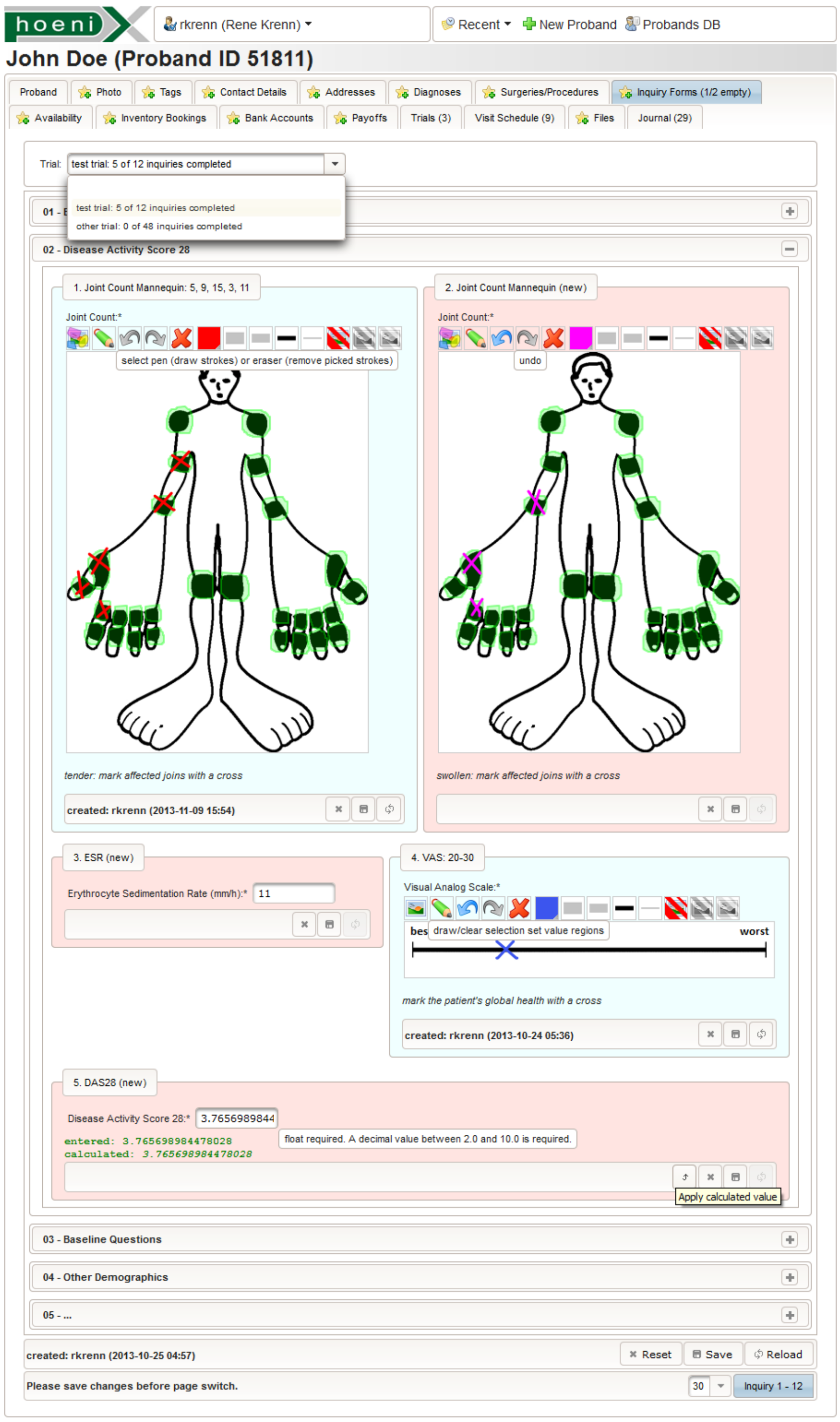
Managed beans are supposed to provide a *string* field for the tag's `value` attribute[94] in order to pass both stroke data and region selection. The JSON data structure shown in listing 4.57 combines an array of stroke paths and a string containing a comma-separated list of `strokeId` UUID values of selected regions. In order to persist `InputFieldValue.selectionValues`, `InputFieldSelectionSetValue` items of corresponding regions can be looked up by the `strokeId` key. Handling the JSON format and the mapping of `strokeIds` to `InputFieldSelectionSetValue.ids` is provided by separate web tier utility methods, but could be integrated into the JSF component using a tailored data model class in a future version.

4.8.4.4. Application Example: DAS28

The implementation of the sketch input field type is simple but turned out flexible enough for the problem domain. It finally allows database queries against captured data, which is coded by selecting background image areas. Users can literally mark a region to select it, in the intuitive way of drawing a ckeckmark. However, performance drops with a large number of regions in a single sketch or when using System-on-a-Chip (SoC)-based mobile platform hardware[95]. Aside the performance-related refactoring, future enhancements could comprise an additional region selection mode, which supports selecting multiple areas at once by drawing an enclosing path.

As a concluding demonstration, the DAS28 parameter is considered again, which could be relevant in the context of recruitment for a trial related to rheumatoid arthritis. The Phoenix CTMS allows users to design a complete input form for capturing the DAS28 parameter (figure 4.115) by combining sketch input fields and a calculated field value.

[94]During a JSF request lifecycle, the component instance's value is stored and retrieved by `UIInput.setValue` and `getValue`.
[95]e.g. ARM- or Intel Atom-based devices.



John Doe (Proband ID 51811)

Trial: test trial: 5 of 12 inquiries completed

test trial: 5 of 12 inquiries completed
other trial: 0 of 48 inquiries completed

01 - E

02 - Disease Activity Score 28

1. Joint Count Mannequin: 5, 9, 15, 3, 11

Joint Count:*

tender: mark affected joins with a cross

created: rkrenn (2013-11-09 15:54)

2. Joint Count Mannequin (new)

Joint Count:*

swollen: mark affected joins with a cross

3. ESR (new)

Erythrocyte Sedimentation Rate (mm/h):* 11

4. VAS: 20-30

Visual Analog Scale:*

mark the patient's global health with a cross

created: rkrenn (2013-10-24 05:36)

5. DAS28 (new)

Disease Activity Score 28:* 3.7656989844

float required. A decimal value between 2.0 and 10.0 is required.

```
entered:  3.765698984478028
calculated:  3.765698984478028
```

03 - Baseline Questions

04 - Other Demographics

05 - ...

created: rkrenn (2013-10-25 04:57)

Please save changes before page switch.

Figure 4.115.: The input form section shown groups inquiry input fields for capturing the DAS28 parameter, as outlined by the expression tree example from figure 4.99. The "tender" and "swollen" joint counts are documented and determined by marking affected joints in a sketch input field with a mannequin background image. The proband could fill in the VAS input field himself when asked for the subjective assessment of recent disease activity. The regions of the VAS input field can therefore be hidden in this case. The resulting DAS28 value is persisted using an input field for decimal numbers. Its value is computed by the value expression from listing 4.40. Whenever a stroke is drawn on one of the mannequin diagram sketches, the DAS28 value is recalculated and displayed.

4.9. Query Designer

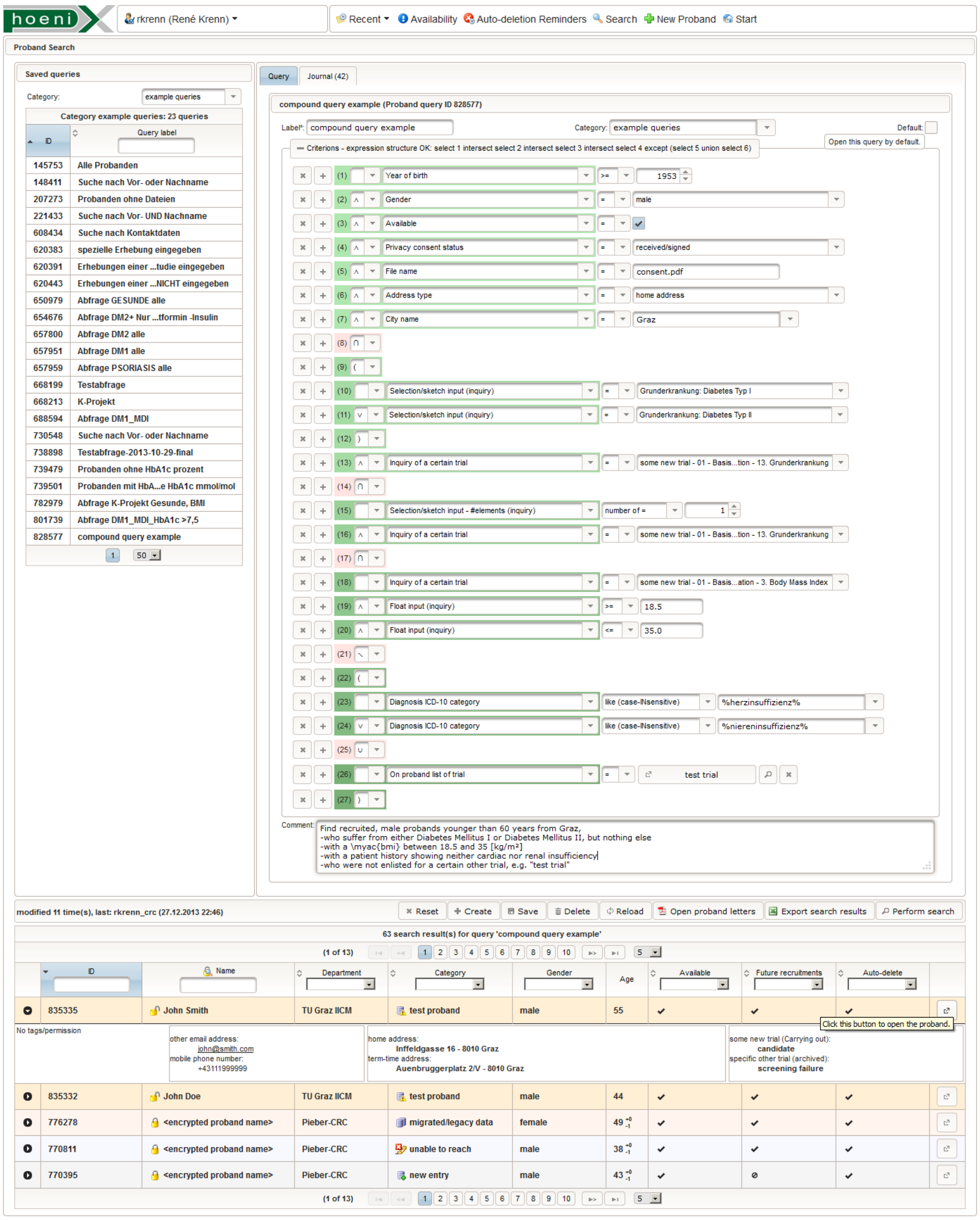
Functionality indispensable to enterprise applications is searching the database. Users should be able to run queries *instantly* against any of the Phoenix CTMS database modules in order to look up records explicitly and overview (total) sets of existing records. As a top level design decision, a query should list a module's *root entity* records only, but not its detail entity records. However, any detail entity field of the entire domain model can serve as a designated query criterion. Surveying particular detail records is supported separately by several dedicated UI views, which were presented in the previous sections:

- System Modules Overview - tag clouds (section 4.6.3.1)
- Notification messages (section 4.6.3.2)
- Calendar Views (section 4.6.5)
- Timeline View (section 4.6.6)
- Overview Lists (section 4.6.7)

In order to perform dynamic database queries, an UI for entering a number of user-defined search criterion expressions is desired instead of static search masks with hard-wired search criteria. As shown in subsequent sections, this comes down to the implementation of a *query editor* for entering search criteria in a structure basically known from SQL statements. This concept will encounter the following basic challenges:

- handling a large number of detail entity fields available to queries
- extendable query infrastructure by means of configuration

To generalize implementation and facilitate a uniform UI appearance and usage across system modules, each system module gets its own query editor view instance with a tailored setup. Particular attention is paid to the proband's module query editor (figure 4.116), since it provides the representation of subject eligibility criteria. Performing a query with eligibility criteria implies to run database queries against EAV data model tables (section 4.1.1.3), which in turn has crucial implementation impacts.



Proband Search

Saved queries

Category: example queries

Category example queries: 23 queries

ID	Query label
145753	Alle Probanden
148411	Suche nach Vor- oder Nachname
207273	Probanden ohne Dateien
221433	Suche nach Vor- UND Nachname
608434	Suche nach Kontaktdaten
620383	spezielle Erhebung eingegeben
620391	Erhebungen einer ...tudie eingegeben
620443	Erhebungen einer ...NICHT eingegeben
650979	Abfrage GESUNDE alle
654676	Abfrage DM2+ Nur ...tformin -Insulin
657800	Abfrage DM2 alle
657951	Abfrage DM1 alle
657959	Abfrage PSORIASIS alle
668199	Testabfrage
668213	K-Projekt
688594	Abfrage DM1_MDI
730548	Suche nach Vor- oder Nachname
738898	Testabfrage-2013-10-29-final
739479	Probanden ohne HbA1c prozent
739501	Probanden mit HbA...e HbA1c mmol/mol
782979	Abfrage K-Projekt Gesunde, BMI
801739	Abfrage DM1_MDI_HbA1c >7,5
828577	compound query example

1 50

compound query example (Proband query ID 828577)

Label*: compound query example Category: example queries Default: Open this query by default.

Criterions - expression structure OK: select 1 intersect select 2 intersect select 3 intersect select 4 except (select 5 union select 6)

(1)		Year of birth	>=	1953
(2)	∧	Gender	=	male
(3)	∧	Available	=	✓
(4)	∧	Privacy consent status	=	received/signed
(5)	∧	File name	=	consent.pdf
(6)	∧	Address type	=	home address
(7)	∧	City name	=	Graz
(8)	∩			
(9)	(			
(10)		Selection/sketch input (inquiry)	=	Grunderkrankung: Diabetes Typ I
(11)	∨	Selection/sketch input (inquiry)	=	Grunderkrankung: Diabetes Typ II
(12)	)			
(13)	∧	Inquiry of a certain trial	=	some new trial - 01 - Basis...tion - 13. Grunderkrankung
(14)	∩			
(15)		Selection/sketch input - #elements (inquiry)	number of =	1
(16)	∧	Inquiry of a certain trial	=	some new trial - 01 - Basis...tion - 13. Grunderkrankung
(17)	∩			
(18)		Inquiry of a certain trial	=	some new trial - 01 - Basis...ation - 3. Body Mass Index
(19)	∧	Float input (inquiry)	>=	18.5
(20)	∧	Float input (inquiry)	<=	35.0
(21)	\			
(22)	(			
(23)		Diagnosis ICD-10 category	like (case-INsensitive)	%herzinsuffizienz%
(24)	∨	Diagnosis ICD-10 category	like (case-INsensitive)	%niereninsuffizienz%
(25)	∪			
(26)		On proband list of trial	=	test trial
(27)	)			

Comment:
Find recruited, male probands younger than 60 years from Graz,
-who suffer from either Diabetes Mellitus I or Diabetes Mellitus II, but nothing else
-with a \myac{bmi} between 18.5 and 35 [kg/m²]
-with a patient history showing neither cardiac nor renal insufficiency
-who were not enlisted for a certain other trial, e.g. "test trial"

modified 11 time(s), last: rkrenn_crc (27.12.2013 22:46)

63 search result(s) for query 'compound query example'

(1 of 13) 1 2 3 4 5 6 7 8 9 10 5

	ID	Name	Department	Category	Gender	Age	Available	Future recruitments	Auto-delete
	835335	John Smith	TU Graz IICM	test proband	male	55	✓	✓	✓
No tags/permission		other email address: john@smith.com mobile phone number: +43111999999		home address: Inffeldgasse 16 - 8010 Graz term-time address: Auenbruggerplatz 2/V - 8010 Graz			some new trial (Carrying out): candidate specific other trial (archived): screening failure		
	835332	John Doe	TU Graz IICM	test proband	male	44	✓	✓	✓
	776278	<encrypted proband name>	Pieber-CRC	migrated/legacy data	female	49 $^{+0}_{-1}$	✓	✓	✓
	770811	<encrypted proband name>	Pieber-CRC	unable to reach	male	38 $^{+0}_{-1}$	✓	✓	✓
	770395	<encrypted proband name>	Pieber-CRC	new entry	male	43 $^{+0}_{-1}$	✓	⊘	✓

Click this button to open the proband.

(1 of 13) 1 2 3 4 5 6 7 8 9 10 5

Figure 4.116.: The query editor UI is common to all system modules. It allows composing database queries by building a query expression block-wise. Queries can be loaded and executed to show a paginated result list below or export results in Excel format. PDF export is available for document types with mail merge support (section 4.4.1). Privileged users can persist queries and organize them into category folders. The proband module example query shown expresses some minimalistic subject eligibility criteria to list a result set of potential subject candidates and will be examined in subsequent sections.

4.9.1. Query Model

The fundamental expression power of queries is determined by capabilities of SQL. SQL database queries represent *relational tuple calculus expressions* described for example in (Kemper and Eickler, 2011, pp. 99). They define how to construct result set of tuples according to predicates for tuple attributes. For *safe* tuple calculus expressions, intermediate and result sets are limited by the value domain specified for the expression. The value domain is determined by the actual content of involved database tables.

Queries will effectively be performed by means of Hibernate (HQL/Criteria API) and the RDBMS (SQL) respectively. For a doable implementation with respect to expected use cases, the following common SQL language features available to `SELECT` statements are restricted:

- aggregate functions[96]
- `GROUP BY/HAVING` clause
- subqueries[96]
- projection
- condition terms of the form `<column> <comparison operator> <constant value>` only

In order to support managing and editing database queries, a data model capable of persisting a query definition is required as a starting point. The structure of queries is analyzed to elaborate a suitable model.

4.9.1.1. Criterion, Query, Compound Query

To retrieve the query result list, a HQL `select` statement is run against the root entity table (e.g. the `Proband` table). A *criterion* represents a basic condition term c of the statement's `where` clause, defined by an entity column $p \in P$, a comparison operator $r \in R_p$ and a literal comparison value v:

$$c := (p, r, v)$$

The set P of available entity fields relevant to a system module represents master data. Each imported entity field p references a set R_p of applicable comparison operators (predicates). Data types of p and v have to match, unless v is omitted for unary operators r (e.g. `IS NULL`, `>= TODAY()`).

Multiple criterion terms are combined by Boolean `and` ($\wedge$) and `or` ($\vee$) operators to form a *query*. A record is part of the result list only if its column values satisfy the entire query criteria expression. As usual, the $\wedge$ operator has a higher precedence than $\vee$. Nevertheless users need not enter expanded query criteria, since the order of criterion evaluation can optionally be controlled using parentheses.

If the query involves detail entity's columns, the detail entity table has to be joined. Any type of SQL table join of two tables M and N gives a subset of rows from the Cartesian product $M \times N$. This virtual *join table* therefore contains columns of both joined tables. The HQL query statement flattens join table columns again by projecting them onto the root entity columns (listing 4.58 line 2). As a consequence, the SQL join of a master-detail association will inherently cause duplicate root entity records in the result set. Since these are not desired in result list presented to users, they are removed using the `distinct` keyword (listing 4.58 line 1). This also resolves the problem of temporal coalescing (Zhou, F. Wang, and Zaniolo, 2006), coming from detail entities representing a period of time. The result list can therefore always be considered as a set in strict mathematical sense. As a downside, result set sorting by columns not part of the effective SQL `SELECT` statement's projection column list is not possible.

[96]The HQL `size(x) = y` construct is supported, which generates a correlated subquery: `(SELECT COUNT(x1.id) FROM x_table AS x1 WHERE x1.id = x.id) = y`

```
 1  select distinct //no duplicate records
 2  proband from at.zmf.crc.ctsms.ProbandImpl as proband //project result set onto root entity columns
 3         left join particulars as _0 //one-to-one
 4         left join inquiryValues as _1 //one-to-many - not supported by Hibernate's implicit joins
 5         left join _1.inquiry as _2 //many-to-one
 6         left join _1.value as _3 //one-to-one
 7  where
 8             proband.available = true //a criterion term of a master entity primitive field requires no join
 9         and _0.gender = Sex.MALE //alternatively, proband.particulars.gender = Sex.Male resolves to an inner join
10         and _2.id = 1233 //narrow result set to inquiry "Körpergröße" ...
11         and _3.longValue >= 180 //... with captured values of greater than or equal to 180 cm
```

Listing 4.58: This example shows a basic HQL `select` statement with a `where` clause containing criterion terms combined using Boolean operators. Tables of referred detail entity tables are left-joined.

HQL extends the syntax of SQL by accepting association path notations to a limited extent (King et al., 2004, chapter 14). To utilize association paths, relevant entity associations have to be named and navigatable. This was therefore considered for entity associations throughout the system's domain model (section 4.1.2) in order to avoid limitations when preparing definitions of entity columns available to criterion terms. Hibernate generates implicit joins (SQL `INNER JOIN`) for entity columns expressed by an association path when converting the HQL query into SQL. Unfortunately, this is supported for x-to-one associations (e.g. `Proband.particulars`) only, but not for collection-valued fields of x-to-many associations known from master-detail relationships. Since the idea of consistently identifying an entity column p using its association path is ideal for a data model holding query criteria, it was decided to overcome this limitation. A preprocessing step transforms *any* association path into `LEFT JOIN`s (listing 4.58 line 3-6). By using *left* joins, a criterion comparing column values with a `NULL` value will also match non-existent detail records.

```
1  distinct proband from at.zmf.crc.ctsms.ProbandImpl as proband where
2             proband.available = true
3         and proband.particulars.gender = Sex.MALE
4         and proband.inquiryValues.inquiry.id = 1233
5         and proband.inquiryValues.value.longValue >= 180
```

Listing 4.59: The HQL statement from listing 4.58 is abbreviated by introducing association paths. However, association paths covering x-to-many associations (*italic font*) are not supported by Hibernate.

After introducing unrestricted association paths, a HQL query can be written in shorter form shown in listing 4.59. The HQL artefact denoted by listing 4.59 line 4-5 shows how to define criteria for filtering by a single captured inquiry value ($c_A : \textit{body size} \geq 180[cm]$). Assuming another inquiry question for body weight (e.g. `Inquiry.id` $= 1232$) and another criteria $c_B : \textit{body weight} \geq 90[kg]$, the Boolean combination $c_A \vee c_B$ will look up subjects satisfying c_A *or* c_B or both. The result set would be equal the *union* set $A \cup B$ of probands A satisfying c_A and probands B satisfying c_B. However, using $c_A \wedge c_B$ respectively for finding probands satisfying both c_A *and* c_B cannot work. Since a join table row can never satisfy both criterion terms `proband.inquiryValues.inquiry.id` $= 1232 \wedge$ `proband.inquiryValues.inquiry.id` $= 1233$, the result set would be empty. The correct result is given by the *intersection* set $A \cap B$. This nature of database queries performed against EAV data model tables was the crucial reason for the decision to implement *set operation* support. Although specified by SQL-92, set operations are not necessarily supported by RDBMS[97] and not supported by Hibernate either. Therefore, the implementation comes up with a resourceful algorithm to circumvent Hibernate's shortcomings regarding set operations, described in detail in section 4.9.2.2.

[97]For example, MySQL supports `UNION` only.A *self-join* query could be used as a workaround for set intersections though. Difference sets can be expressed by the identity $A \setminus B \equiv A \cap \overline{B}$. The complement $\overline{B}$ results from inverting the condition specified by a `WHERE` clause.

operator	SQL	precedence	commutative	description
$\wedge$	`and`	4	✓	$c_1 \wedge c_2$: put a row on the query result list, if it satisfies both criterion terms c_1 *and* c_2
$\vee$	`or`	3	✓	$c_1 \vee c_2$: put a row on the result list, if it satisfies either c_1 or c_2 or both
$\cap$	`intersect`	2	✓	$A \cap B$: the compound query result list contains rows which appear in both result lists A and B only
$\cup$	`union`	1	✓	$A \cup B$: the result list contains rows which appear in either A or B, or both
$\setminus$	`except`	1	✗	$A \setminus B$: the result list contains rows which appear in A, but not B

Table 4.16.: Query criterion operators and their precedence.

Multiple criterion terms can be combined by `union` ($\cup$), `intersect` ($\cap$) and `except` ($\setminus$) operators to form a *compound query* (table 4.16). The $\cap$ operator has a higher precedence than $\cup$ and $\setminus$, which both have the same precedence (left-associative). The evaluation order of set operations can be controlled using parentheses. Aside the requirement in the context of the EAV data model, set operations allow to express queries with eligibility criteria in an elegant way, which showed up as a concept intuitive to users (figure 4.117). A more complex compound query example is given for demonstration:

Find recruited, male probands younger than 60 years from Graz (listing 4.60 line 2-11),

- *who suffer from either Diabetes Mellitus I or Diabetes Mellitus II, but nothing else (line 13-27)*
- *with a BMI between* 18.5 *and* $35kg/m^2$ *(line 29-33)*
- *with a patient history showing neither cardiac nor renal insufficiency (line 36-42)*
- *who were not enrolled for a certain other trial, e.g. "test trial" (line 44-46)*

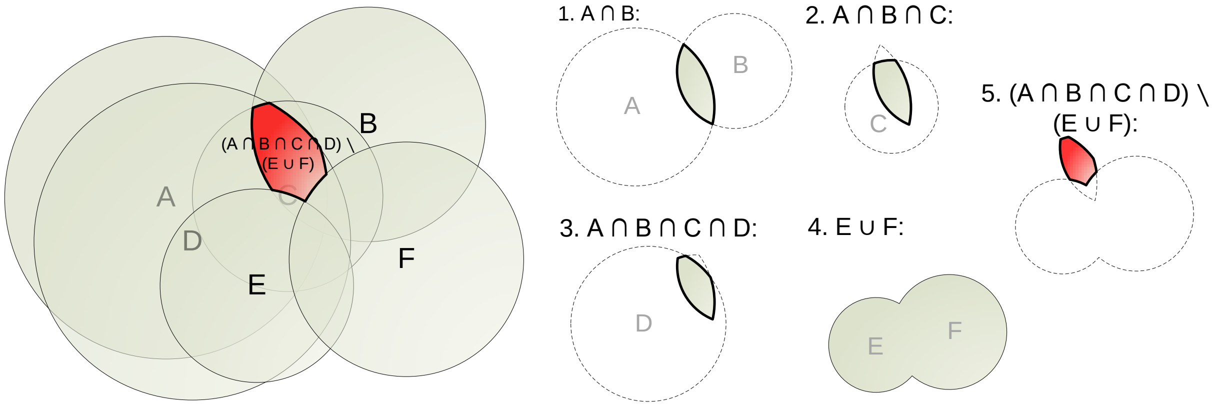


Figure 4.117.: The construction of the compound query example from listing 4.60 can be visualized using *Venn diagrams*.

```
(//compound proband query example:
    distinct proband from at.zmf.crc.ctsms.ProbandImpl as proband where //query A: subjects with ...
           proband.particulars.yearOfBirth >= 1953 //(c1) ... a max. age of 60 years as of 2013-01-01
       and proband.particulars.gender = Sex.MALE //(c2) ... male gender
       and proband.available = true //(c3) ... available flag set to true
       and proband.privacyConsentStatus.id = 123 //(c4) ... privacy consent state set to "received/signed"
       and proband.files.fileNameHash = md5("consent.pdf") //(c5) ... a certain PDF document attachment
       and proband.addresses.type.id = 456 //(c6) ... at least a single "home addess" detail record, and the
       and proband.addresses.cityNameHash = md5("Graz") //(c7) ... "home address" city name is "Graz"
       //optional datatable filter terms F, e.g.
       //and proband.particulars.firstNameHash = md5("John")
intersect
    distinct proband from at.zmf.crc.ctsms.ProbandImpl as proband where //query B: subjects with ...
       (
               proband.inquiryValues.value.selectionValues.id = 54321 //(c8) ... "Diabetes Typ I" plus/or
           or  proband.inquiryValues.value.selectionValues.id = 54322 //(c9) ... "Diabetes Typ II" checked
       )
       and proband.inquiryValues.inquiry.id = 1234 //(c10) for the a multiple selection inquiry "Grunderkrankung"
       //optional datatable filter terms F
intersect
    distinct proband from at.zmf.crc.ctsms.ProbandImpl as proband where //query C: subjects where ...
           size(proband.inquiryValues.value.selectionValues) = 1 //(c11) ... a single value is checked only from
       and proband.inquiryValues.inquiry.id = 1234 //(c12) available "Grunderkrankung" selection values:
                //"Gesund", "Diabetes Typ I", "Diabetes Typ II", "Psoriasis", etc.
       //optional datatable filter terms F
       //query B intersect query C will list subjects with nothing but *either* "Diabetes Typ I" *or*
       //"Diabetes Typ II" checked
intersect
    distinct proband from at.zmf.crc.ctsms.ProbandImpl as proband where //query D: subjects with ...
           proband.inquiryValues.inquiry.id = 1235 //(c13) ... a BMI value in a range from
       and proband.inquiryValues.value.floatValue >= 18.5 //(c14) 18.5 kg/m² to
       and proband.inquiryValues.value.floatValue <= 35.0 //(c15) 35.0 kg/m²
       //optional datatable filter terms F
except
    (
            distinct proband from at.zmf.crc.ctsms.ProbandImpl as proband where //query E: subjects with one or
                        //more diagnoses of a ICD-10 category containing ...
                   proband.diagnoses.code.systematics.categories.preferredRubricLabel ilike =
                        "%herzinsuffizienz%" //(c16) the word "herzinsuffizienz" or
               or  proband.diagnoses.code.systematics.categories.preferredRubricLabel ilike =
                        "%niereninsuffizienz%" //(c17) "niereninsuffizienz"
               //optional datatable filter terms F
        union
            distinct proband from at.zmf.crc.ctsms.ProbandImpl as proband where //query F: subjects, ...
                   proband.trialParticipations.trial.id = 12345 //(c18) ... who participated "test trial"
               //optional datatable filter terms F
    )
)
//optional datatable sorting by root entity field, e.g.: order by department
//optional datatable pagination by root entity field, e.g.: limit 0, 20
```

Listing 4.60: This listing shows the abbreviated HQL query of the compound query example shown in figure 4.116.

When hiding obligatory parts from the HQL statements of the compound query example from listing 4.60, the query structure can be depicted by the syntax tree shown in figure 4.118. Likewise, the query can be denoted by an an infix expression:

$$c_1 \wedge c_2 \wedge c_3 \wedge c_4 \wedge c_5 \wedge c_6 \wedge c_7 \cap (c_8 \vee c_9) \wedge c_{10} \cap c_1 \wedge c_{12} \cap c_{13} \wedge c_{14} \wedge c_{15} \setminus (c_{16} \vee c_{17} \cup c_{18})$$

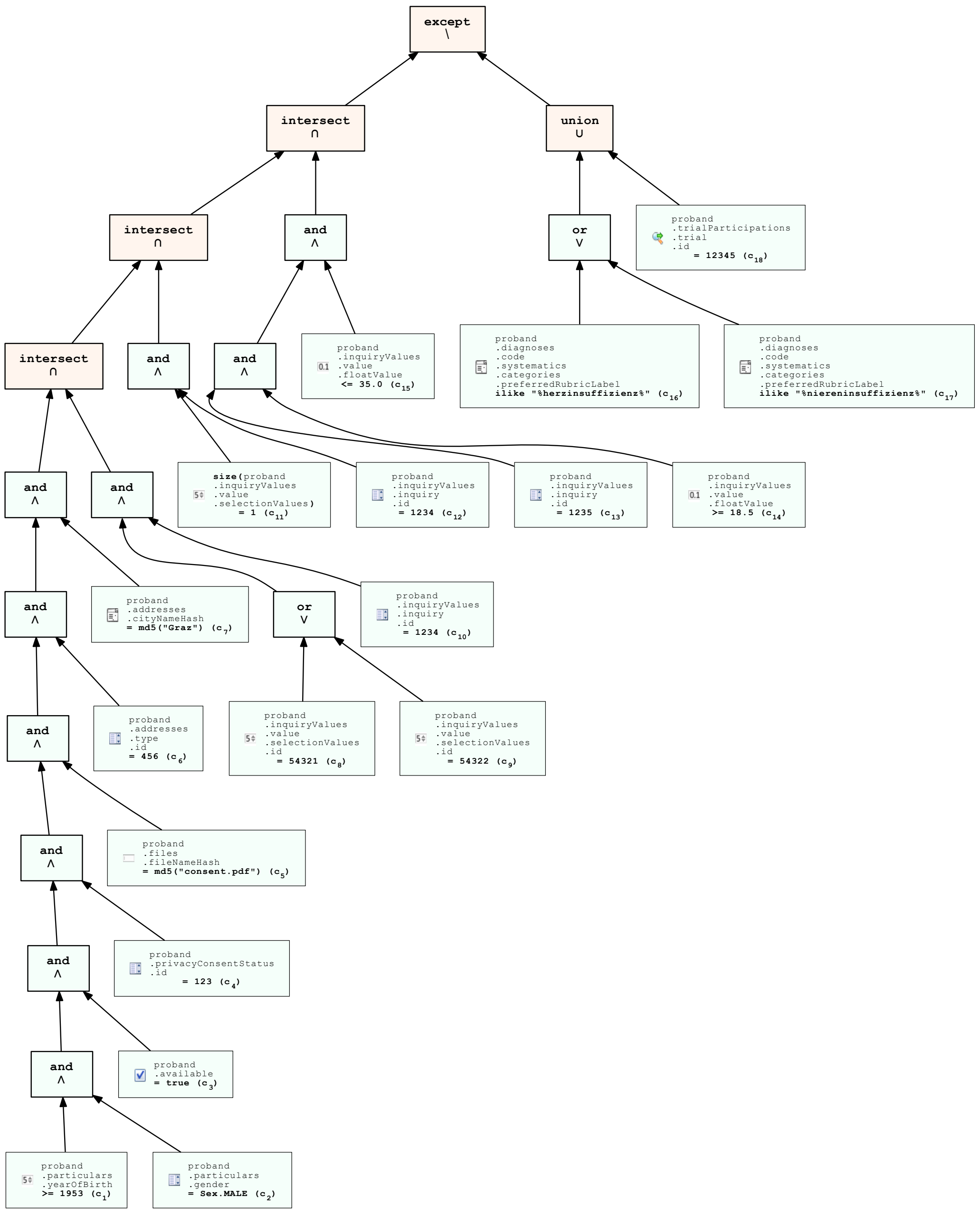
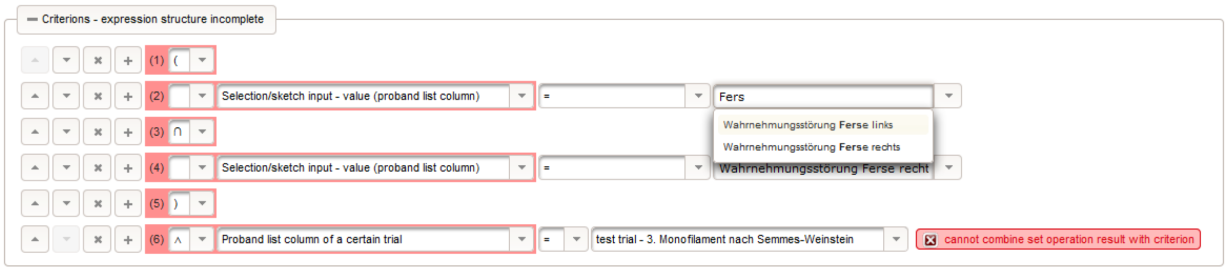

except \
intersect ∩
union ∪
intersect ∩
and ∧
or ∨
proband .trialParticipations .trial .id = 12345 (c18)
intersect ∩
and ∧
and ∧
proband .inquiryValues .value .floatValue <= 35.0 (c15)
proband .diagnoses .code .systematics .categories .preferredRubricLabel ilike "%herzinsuffizienz%" (c16)
proband .diagnoses .code .systematics .categories .preferredRubricLabel ilike "%niereninsuffizienz%" (c17)
and ∧
and ∧
size(proband .inquiryValues .value .selectionValues) = 1 (c11)
proband .inquiryValues .inquiry .id = 1234 (c12)
proband .inquiryValues .inquiry .id = 1235 (c13)
proband .inquiryValues .value .floatValue >= 18.5 (c14)
and ∧
proband .addresses .cityNameHash = md5("Graz") (c7)
or ∨
proband .inquiryValues .inquiry .id = 1234 (c10)
and ∧
proband .addresses .type .id = 456 (c6)
proband .inquiryValues .value .selectionValues .id = 54321 (c8)
proband .inquiryValues .value .selectionValues .id = 54322 (c9)
and ∧
proband .files .fileNameHash = md5("consent.pdf") (c5)
and ∧
proband .privacyConsentStatus .id = 123 (c4)
and ∧
proband .available = true (c3)
proband .particulars .yearOfBirth >= 1953 (c1)
proband .particulars .gender = Sex.MALE (c2)

Figure 4.118.: The query expression's syntax tree elimiantes parentheses while preserving the correct order of operations.

Expression token types comprise criterion, ∧, ∨, ∩, ∪, \, left parenthesis (and right parenthesis). Boolean operators have a higher precedence than set operators (table 4.16), thus set operators delimit basic queries within a compound query. By supporting parentheses for grouping, query expressions are basically prone to *semantic errors*, because there are two operand types (criterion terms C and result sets S) and a limitation of supported constructs:

- Boolean operators: $C \times C \mapsto C$
- set operators: $C \times S \mapsto S$, $C \times C \mapsto S$, $S \times S \mapsto S$, $S \times C \mapsto S$

Since Boolean operators cannot accept a result set operand, it is possible to compose malformed expressions. The trivial solution to this situation is `trying` to simply ignore it in application code and just propagate the Hibernate/RDBMS error message. The average user who is already faced with the challenging task of composing queries will not be able to identify the problem cause from this message, which will refer application logic internals. Known from applications with DSL editors, syntax checking features are a preferred but expensive alternative in this case. Nevertheless, a custom *recognizer* was developed to add these features to the query editor UI (figure 4.119).


Criterions - expression structure incomplete

(1) (

(2) Selection/sketch input - value (proband list column) = Fers

Wahrnehmungsstörung Ferse links

Wahrnehmungsstörung Ferse rechts

(3) ∩

(4) Selection/sketch input - value (proband list column) = Wahrnehmungsstörung Ferse recht

(5))

(6) ∧ Proband list column of a certain trial = test trial - 3. Monofilament nach Semmes-Weinstein

cannot combine set operation result with criterion

Figure 4.119.: A recognizer for query expressions allows identifying semantic errors.

4.9.1.2. Data Model

In order to store query expressions, the data model shown in figure 4.120 is introduced. The `Criterion` entity is used to tokenize a compound query. For a more compact UI presentation, each `Criterion` instance can hold a criterion token and a leading operator/parenthesis token, which are unfolded before parsing.

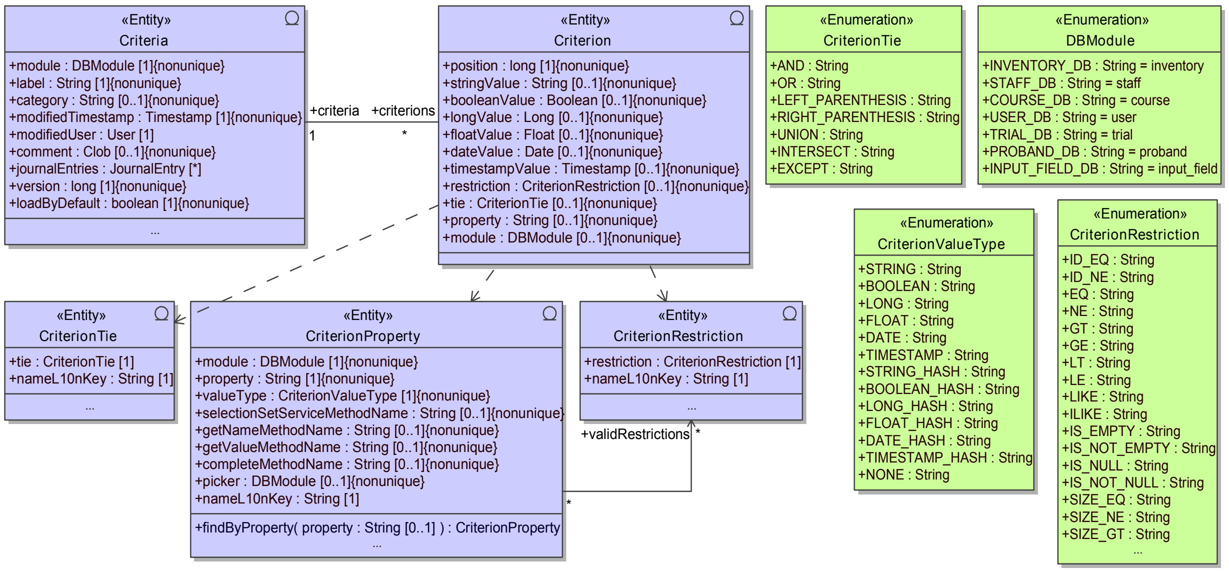

«Entity»
Criteria
+module : DBModule [1]{nonunique}
+label : String [1]{nonunique}
+category : String [0..1]{nonunique}
+modifiedTimestamp : Timestamp [1]{nonunique}
+modifiedUser : User [1]
+comment : Clob [0..1]{nonunique}
+journalEntries : JournalEntry [*]
+version : long [1]{nonunique}
+loadByDefault : boolean [1]{nonunique}
...
+criteria
1
+criterions
*
«Entity»
Criterion
+position : long [1]{nonunique}
+stringValue : String [0..1]{nonunique}
+booleanValue : Boolean [0..1]{nonunique}
+longValue : Long [0..1]{nonunique}
+floatValue : Float [0..1]{nonunique}
+dateValue : Date [0..1]{nonunique}
+timestampValue : Timestamp [0..1]{nonunique}
+restriction : CriterionRestriction [0..1]{nonunique}
+tie : CriterionTie [0..1]{nonunique}
+property : String [0..1]{nonunique}
+module : DBModule [0..1]{nonunique}
«Enumeration»
CriterionTie
+AND : String
+OR : String
+LEFT_PARENTHESIS : String
+RIGHT_PARENTHESIS : String
+UNION : String
+INTERSECT : String
+EXCEPT : String
«Enumeration»
DBModule
+INVENTORY_DB : String = inventory
+STAFF_DB : String = staff
+COURSE_DB : String = course
+USER_DB : String = user
+TRIAL_DB : String = trial
+PROBAND_DB : String = proband
+INPUT_FIELD_DB : String = input_field
«Entity»
CriterionTie
+tie : CriterionTie [1]
+nameL10nKey : String [1]
...
«Entity»
CriterionProperty
+module : DBModule [1]{nonunique}
+property : String [1]{nonunique}
+valueType : CriterionValueType [1]{nonunique}
+selectionSetServiceMethodName : String [0..1]{nonunique}
+getNameMethodName : String [0..1]{nonunique}
+getValueMethodName : String [0..1]{nonunique}
+completeMethodName : String [0..1]{nonunique}
+picker : DBModule [0..1]{nonunique}
+nameL10nKey : String [1]
+findByProperty(property : String [0..1]) : CriterionProperty
...
«Entity»
CriterionRestriction
+restriction : CriterionRestriction [1]
+nameL10nKey : String [1]
...
+validRestrictions
*
*
«Enumeration»
CriterionValueType
+STRING : String
+BOOLEAN : String
+LONG : String
+FLOAT : String
+DATE : String
+TIMESTAMP : String
+STRING_HASH : String
+BOOLEAN_HASH : String
+LONG_HASH : String
+FLOAT_HASH : String
+DATE_HASH : String
+TIMESTAMP_HASH : String
+NONE : String
«Enumeration»
CriterionRestriction
+ID_EQ : String
+ID_NE : String
+EQ : String
+NE : String
+GT : String
+GE : String
+LT : String
+LE : String
+LIKE : String
+ILIKE : String
+IS_EMPTY : String
+IS_NOT_EMPTY : String
+IS_NULL : String
+IS_NOT_NULL : String
+SIZE_EQ : String
+SIZE_NE : String
+SIZE_GT : String
...

Figure 4.120.: A database query definition is persisted by an `Criteria` entity record and its associated `Criterion` instances. `CriterionProperty` and `CriterionRestriction` represent master data entities holding definitions of entity columns available to criterion terms.

`Criteria` is a master entity which represents a compound query. It groups a list of `Criterion` detail objects, which are ordered according to their `position` field values. The `CriterionProperty` entity holds available entity column definitions P, which are populated by importing master data (table 4.17) using the CLI tool.

entity column p	data type	`SelectionSetService` method name	`NameMethodName`	`ValueMethodName`	`ToolsService` method name	picker module	R_p
`proband.inquiryValues.inquiry.id`	`LONG`	`getAllInquiries`	`getUniqueName`	`getId`			`EQ`
							`NE`
`proband.adresses.zipCodeHash`	`STRING_HASH`				`completeZipCode`		`EQ`
							`NE`
`staff.parent.id`	`LONG`					`STAFF_DB`	`EQ`
							`NE`

Table 4.17.: Query column definitions are imported using the tabular format shown. There is no need to modify application code when extending search capabilities. Whenever the system's data model is extended, new detail entity columns to search for can be made available to the query editor by importing the updated definitions.

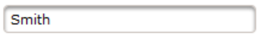
`Criterion.property` references a criterion term's entity column p using its unique association path. This results in an unconstrained association (section 4.1.1.4), which allows importing updated or extended entity column definitions hassle-free, even if a user has saved a query and `Criterion` records referencing entity column definitions have already been created. The comparison operator is stored by the `Criterion.restriction` field. Supported comparison operators are enumerated (`CriterionRestriction` enumeration type), but a separate `CriterionRestriction` entity is required for persisting applicable comparison operators R_p in the `CriterionRestriction.validRestrictions` many-to-many association.[8] Although the same does not apply for the association with criterion term conjunctions (Boolean operators, set operators and parentheses), they were modelled in the same way (`CriterionTie` enumeration type and entity). The criterion's comparison value v is stored in a `Criterion` field of the data type which corresponds the criterion's entity columns data type (`stringValue`, `floatValue`, . . .).

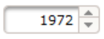
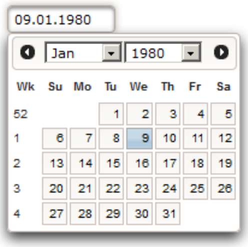
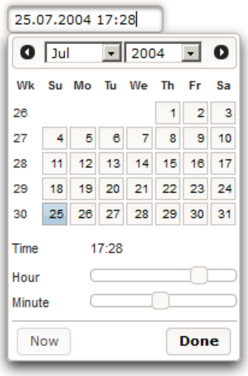
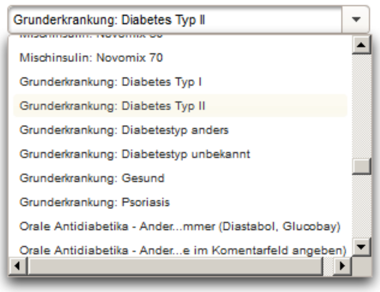
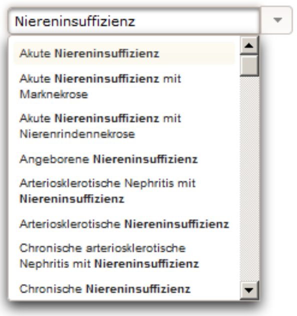
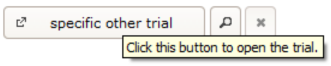
The query editor UI facilitates entering v by providing suitable input elements matching the column data type (`CriterionProperty.valueType`). Aside basic input elements for entering literal comparison values, the last three input elements shown in table 4.18 help users to *select* a comparison value:

Root entity picker Throughout the system, user are never required to enter a referenced record ID manually. Accordingly, the entity picker is utilized to specify a root entity record ID as a comparison value. A root record is selected from the result list of another query editor opened in the transient picker window (figure 4.71). `CriterionProperty.picker` defines the system module to search in. The `Criterion.longValue` field is used to persist the referenced record's ID value. When the referenced item is deleted, v is not updated but the missing or wrong reference is indicated by the query editor UI.

Item selector Aside root entity records, selecting a master data record ID for comparison will be a frequent case. `SelectionSetService` encapsualtes all retrieval operations for populating option items of dropdown spread across the UI. The query editor UI is supposed to display any of these dropdown option items as well. Therefore, a generic `p:selectOneMenu` dropdown is provided, whose option items are populated by invoking a `SelectionSetService` service method using *reflection*. The particular service method name belongs to the imported column configuration and is stored in the `CriterionProperty.selectionSetServiceMethodName` field. The service method's return value will be a collection of a corresponding OutVO type. To populate the dropdown's option items, each option item's ID (value) and label (name) is gathered by applying `getNameMethodName` and `getValueMethodName` while iterating the collection.

Autocomplete As outlined in section 4.1.1.4, master data records such as street names are not associated but suggested and field values are copied finally. In this case, a criterion has to compare the persisted value (e.g. `proband.adresses.cityNameHash`) instead of an ID. In order to provide users with corresponding suggestions for the exact spelling of persisted values, a generic `p:autoComplete` input is desired to facilitate looking up and correcting an entered string value fragment (query string). Retrieval operations for populating suggestions are exposed by `ToolsService.complete*` service methods. When populating suggestion, the method specified by `CriterionProperty.completeMethodName` is invoked with the entered query string argument.

Single-line text
`CriterionValueType` enumeration constants: `STRING`, `STRING_HASH`
entity column examples: *`trial.name`*, `trial.tagValues.value`, `proband.particulars.lastNameHash`, `proband.inquiryValues.value.stringValue`  Smith
	Yes/No
	`BOOLEAN`, `BOOLEAN_HASH`
	`proband.available`, `proband.inquiryValues.value.booleanValue`

Integer `LONG`, `LONG_HASH` `proband.id`, `proband.inquiryValues.value.longValue`  1972	**Decimal** `FLOAT`, `FLOAT_HASH` `proband.inquiryValues.value.floatValue` 37.2
Date `DATE`, `DATE_HASH`  09.01.1980 `proband.inquiryValues.value.dateValue`	**Datetime** `TIMESTAMP`, `TIMESTAMP_HASH`  25.07.2004 17:28 `proband.inquiryValues.value.timestampValue`
Selection `LONG`  Grunderkrankung: Diabetes Typ II `trial.tagValues.tag.id`, `proband.addresses.type.id`, `proband.inquiryValues.value.selectionValues.id`	**Autocomplete** `STRING`, `STRING_HASH`  Niereninsuffizienz `proband.addresses.cityNameHash`, `proband.inquiryValues.value.selectionValues.value`, `proband.diagnoses.code.systematics.categories.preferredRubricLabel`
Root entity `LONG` `inventory.parent.id`, `proband.trialParticipations.trial.id`  specific other trial	

Jan
1980
Wk Su Mo Tu We Th Fr Sa
Jul
2004
Time 17:28
Hour
Minute
Now
Done
Mischinsulin: Novomix 70
Grunderkrankung: Diabetes Typ I
Grunderkrankung: Diabetes Typ II
Grunderkrankung: Diabetestyp anders
Grunderkrankung: Diabetestyp unbekannt
Grunderkrankung: Gesund
Grunderkrankung: Psoriasis
Orale Antidiabetika - Ander...mmer (Diastabol, Glucobay)
Orale Antidiabetika - Ander...e im Komentarfeld angeben)
Akute Niereninsuffizienz
Akute Niereninsuffizienz mit Marknekrose
Akute Niereninsuffizienz mit Nierenrindennekrose
Angeborene Niereninsuffizienz
Arteriosklerotische Nephritis mit Niereninsuffizienz
Arteriosklerotische Niereninsuffizienz
Chronische arteriosklerotische Nephritis mit Niereninsuffizienz
Chronische Niereninsuffizienz
Click this button to open the trial.

Table 4.18.: Query editor UI input elements for entering comparison values.

Return values of `SelectionSetService` and `ToolsService.complete*` form sets of literal tokens, backed by the database. Putting things together, the database query concept presented can be finally summarized by the grammar shown in figure 4.121 and 4.122. In the context of subject eligibility criteria, it defines production rules of an "ad hoc expression language" according to (Weng et al., 2010, p. 6).

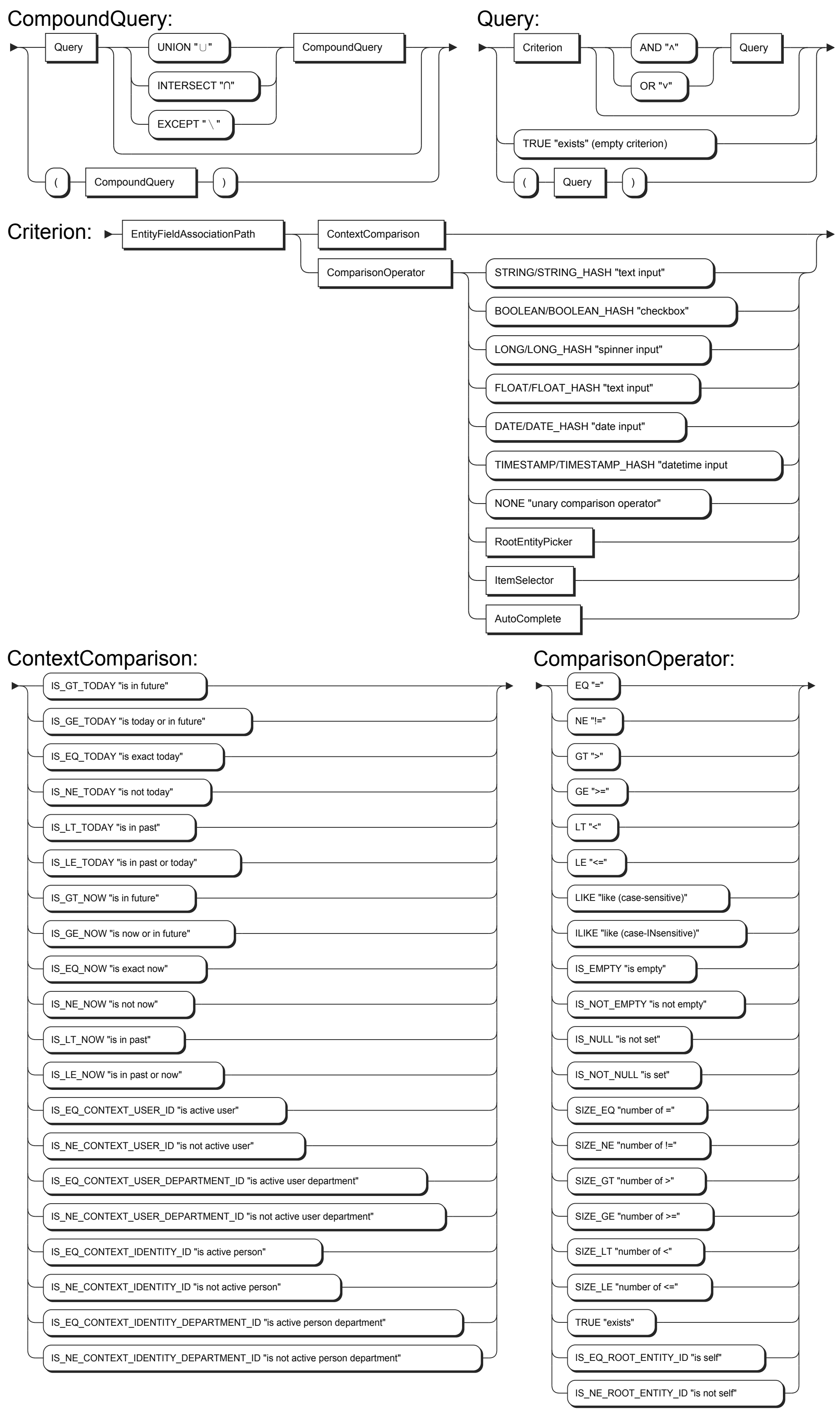

CompoundQuery:
Query
UNION "∪"
INTERSECT "∩"
EXCEPT "\"
CompoundQuery
(
CompoundQuery
)
Query:
Criterion
AND "∧"
OR "∨"
Query
TRUE "exists" (empty criterion)
(
Query
)
Criterion:
EntityFieldAssociationPath
ContextComparison
ComparisonOperator
STRING/STRING_HASH "text input"
BOOLEAN/BOOLEAN_HASH "checkbox"
LONG/LONG_HASH "spinner input"
FLOAT/FLOAT_HASH "text input"
DATE/DATE_HASH "date input"
TIMESTAMP/TIMESTAMP_HASH "datetime input
NONE "unary comparison operator"
RootEntityPicker
ItemSelector
AutoComplete
ContextComparison:
IS_GT_TODAY "is in future"
IS_GE_TODAY "is today or in future"
IS_EQ_TODAY "is exact today"
IS_NE_TODAY "is not today"
IS_LT_TODAY "is in past"
IS_LE_TODAY "is in past or today"
IS_GT_NOW "is in future"
IS_GE_NOW "is now or in future"
IS_EQ_NOW "is exact now"
IS_NE_NOW "is not now"
IS_LT_NOW "is in past"
IS_LE_NOW "is in past or now"
IS_EQ_CONTEXT_USER_ID "is active user"
IS_NE_CONTEXT_USER_ID "is not active user"
IS_EQ_CONTEXT_USER_DEPARTMENT_ID "is active user department"
IS_NE_CONTEXT_USER_DEPARTMENT_ID "is not active user department"
IS_EQ_CONTEXT_IDENTITY_ID "is active person"
IS_NE_CONTEXT_IDENTITY_ID "is not active person"
IS_EQ_CONTEXT_IDENTITY_DEPARTMENT_ID "is active person department"
IS_NE_CONTEXT_IDENTITY_DEPARTMENT_ID "is not active person department"
ComparisonOperator:
EQ "="
NE "!="
GT ">"
GE ">="
LT "<"
LE "<="
LIKE "like (case-sensitive)"
ILIKE "like (case-INsensitive)"
IS_EMPTY "is empty"
IS_NOT_EMPTY "is not empty"
IS_NULL "is not set"
IS_NOT_NULL "is set"
SIZE_EQ "number of ="
SIZE_NE "number of !="
SIZE_GT "number of >"
SIZE_GE "number of >="
SIZE_LT "number of <"
SIZE_LE "number of <="
TRUE "exists"
IS_EQ_ROOT_ENTITY_ID "is self"
IS_NE_ROOT_ENTITY_ID "is not self"

Figure 4.121.: Railroad diagram for query expressions, part 1.

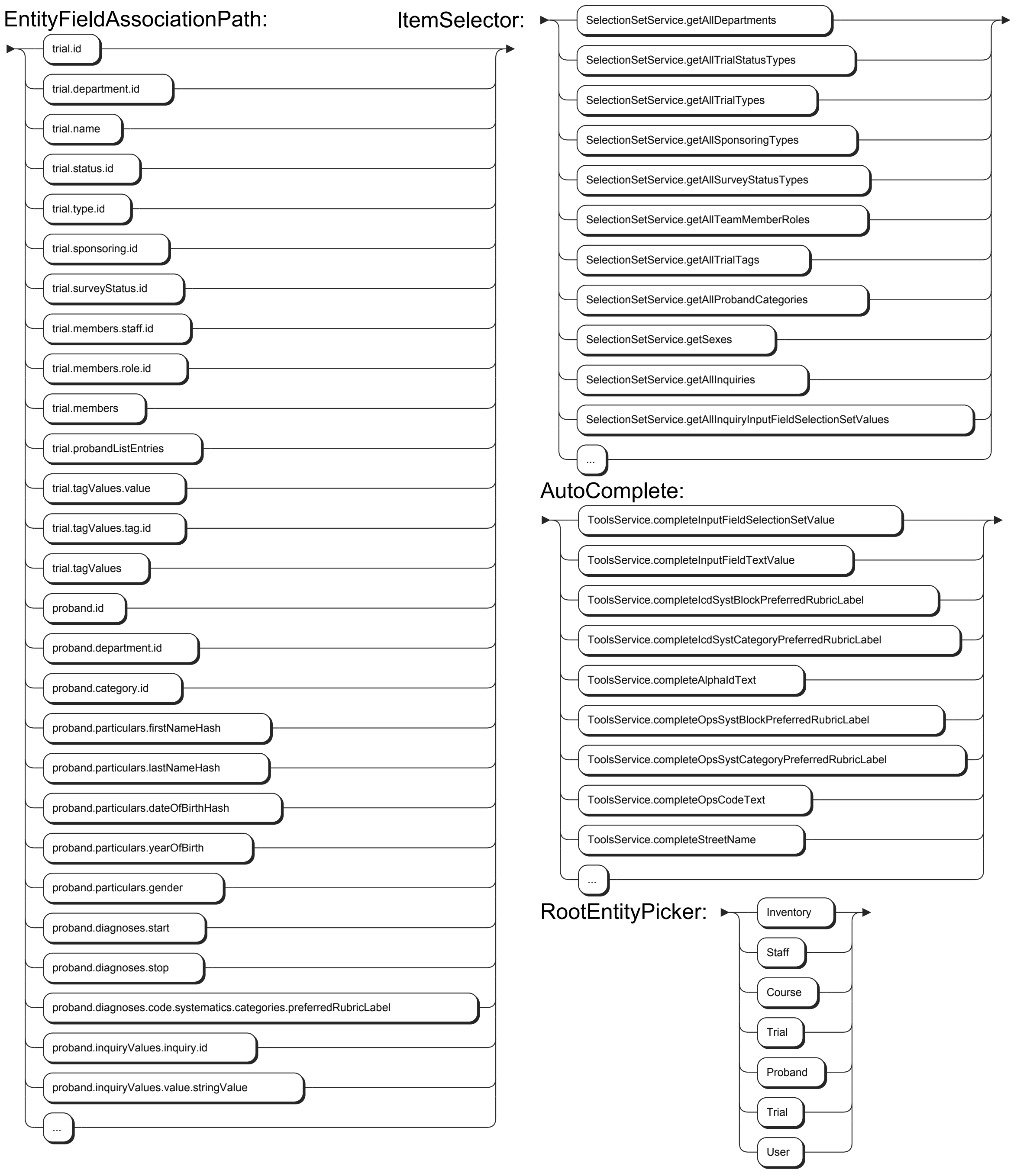
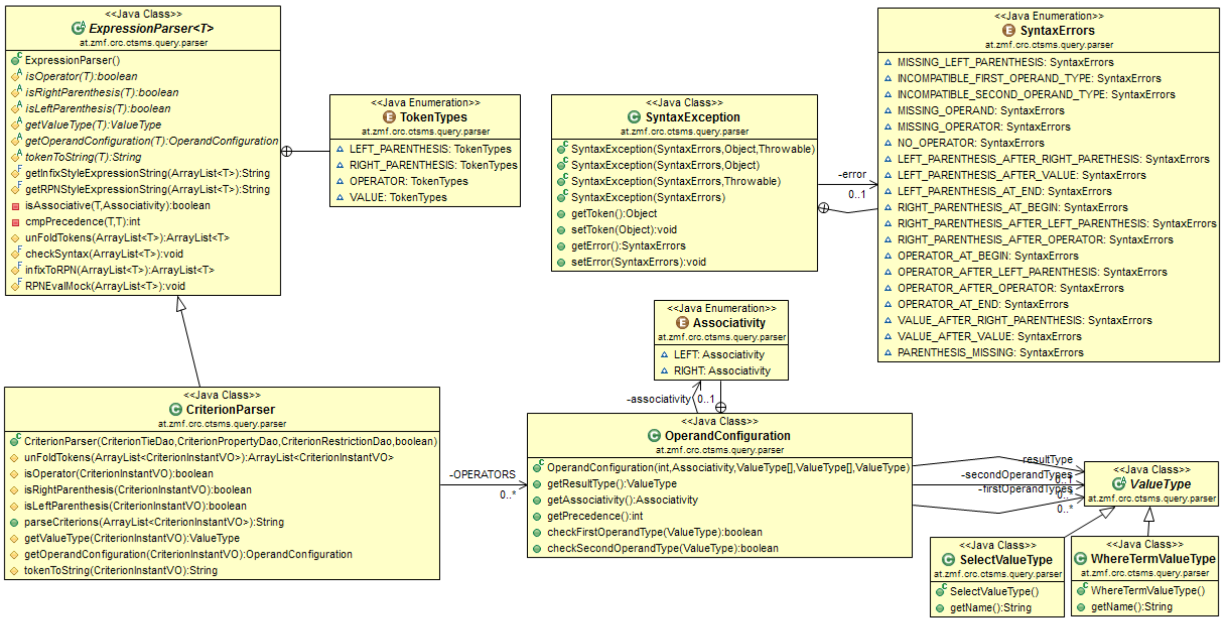

EntityFieldAssociationPath:
trial.id
trial.department.id
trial.name
trial.status.id
trial.type.id
trial.sponsoring.id
trial.surveyStatus.id
trial.members.staff.id
trial.members.role.id
trial.members
trial.probandListEntries
trial.tagValues.value
trial.tagValues.tag.id
trial.tagValues
proband.id
proband.department.id
proband.category.id
proband.particulars.firstNameHash
proband.particulars.lastNameHash
proband.particulars.dateOfBirthHash
proband.particulars.yearOfBirth
proband.particulars.gender
proband.diagnoses.start
proband.diagnoses.stop
proband.diagnoses.code.systematics.categories.preferredRubricLabel
proband.inquiryValues.inquiry.id
proband.inquiryValues.value.stringValue
...
ItemSelector:
SelectionSetService.getAllDepartments
SelectionSetService.getAllTrialStatusTypes
SelectionSetService.getAllTrialTypes
SelectionSetService.getAllSponsoringTypes
SelectionSetService.getAllSurveyStatusTypes
SelectionSetService.getAllTeamMemberRoles
SelectionSetService.getAllTrialTags
SelectionSetService.getAllProbandCategories
SelectionSetService.getSexes
SelectionSetService.getAllInquiries
SelectionSetService.getAllInquiryInputFieldSelectionSetValues
...
AutoComplete:
ToolsService.completeInputFieldSelectionSetValue
ToolsService.completeInputFieldTextValue
ToolsService.completeIcdSystBlockPreferredRubricLabel
ToolsService.completeIcdSystCategoryPreferredRubricLabel
ToolsService.completeAlphaIdText
ToolsService.completeOpsSystBlockPreferredRubricLabel
ToolsService.completeOpsSystCategoryPreferredRubricLabel
ToolsService.completeOpsCodeText
ToolsService.completeStreetName
...
RootEntityPicker:
Inventory
Staff
Course
Trial
Proband
Trial
User

Figure 4.122.: Railroad diagram for query expressions, part 2.

4.9.2. Implementation

4.9.2.1. Recognizer

The implemented recognizer is capable of detecting syntax and semantic errors before a query statement will be compiled and passed to the RDBMS. It generates an appropriate error message including the error's position, if a malformed expression is detected. The `CriterionParser.parseCriterions` method throws a `SyntaxException` containing the parsing error report, which can be caught by service methods to throw a `ServiceException` finally.



<<Java Class>>
ExpressionParser<T>
at.zmf.crc.ctsms.query.parser
ExpressionParser()
isOperator(T):boolean
isRightParenthesis(T):boolean
isLeftParenthesis(T):boolean
getValueType(T):ValueType
getOperandConfiguration(T):OperandConfiguration
tokenToString(T):String
getInfixStyleExpressionString(ArrayList<T>):String
getRPNStyleExpressionString(ArrayList<T>):String
isAssociative(T,Associativity):boolean
cmpPrecedence(T,T):int
unFoldTokens(ArrayList<T>):ArrayList<T>
checkSyntax(ArrayList<T>):void
infixToRPN(ArrayList<T>):ArrayList<T>
RPNEvalMock(ArrayList<T>):void
<<Java Enumeration>>
TokenTypes
at.zmf.crc.ctsms.query.parser
LEFT_PARENTHESIS: TokenTypes
RIGHT_PARENTHESIS: TokenTypes
OPERATOR: TokenTypes
VALUE: TokenTypes
<<Java Class>>
SyntaxException
at.zmf.crc.ctsms.query.parser
SyntaxException(SyntaxErrors,Object,Throwable)
SyntaxException(SyntaxErrors,Object)
SyntaxException(SyntaxErrors,Throwable)
SyntaxException(SyntaxErrors)
getToken():Object
setToken(Object):void
getError():SyntaxErrors
setError(SyntaxErrors):void
-error
0..1
<<Java Enumeration>>
SyntaxErrors
at.zmf.crc.ctsms.query.parser
MISSING_LEFT_PARENTHESIS: SyntaxErrors
INCOMPATIBLE_FIRST_OPERAND_TYPE: SyntaxErrors
INCOMPATIBLE_SECOND_OPERAND_TYPE: SyntaxErrors
MISSING_OPERAND: SyntaxErrors
MISSING_OPERATOR: SyntaxErrors
NO_OPERATOR: SyntaxErrors
LEFT_PARENTHESIS_AFTER_RIGHT_PARENTHESIS: SyntaxErrors
LEFT_PARENTHESIS_AFTER_VALUE: SyntaxErrors
LEFT_PARENTHESIS_AT_END: SyntaxErrors
RIGHT_PARENTHESIS_AT_BEGIN: SyntaxErrors
RIGHT_PARENTHESIS_AFTER_LEFT_PARENTHESIS: SyntaxErrors
RIGHT_PARENTHESIS_AFTER_OPERATOR: SyntaxErrors
OPERATOR_AT_BEGIN: SyntaxErrors
OPERATOR_AFTER_LEFT_PARENTHESIS: SyntaxErrors
OPERATOR_AFTER_OPERATOR: SyntaxErrors
OPERATOR_AT_END: SyntaxErrors
VALUE_AFTER_RIGHT_PARENTHESIS: SyntaxErrors
VALUE_AFTER_VALUE: SyntaxErrors
PARENTHESIS_MISSING: SyntaxErrors
<<Java Enumeration>>
Associativity
at.zmf.crc.ctsms.query.parser
LEFT: Associativity
RIGHT: Associativity
-associativity
0..1
<<Java Class>>
CriterionParser
at.zmf.crc.ctsms.query.parser
CriterionParser(CriterionTieDao,CriterionPropertyDao,CriterionRestrictionDao,boolean)
unFoldTokens(ArrayList<CriterionInstantVO>):ArrayList<CriterionInstantVO>
isOperator(CriterionInstantVO):boolean
isRightParenthesis(CriterionInstantVO):boolean
isLeftParenthesis(CriterionInstantVO):boolean
parseCriterions(ArrayList<CriterionInstantVO>):String
getValueType(CriterionInstantVO):ValueType
getOperandConfiguration(CriterionInstantVO):OperandConfiguration
tokenToString(CriterionInstantVO):String
-OPERATORS
0..*
<<Java Class>>
OperandConfiguration
at.zmf.crc.ctsms.query.parser
OperandConfiguration(int,Associativity,ValueType[],ValueType[],ValueType)
getResultType():ValueType
getAssociativity():Associativity
getPrecedence():int
checkFirstOperandType(ValueType):boolean
checkSecondOperandType(ValueType):boolean
-resultType
-secondOperandTypes
-firstOperandTypes
0..1
0..*
0..*
<<Java Class>>
ValueType
at.zmf.crc.ctsms.query.parser
<<Java Class>>
SelectValueType
at.zmf.crc.ctsms.query.parser
SelectValueType()
getName():String
<<Java Class>>
WhereTermValueType
at.zmf.crc.ctsms.query.parser
WhereTermValueType()
getName():String

Figure 4.123.: The `ExpressionParser` abstract base class implements the shunting-yard and RPN evaluation algorithms. `CriterionParser` provides handling of `CriterionInstantVO` tokens and operator definitions from table 4.16.

Since a query definition is always passed in tokenized form instead of expression strings, the decision was to develop a simple recognizer from scratch instead of using a parser generator framework. The recognizer algorithm works in two intuitive steps, forming an *interpreter pattern*:

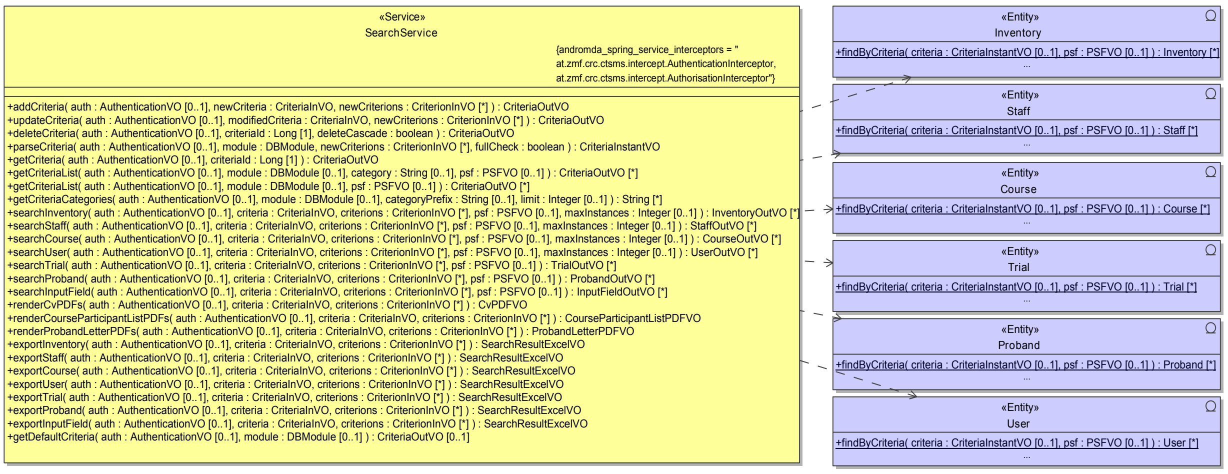
1. `ExpressionParser.infixToRPN` implements the *shunting-yard* algorithm (Dijkstra, 1961) to transform an infix query into RPN, e.g.

$$c_{18}, c_{17}, c_{16}, \vee, \cup, c_{15}, c_{14}, c_{13}, \wedge, \wedge, c_{12}, c_{11}, \wedge, c_{10}, c_9, c_8, \vee, \wedge,$$
$$c_7, c_6, c_5, c_4, c_3, c_2, c_1, \wedge, \wedge, \wedge, \wedge, \wedge, \wedge, \cap, \cap, \cap, \setminus$$

2. `ExpressionParser.RPNEvalMock` represents a simple mock evaluator for processing the RPN result. It will not compute any query result set, but can check the compatibility of operand types. An error location is given by the mismatching token's position, since token objects carry a value representing their position in the original infix query.

4.9.2.2. Query Processing

A separate service interface (`SearchService`, figure 4.124) is introduced to encapsulate CRUD service methods for managing queries persisted with the query data model presented in section 4.9.1.2.



```
«Service»
SearchService
{andromda_spring_service_interceptors = "
at.zmf.crc.ctsms.intercept.AuthenticationInterceptor,
at.zmf.crc.ctsms.intercept.AuthorisationInterceptor"}

+addCriteria( auth : AuthenticationVO [0..1], newCriteria : CriteriaInVO, newCriterions : CriterionInVO [*] ) : CriteriaOutVO
+updateCriteria( auth : AuthenticationVO [0..1], modifiedCriteria : CriteriaInVO, newCriterions : CriterionInVO [*] ) : CriteriaOutVO
+deleteCriteria( auth : AuthenticationVO [0..1], criteriaId : Long [1], deleteCascade : boolean ) : CriteriaOutVO
+parseCriteria( auth : AuthenticationVO [0..1], module : DBModule, newCriterions : CriterionInVO [*], fullCheck : boolean ) : CriteriaInstantVO
+getCriteria( auth : AuthenticationVO [0..1], criteriaId : Long [1] ) : CriteriaOutVO
+getCriteriaList( auth : AuthenticationVO [0..1], module : DBModule [0..1], category : String [0..1], psf : PSFVO [0..1] ) : CriteriaOutVO [*]
+getCriteriaList( auth : AuthenticationVO [0..1], module : DBModule [0..1], psf : PSFVO [0..1] ) : CriteriaOutVO [*]
+getCriteriaCategories( auth : AuthenticationVO [0..1], module : DBModule [0..1], categoryPrefix : String [0..1], limit : Integer [0..1] ) : String [*]
+searchInventory( auth : AuthenticationVO [0..1], criteria : CriteriaInVO, criterions : CriterionInVO [*], psf : PSFVO [0..1], maxInstances : Integer [0..1] ) : InventoryOutVO [*]
+searchStaff( auth : AuthenticationVO [0..1], criteria : CriteriaInVO, criterions : CriterionInVO [*], psf : PSFVO [0..1], maxInstances : Integer [0..1] ) : StaffOutVO [*]
+searchCourse( auth : AuthenticationVO [0..1], criteria : CriteriaInVO, criterions : CriterionInVO [*], psf : PSFVO [0..1], maxInstances : Integer [0..1] ) : CourseOutVO [*]
+searchUser( auth : AuthenticationVO [0..1], criteria : CriteriaInVO, criterions : CriterionInVO [*], psf : PSFVO [0..1], maxInstances : Integer [0..1] ) : UserOutVO [*]
+searchTrial( auth : AuthenticationVO [0..1], criteria : CriteriaInVO, criterions : CriterionInVO [*], psf : PSFVO [0..1] ) : TrialOutVO [*]
+searchProband( auth : AuthenticationVO [0..1], criteria : CriteriaInVO, criterions : CriterionInVO [*], psf : PSFVO [0..1] ) : ProbandOutVO [*]
+searchInputField( auth : AuthenticationVO [0..1], criteria : CriteriaInVO, criterions : CriterionInVO [*], psf : PSFVO [0..1] ) : InputFieldOutVO [*]
+renderCvPDFs( auth : AuthenticationVO [0..1], criteria : CriteriaInVO, criterions : CriterionInVO [*] ) : CvPDFVO
+renderCourseParticipantListPDFs( auth : AuthenticationVO [0..1], criteria : CriteriaInVO, criterions : CriterionInVO [*] ) : CourseParticipantListPDFVO
+renderProbandLetterPDFs( auth : AuthenticationVO [0..1], criteria : CriteriaInVO, criterions : CriterionInVO [*] ) : ProbandLetterPDFVO
+exportInventory( auth : AuthenticationVO [0..1], criteria : CriteriaInVO, criterions : CriterionInVO [*] ) : SearchResultExcelVO
+exportStaff( auth : AuthenticationVO [0..1], criteria : CriteriaInVO, criterions : CriterionInVO [*] ) : SearchResultExcelVO
+exportCourse( auth : AuthenticationVO [0..1], criteria : CriteriaInVO, criterions : CriterionInVO [*] ) : SearchResultExcelVO
+exportUser( auth : AuthenticationVO [0..1], criteria : CriteriaInVO, criterions : CriterionInVO [*] ) : SearchResultExcelVO
+exportTrial( auth : AuthenticationVO [0..1], criteria : CriteriaInVO, criterions : CriterionInVO [*] ) : SearchResultExcelVO
+exportProband( auth : AuthenticationVO [0..1], criteria : CriteriaInVO, criterions : CriterionInVO [*] ) : SearchResultExcelVO
+exportInputField( auth : AuthenticationVO [0..1], criteria : CriteriaInVO, criterions : CriterionInVO [*] ) : SearchResultExcelVO
+getDefaultCriteria( auth : AuthenticationVO [0..1], module : DBModule [0..1] ) : CriteriaOutVO [0..1]

«Entity»
Inventory
+findByCriteria( criteria : CriteriaInstantVO [0..1], psf : PSFVO [0..1] ) : Inventory [*]
...

«Entity»
Staff
+findByCriteria( criteria : CriteriaInstantVO [0..1], psf : PSFVO [0..1] ) : Staff [*]
...

«Entity»
Course
+findByCriteria( criteria : CriteriaInstantVO [0..1], psf : PSFVO [0..1] ) : Course [*]
...

«Entity»
Trial
+findByCriteria( criteria : CriteriaInstantVO [0..1], psf : PSFVO [0..1] ) : Trial [*]
...

«Entity»
Proband
+findByCriteria( criteria : CriteriaInstantVO [0..1], psf : PSFVO [0..1] ) : Proband [*]
...

«Entity»
User
+findByCriteria( criteria : CriteriaInstantVO [0..1], psf : PSFVO [0..1] ) : User [*]
...
```

Figure 4.124.: Service methods for managing and executing database queries are provided by `SearchService`.

Thus, the invocation of `SearchService` methods can be authorized for a user independent of granted privileges for modifying a module's records. The parsing report can be generated by a separate service method (`parseCriteria`), which is used to display errors and highlight query syntax in the UI after any modification to the query while composing it. `search*` service methods finally perform a persisted query for the corresponding root entity type. Privileged users may also run instant queries before saving them, e.g. when designing a new query. Returned result lists can be paginated, sorted and filtered. Likewise, `export*` service methods are used to export the query's entire result set in Excel format (section 4.3.2).

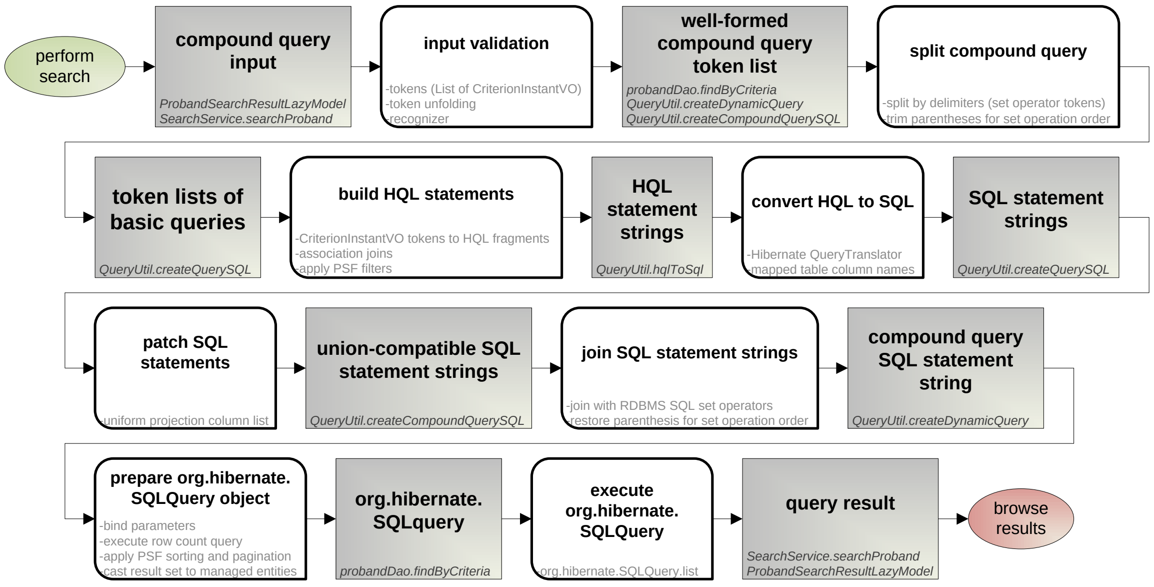

perform search
compound query input
ProbandSearchResultLazyModel
SearchService.searchProband
input validation
-tokens (List of CriterionInstantVO)
-token unfolding
-recognizer
well-formed compound query token list
probandDao.findByCriteria
QueryUtil.createDynamicQuery
QueryUtil.createCompoundQuerySQL
split compound query
-split by delimiters (set operator tokens)
-trim parentheses for set operation order
token lists of basic queries
QueryUtil.createQuerySQL
build HQL statements
-CriterionInstantVO tokens to HQL fragments
-association joins
-apply PSF filters
HQL statement strings
QueryUtil.hqlToSql
convert HQL to SQL
-Hibernate QueryTranslator
-mapped table column names
SQL statement strings
QueryUtil.createQuerySQL
patch SQL statements
-uniform projection column list
union-compatible SQL statement strings
QueryUtil.createCompoundQuerySQL
join SQL statement strings
-join with RDBMS SQL set operators
-restore parenthesis for set operation order
compound query SQL statement string
QueryUtil.createDynamicQuery
prepare org.hibernate. SQLQuery object
-bind parameters
-execute row count query
-apply PSF sorting and pagination
-cast result set to managed entities
org.hibernate. SQLquery
probandDao.findByCriteria
execute org.hibernate. SQLQuery
org.hibernate.SQLQuery.list
query result
SearchService.searchProband
ProbandSearchResultLazyModel
browse results

Figure 4.125.: This flow diagram illustrates processing steps and intermediate structures when building and executing dynamic queries. The pipeline is run through whenever `SearchService.search*` or `SearchService.export*` methods are invoked.

```
public class ProbandDaoImpl extends ProbandDaoBase {

    ...

    @Override
    protected Collection<Proband> handleFindByCriteria(CriteriaInstantVO criteria, PSFVO psf) throws Exception {
        //build the org.hibernate.SQLQuery object:
        Query query = QueryUtil.createSearchQuery(criteria, DBModule.PROBAND_DB, psf, this.getSessionFactory(), this.
              getCriterionTieDao(), this.getCriterionPropertyDao(), this.getCriterionRestrictionDao());
        return query.list(); //execute the query
    }

    ...
```

Listing 4.61: A root entity's `findByCriteria` finder method represents the main entry point for executing dynamic database queries.

`search*` or `export*` service methods validate the query input and execute it by invoking the corresponding root entity's `findByCriteria` finder method (listing 4.61) afterwards. In contrast to regular finder methods, which utilize Hibernate's Criteria API to list records, `findByCriteria` methods represent the main entry point for the more complex dynamic query processing logic (figure 4.125). Aside the PSF parameter described in section 4.2.4, the query definition is passed in form of a `CriteriaInstantVO` VO parameter.

As indicated in the previous section, the basic idea is to build a HQL query statement string, which is executed against the database to retrieve query results. The `CriteriaInstantVO` parameter comprises the compound query's input tokens in form of an ordered list of `CriterionInstantVO` objects. This token list could be converted to a HQL statement by unfolding[98] and iterating it straight-forward to substitute each token by a corresponding HQL fragment (listing 4.63):

- Boolean operators and parentheses
- column names from association paths
- comparison operators with placeholder (?) to bind comparison values (query parameters)

After expanding association paths of criterion columns as shown in listing listing 4.58, the HQL statement is valid and accepted by `org.hibernate.createQuery`. While this suffices simple queries without set operators, compound queries containing set operators require additional processing. Set operators are allowed in raw SQL statements only, so the workaround is to compile to SQL and insert set operator keywords explicitly.

```
//located in public final class at.zmf.crc.ctsms.query.QueryUtil
private static String hqlToSql(String hqlQueryText, SessionFactory sessionFactory) {
    final QueryTranslatorFactory translatorFactory = new ASTQueryTranslatorFactory();
    final SessionFactoryImplementor factory = (SessionFactoryImplementor) sessionFactory;
    final QueryTranslator translator = translatorFactory.createQueryTranslator(hqlQueryText, hqlQueryText, Collections.
            EMPTY_MAP, factory);
    translator.compile(Collections.EMPTY_MAP, false); //named bound params are substituted with "?"
    return translator.getSQLString();
}
```

Listing 4.62: This utility method utilizes Hibernate's `QueryTranslator` to convert a HQL statement string to SQL (Cinar, 2009).

The implementation therefore provides an algorithm comprising the piece-wise conversion of the compound query into native SQL query statements. It is possible to compile a HQL statement into SQL explicitly using `org.hibernaet.hql.QueryTranslator`, which is utilized by the method shown in listing 4.62.

[98]Each `CriterionInstantVO` containing an operator/parenthesis *and* criterion is split.

```
//located in at.zmf.crc.ctsms.query.QueryUtil
private static String createQuerySQL(ArrayList<CriterionInstantVO> sortedCriterions, DBModule module, PSFVO psf, SessionFactory sessionFactory,
    ArrayList<QueryParameterValue> queryValues, //a list of QueryParameterValue helper objects is populated for query parameter binding
    HashMap<Long, at.zmf.crc.ctsms.enumeration.CriterionTie> tieMap,
    HashMap<Long, CriterionProperty> propertyMap,
    HashMap<Long, at.zmf.crc.ctsms.enumeration.CriterionRestriction> restrictionMap) throws Exception {

    StringBuilder hqlStatement = new StringBuilder("select distinct ");
    StringBuilder sqlSelectStatement = new StringBuilder(); //the final query SQL result
    StringBuilder hqlWhereClause = new StringBuilder(); //the HQL "where" clause: query criteria and optional filter criteria

    HashMap<String,AssociationPath> explicitJoinsMap = new HashMap<String,AssociationPath>(); //a map for association path strings and corresponding
            AssociationPath helper objects
    HashMap<String,Class> propertyClassMap = new HashMap<String,Class>(); //a map for association path strings and corresponding field types, required
            for applying PSF filters
    Class entityClass = getRootEntityClass(module); //e.g. ProbandImpl for DBModules.PROBAND
    String entityClassName = entityClass.getCanonicalName(); //e.g. "at.zmf.crc.ctsms.domain.ProbandImpl"
    String entityName = ENTITY_NAMES.get(module); // "proband"  for DBModules.PROBAND
    hqlStatement.append(entityName).append(" from ").append(entityClassName).append(" as ").append(entityName);

    boolean whereAppended = false;
    if (sortedCriterions.size() > 0) { hqlWhereClause.append(" where ("); whereAppended = true; }

    for (int i = 0; i < sortedCriterions.size(); i++) { //iterate tokens in order build a HQL "where" clause string from listing 4.58
        CriterionInstantVO criterion = sortedCriterions.get(i);
            if (criterion.getTieId() != null) {
                at.zmf.crc.ctsms.enumeration.CriterionTie tie = tieMap.get(criterion.getTieId());
            switch (tie) { //append criterion term conjunction:
                case AND: hqlWhereClause.append(" and "); break;
                case OR:  hqlWhereClause.append(" or "); break;
                case LEFT_PARENTHESIS: hqlWhereClause.append("("); break;
                case RIGHT_PARENTHESIS: hqlWhereClause.append(")"); break;
                default: throw new IllegalArgumentException(L10nUtil.getMessage(MessageCodes.UNSUPPORTED_CRITERION_TIE, DefaultMessages.
                        UNSUPPORTED_CRITERION_TIE,new Object[]{tie.toString()}));
            }
        }
            CriterionProperty property = null; //criterion entity field
            if (criterion.getPropertyId() != null) { property = propertyMap.get(criterion.getPropertyId()); }
            at.zmf.crc.ctsms.enumeration.CriterionRestriction restriction = null; //criterion comparison operator
            if (criterion.getRestrictionId() != null) { restriction = restrictionMap.get(criterion.getRestrictionId()); }
            if (property != null  restriction != null) {
                AssociationPath propertyNameAssociationPath = new AssociationPath(property.getProperty());
                //An association path is analyzed to identify required table joins for collection-valued association ends, which cannot be dereferenced
                //by Hibernate otherwise. As a result, an numbered alias is generated per joined association and has to be used to refer to the criterion
                //entity field instead of the original association path. For example, a proband query with a single criterion
                //"proband.inquiryValues.value.longValue > 0" produces the HQL query statement below:
                //  distinct proband from at.zmf.crc.ctsms.domain.ProbandImpl as proband
                //    left join proband.inquiryValues as _0 left join _0.value as _1 where (_1.longValue > ?)
                //Each alias is registered and reused when encountered in subsequent association paths:
                //  proband.inquiryValues -> "_0", proband.inquiryValues.value -> "_1"
                String propertyName = aliasPropertyName(entityClass, propertyNameAssociationPath, entityName, explicitJoinsMap, propertyClassMap); //
                        returns "_3"
            switch (restriction) { //append comparison operator with positioned query parameter binding:
                case EQ: hqlWhereClause.append(propertyName); hqlWhereClause.append(" = ?"); break;
                case ILIKE: hqlWhereClause.append("lower("); hqlWhereClause.append(propertyName); hqlWhereClause.append(") like lower(?)"); break;
                case SIZE_EQ: hqlWhereClause.append("size("); hqlWhereClause.append(propertyName); hqlWhereClause.append(") = ?"); break;
                case IS_EQ_CONTEXT_USER_ID: hqlWhereClause.append(propertyName); hqlWhereClause.append(" = ?"); queryValues.add(new QueryParameterValue(
                        QueryParameterValueType.CONTEXT_USER_ID)); break;
                ...
                default: throw new IllegalArgumentException(L10nUtil.getMessage(MessageCodes.UNSUPPORTED_CRITERION_RESTRICTION, DefaultMessages.
                        UNSUPPORTED_CRITERION_RESTRICTION,new Object[]{restriction.toString()}));
            }
            if (!CommonUtil.isUnaryCriterionRestriction(restriction)) { //unless unary, add the comparison value for query parameter binding:
                    queryValues.add(new QueryParameterValue(propertyNameAssociationPath.getFullQualifiedPropertyName(), property.getValueType(),
                            criterion));
                }
        }
    }
    if (whereAppended) { hqlWhereClause.append(")"); } //close criterion terms where clause fragment
    if (psf != null) { //process filter criterion terms, if provided by the PSFVO passed:
                boolean firstFilter = true;
                Iterator<Map.Entry<String,String>> filterIt = psf.getFilters().entrySet().iterator();
        while (filterIt.hasNext()) {
            Map.Entry<String,String> filter = (Map.Entry<String,String>) filterIt.next();
            AssociationPath filterFieldAssociationPath = new AssociationPath(filter.getKey());
            //A filter association path is processed according to association paths used in criterion terms before. Field types of inspected association
                    ends are recorded (propertyClassMap) since they are required to build appropriate filter terms when applying PSF filters.
            String filterField = aliasPropertyName(entityClass,filterFieldAssociationPath,entityName,explicitJoinsMap,propertyClassMap);
            if (firstFilter) {
                if (whereAppended) {
                    hqlWhereClause.append(" and (");
                } else {
                    hqlWhereClause.append(" where ("); whereAppended = true;
                }
                firstFilter = false;
            } else {
                hqlWhereClause.append(" and ");
            }
            //The HQL applyFilter method builds and appends filter criteria fragments to the statement. This is done to produce results identical to the
                    Criteria API applyFilter method from listing 4.17:
            applyFilter(hqlWhereClause, queryValues, filterField, propertyClassMap.get(filterFieldAssociationPath.getFullQualifiedPropertyName()), filter
                    .getValue());
            if (!filterIt.hasNext()) { hqlWhereClause.append(")"); } //close filter criterion terms where clause fragment
        }
    }
    appendJoins(hqlStatement,explicitJoinsMap); //put association joins into place
    hqlStatement.append(hqlWhereClause);
    sqlSelectStatement.append(hqlToSql(hqlStatement.toString(),sessionFactory)); //select distinct probandimp0_.ID as ID14_, probandimp0_.
            MODIFIED_TIMESTAMP as MODIFIED2_14_, ...
    //sanitize column names generated by the HQL to SQL conversion to exactly match the entity's original column names:
    return sanitizeColumnProjectionList(sqlSelectStatement.toString()); //select distinct probandimp0_.ID as ID, probandimp0_.MODIFIED_TIMESTAMP as
            MODIFIED_TIMESTAMP, ...
    //The compound query result set will keep column names only if they are intentical among any single query. This is required to apply an ORDER BY
            clause and finally cast back to Hibernate managed entity objects.
}
```

Listing 4.63: The `createQuerySQL` helper method processes tokens of a basic query and builds a HQL statement string.

```
//located in at.zmf.crc.ctsms.query.QueryUtil
private static StringBuilder createCompoundQuerySQL(CriteriaInstantVO criteriaInstantVO, //query definition
    DBModule module, PSFVO psf, SessionFactory sessionFactory, ArrayList<QueryParameterValue> queryValues, //createQuerySQL parameters
    HashMap<Long, at.zmf.crc.ctsms.enumeration.CriterionTie> tieMap, //cache map for fast lookup by CriterionTie.id
    HashMap<Long, CriterionProperty> propertyMap, //cache map for fast lookup by CriterionProperty.id
    HashMap<Long, at.zmf.crc.ctsms.enumeration.CriterionRestriction> restrictionMap) throws Exception { //cache map for fast lookup by
           CriterionRestriction.id

    ArrayList<CriterionInstantVO> sortedCriterions = new ArrayList<CriterionInstantVO>(criteriaInstantVO.getCriterions());
    Collections.sort(sortedCriterions, new CriterionInstantVOComparator()); //ensure tokens are sorted by position

    StringBuilder compoundSQLStatement = new StringBuilder(); //the compound query's SQL statement result
    int queryIndex = 0; //the query index is used to group the compound query's tokens by query for syntax highlighting
    int lastSetOperatorIndex = -1;

    for (int i = 0; i < sortedCriterions.size(); i++) { //iterate tokens in order to split tokens by set operators:
        CriterionInstantVO criterion = sortedCriterions.get(i);
        if (criterion.getTieId() != null) {
            String sqlSetKeyWord = null;
            boolean isSetOperator = false;
            at.zmf.crc.ctsms.enumeration.CriterionTie tie = tieMap.get(criterion.getTieId());
            switch (tie) {
                case EXCEPT:
                    isSetOperator = true;
                    sqlSetKeyWord = Settings.getString(SettingCodes.SQL_EXCEPT_KEYWORD, Bundle.SETTINGS, null);
                    break;
                case INTERSECT:
                    isSetOperator = true;
                    sqlSetKeyWord = Settings.getString(SettingCodes.SQL_INTERSECT_KEYWORD, Bundle.SETTINGS, null);
                    break;
                case UNION:
                    isSetOperator = true;
                    sqlSetKeyWord = Settings.getString(SettingCodes.SQL_UNION_KEYWORD, Bundle.SETTINGS, null);
                    break;
                case LEFT_PARENTHESIS:
                case RIGHT_PARENTHESIS:
                default: //Boolean operators
                    break;
            }
            if (isSetOperator) { //another set operator is detected:
                //count parentheses between the last set operator token and the current:
                int offset = lastSetOperatorIndex + 1; int count = i - offset;
                int leftParenthesisCount = getCriterionTieCount(sortedCriterions, offset, LEFT_PARENTHESIS,tieMap);
                int rightParenthesisCount = getCriterionTieCount(sortedCriterions, offset, RIGHT_PARENTHESIS,tieMap);
                if (leftParenthesisCount > rightParenthesisCount) { //the leading query contains opening parentheses not closed yet:
                    int balance = leftParenthesisCount - rightParenthesisCount;
                    for (int j = 0; j < balance; j++) { compoundSQLStatement.append("("); } //dump the opening parentheses
                    //convert and dump the leading query:
                    compoundSQLStatement.append(createQuerySQL(CoreUtil.getSubList(sortedCriterions, offset + balance, count - balance), module, psf,
                           sessionFactory, queryValues, tieMap, propertyMap, restrictionMap));
                } else if (leftParenthesisCount < rightParenthesisCount) { //opening parenteses were closed in the the leading query:
                    int balance = rightParenthesisCount - leftParenthesisCount;
                    //convert and dump the leading query:
                    compoundSQLStatement.append(createQuerySQL(CoreUtil.getSubList(sortedCriterions, offset, count - balance), module, psf,
                           sessionFactory, queryValues, tieMap, propertyMap, restrictionMap));
                    for (int j = 0; j < balance; j++) { compoundSQLStatement.append(")"); } //dump the closing parentheses
                } else { //a balanced number of parentheses were found since the last set operator:
                    //convert and dump the leading query:
                    compoundSQLStatement.append(createQuerySQL(CoreUtil.getSubList(sortedCriterions, offset, count), module, psf, sessionFactory,
                           queryValues, tieMap, propertyMap, restrictionMap));
                }
                compoundSQLStatement.append(" "); //append the SQL set operator keyword:
                if (sqlSetKeyWord != null  sqlSetKeyWord.length() > 0) {
                    compoundSQLStatement.append(sqlSetKeyWord);
                } else { //report an error, if a set operator keyword is not configured:
                    throw new IllegalArgumentException(L10nUtil.getMessage(MessageCodes.SQL_SET_OPERATION_KEYWORD_UNDEFINED, DefaultMessages.
                           SQL_SET_OPERATION_KEYWORD_UNDEFINED,new Object[]{ tie.toString()}));
                }
                compoundSQLStatement.append(" ");
                lastSetOperatorIndex = i;
                queryIndex++;
            } else {
                criterion.setSelectStatementIndex(queryIndex); //mark criterion with the current query index for syntax highlighting
            }
        } else {
            criterion.setSelectStatementIndex(queryIndex); //mark criterion with the current query index for syntax highlighting
        }
    }
    if (lastSetOperatorIndex >= 0) { //process the last, trailing query
        int offset = lastSetOperatorIndex + 1; int count = sortedCriterions.size() - offset;
        int leftParenthesisCount = getCriterionTieCount(sortedCriterions, offset, LEFT_PARENTHESIS,tieMap);
        int rightParenthesisCount = getCriterionTieCount(sortedCriterions, offset, RIGHT_PARENTHESIS,tieMap);
        if (leftParenthesisCount < rightParenthesisCount) {
            int balance = rightParenthesisCount - leftParenthesisCount;
            compoundSQLStatement.append(createQuerySQL(CoreUtil.getSubList(sortedCriterions, offset, count - balance), module, psf, sessionFactory,
                   queryValues, tieMap, propertyMap, restrictionMap));
            for (int j = 0; j < balance; j++) { compoundSQLStatement.append(")"); }
        } else if (leftParenthesisCount == rightParenthesisCount) {
            compoundSQLStatement.append(createQuerySQL(CoreUtil.getSubList(sortedCriterions, offset, count), module, psf, sessionFactory, queryValues,
                   tieMap, propertyMap, restrictionMap));
        } else { //there must not be any opening parentheses left:
            throw new IllegalArgumentException(L10nUtil.getMessage(MessageCodes.UNBALANCED_SET_PARENTHESES, DefaultMessages.UNBALANCED_SET_PARENTHESES));
        }
    } else {
        compoundSQLStatement.append(createQuerySQL(sortedCriterions, module, psf, sessionFactory, queryValues, tieMap, propertyMap, restrictionMap));
    }
    return compoundSQLStatement;
}
```

Listing 4.64: The `createCompoundQuerySQL` utility method splits a compound query by set operations and applies `createQuerySQL` to each basic query to build the compound query's SQL statement.

Assuming an RDBMS which supports set operations by explicit operator keywords[99], a SQL statement for the whole compound query is finally assembled by concatenating queries statements again and inserting set operator keywords (listing 4.64). The algorithm maintains parentheses for grouping of set operations and relies on well-formed query input, which is guaranteed after the aforegoing validation by the service method implementation. The native compound SQL query result statement is prepared using `org.hibernate.createSQLQuery`, ready for binding query parameter values and execution (listing 4.65). Result list filtering is implemented in `createQuerySQL` by injecting filter terms to each basic query of the compound query as shown in listing 4.60 line 10, 11, 19, 25, 33, 42 and 46.

```java
public static Query createDynamicQuery(CriteriaInstantVO criteriaInstantVO, DBModule module, PSFVO psf, SessionFactory sessionFactory,
    CriterionTieDao criterionTieDao, CriterionPropertyDao criterionPropertyDao, CriterionRestrictionDao criterionRestrictionDao) throws Exception {

    ArrayList<QueryParameterValue> queryValues = new ArrayList<QueryParameterValue>(); //a list of QueryParameterValue helper objects will be populated
         for query parameter binding
    HashMap<NamedParameterValues,Object> namedParameterValuesCache = new HashMap<NamedParameterValues,Object>(); //a cache for derived comparison values
         (NamedParameterValues enumeration: TIME, USER, USER_DEPARTMENT, IDENTITY, IDENTITY_DEPARTMENT)
    //build the SQL query statement string:
    StringBuilder sqlQuery = createCompoundQuerySQL(criteriaInstantVO, module, psf, sessionFactory, queryValues,
        createCriterionTieMap(criterionTieDao),
        createCriterionPropertyMap(module, criterionPropertyDao),
        createCriterionRestrictionMap(criterionRestrictionDao));

    Class rootEntityClass = getRootEntityClass(module);
    SQLQuery query;
    if (psf != null) { //handle pagination and sorting:
        if (psf.getUpdateRowCount()) { //perform a separate count query to obtain the total number of records
            StringBuilder countStatement = new StringBuilder("select count(*) from (");
            countStatement.append(sqlQuery);
            countStatement.append(") as resultset");
            Query countQuery = sessionFactory.getCurrentSession().createSQLQuery(countStatement.toString());
            setQueryValues(countQuery,queryValues,namedParameterValuesCache);
            psf.setRowCount(new Long(countQuery.uniqueResult().toString()));
        }

        String sortField = psf.getSortField();
            if (sortField != null  sortField.length() > 0) { //sort by a root entity column:
                sqlQuery.append(" order by "); //SQL-92, thus supported by majority of RDBMSs
                //convert the sort field name (association path) to the unified root column name returned by createQuerySQL, e.g.
                //"department" -> "department_fk"
                sqlQuery.append(getPropertyColumnName(rootEntityClass, sortField, sessionFactory));
                    if (psf.getSortOrder()) { //sort order:
                        sqlQuery.append(" asc");
                    } else {
                        sqlQuery.append(" desc");
                    }
                }

        //create the final org.hibernate.Query object:
            query = sessionFactory.getCurrentSession().createSQLQuery(sqlQuery.toString());
        //pagination:
                if (psf.getFirst() != null) {
                    query.setFirstResult(psf.getFirst());
                }
                if (psf.getPageSize() != null) {
                    query.setMaxResults(psf.getPageSize());
                }

    } else { //no PSF
        query = sessionFactory.getCurrentSession().createSQLQuery(sqlQuery.toString());
    }

    setQueryValues(query, queryValues, namedParameterValuesCache); //bind query parameter values
    //The custom query result of a native SQL query will be a list of Object[]. A "cast" to List<Entity> can be set up using the org.hibernate.Query.
         addEntity, since result set columns were named properly (listing 4.63).
    query.addEntity(rootEntityClass);

    return query;
}
```

Listing 4.65: The `createDynamicQuery` creates the compound query statement by invoking `createCompoundQuerySQL`. It finally binds query parameter values and applies sorting and pagination. The return value is an `org.hibernate.SQLQuery` object capable of listing root entity instances, ready for use by `findByCriteria` (listing 4.61).

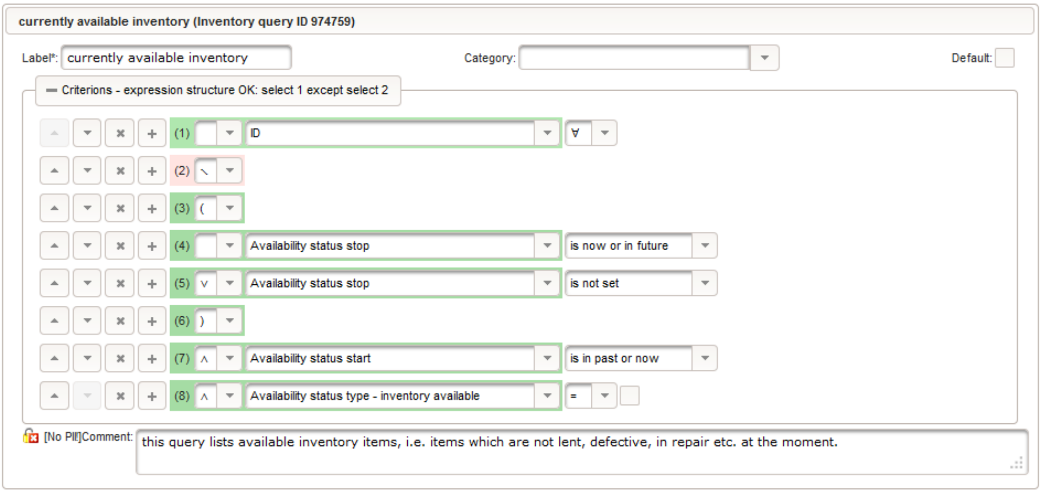
In order to successfully execute a compound SQL query, RDBMS require SQL queries as returned by `createQuerySQL`, producing *union compatible* result sets. This is achieved by patching the compiled SQL `SELECT` statement's projection column list. In addition, the sanitized column name list ensures uniform column names to make `org.hibernate.SQLQuery.addEntity` work. This will cause Hibernate to "cast" a native SQL query's result list (row as `Object[]`) by marshalling it into a collection of root entity objects (row as managed entity object). Hereby, the presented dynamic query implementation behaves like any regular Criteria API finder method after all.

[99]Set operator keyword can be specified in a configuration file. PostgreSQL: `INTERSECT`, `UNION`, `EXCEPT`; Oracle Database: `INTERSECT`, `UNION` and `EXCEPT` or `MINUS`.

4.9.3. Application Examples

Examples in previous sections emphasized the use case of subject eligibility criteria, which is supported by the query editor of the proband module. Nevertheless, query editors of remaining application modules turned out indispensable in order to keep control of data managed by the system overall. This includes abilities to identify inconsistent, missing or duplicate data entered by users as well as performing basic *reporting* tasks.

Short example queries shown in figure 4.126, 4.127, 4.128, 4.129, 4.130 and 4.131 demonstrate how users are able to achieve these tasks themselves by utilizing query editors, without the need of interventions by a database administrator.


currently available inventory (Inventory query ID 974759)

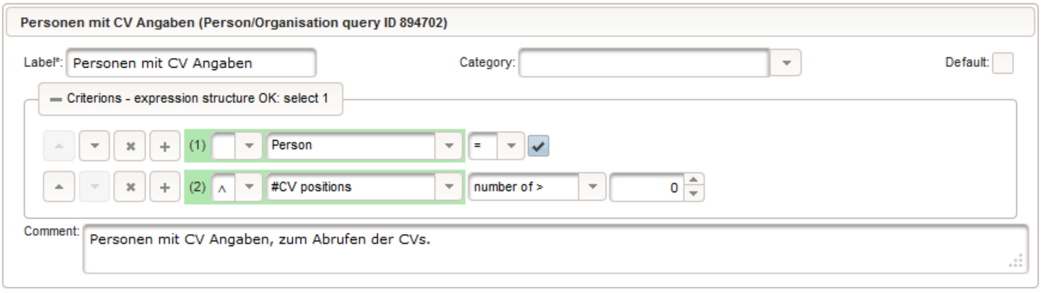
Label*: currently available inventory
Category:
Default:

Criterions - expression structure OK: select 1 except select 2

(1)		ID	∀
(2)	∖		
(3)	(		
(4)		Availability status stop	is now or in future
(5)	∨	Availability status stop	is not set
(6)	)		
(7)	∧	Availability status start	is in past or now
(8)	∧	Availability status type - inventory available	=

[No PII]Comment: this query lists available inventory items, i.e. items which are not lent, defective, in repair etc. at the moment.

Figure 4.126.: When selecting an inventory in the booking calendar, it is helpful to pick from inventory items currently not N/A. This compound query shows how to express the *set complement* construct.


Personen mit CV Angaben (Person/Organisation query ID 894702)

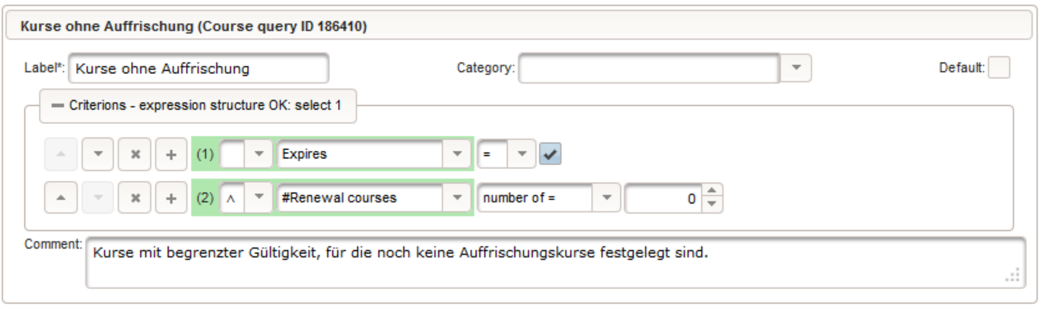
Label*: Personen mit CV Angaben
Category:
Default:

Criterions - expression structure OK: select 1

(1)		Person	=	✓
(2)	∧	#CV positions	number of >	0

Comment: Personen mit CV Angaben, zum Abrufen der CVs.

Figure 4.127.: This person/organisation query will list `Staff` records with a non-empty CV. It can be used to generate a single PDF document containing CV's of persons listed in the search result.



Kurse ohne Auffrischung (Course query ID 186410)

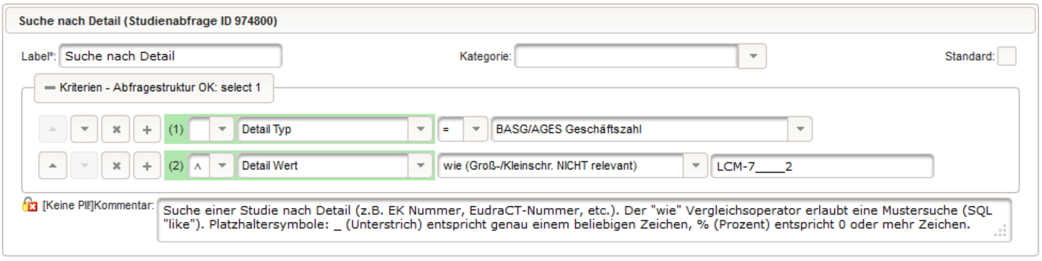
Label*: Kurse ohne Auffrischung
Category:
Default:

Criterions - expression structure OK: select 1

(1) Expires = ✓
(2) ∧ #Renewal courses number of = 0

Comment: Kurse mit begrenzter Gültigkeit, für die noch keine Auffrischungskurse festgelegt sind.

Figure 4.128.: The course query shown is an alternative to course overview lists from section 4.6.7.



Suche nach Detail (Studienabfrage ID 974800)

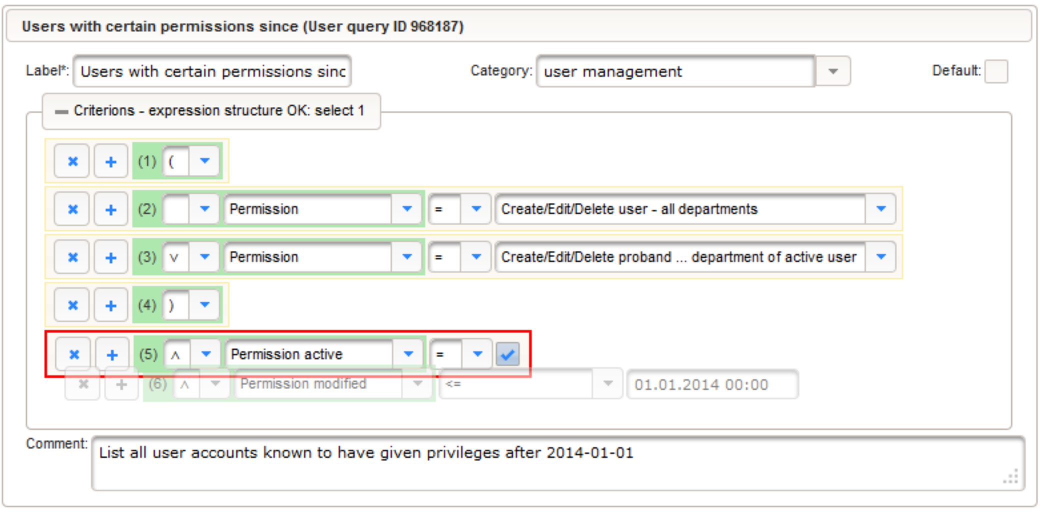
Label*: Suche nach Detail
Kategorie:
Standard:

Kriterien - Abfragestruktur OK: select 1

(1) Detail Typ = BASG/AGES Geschäftszahl
(2) ∧ Detail Wert wie (Groß-/Kleinschr. NICHT relevant) LCM-7____2

[Keine PII]Kommentar: Suche einer Studie nach Detail (z.B. EK Nummer, EudraCT-Nummer, etc.). Der "wie" Vergleichsoperator erlaubt eine Mustersuche (SQL "like"). Platzhaltersymbole: _ (Unterstrich) entspricht genau einem beliebigen Zeichen, % (Prozent) entspricht 0 oder mehr Zeichen.

Figure 4.129.: This trial query finds an trial with the specified property value, persisted by a `TrialTagValue` detail record.



Users with certain permissions since (User query ID 968187)

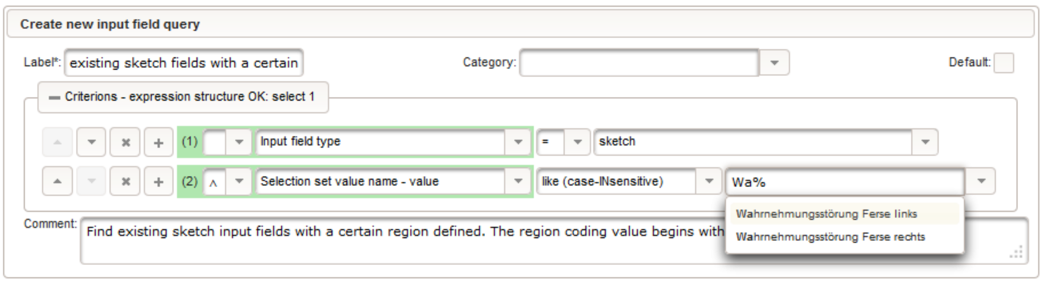
Label*: Users with certain permissions sinc
Category: user management
Default:

Criterions - expression structure OK: select 1

(1) (
(2) Permission = Create/Edit/Delete user - all departments
(3) ∨ Permission = Create/Edit/Delete proband ... department of active user
(4))
(5) ∧ Permission active = ✓
(6) ∧ Permission modified <= 01.01.2014 00:00

Comment: List all user accounts known to have given privileges after 2014-01-01

Figure 4.130.: This screen shows a query to identify user accounts with specific privileges. Query editor UIs support rearranging criterion input rows by drag-and-drop.


Create new input field query

Label*: existing sketch fields with a certain

Category:

Default:

Criterions - expression structure OK: select 1

(1) Input field type = sketch

(2) ∧ Selection set value name - value like (case-INsensitive) Wa%

Wahrnehmungsstörung Ferse links

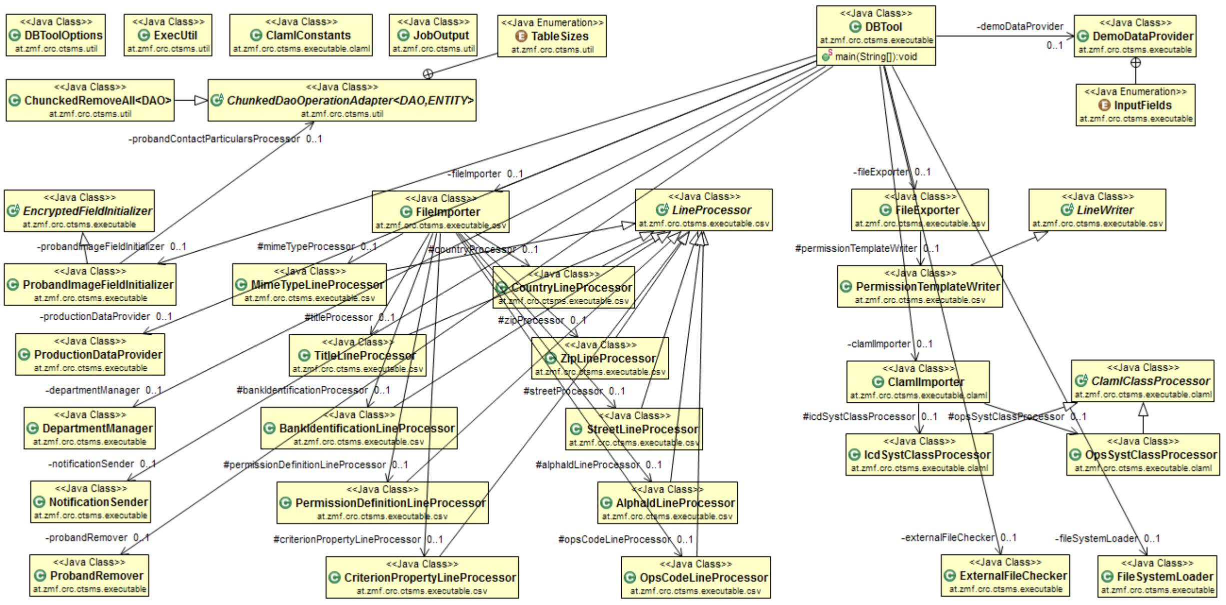
Wahrnehmungsstörung Ferse rechts

Comment: Find existing sketch input fields with a certain region defined. The region coding value begins with

Figure 4.131.: Before creating another input field for a new inquiry form, users can check for existing input fields suitable for storing the specific subject data they intend to capture.

4.10. Command Line Interface (CLI)

Setup and maintenance of a database application adds up to an appreciable issue for a system of considerable size. Creating the RDBMS schema might be as simple as executing the `schema-create.sql` SQL script generated by AndroMDA, but system setup usually comprises more complex initialization steps such as loading master data from XML data sources into tables, typically requiring an importer program. While these tasks are manual and nonrecurring interventions done by a system administrator, enterprise applications tend to comprise automated jobs, running periodically. Simple recurring maintenance tasks like creating SQL dumps for backup purposes can be solved using the server OS' built-in batch processing and job scheduling capabilities. Other scheduled tasks run application logic not wired with the UI directly, such as queue processing.



<<Java Class>> DBToolOptions at.zmf.crc.ctsms.util
<<Java Class>> ExecUtil at.zmf.crc.ctsms.util
<<Java Class>> ClamlConstants at.zmf.crc.ctsms.executable.claml
<<Java Class>> JobOutput at.zmf.crc.ctsms.util
<<Java Enumeration>> TableSizes at.zmf.crc.ctsms.util
<<Java Class>> DBTool at.zmf.crc.ctsms.executable
main(String[]):void
-demoDataProvider 0..1
<<Java Class>> DemoDataProvider at.zmf.crc.ctsms.executable
<<Java Enumeration>> InputFields at.zmf.crc.ctsms.executable
<<Java Class>> ChunckedRemoveAll<DAO> at.zmf.crc.ctsms.util
<<Java Class>> ChunkedDaoOperationAdapter<DAO,ENTITY> at.zmf.crc.ctsms.util
-probandContactParticularsProcessor 0..1
-fileImporter 0..1
-fileExporter 0..1
<<Java Class>> EncryptedFieldInitializer at.zmf.crc.ctsms.executable
<<Java Class>> FileImporter at.zmf.crc.ctsms.executable.csv
<<Java Class>> LineProcessor at.zmf.crc.ctsms.executable.csv
<<Java Class>> FileExporter at.zmf.crc.ctsms.executable.csv
<<Java Class>> LineWriter at.zmf.crc.ctsms.executable.csv
-probandImageFieldInitializer 0..1
#mimeTypeProcessor 0..1
#countryProcessor 0..1
#permissionTemplateWriter 0..1
<<Java Class>> ProbandImageFieldInitializer at.zmf.crc.ctsms.executable
<<Java Class>> MimeTypeLineProcessor at.zmf.crc.ctsms.executable.csv
<<Java Class>> CountryLineProcessor at.zmf.crc.ctsms.executable.csv
<<Java Class>> PermissionTemplateWriter at.zmf.crc.ctsms.executable.csv
-productionDataProvider 0..1
#titleProcessor 0..1
#zipProcessor 0..1
-clamlImporter 0..1
<<Java Class>> ProductionDataProvider at.zmf.crc.ctsms.executable
<<Java Class>> TitleLineProcessor at.zmf.crc.ctsms.executable.csv
<<Java Class>> ZipLineProcessor at.zmf.crc.ctsms.executable.csv
<<Java Class>> ClamlImporter at.zmf.crc.ctsms.executable.claml
<<Java Class>> ClamlClassProcessor at.zmf.crc.ctsms.executable.claml
-departmentManager 0..1
#bankIdentificationProcessor 0..1
#streetProcessor 0..1
#icdSystClassProcessor 0..1
#opsSystClassProcessor 0..1
<<Java Class>> DepartmentManager at.zmf.crc.ctsms.executable
<<Java Class>> BankIdentificationLineProcessor at.zmf.crc.ctsms.executable.csv
<<Java Class>> StreetLineProcessor at.zmf.crc.ctsms.executable.csv
<<Java Class>> IcdSystClassProcessor at.zmf.crc.ctsms.executable.claml
<<Java Class>> OpsSystClassProcessor at.zmf.crc.ctsms.executable.claml
-notificationSender 0..1
#permissionDefinitionLineProcessor 0..1
#alphaIdLineProcessor 0..1
<<Java Class>> NotificationSender at.zmf.crc.ctsms.executable
<<Java Class>> PermissionDefinitionLineProcessor at.zmf.crc.ctsms.executable.csv
<<Java Class>> AlphaIdLineProcessor at.zmf.crc.ctsms.executable.csv
-probandRemover 0..1
#criterionPropertyLineProcessor 0..1
#opsCodeLineProcessor 0..1
-externalFileChecker 0..1
-fileSystemLoader 0..1
<<Java Class>> ProbandRemover at.zmf.crc.ctsms.executable
<<Java Class>> CriterionPropertyLineProcessor at.zmf.crc.ctsms.executable.csv
<<Java Class>> OpsCodeLineProcessor at.zmf.crc.ctsms.executable.csv
<<Java Class>> ExternalFileChecker at.zmf.crc.ctsms.executable
<<Java Class>> FileSystemLoader at.zmf.crc.ctsms.executable

Figure 4.132.: The `at.zmf.crc.ctsms.executable` namespace is part of the core tier Java project and organizes logic wired to the `DBTool.main` entry point into numerous classes.

```
ctsms@phoenix.joanneum.at:~$ /ctsms/dbtool.sh --help
14-01-03 23:25:44: starting...
Phoenix Clinical Trial Management System
version: 1.0.8
site: phoenix.joanneum.at
task: print help
usage: dbtool [task <arg>] [option1 <arg> option2 <arg> ...]
 -air,--alpha_id_revision <arg>                        option: alpha id catalogue revision
 -c,--clear                                            task: clear database
 -cd,--create_department                               task: create new department
 -cdi,--create_department_interactive                  task: create new department (interactive)
 -cdif,--create_demo_input_fields                      task: create demo input fields
 -cdp,--change_department_password                     task: change department password
 -cdpi,--change_department_password_interactive        task: change department password (interactive)
 -cdpu,--change_department_password_user               task: change department password (user authentication)
 -cu,--create_user                                     task: create new user
 -cui,--create_user_interactive                        task: create new user (interactive)
 -dlk,--department_l10n_key <arg>                      option: department l10n name
 -dm,--delete_missing                                  task: scan for missing external files and delete references from
      DB
 -do,--delete_orphaned                                 task: scan for orphaned external files and delete them
 -dp,--department_password <arg>                       option: department password
 -e,--encoding <arg>                                   option: encoding of csv/text file to import
 -epdt,--export_permission_definition_template <arg>   task: export permission definition template as csv/text file
 -er,--email_recipients <arg>                          option: send output to email recipients
 -f,--force                                            option: skip confirmation promt
 -h,--help                                             task: print help
 -i,--init                                             task: initialize db by insertig required setup records
 -iai,--import_alpha_ids <arg>                         task: import alpha ids
 -ib,--import_bank <arg>                               task: import bank identifications from csv/text file
 -ic,--import_country <arg>                            task: import country names from csv/text file
 -icf,--import_course_files <arg>                      task: import documents and files for a course entity
 -icp,--import_criterion_properties <arg>              task: import criterion properties tables
 -id,--entity_id <arg>                                 option: record id of entity
 -iif,--import_inventory_files <arg>                   task: import documents and files for an inventory entity
 -iis,--import_icd_syst <arg>                          task: import icd systematics categories
 -imc,--import_mime_course <arg>                       task: import course file mime types from csv/text file
 -imi,--import_mime_inventory <arg>                    task: import inventory file mime types from csv/text file
 -imifi,--import_mime_input_field_image <arg>          task: import input field image file mime types from csv/text file
 -imp,--import_mime_proband <arg>                      task: import proband file mime types from csv/text file
 -impi,--import_mime_input_field_image <arg>           task: import proband image file mime types from csv/text file
 -ims,--import_mime_staff <arg>                        task: import staff file mime types from csv/text file
 -imsi,--import_mime_staff_image <arg>                 task: import input staff image file mime types from csv/text file
 -imt,--import_mime_trial <arg>                        task: import trial file mime types from csv/text file
 -ioc,--import_ops_codes <arg>                         task: import ops codes
 -ios,--import_ops_syst <arg>                          task: import ops systematics categories
 -ipd,--import_permission_definitions <arg>            task: import permission definitions from csv/text file
 -ipf,--import_proband_files <arg>                     task: import documents and files for a proband entity
 -ipif,--initialize_proband_image_fields               task: initialize proband image fields
 -is,--import_street <arg>                             task: import street names from csv/text file
 -isf,--import_staff_files <arg>                       task: import documents and files for a staff entity
 -isr,--icd_syst_revision <arg>                        option: icd systematics catalogue revision
 -it,--import_title <arg>                              task: import titles from csv/text file
 -itf,--import_trial_files <arg>                       task: import documents and files for a trial entity
 -iz,--import_zip <arg>                                task: import zip codes from csv/text file
 -l,--limit <arg>                                      option: limit for number of (processed) records
 -ldd,--load_demo_data                                 task: load db with demo data records
 -ndp,--new_department_password <arg>                  option: new department password
 -ocr,--ops_code_revision <arg>                        option: ops catalogue revision
 -odp,--old_department_password <arg>                  option: old department password
 -osr,--ops_syst_revision <arg>                        option: ops systematics catalogue revision
 -p,--password <arg>                                   option: user password
 -pn,--prepare_notifications                           task: prepare notifications
 -pp,--permission_profiles <arg>                       option: list of permission profiles
 -rcf,--remove_course_files                            task: remove all files of a course entity
 -rif,--remove_inventory_files                         task: remove all files of an inventory entity
 -rp,--remove_probands                                 task: remove probands pending for auto-delete
 -rpf,--remove_proband_files                           task: remove all files of a proband entity
 -rsf,--remove_staff_files                             task: remove all files of a staff entity
 -rtf,--remove_trial_files                             task: remove all files of a trial entity
 -sl,--syst_lang <arg>                                 option: icd/ops systematics label language
 -sm,--scan_missing                                    task: scan for missing external files
 -sn,--send_notifications                              task: send notifications
 -so,--scan_orphaned                                   task: scan for orphaned external files
 -u,--username <arg>                                   option: username
```

Listing 4.66: A batch file[100] is provided for launching the Java interpreter to execute `DBTool.main` of the `ctsms-core-x.y.z.jar` package from a system console with a simple "`dbtool`" command. When executing the program, a program mode is selected by specifying a *task* command line parameter. Additional *option* parameters are available or even mandatory, depending on the task. Both task and option parameters can either expect an additional argument or not (switch).

A final, remarkable component was introduced in order to facilitate the matters mentioned before. Instead of preparing SQL scripts or separate programs written in a scripting language[101] or using third-party tools, it was decided to utilize the system's DAO and service layer as well as existing utility classes, located in the application's core tier and common projects (figure 2.16). The core tier's `at.zmf.crc.ctsms.executable` namespace (figure 4.132) provides an entry point (`DBTool.main`) for running a CLI program by invoking the Java interpreter program. This program utilizes Apache Commons CLI (Strachan et al., 2013) to expose an extensible suite of features using command line parameters (listing 4.66), which are outlined in subsequent sections. Each of its program modes ("tasks") allows to initialize or modify a specific group of related tables. To manipulate records, DAO operations are invoked directly, unless transactional rollback is desired for error scenarios of critical tasks, operating on production data. In this case, service implementation classes are instantiated Spring (`@Autowired`) to harness the system's service layer. The CLI tool's logic invokes regular service methods that are designed to support the presentation layer, as well as specificially implemented ones (e.g. `ToolsService.changeDepartmentPassword`, section 4.10.2) in order to be able to abort and revert sequences of CRUD operations. Whenever a potentially large number of records is affected, an additional confirmation prompt requires the operating administrator's attention to continue, unless the `--force` option switch was specified.

4.10.1. Database Initialisation

4.10.1.1. Configuration Records

Master data comprising itemized records, expected to be modified rarely and manually, can be created by running the `--init` task. This program task is meant to be run once after RDBMS schema creation only. Record contents are hard-coded in `ProductionDataProvider` and actually cover most master data entites from table 4.2 and 4.3, which are linked to domain entities by a regular association:

`InventoryCategory`	`TrialStatusType`	`PrivacyConsentStatusType`
`InventoryTag`	`TrialStatusAction`	`ProbandCategory`
`InventoryStatusType`	`TrialType`	`ProbandTag`
`MaintenanceType`	`SurveyStatusType`	`ProbandStatusType`
`StaffCategory`	`SponsoringType`	`AddressType`
`StaffTag`	`TrialTag`	`ContactDetailType`
`StaffStatusType`	`TeamMemberRole`	`FileFolderPreset`
`CvSection`	`TimelineEventType`	`HyperlinkCategory`
`CourseCategory`	`VisitType`	`JournalCategory`
`LecturerCompetence`	`ProbandListStatusType`	`NotificationType`
`CourseParticipationStatusType`	`ProbandListStatusLogLevel`	`Holiday`

4.10.1.2. Clear Database

While implementing the `dbtool` mode for creating configuration records described in the previous section, methods for removing created records were provided for testing purposes. This was finally extended and exposed by a separate task (`--clear`), allowing to empty *all* database tables. Basically, this is actually redundant since purging database contents can be done more quickly by executing `schema-drop.sql` and recreating the database schema subsequently. However, `ProductionDataProvider` methods for removing records are required by other parts of the CLI tool implementation anyway.

[100]`dbtool.bat` (Microsoft Windows), `dbtool.sh` (Linux, appendix C)
[101]e.g. Linux Shell, Perl or Microsoft Windows Batch scripts

```
public class ProductionDataProvider {

    @Autowired
    protected FileDao fileDao;

    ...

    private void removeFileRecords() {
        //the code below is equivalent to: (new ChunckedRemoveAll(fileDao)).processPages(TableSizes.BIG, null);
        ChunkedDaoOperationAdapter<FileDao, File> fileProcessor = new ChunkedDaoOperationAdapter<FileDao, File>(fileDao) {

            @Override
            protected Collection<File> loadAllSorted(int pageNumber, int pageSize) throws Exception {
                this.dao.loadAll(pageNumber, pageSize);
            }

            @Override //process records one-by-one: fileProcessor.processEach
            protected void process(File entity, Object passThrough) throws Exception {
                //this.dao.remove(entity);
            }

            @Override //process records page-wise: fileProcessor.processPages
            protected void process(Collection<File> page, Object passThrough) throws Exception {
                this.dao.remove(page); //one transaction per page of records
                //association ends are not updated
            }

        };
        Object passThrough = null;
        fileProcessor.processPages(TableSizes.BIG, passThrough);
    }

    ...
```

Listing 4.67: `ChunkedDaoOperationAdapter` provides basic functionality for iterating entities managed by a dedicated DAO. This example demonstrates how to remove `File` records page-wise by overriding `ChunkedDaoOperationAdapter` methods using an anonymous inner class[102]. Since purging tables is required frequently, it is implemented explicitly by `ChunkedRemoveAll`.

In order to empty tables, DAO `remove` methods were used, causing a `DELETE FROM ... WHERE id = ...` SQL statement per entity instance to delete. This reveals another drawback of ORM - the lack of bulk DML statement[103] support. Records have to be deleted on-by-one by iterating collections of managed entity instances. Even when tolerating low performance, the problem of a memory bottleneck remains if collections get large. An adapter allowing to perform DAO operations page-wise (`ChunkedDaoOperationAdapter`) was introduced as a solution for this case. The concrete `ChunkedRemoveAll` adapter implementation allows removing records safely (listing 4.67).

4.10.1.3. Test Data Generator

After configuration records have been created, a test data generator (`DemoDataProvider`) can be started using the `-ldd` task switch. During this long-running task, adjustable numbers of records are created for most relevant domain entities across system modules. Although data is generated randomly, the resulting content is meaningful in order to serve demonstration purposes. For instance, most frequent German first and last names are combined randomly for proband and employee names, or demographic distributions of academic degrees are considered etc. Essential workflows are simulated by updating records of stateful entities step-wise in order to produce histories as expected during real-life usage (e.g. figure 4.133).

[102]Apart from that, anonymous inner classes are avoided in general.
[103]e.g. `TRUNCATE TABLE` or `UPDATE` and `DELETE` statements without `WHERE` clause.

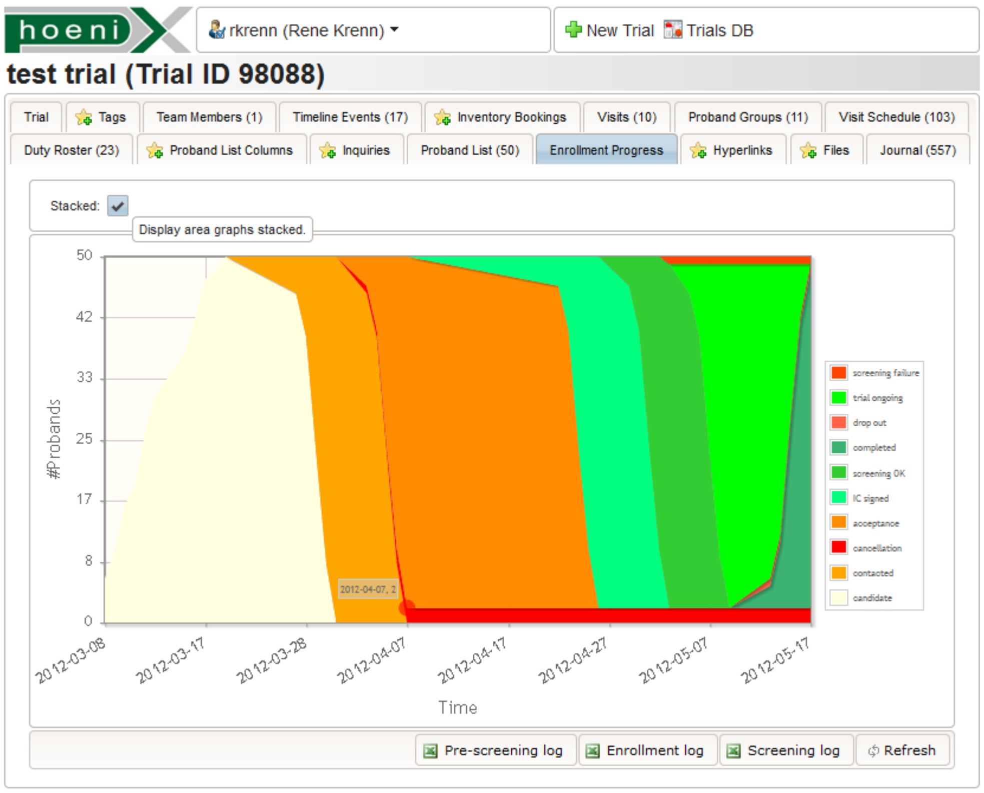


Figure 4.133.: This *enrollment chart* shows the enrollment progress of a trial with a pre-screening target of 50 recruitment probands. In "stacked" mode, it displays the distribution of probands per enrollment state over time. When considering the enrollment status as a *Markov chain* (Häggström, 2002, pp. 8-10), the trial simulation shown uses a probability of 0.2 for entering drop out states ("cancellation", "screening failure", "drop out" from figure 4.43) and equal probability for remaining states. However, probabilities are not reflected by the graph in general since the simulation retries transitions if a state was already visited before.

A test case against the heavily loaded EAV data model reveals expectations about general performance and required hardware resources:

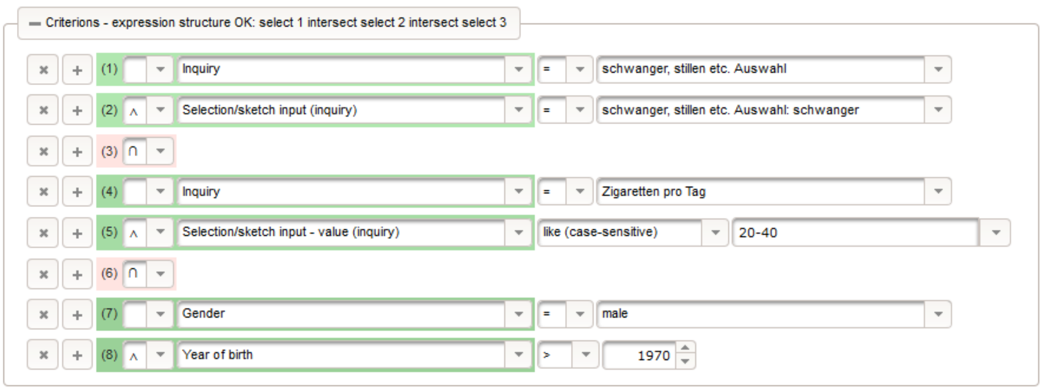
setup:

- 53500 subjects from a repository of 99.999 probands are recruited for 9 trials.
- Each trial has defines the same inquiry input form consisting of 48 input fields.
- This results in 2568000 randomly generated form values (`InquiryValue` records). Aside, app. $6.8 \cdot 10^6$ journal entries are created overall.

results:

- The PostgreSQL tablespace requires app. 5GB of disk storage.
- The test data load procedure was performed in three stages and took 23 hours.
- A simple eligibility criteria query (figure 4.134) hits several hundred probands and takes between 2.3s and 2.9s to display a result page.[104]

[104]For the test environment, application components are deployed on a single RHEL 6.2 x64 instance (appendix C). VMware virtualisation is used to create disposable test evironment images (CPU: two physical Intel Sandy Bridge i7 cores, RAM: 4GB, storage: FireWire 800 RAID 0.).



Criterions - expression structure OK: select 1 intersect select 2 intersect select 3

(1)		Inquiry	=	schwanger, stillen etc. Auswahl
(2)	∧	Selection/sketch input (inquiry)	=	schwanger, stillen etc. Auswahl: schwanger
(3)	∩			
(4)		Inquiry	=	Zigaretten pro Tag
(5)	∧	Selection/sketch input - value (inquiry)	like (case-sensitive)	20-40
(6)	∩			
(7)		Gender	=	male
(8)	∧	Year of birth	>	1970

Figure 4.134.: This proband query example was used to demonstrate the system's performance for the critical case of high-volume EAV data. The execution duration of the compound query depends on the result set sizes of basic queries, which are prone to slow full table scans and joining large tables. The data model design allows PostgreSQL's query planning to perform well, hence this query completes in seconds even in the case of millions of captured form values.

Running the test data generator is supposed to represent a basic *integration test* of system modules at the service layer level. Although there is no result assertion, there is a high probability the program execution will fail if a service method implementation was changed. However, `DemoDataProvider` currently covers 7.6% of overall service methods only:

```
ToolsService.addUser
ToolsService.completeTitle
SelectionSetService.getStaffCategories
StaffService.addStaff
UserService.updateUser
SelectionSetService.getStaffCategories
SelectionSetService.getInventoryCategories
InventoryService.addInventory
CourseService.getCourse
SelectionSetService.getCourseCategories
SelectionSetService.getCvSections
CourseService.addCourse
CourseService.addCourseParticipationStatusEntry
SelectionSetService.getProbandCategories
ProbandService.addProband
InputFieldService.addInputField
InputFieldService.addSelectionSetValue
InputFieldService.getInputField
TrialService.addInquiry
TrialService.addProbandListEntryTag
SelectionSetService.getInitialTrialStatusTypes
SelectionSetService.getTrialTypes
SelectionSetService.getSponsoringTypes
SelectionSetService.getSurveyStatusTypes
TrialService.addTrial
TrialService.addTeamMember
TrialService.addVisit
TrialService.addProbandGroup
TrialService.addVisitScheduleItem
TrialService.updateTrial
TrialService.addProbandListEntry
SelectionSetService.getAllProbandListStatusTypes
TrialService.setProbandListEntryTagValues
ProbandService.setInquiryValues
TrialService.addProbandListStatusEntry
TrialService.getProbandListEntry
SelectionSetService.getProbandListStatusTypeTransitions
TrialService.addTimelineEvent
TrialService.getVisitScheduleItemInterval
StaffService.addDutyRosterTurn
FileService.getFileFolders
FileService.addFile
```

Apart from that, unit test stubs for DAO transformation and service methods are generated by AndroMDA. This is achieved with additional code generation templates[105] registered at the `<!-- cartridge-template merge-point -->` in the Spring/Hibernate cartridge merge mappings. Implementation class stubs for `org.springframework.test.context.testng.AbstractTransactionalTestNGSpringContextTests` unit tests are generated once initially according to DAO and service interfaces implementation class stubs. Hand-crafted unit test code can then be placed in these test implementation classes to create a unit test suite for the core tier Java project. The test suite can be invoked from command-line using the Maven `test` build goal option finally.

[105] The templates were taken from an early development snapshot of the Reaction (Spat et al., 2012) server Java project codebase, provided by JR Health.

4.10.1.4. Encrypted Field Value Initialisation

Since the Phoenix CTMS is still a growing software project, new features are introduced with every iteration. Domain model extension are expected and require to update the existing database schema when a new version is rolled out. During application startup, Hibernate handles schema updates silently as far as possible[106], e.g. by performing an `ALTER TABLE` DDL statement for adding another table column to fit a modified entity. An obvious problem arises, if the table already contains records and a new column has to be declared `NOT NULL`, such as *encrypted* columns. The application's encryption utilities are needed to be able to initialize existing records with encrypted `null` values. For this purpose, the abstract `EncryptedFieldInitializer` can be extended to implement a migration task for a specific version rollout, as it was the case for example with the introduction of (encrypted) proband photos (`ProbandImageFieldInitializer`).

4.10.2. Interactive Procedures

An interactive console dialog allows a system administrator to supply credentials for `dbtool` modes, which require to invoke cryptographic operations:

Create department: The `-cdi` tasks starts an interactive session to create a new department. In order to create and encrypt the new department's key as outlined in section 4.2.3, the user is prompted for the department's identifier for l10n bundles and a department passphrase, which has to be confirmed.

Change department password: A department's passphrase can be changed using the `-cdpi` mode. This requires selecting an existing department and specifying the existing and new passphrase to start a complex procedure comprising the re-encryption of user password histories.

Create user: After a new department has been created, an initial user account is required to log in using the web frontend and set up additional user accounts. The `-cui` mode starts a dialog to create a user including password and permission profiles (listing 4.68). The user's key pair is created behind the scenes (section 4.3.3).

[106]This behaviour is controlled with the `hibernate.hbm2ddl.auto` setting.

```
ctsms@phoenix.joanneum.at:~$ /ctsms/dbtool.sh --cui
14-01-06 00:34:03: starting...
Phoenix Clinical Trial Management System
version: 1.0.8
site: phoenix.joanneum.at
task: create new user (interactive)
departments:
ID 145282: zmf_pieber_crc - Pieber-CRC
ID 145283: zmf - ZMF
ID 145284: jr_health - Joanneum Research HEALTH
ID 145285: tu_graz_iicm - TU Graz IICM
ID 145286: test_department - Test Department
ID 809826: diabetes_outpatient_clinic - Diabetes Outpatient Clinic
enter department id for new user:809826
enter department password:***secrect department passphrase***
enter new username:newuser
password policy requirements:
a minimum of 8 characters
a maximum of 32 characters
a minimum of 2 small letters
a minimum of 1 digits
a minimum of 1 special characters ^°!"§$%&/{([])}=?\‘+*-#’-_.,:;|<>
admins may ignore this password policy when setting passwords
a random password that meets password policy requirements:
-6tobP*7
enter desired user password:-6tobP*8
permission profiles:
...
USER_USER_DEPARTMENT: Users - Create/Edit/Delete user - department of active user
...
enter separated list of permission profiles:..., USER_USER_DEPARTMENT, ...
user created
permission profile INVENTORY_VIEW_USER_DEPARTMENT added
permission profile STAFF_DETAIL_IDENTITY added
permission profile COURSE_VIEW_USER_DEPARTMENT added
permission profile TRIAL_VIEW_USER_DEPARTMENT added
permission profile PROBAND_VIEW_USER_DEPARTMENT added
permission profile USER_USER_DEPARTMENT added
permission profile INPUT_FIELD_VIEW added
permission profile INVENTORY_NO_SEARCH added
permission profile STAFF_NO_SEARCH added
permission profile COURSE_NO_SEARCH added
permission profile TRIAL_NO_SEARCH added
permission profile USER_MASTER_SEARCH added
permission profile INPUT_FIELD_NO_SEARCH added
permission profile PROBAND_NO_SEARCH added
```

Listing 4.68: To be able to manage user accounts using the web UI, an initial user has to be created at first. This transcript shows how to create a new user with the interactive `dbtool -cui` task. The system administrator is prompted to select a department and specify the username and password.

To automate the entire user setup by means of batch processing, a corresponding non-interactive task is provided for each procedure presented (`-cd`, `-cdp`, `-cu`). Credentials are specified by option arguments in this case.

4.10.3. Importers

A series of *importers* allow to import master data from table 4.3, whose records are linked to domain entities by unconstrained associations. According to the format of master data sources, the importers provided can be divided into three groups: CSV, Classification Markup Language (ClaML) and file system importers. Basically, all importer tasks have a stable memory consumption and are implemented using the *visitor pattern*.

4.10.3.1. Character-Separated Values (CSV)

The CSV importer logic is encapsulated by `FileImporter` and the abstract `LineProcessor` class, capable of processing a large number of rows from a file in a tabular text format. `LineProcessor` implementations map a data file's rows to records of corresponding database tables. A row's column values are converted into managed entities to finally insert records using DAO create operations. More in detail, a `LineProcessor` is required to support features below:

- column ordering
- comment stripper
- removal of duplicates
- row skipping and aggregation
- column value enclosing characters and trimming
- column value and row delimiter characters
- validation of column values and row consistency
- character encoding (`--encoding` option switch)
- row count limitation (`--limit` option switch)

Since imports will be performed occasionally to update master data, populated tables contents are usually purged before any `INSERT`. In contrast to Configuration Records (section 4.10.1.1), involved types of master data cause no foreign key constraint violation when removed. While executing the first three CSV imports presented next is mandatory for proper operation, the remaining ones are optional and may require licensed source data.

Search criteria importer The `CriterionPropertyLineProcessor LineProcessor` is used to import entity column definitions required by the dynamic database query feature (table 4.17). A CSV file listing all entity columns to be made available to the query editor is prepared using spreadsheet software. To import the file, `dbtool` is launched with the `-icp` task argument and the file's name.

Permission definitions importer Permission definitions required by the authorization layer can be provided by a CSV file, processed by the `PermissionDefinitionLineProcessor` when starting the `-ipd` task. As stated in section 4.5.2, a service method is considered unauthorized by default, if no permission could be found. While it is good practice to deny by default if a `User` item lacks `UserPermissionProfile` records, missing permission definitions have the same effect. As a consequence, a service method accidentally left out in the permission definition file will be always blocked, disregarding a user's privileges. Since this was an issue during development and the definition file will contain a considerable number of rows, an exporter was implemented (`PermissionTemplateWriter`). When executing the `-epdt` task, it generates a *template* CSV file by utilizing reflection to list all service methods requiring authorization. For each service method, corresponding input parameter names are included to found the structure of the format expected by the importer (table 4.9).

Mime type definitions importer For document management, a list of accepted file content types needs to be registered per system module (section 4.7.2). Unless a restriction to specific file types is desired, a list as complete as possible should be loaded using the import task for each corresponding system module (`-imi`, `-ims`, `-imc`, `-imt`, `-imp`). Furthermore, supported `image/*` file types have to be defined to enable uploading of proband (`-impi`) and employee (`-imsi`) photos as well as sketch background images (`-imifi`). A list of content type names according to RFC RFC 4288 including known file name extensions can be found in the `mime.types` file, contained in distributions of the Apache HTTP Server. This tab-separated text file is processed by `MimeTypeLineProcessor`.

Bank identification importer The UI provides `p:autoComplete` input elements for specifying banking institutions by bank name and either bank code number or Bank Identification Code (BIC). If a source file for this kind of master data is available, it can be imported into a corresponding master data table for suggestions by utilizing `BankIdentificationLineProcessor` when running the `-ib` task.

Titles/academic degrees importer A single-column text file containing desired suggestions for titles and academic degrees is imported by the `TitleLineProcessor` when running the `-it` task.

Importers for postal address master data Master data for suggestions when typing parts of a postal address (figure 4.32) is organized by three tables (figure 4.135, 4.136, 4.137).

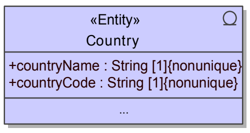

«Entity»
Country
+countryName : String [1]{nonunique}
+countryCode : String [1]{nonunique}
...

Figure 4.135.: The `Country` master data entity.

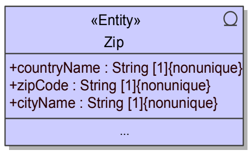

«Entity»
Zip
+countryName : String [1]{nonunique}
+zipCode : String [1]{nonunique}
+cityName : String [1]{nonunique}
...

Figure 4.136.: The `Zip` master data entity.

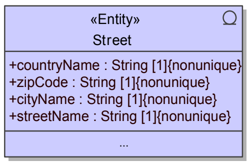

«Entity»
Street
+countryName : String [1]{nonunique}
+zipCode : String [1]{nonunique}
+cityName : String [1]{nonunique}
+streetName : String [1]{nonunique}
...

Figure 4.137.: The `Street` master data entity.

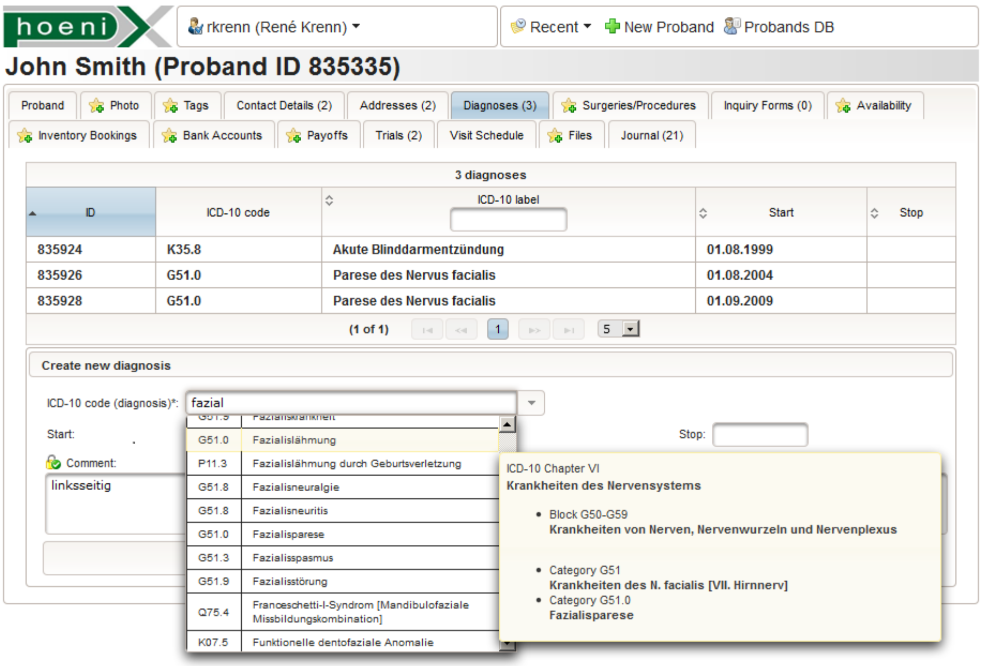
Each table can be populated with a separate importer task:

Country names importer: A two-column CSV file with country names and (optional) ISO 3166 country codes can be imported into the `Country` entity table (figure 4.135) by running the `-ic` task.

ZIP codes/city names importer: By using the `-iz` task, the current implementation of `ZipLineProcessor` extracts ZIP codes and city names when importing the dump of a national address directory, as available from Bundesanstalt Statistik Österreich (ÖSTAT) (Statistik Austria, 2011). To support establishing a multi-national address directory, the `Zip` table (figure 4.136) contains an additional country discriminator (`Zip.countryName`). In order to uniquely identify a record by `countryName`, `zipCode` and `cityName`, duplicates are filtered during the import.

Street names importer: The `-is` task will import the complete list of street names from the same address directory dump used before. `StreetLineProcessor` filters duplicates to uniquely identify a `Street` record by `countryName`, `zipCode`, `cityName` and `streetName`. This is particularly necessary since the dump will most likely contain additional columns for more detail such as commune names or even house numbers.

Alpha-ID identification code for diagnoses (Alpha-ID) importer Since longitudinal data entry is currently not supported by EDC features of the Phoenix CTMS, separate detail entities such as `Diagnosis` or `Procedure` (figure 4.139) are introduced to store particular longitudinal records of probands as required. In addition, the concerned data model is supposed to "encode" information by referencing standardized master data. Capturing and coding diagnoses is outlined in detail for example in (Haas, 2004, pp. 405). After separate investigation, it was decided to encode diagnoses by referencing Alpha-ID identification code for diagnoses (Alpha-ID) records (*Alpha-ID - Identifikationsnummer für Diagnosen* 2013) when entering a proband's patient history (figure 4.138).


rkrenn (René Krenn) ▾ | Recent ▾ | New Proband | Probands DB

John Smith (Proband ID 835335)

Proband | Photo | Tags | Contact Details (2) | Addresses (2) | Diagnoses (3) | Surgeries/Procedures | Inquiry Forms (0) | Availability | Inventory Bookings | Bank Accounts | Payoffs | Trials (2) | Visit Schedule | Files | Journal (21)

3 diagnoses

ID	ICD-10 code	ICD-10 label	Start	Stop
835924	K35.8	Akute Blinddarmentzündung	01.08.1999	
835926	G51.0	Parese des Nervus facialis	01.08.2004	
835928	G51.0	Parese des Nervus facialis	01.09.2009	

(1 of 1) 1 5

Create new diagnosis

ICD-10 code (diagnosis)*: fazial

Start:

Stop:

Comment: linksseitig

G51.0	Fazialislähmung
P11.3	Fazialislähmung durch Geburtsverletzung
G51.8	Fazialisneuralgie
G51.8	Fazialisneuritis
G51.0	Fazialisparese
G51.3	Fazialisspasmus
G51.9	Fazialisstörung
Q75.4	Franceschetti-I-Syndrom [Mandibulofaziale Missbildungskombination]
K07.5	Funktionelle dentofaziale Anomalie

ICD-10 Chapter VI
Krankheiten des Nervensystems

- Block G50-G59
  Krankheiten von Nerven, Nervenwurzeln und Nervenplexus

- Category G51
  Krankheiten des N. facialis [VII. Hirnnerv]
- Category G51.0
  Fazialisparese

Figure 4.138.: Diagnoses are represented by the `Diagnosis` proband detail entity. Each `Diagnosis` record refers a Alpha-ID master data record and holds optional time interval information. To specify the Alpha-ID reference, users selects it after looking up a matching `AlphaId.text` using a `p:autoComplete` input element. The ICD-10 systematics entry belonging to the Alpha-ID record is displayed in a `itemtip` box when moving over an entry in the suggestion list.

The Alpha-ID source file (`AlphaId.text`) is amended and released yearly by Deutsches Institut für Medizinische Dokumentation und Information (DIMDI). It alphabetically lists common names and synonyms for all diseases and related health problems compiled in the International Classification of Diseases (ICD-10)-GM (German Modification) systematics. The *ICD-10 code* (`primaryCode` or `optionalCode`) refers an ICD-10 systematics entry, mapped by the `IcdSyst` graph (figure 4.141).

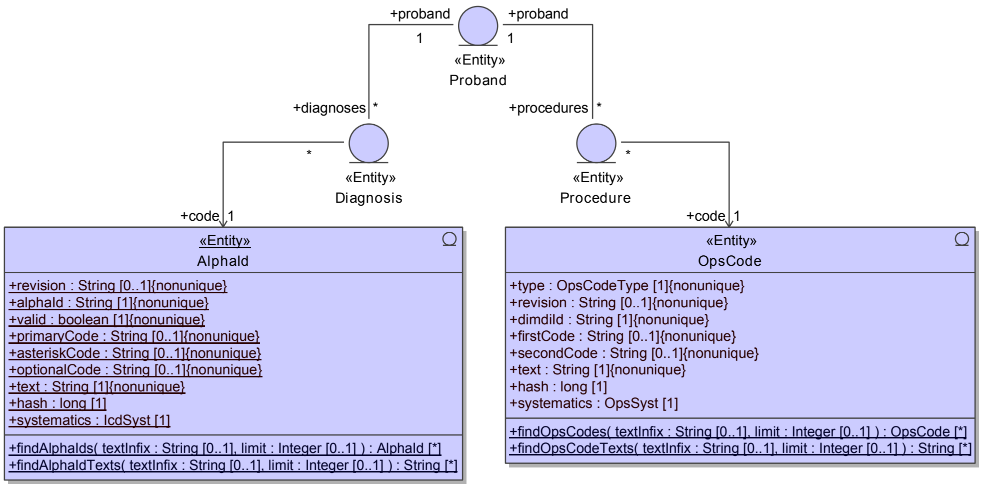
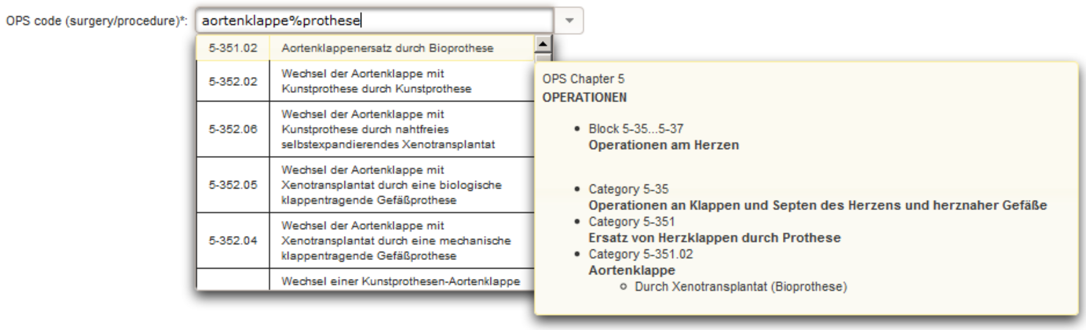

+proband
1
+proband
1
«Entity»
Proband
+diagnoses *
+procedures *
*
«Entity»
Diagnosis
*
«Entity»
Procedure
+code 1
+code 1
«Entity»
AlphaId
+revision : String [0..1]{nonunique}
+alphaId : String [1]{nonunique}
+valid : boolean [1]{nonunique}
+primaryCode : String [0..1]{nonunique}
+asteriskCode : String [0..1]{nonunique}
+optionalCode : String [0..1]{nonunique}
+text : String [1]{nonunique}
+hash : long [1]
+systematics : IcdSyst [1]
+findAlphaIds(textInfix : String [0..1], limit : Integer [0..1]) : AlphaId [*]
+findAlphaIdTexts(textInfix : String [0..1], limit : Integer [0..1]) : String [*]
«Entity»
OpsCode
+type : OpsCodeType [1]{nonunique}
+revision : String [0..1]{nonunique}
+dimdiId : String [1]{nonunique}
+firstCode : String [0..1]{nonunique}
+secondCode : String [0..1]{nonunique}
+text : String [1]{nonunique}
+hash : long [1]
+systematics : OpsSyst [1]
+findOpsCodes(textInfix : String [0..1], limit : Integer [0..1]) : OpsCode [*]
+findOpsCodeTexts(textInfix : String [0..1], limit : Integer [0..1]) : String [*]

Figure 4.139.: `AlphaId` and `OpsCode` master data entities are referred by corresponding `Diagnosis` and `Procedure` proband detail entities using many-to-one associations.

Associations using alphanumeric `AlphaId.alphaid` keys are preferred over ICD-10 codes in applications requiring to reference ICD-10 diagnoses. For the Phoenix CTMS, the `Diagnoses` detail entity could therefore show an unconstrained association. To include diagnoses in dynamic database queries (listing 4.60, line 38-41), regular associations are required to denote the association path of entity columns for criterion terms however (figure 4.139). Hence it is not possible to purge the `alphaid` table before starting the task for importing a new version of the Alpha-ID file (`-iai`). Records are appended with another value for the `revision` discriminator instead, specified by the `-air` option argument. For populating the `AlphaId` table, the `AlphaIdLineProcessor` requires the ICD-10 systematics master data to be present.

OPS code importer Aside diagnoses, patient history information comprises surgeries and medical procedures. At the moment, the Phoenix CTMS supports encoding them (figure 4.140) by referencing Operationen- und Prozedurenschlüssel (OPS) code records (*Operationen- und Prozedurenschlüssel* 2013).



OPS code (surgery/procedure)*: aortenklappe%prothese

5-351.02	Aortenklappenersatz durch Bioprothese
5-352.02	Wechsel der Aortenklappe mit Kunstprothese durch Kunstprothese
5-352.06	Wechsel der Aortenklappe mit Kunstprothese durch nahtfreies selbstexpandierendes Xenotransplantat
5-352.05	Wechsel der Aortenklappe mit Xenotransplantat durch eine biologische klappentragende Gefäßprothese
5-352.04	Wechsel der Aortenklappe mit Xenotransplantat durch eine mechanische klappentragende Gefäßprothese
	Wechsel einer Kunstprothesen-Aortenklappe

OPS Chapter 5
OPERATIONEN

- Block 5-35...5-37
  Operationen am Herzen

- Category 5-35
  Operationen an Klappen und Septen des Herzens und herznaher Gefäße
- Category 5-351
  Ersatz von Herzklappen durch Prothese
- Category 5-351.02
  Aortenklappe
  - Durch Xenotransplantat (Bioprothese)

Figure 4.140.: According to the diagnoses tab from figure 4.138, the procedures tab shows a `p:autoComplete` input element for selecting the OPS code master data record. The OPS systematics entry belonging to the OPS code record is displayed in a `itemtip` box when moving over an entry in the suggestion list.

Likewise, the OPS source file (`OpsCode.text`) is amended and released yearly by DIMDI. This file alphabetically lists names[107] for all surgeries, procedures und medical interventions compiled in the OPS systematics. The `firstCode` or `secondCode` refers an OPS systematics entry, mapped by the `OpsSyst` graph.

The structure of OPS codes and OPS systematics master data is congruent with the structure of Alpha-ID and ICD-10 systematics and so is the data model (figure 4.139) and importer implementation. The import is started by passing the `OpsCode.text` file location as argument for the `-ioc` task and requires the OPS systematics master data to be present.

4.10.3.2. Classification Markup Language (ClaML)

Both ICD-10-GM and OPS systematics mentioned in the previous section are provided by DIMDI in ClaML format (*ISO 13120:2013 - Health informatics - Syntax to represent the content of healthcare classification systems - Classification Markup Language (ClaML)* 2013). As a brief overview, this XML format for medical classification schemes is used by DIMDI to distribute the catalogues in a mono hierarchy of chapters, levels of blocks (code ranges) and levels of categories (codes). For a more compact representation, a "modifier" can be applied to an item of a hierarchy level in order to place common child items. In order to import the complex ClaML format, Simple API for XML (SAX) is utilized by `ClamlImporter` to parse XML and the accompanying Document Type Definition (DTD) source file. `ClamlClassProcessor`

[107]In contrast to Alpha-ID, searching using `p:autoComplete` inputs is cumbersome for users, since terms are too detailed. Manual tagging of relevant items could be a solution.

processes XML nodes and populates `ClamlImporter`'s internal data structures[108]. According to CSV importers, concrete implementations of `ClamlClassProcessor` persist records and differentiate into importers for ICD-10 and OPS systematics master data.

ICD-10-GM systematics importer An ICD-10 systematics entry is mapped by an `IcdSyst` graph, breaking down the hierarchy example below for displaying it in the UI as shown in figure 4.138:

- chapter "IV" - "Krankheiten des Nervensystems"
- block "G50-G59" - "Krankheiten von Nerven, Nervenwurzeln und Nervenplexus"
- category "G51" - "Kranheiten des N. facialis [VII. Hirnnerv]"
- category "**G51.0**" - "Facialisparese"

The lowest category's `icdsystcategory.code` (e.g. G51.0) corresponds the ICD-10 entry referred by an Alpha-ID record. Since Alpha-ID records are associated with `IcdSyst` (figure 4.139), the popuplated data model shown in figure 4.141 will store the catalogue in de-normalized form for improved query performance.

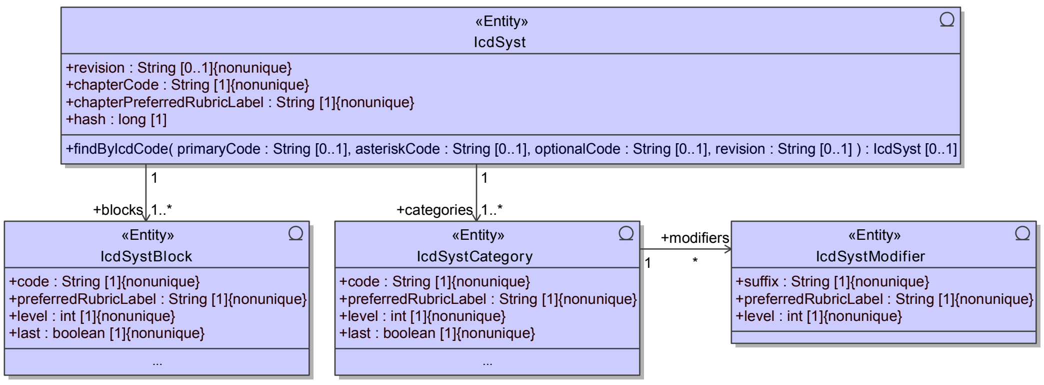

«Entity»
IcdSyst
+revision : String [0..1]{nonunique}
+chapterCode : String [1]{nonunique}
+chapterPreferredRubricLabel : String [1]{nonunique}
+hash : long [1]
+findByIcdCode(primaryCode : String [0..1], asteriskCode : String [0..1], optionalCode : String [0..1], revision : String [0..1]) : IcdSyst [0..1]
1
+blocks
1..*
1
+categories
1..*
«Entity»
IcdSystBlock
+code : String [1]{nonunique}
+preferredRubricLabel : String [1]{nonunique}
+level : int [1]{nonunique}
+last : boolean [1]{nonunique}
...
«Entity»
IcdSystCategory
+code : String [1]{nonunique}
+preferredRubricLabel : String [1]{nonunique}
+level : int [1]{nonunique}
+last : boolean [1]{nonunique}
...
+modifiers
1
*
«Entity»
IcdSystModifier
+suffix : String [1]{nonunique}
+preferredRubricLabel : String [1]{nonunique}
+level : int [1]{nonunique}

Figure 4.141.: The domain model shown is designed to store and efficiently query relevant parts of the ICD-10 systematics master data, provided in ClaML format.

The ICD-10 systematics import is started by specifying the `-iis` task with the XML file name argument. Since the ClaML format supports multiple languages, the language of extracted `preferredRubricLabels` can be selected using the `-sl` option argument.

OPS systematics importer According to ICD-10 master data, an OPS systematics entry is mapped by a corresponding `OpsSyst` graph with identical structure, breaking down the hierarchy for displaying it in the UI as shown in figure 4.140. While `IcdSystClassProcessor` is utilized for the ICD-10 ClaML import, OPS master data records will effectively be created by the `OpsSystClassProcessor` XML node processor. The OPS systematics import is started by specifying the `-ios` task.

4.10.3.3. File System

The file system importer allows to access entire directories online using the file manager UI. A directory in the system server's file system can be specified by the `-iif`, `-isf`, `-icf`, `-itf` or `-ipf` task's argument value to add its structure and content to the document repository of a module's root entity, specified by

[108]Modifiers are inherited to items of lower level, which requires the importer logic to build Cartesian products of leaf items across hierarchy levels.

the mandatory `-id` option argument. Due to encryption, importing files of a proband (`-ipf`) additionally requires to specify user credentials (`--user`, `--password`). The directory's content is effectively *copied*, thus it can be removed or unmounted after importing. The importer logic of `FileSystemLoader` traverses the given directory and adds each file using the `FileService.addFile` service method. This way, the root entity's file repository is merged with the directories content, but conflicts known from file system operations are inherently excluded. If a file with the same path and name is already present, two files will show up in the file browser UI after importing. `addFile` requires the caller to specify the file's content type, given by HTTP header values in the case of uploading document files. For the file system importer, a file's mime type is determined by its content using Apache Tika (Apache Tika Contributors, 2012, documentation section "Mime Magic Detection").

External filebases located on file shares were established over years of business. A simple synchronisation technique would require automating file system imports by means of job scheduling. Supplementary tasks (`-rif`, `-rsf`, `-rcf`, `-rtf`, `-rpf`) are prepared to empty file repositories again. Since a single repository can comprise thousands of `File` records, their removal is implemented using `ChunkedDaoOperationAdapter`, as described in section 4.10.1.2.

4.10.4. Scheduled Tasks

In enterprise applications, asynchronous processing can be accomplished by means of scheduled jobs. A job schedule defines when to run each recurring job. The server OS provides built-in job scheduler such as cron or Windows Task Scheduler for triggering jobs according to the registered job schedule. While OS job schedulers run a job as separate process, the Quartz scheduler (House and Quartz Contributors, 2013) can be used in Java applications to run jobs in form of threads spawned aside servlet container threads for serving HTTP requests. Although Quartz would be basically suitable, OS job schedulers where chosen over Quartz for multiple reasons such as:

- separate control of jobs and servlet container (application frontend)
- the server's unified job schedule will also include invocations of non-Java programs like shell/batch scripts, e.g. for creating database snapshots (backups/restore points)
- instead of inter-thread communication, database tables and transactions are used to synchronize data

As required by cron and counterparts, `dbtool` allows selecting and running a job as a process by command-line invocation. Task modes presented in this section are used for several scheduled jobs, the Phoenix CTMS environment relies on. The job invocation command will redirect a task's output into a separate file for logging purposes. Additionally, a job's output can be sent to email recipients, if specified using the `-er` option argument.

Notification processing As outlined in section 4.6.3.2, notification messages are generated and sent in form of emails using a consumer-producer pattern. On producer-side, notifications of relevant CRUD events are created by service method logic, triggered by UI interaction of a user. Additionally, a daily job generates notifications marked with "job" in table 4.12. The job runs the `dbtool -pn` task to invoke `ToolsService.prepareNotifications`. This service method will create corresponding notifications, if the current date exceeds registered dates such as the beginning of a reminder period or an expiry date.

On consumer-side, notifications are processed by a separate job executing the `-sn` task in short intervals. `NotificationSender` is set up to use `NotificationEmailSender` (figure 4.74) for converting filtered notifications into message texts and sending them to recipients using the outgoing email server specified by the configuration, such as `smtp.medunigraz.at`.

Deletion of probands When a recruited proband is registered in the Phoenix CTMS, a proband letter can be generated and printed (section 4.4), containing a written data privacy consent. To comply with data privacy regulations, it has to be own-hand signed and returned by the proband to be finally filed. In the end, the proband database must not contain any proband record with the privacy consent document missing. Therefore, a rigorous workflow is established to ensure the existence of this document (figure 4.142), ending with the deletion of any incomplete proband registration.

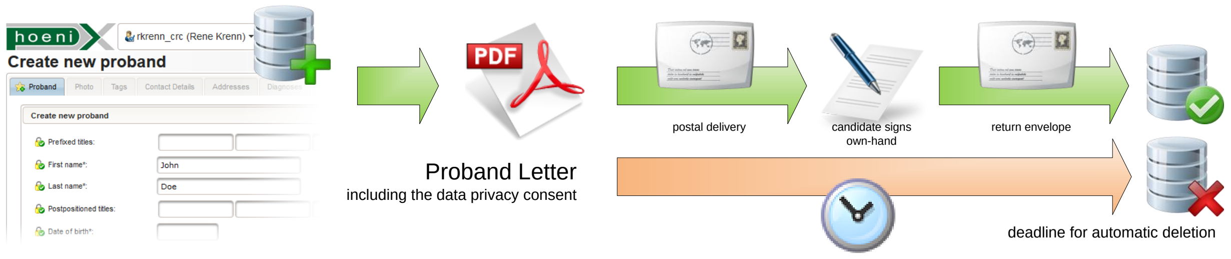


Figure 4.142.: After a new proband entry is entered, a proband letter PDF document can be generated. It includes the printable data privacy consent, which is sent to the candidate. If it is not returned within a given time, the system deletes any data that was entered until then.

When a proband is entered, a deadline is set until the privacy consent must be present. Users responsible for proband recruitment print and send the proband letters. When checking the daily mail, they scan returned privacy consents and upload them to the probands' document repository. The proband's privacy consent status value reflects the existence of the returned privacy consent. An overview list (figure 4.92) allows tracking awaited reconsignments. The `-rp` task is executed daily to purge proband records with expired deadline. Since consistent removal of a single proband entity graph results in deleting multiple records, a transactional delete operation is desired. Instead of performing separate DAO operations, `ProbandRemover` therefore encapsulates the invocation of an extra service method for that purpose (`DBTool.deleteProbands`). The `--limit` option switch allows to limit the number of probands deleted. As another option for phasing automated proband deletions, a department's l10n identifier can be specified by the `--dlk` option argument in order to process probands of a certain department only.

File system maintenance The default configuration option for the document management feature will cause files to be stored in the system server's file system. As described in section 4.7.1, a file delete operation will not remove the referenced file from the file system in order to enforce consistency at any given time. `ExternalFileChecker` provides methods with directory traversal logic for dealing with resulting orphaned files. These methods are exposed by `dbtool` task switches:

Scan for orphaned files: The `-so` task identifies and lists orphaned files.
Delete orphaned files: Orphaned files can be identified and removed at once using the `-do` task.

Removing orphaned files can be registered as weekly or monthly job in order to preserve disk space. A more serious situation arises if referenced document files get lost, e.g. due to disk failure. Additional tasks are provided as an aid for administrators in this case, allowing to check and restore database consistency:

Scan for missing files: The `-sm` task lists `File` records, whose referenced files cannot be found in the file system.
Purge `File` records referencing missing files: `File` records without valid file references are identified and removed at once using the `-dm` task.

5. Results

5.1. Rollout

The final result presented in this work is a production-grade application, which was finally rolled out at MUG trial sites located at the ZMF and the Division of Endocrinology and Metabolism. As a start, it was decided to run a single instance of the Phoenix CTMS. The monolithic deployment integrates all system components on a single physical server:

- PostgreSQL RDBMS
- fronting HTTP server
- Tomcat servlet container
- storage partitions
- automated jobs

Amazon EC2 In order enable users to evaluate demonstrated features themselves, a temporary test environment was initialized at first. The test environment was established in September 2012, when the prototype's number of implemented features covered a majority of requirements. For the test environment, a single Windows Server instance running in the Amazon Elastic Compute Cloud (EC2) platform was hired. Disk storage for images is required separately and was set up using Amazon Elastic Block Store (EBS). The Windows Task Scheduler was configured to run jobs in order to constitute an almost complete system setup, without a fronting HTTP server. The system contained test data only, allowing to reset and update the database several times while development was still in progress. The main purpose of the test environment was to identify and resolve basic issues such as:

- compatibility with workstation browsers
- bugs that manifest with multi-user utilisation
- performance bootlenecks
- adjust and document application-specific configuration options (*configuration mgmt*)

A structured User Acceptance Test (UAT) was intended but never completed since it turned out too time-consuming for involved trial site staff. Hence, the utilisation of the test environment was lower than expected.

phoenix.joanneum.at The test environment was replaced by the first production environment, located at `https://phoenix.joanneum.at`. This virtualized Linux server was hosted by JR Health from December 2012 to January 2014. The system setup was elaborated for both Red Hat Enterprise Linux (RHEL) and Debian, before RHEL was given preferrence. Appendix C gives a basic description of the system installation procedure for RedHat-based Linux environments.

domain entity	**number of records**	**domain entity**	**number of records**
Inventory		*Courses*	
inventory items	211	courses	95
detail properties	406	lecturers	116
availability status entries	28	course participations	286
maintenance reminders	156	hyperlinks	0
bookings	409	system messages	1744
hyperlinks	0	manual journal entries	0
system messages	3025	files	6
manual journal entries	0	queries	7
files	230	*Trials*	
queries	7	trial	72
Persons and Organisations		trial tags	120
persons	121	team members	374
organisations	15	timelineevents	383
photos	3	visits	161
detail properties	141	groups	117
absence status entries	223	visit schedule	1169
contact details	155	proband list columns	38
addresses	145	column values	343
shifts	1383	inquiry questions	33
CV positions	450	proband list entries	587
hyperlinks	0	enrollment status entries	1728
system messages	11869	hyperlinks	
manual journal entries	0	system messages	61806
files	325	manual journal entries	0
queries	11	files	72
Probands		queries	5
probands	1082	*Input Fields*	
detail properties	5	input fields	67
contact details	2039	option items	181
addresses	1078	system messages	369
diagnoses	2544	manual journal entries	0
procedures	144	queries	1
absence status entries	26	*Users*	
inquiry form values	28835	user accounts	117
bank accounts	3	passwords	433
system messages	58503	system messages	19867
manual journal entries	587	manual journal entries	0
files	1924	queries	4
queries	44	*Departments*	3

Table 5.1.: Summary of production data assets as of April 2014.

With the go-live of phoenix.joanneum.at, users started entering assets into the system. SOPs for data entry were created to prepare the regular utilisation of system modules. This was accomplished in stages, starting with the user and person/organisation modules. When introducing the proband module in August 2013 (figure 5.1), existing subject data was entered initially. At this point, the most essential use cases of the trial module were ready for production use for the first time. Full-scale utilization of all modules was finally reached in the course of a large commercial trial starting in November 2013, which was entirely managed using the Phoenix CTMS. A sequence of version upgrades were released to put refinements into production, while existing production data had to stay intact.

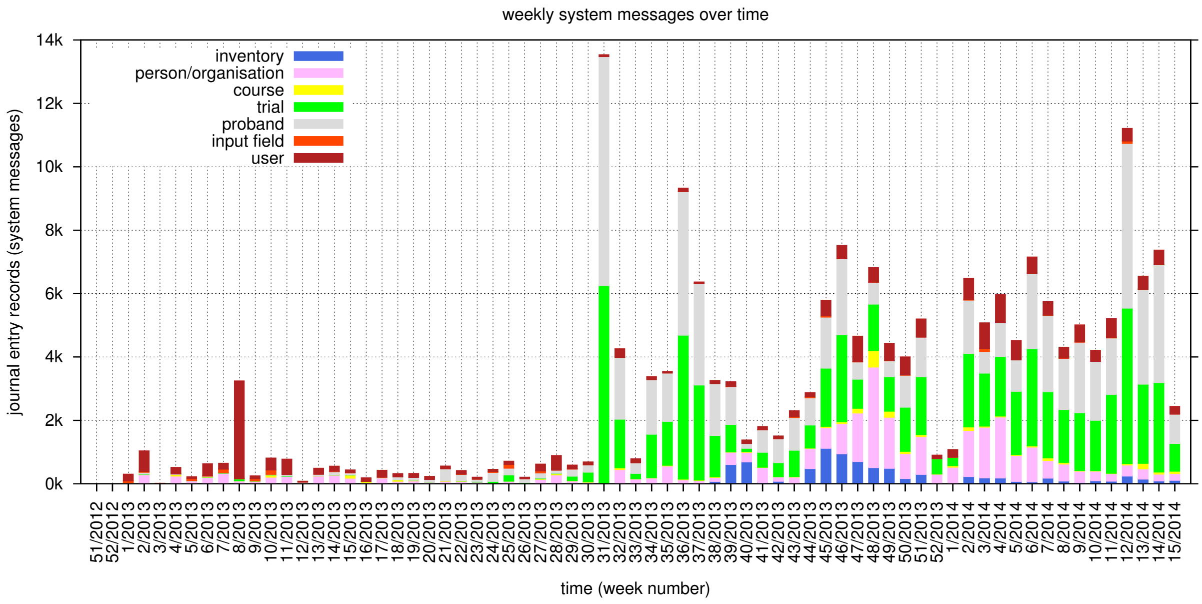


Figure 5.1.: This histogram shows the trend of system generated journal messages per week. It basically indicates the frequency of CRUD operations performed by users.

phoenix.medunigraz.at In January 2014, the production environment was finally moved and is now hosted at `https://phoenix.medunigraz.at`. The migration of production data (table 5.1) was prepared to minimize the duration of interruption. The new server is running SUSE Linux Enterprise Server (SLES) on VMware virtualisation as a part of the MUG IT infrastructure, governed by the ZID. At this point, the roll-out was considered completed and corporate usage officially appreciated. The Phoenix CTMS is fully integrated into the trial site's daily business, which is confirmed by the constant utilization shown in figure 5.2. The application undergoes continuous refinements and feature enhancements in the course of software maintenance.

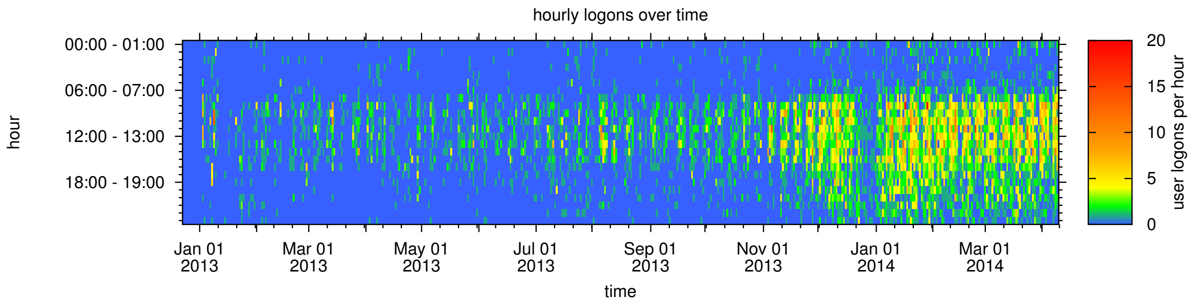


Figure 5.2.: This heat map shows successful user logons per hour over the time of day. It illustrates system utilization based on the number of concurrently active users.

5.2. Ratio of Generated Artefacts

Whenever the software development process includes code generation techniques, the question of the effectiveness of this approach arises. In this case, the ratio of artefacts generated with AndroMDA are an interesting metric to look at retrospectively.

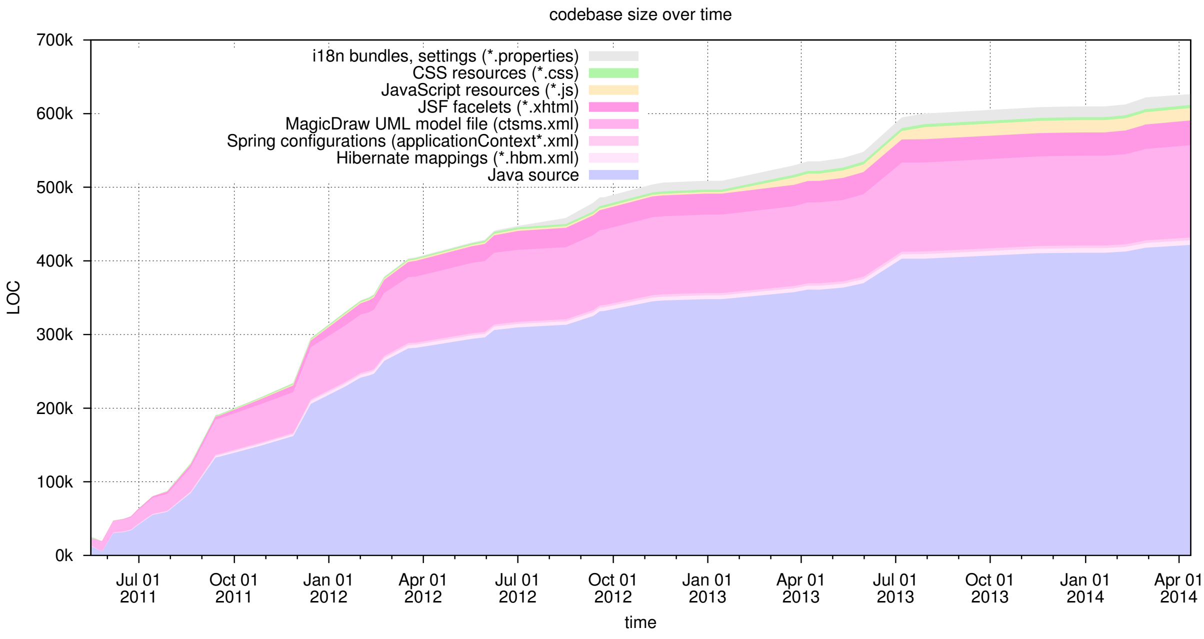


Figure 5.3.: This stacked area chart illustrates essential types of sources that constitute the project codebase. To compare different langauges, source code volumes are given in *physical* lines of code[1].

The project's codebase comprises various types of source files (figure 5.3). Java source code represents the most extensive part and comprises both generated and hand-crafted artefacts. Figure 5.4 shows the proportions of Java packages and classes, which are organised into Maven projects from figure 2.16.

[1]Lines of source code files are counted using `cloc` (Danial and CLOC Contributors, 2013): no code style normalization, excluding comments, including blank lines (19%).

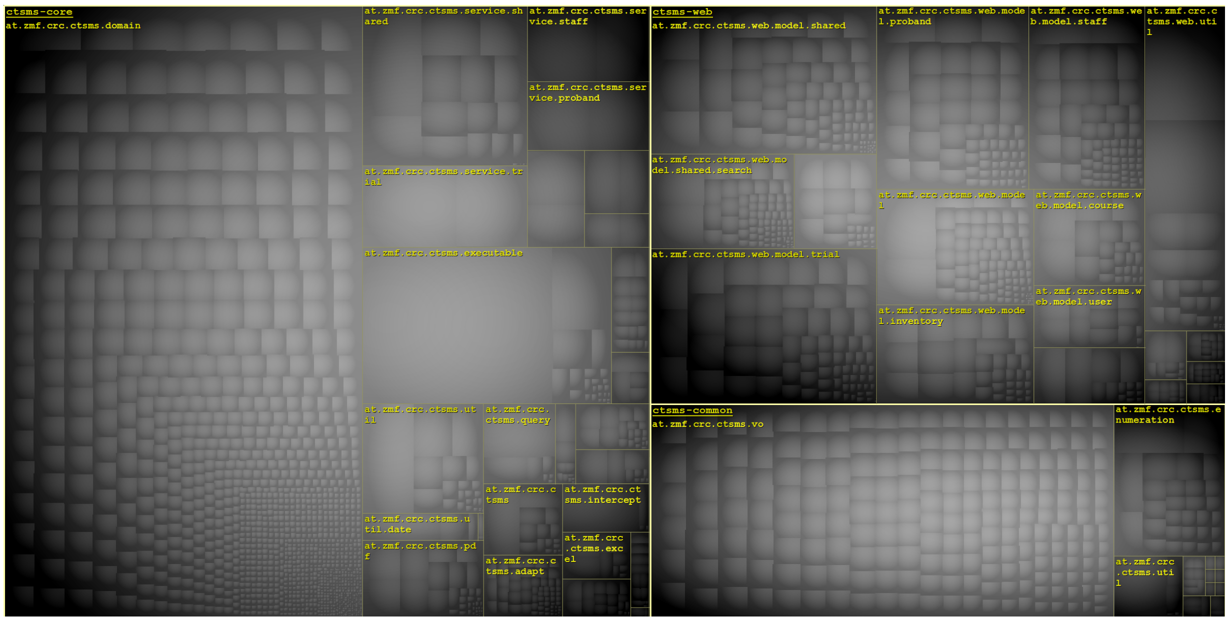


Figure 5.4.: This tree map illustrates proportions of Java projects (`ctsms-core`, `ctsms-common`, `ctsms-web`). Rectangles represent contained packages and classes. The rectangle area reflects the artefact's size in *logical* lines of code[2]. The `ctsms-web` project is entirely hand-crafted and makes up 30.5% of the overall Java code.

When looking at the Java sources of Maven projects as a whole (figure 5.5), the percentage ratio of code generated with AndroMDA to the total volume settled at app. 60% (figure 5.6). It indicates the considerable fraction of boilerplate code inherent to data-driven Java web applications. The ratio value would be even higher when using code generation for all application layers, including web tier artefacts.

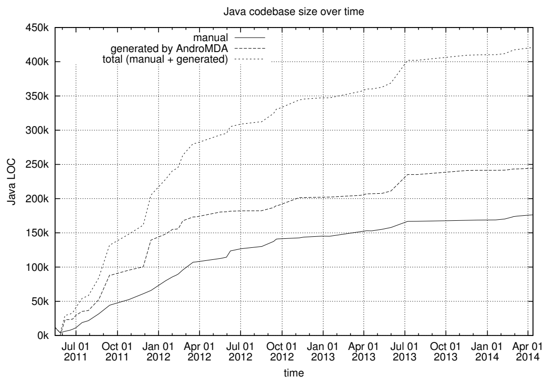


Figure 5.5.: Generated and hand-crafted portions of the Java codebase in physical lines of code.

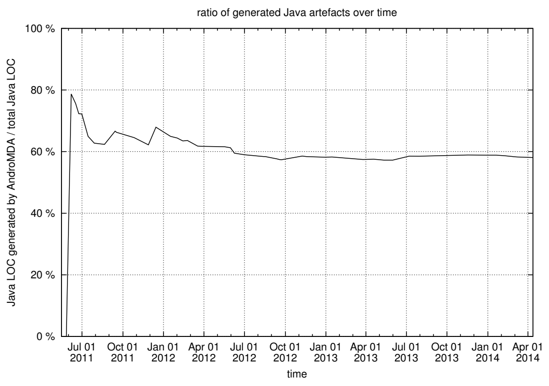


Figure 5.6.: After a familiarization phase, the ratio of generated portions from figure 5.5 got stable. This indicates short MDD iterations.

[2]Lines of source code files are counted using JArchitect (CoderGears, 2014): noramlized coding style, excluding comments, excluding blank lines.

6. Conclusion

6.1. Summary

The Phoenix CTMS is a tailored software system to cover major workflows of clinical trials conducted at MUG trial sites. This work described how the system was designed, implemented and finally rolled out. After a brief introduction into the terminology of clinical trials and related software systems, approaches of other Austrian medical schools were outlined. This includes a feature comparison of the EDC capabilities common to presented systems.

A precondition was to create a web application based upon an established Java technology stack using the MDA approach. The basic multi-tier architecture of the Java web application is dictated by AndroMDA. AndroMDA facilitates a practicable MDD process, which was used to develop the backend layers. The skeleton is generated according to UML class diagrams and has to be completed by hand-crafted application logic. The web presentation layer is completely hand-crafted using the JSF framework.

A detailed requirement analysis was elaborated in corporation with CRC trial site staff during an intense design phase at the beginning of the project. Results were documented in a requirement analysis document in form of textual descriptions of planned features. With the focus on the data model, it outlines a separation of required features into seven system modules:

- inventory
- person and organsations
- courses and trainings
- clinical trials
- subject database
- users
- EDC input fields

A remarkable set of features implements or is subject to regulations such as GCP or DSG. These were considered in expert staff sessions throughout the agile, user-centered development process.

With respect to the application size, a number of design considerations were found necessary for a successful result. Hence, the implementation chapter (section 4) deals with a number of elaborated styles and patterns that apply to backend layers:

- data model structures
- VO cycles
- i18n strategy
- finder methods
- access control concept
- application-level encryption
- PDF and Excel exporter infrastructure.

Accordingly, implementation details of JSF web tier components such as managed beans, templates or UI organisation were given. The AJAX-driven RIA UI reflects the separation into system modules and comprises over 100 distinct screens. Using the PrimeFaces JSF component library allows to integrate complex views such as calendars or a timeline. Finally, the design and implementation of three remarkable features is described in greater detail:

- DMS (file managers)
- dynamic database queries (query editors)
- EDC (input forms, field value calculation, sketches)

At last, each mode of the implemented CLI tool was described. The project result was summarized with insights on the system rollout, utilisation and basic codebase metrics.

6.2. Discussion

6.2.1. Project Outcome

The Phoenix CTMS was designed according to processes established over years and numerous clinical trials conducted by the CRC. Beside the implementation quality, the range and level of detail of features showed sufficient for a productive tool that gets accepted by users. Large-scale commercial trials are supported as well as trials of an individual graduand or researcher. A standardized baseline of information is introduced, which will help to avoid mistakes and safe time when users are expected to collaborate. It quickly turned into the key application in the trial site's daily business. In fact, the platform will prepare the trial site for the future by enabling to manage

- a growing number of trial site employees,
- more clinical trials conducted simultaneously,
- a large candidate repository,
- and extensive clinical trials with large numbers of subjects

"Mit der Phoenix Datenbank wurde ein Tool geschaffen, das die tägliche Arbeit in hohem Maße unterstützt und die Planung und Koordination von klinischen Studien mit höchster Professionalität und Qualität ermöglicht." (Martina Brunner, MSc. - Abteilung Core Facility CRC, April 2014)

In 2013, the project was submitted to the seventh Science Park Graz Idea Contest (Science Park Graz GmbH, 2013, appendix A) and became a top-ten finalist among 105 contributions.

6.2.2. Java Technology Stack

Java is a mature, strongly typed programming language, taught at schools and universities.

mature: Proven third-party libs and frameworks for almost any detail problem are available.
strongly typed: Strongly typed languages facilitate layering and collaborative application development.
covered by curriculas: Java skills are a common prerequisiste for a developer nowadays, and are therefore easy to find.

These properties might be the essential reasons for the success and wide-spread use of Java technology stacks for enterprise web applications. Frameworks and code generation aim at artefacts of straightforward data-driven applications, which means a benefit when creating large systems. However, Java web application development requires specific skills to cope with tools and the numerous frameworks. They are considered as de-facto standards frequently, but have steep learning curves and do not necessarily mean a simplification for every project. Project archetypes reflect heavyweight application architectures and may suggest characteristics of an over-engineered platform, less suitable for small projects or minimal time-to-market.

Despite experience with web development, it took considerable efforts to learn the Java technology stack details while creating a production-grade result at the same time. Due to the size of the Phoenix CTMS project, a refactoring cycle to eliminate a buggy or suboptimal pattern can easily cause a week of additional work. Aside measures to minimize code duplication, this can be avoided by studying and testing framework usage patterns carefully before implementing them across the application.

6.2.3. AndroMDA

Although the use of AndroMDA added an additional learning curve, there are essential benefits. For beginners, the generated application skeleton is a substantial starting point to study the technology stack and prevent early mistakes. When the codebase starts growing, AndroMDA allows to efficiently generate boilerplate code portions, with patterns applied consistently and without typos. A well known disadvantage with MDA approaches is the generator's processing time, that comes into effect whenever a detail of a large model is edited. For this project, the duration of a complete build-and-deploy cycle takes up to 20 minutes, which was worked around by using virtualized development environments in parallel.

Utilizing AndroMDA beyond the typical MDA-light development process (section 2.3.2) requires deep understanding of its internals, such as the structure of metafacades. While developing new code templates from scratch was not considered for this project, modifying existing templates was required in several situations however. When thinking of a more complete MDA approach, the final JSF/AJAX/PrimeFaces-based web tier can be viewed as a reference implementation for designing a custom catridge and UML profile in order to reproduce and simplify the development of similar presentation layers in other projects.

6.2.4. Expenditures

MUG spent app. 64k€ in form of employer's labour costs. This exceeds the expected volume of 50k€ by app. 30%. The development took place off-site, hence app. 6k€ were additionally spent for required hard- and software and other expenses (EC2, books) by the author. JR Health contributed the MagicDraw seat license. While stakeholders originally rated a total effort of 3k man-hours, it took between 8k and 9k man-hours to complete the project after all.

6.3. Future Outlook

Operating a computerized system for clinical trials comes with impacts and responsibilities. Since the Phoenix CTMS is likely to be utilized in the long term, there is a high probability of future change requests and the need of maintenance support. Pending tasks listed below have accumulated up to now already, although development is frozen with the end of this student project:

mandatory:

- Datenverarbeitungsregister (DVR) registration
- validation
- BASG inspection

optional:

- extensions
- rollout at additional MUG trial sites
- formal usability analysis
- open-source release

This work closes with details on some desired extensions to the Phoenix CTMS application from above. Beside amendments to the original requirement catalogue - most of which were already implemented during refinement iterations - a number of more extensive enhancements were asked for. Most remarkable, two additional system modules were requested:

- Finance
- Publications

While the straightforward publications module could be implemented in a few weeks by reusing existing building blocks, a concept for the more complex finance module was elaborated together with administrative staff and presented to stakeholders (appendix D). Apart from that, the system's web service API was introduced to yield a proof-of-concept and could be extended to introduce new application components:

- fat (desktop) client supporting offline EDC
- mobile app to manage duty allocations personally on the mobile phone
- office software suite plug-in or Web-based Distributed Authoring and Versioning (WebDAV) extension for DMS features[1]

[1]To edit a document file displayed in a file manager (section 4.7), it has to be downloaded to create a local copy. After saving the file in the office application, the modified file has to be uploaded again. In future, this round-trip should be avoided by saving files directly.

List of Figures

List of Tables

List of Listings

Bibliography

Agrawal, Rakesh et al. (2004). “Order preserving encryption for numeric data.” In: *Proceedings of the 2004 ACM SIGMOD international conference on Management of data - SIGMOD'04*. Association for Computing Machinery, p. 563. ISBN: 1581138598. DOI: 10.1145/1007568.1007632. URL: http://dx.doi.org/10.1145/1007568.1007632.

Ahn, L. von et al. (Sept. 2008). “reCAPTCHA: Human-Based Character Recognition via Web Security Measures.” In: *Science* 321.5895, pp. 1465–1468. DOI: 10.1126/science.1160379. URL: http://dx.doi.org/10.1126/science.1160379.

Albert, Yannick (Feb. 2013). *jQuery Base64*. URL: http://plugins.jquery.com/base64/.

Alpha-ID - Identifikationsnummer für Diagnosen (2013). Deutsches Institut für Dokumentation und Information (DIMDI). URL: http://www.dimdi.de/static/de/klassi/alpha-id/index.htm.

Apache FOP Contributors (2013). *The Apache FOP Project*. The Apache Software Foundation. URL: http://xmlgraphics.apache.org/fop/.

Apache Tika Contributors (Mar. 2012). *Apache Tika - a content analysis toolkit*. The Apache Software Foundation. URL: http://tika.apache.org/.

Armour, F. and G. Miller (2001). *Advanced Use Case Modeling: Software Systems*. Object technology series Bd. 1. ADDISON WESLEY Publishing Company Incorporated. ISBN: 9780201615920. URL: http://books.google.at/books?id=VINuQgAACAAJ.

Arshaw, Adam (Sept. 2013). *FullCalendar - Full-sized Calendar jQuery Plugin*. URL: http://www.arshaw.com/fullcalendar/.

Arzneimittelbetriebsordnung 2009 (Sept. 2008). Bundesgesetzblat II, 324. Verordnung. URL: https://www.ris.bka.gv.at/GeltendeFassung.wxe?Abfrage=Bundesnormen&Gesetzesnummer=20005989.

Arzneimittelgesetz (AMG) (Mar. 1983). Bundesgesetzblatt I 185/1983. URL: https://www.ris.bka.gv.at/GeltendeFassung.wxe?Abfrage=Bundesnormen&Gesetzesnummer=10010441.

Atlassian (Feb. 2014). *JIRA - Issue and Project Tracking Software*. Atlassian. URL: https://www.atlassian.com/en/software/jira.

Baranovskiy, Dmitry (2013). *Raphaël - JavaScript Library*. URL: http://www.raphaeljs.com/.

BASG/AGES (Apr. 2012). *Inspektion Computergestützter Systeme*. Bundesamt für Sicherheit im Gesundheitswesen (BASG)/Österreichische Agentur für Gesundheit und Ernährungssicherheit GmbH (AGES). URL: http://www.basg.gv.at/uploads/media/L_I05_Ablauf_IT_Inspektion_01.docx.

Beck, P. et al. (2006). “On-the-fly form generation and on-line metadata configuration–a clinical data management Web infrastructure in Java.” In: *Studies in health technology and informatics* 124, pp. 271–276. ISSN: 0926-9630. URL: http://view.ncbi.nlm.nih.gov/pubmed/17108536.

Bohlen, Matthias et al. (July 2012). *AndroMDA Model Driven Architecture Framework*. URL: http://www.andromda.org.

Boldyreva, Alexandra et al. (2009). “Order-Preserving Symmetric Encryption.” In: *Advances in Cryptology - EUROCRYPT 2009*. Ed. by Antoine Joux. Vol. 5479. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 224–241. ISBN: 978-3-642-01000-2. DOI: 10.1007/978-3-642-01001-9_13. URL: http://dx.doi.org/10.1007/978-3-642-01001-9_13.

Boyd, Norris and Rhino Contributors (June 2012). *Rhino - open-source implementation of JavaScript written entirely in Java*. Mozilla/Sun Microsystems. URL: http://www.mozilla.org/rhino/.

Cattell, Rick (May 2011). “Scalable SQL and NoSQL data stores.” In: *SIGMOD Rec.* 39.4, p. 12. ISSN: 0163-5808. DOI: 10.1145/1978915.1978919. URL: http://dx.doi.org/10.1145/1978915.1978919.

CDISC (Dec. 2013). *Specification for the Operational Data Model (ODM)*. Clinical Data Interchange Standards Consortium (CDISC). URL: http://www.cdisc.org/odm.

CDISC (Mar. 2014). *CDISC Standards and Implementations*. Clinical Data Interchange Standards Consortium (CDISC). URL: http://www.cdisc.org/standards-and-implementations.

Çelik, Tantek, Chris Lilley, and L. David Baron (June 2011). *W3C Recommendation: CSS Color Module Level 3*. World Wide Web Consortium (W3C). URL: http://www.w3.org/TR/css3-color/.

Chapman and Hall (1998). *Algorithms and Theory of Computation Handbook (Chapman & Hall/CRC Applied Algorithms and Data Structures series)*. CRC Press. ISBN: 0849326494. URL: http://www.amazon.com/Algorithms-Computation-Handbook-Chapman-Structures/dp/0849326494?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0849326494.

Cinar, M. Serhat (Jan. 2009). *Hibernate HQL to SQL Translation*. URL: http://narcanti.keyboardsamurais.de/hibernate-hql-to-sql-translation.html.

Çivici, Çagatay et al. (Apr. 2014). *PrimeFaces - Ultimate JSF Component Suite*. PrimeTek Informatics. URL: http://www.primefaces.org/.

Claypool, Kathryn, Bill Szabrak, and Jeff Weaver (Feb. 2011). *Electronic Data Capture At Emory University*. Emory University Research and Health Sciences IT Division. URL: https://med.emory.edu/documents/resources/ram/Redcap%20Presentation.pdf.

CoderGears (Apr. 2014). *JArchitect :: Java Static Analysis Tool*. CoderGears. URL: http://www.javadepend.com/.

Collin, Jean-Marc (Apr. 2010). *AndroMDA Cartridge Development: New Primefaces JSF cartridge*. URL: http://www.andromda.org/forum/viewtopic.php?f=25&t=6772.

Crockford, Douglas (Nov. 2010). *JSON in JavaScript*. URL: https://github.com/douglascrockford/JSON-js.

Danial, Al and CLOC Contributors (Aug. 2013). *CLOC – Count Lines of Code*. URL: http://cloc.sourceforge.net/.

Darcy, Joseph D. (Oct. 2013). *JEP 118: Access to Parameter Names at Runtime*. Oracle. URL: http://openjdk.java.net/jeps/118.

Datenschutzgesetz 2000 (DSG 2000) (Aug. 1999). Bundesgesetzblatt I 165/1999. URL: https://www.ris.bka.gv.at/GeltendeFassung.wxe?Abfrage=bundesnormen&Gesetzesnummer=10001597.

Day, Caroline and Clifford J Bailey (2009). “HbA1c-changing units.” In: *The British Journal of Diabetes & Vascular Disease* 9.3, pp. 134–136. URL: http://dvd.sagepub.com/content/9/3/134.short.

Dijkstra, Edsger W. (1961). *Algol 60 translation : An Algol 60 translator for the x1 and Making a translator for Algol 60*. Tech. rep. 35. Mathematisch Centrum, Amsterdam. URL: http://www.cs.utexas.edu/users/EWD/MCReps/MR35.PDF.

Duftschmid, Georg and Thomas Wrba (2004). “A tool for the design of clinical forms supporting end-user integration.” In: *Informatics for Health and Social Care* 29.1, pp. 29–41. URL: http://informahealthcare.com/doi/abs/10.1080/14639230310001639072.

Dunn, Mike (Feb. 2013). *Simple JS parser to strip comments*. URL: https://github.com/moagrius/stripcomments.

Eisenstein, E. L et al. (Feb. 2008). “Sensible approaches for reducing clinical trial costs.” In: *Clinical Trials* 5.1, pp. 75–84. ISSN: 1740-7745. DOI: 10.1177/1740774507087551. URL: http://dx.doi.org/10.1177/1740774507087551.

Firesmith, Donald G. (Dec. 2003). *Common Concepts Underlying Safety, Security, and Survivability Engineering*. Technical Note CMU/SEI-2003-TN-033. Carnegie Mellon Software Engineering Institute. URL: http://www.sei.cmu.edu/reports/03tn033.pdf.

Fowler, Martin (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional. ISBN: 0321127420. URL: http://www.amazon.com/Patterns-Enterprise-Application-Architecture-Martin/dp/0321127420?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0321127420.

Gall, W et al. (1999). “ArchiMed: a medical information and retrieval system.” In: *Meth Inform Med* 38, pp. 16–24. URL: http://www.ncbi.nlm.nih.gov/pubmed/10339959.

Gaus, Wilhelm and Dagmar Chase (2008). *Klinische Studien: Regelwerke, Strukturen, Dokumente, Daten (German Edition)*. Books On Demand. ISBN: 3833472227. URL: http://www.amazon.com/

Klinische-Studien-Regelwerke-Strukturen-Dokumente/dp/3833472227?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=3833472227.

Gerhold, Lukas (May 2007). "Planung und Realisierung eines webbasierten, medizinischen Informationssystems zur Unterstützung klinischer Multicenterstudien." MA thesis. Institut für Medizinische Informations- und Auswertesysteme der Medizinischen Universität Wien. URL: http://www.meduniwien.ac.at/msi/mias/studarbeiten/2007-DA-Gerhold.pdf.

Gilchrist, Jeff et al. (Oct. 2011). "Performance Evaluation of Various Storage Formats for Clinical Data Repositories." In: *IEEE Transactions on Instrumentation and Measurement* 60.10, pp. 3244–3252. DOI: 10.1109/TIM.2011.2122850. URL: http://dx.doi.org/10.1109/TIM.2011.2122850.

Glass, Dav and YUI Contributors (Apr. 2014). *Yahoo User Interface Library*. Yahoo. URL: http://www.yuilibrary.com/.

Good Manufacturing Practice - Medicinal Products for Human and Veterinary Use Annex 11: Computerised Systems (2011). European Commision - Health and Consumers Directorate - general. URL: http://ec.europa.eu/health/files/eudralex/vol-4/annex11_01-2011_en.pdf.

Gruhn, Volker, Daniel Pieper, and Carsten Röttgers (2006). *MDA®: Effektives Software-Engineering mit UML2® und Eclipse(TM) (Xpert.press) (German Edition)*. Springer. ISBN: 3540287442. URL: http://www.amazon.com/MDA%C3%82%C2%AE-Effektives-Software-Engineering-Eclipse-Xpert-press/dp/3540287442?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=3540287442.

Guidance for Industry - Computerized Systems Used in Clinical Investigations (May 2007). U.S. Food and Drug Administration (FDA). URL: http://www.fda.gov/ohrms/dockets/98fr/04d-0440-gdl0002.pdf.

Haas, Peter (2004). *Medizinische Informationssysteme und Elektronische Krankenakten (German Edition)*. Springer. ISBN: 3540204253. URL: http://www.amazon.com/Medizinische-Informationssysteme-Elektronische-Krankenakten-Edition/dp/3540204253?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=3540204253.

Häggström, Olle (2002). *Finite Markov Chains and Algorithmic Applications (London Mathematical Society Student Texts)*. Cambridge University Press. ISBN: 0521890012. URL: http://www.amazon.com/Algorithmic-Applications-Mathematical-Society-Student/dp/0521890012?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0521890012.

Harris, Paul A. et al. (Apr. 2009). "Research electronic data capture (REDCap) - A metadata-driven methodology and workflow process for providing translational research informatics support." In: *Journal of Biomedical Informatics* 42.2, pp. 377–381. DOI: 10.1016/j.jbi.2008.08.010. URL: http://dx.doi.org/10.1016/j.jbi.2008.08.010.

Hauf, Diet (2007). *Allgemeine Konzepte - K-Anonymity, l-Diversity and T-Closeness*. IPD Uni-Karlsruhe. URL: http://dbis.ipd.uni-karlsruhe.de/img/content/SS07Hauf_kAnonym.pdf.

Haverbeke, Marijn and CodeMirror Contributors (n.d.). *CodeMirror - a versatile text editor implemented in JavaScript for the browser*. URL: http://www.codemirror.net/.

Hightower, Rick (May 2005). *JSF for Nonbelievers: The JSF Application Lifecycle*. DeveloperTutorials.com. URL: http://www.developertutorials.com/tutorials/java/jsp-application-lifecycle-050504-1203/.

Holzinger, A., K.H. Struggl, and M. Debevc (2010). "Applying Model-View-Controller (MVC) in design and development of information systems: An example of smart assistive script breakdown in an e-Business application." In: *e-Business (ICE-B), Proceedings of the 2010 International Conference on*, pp. 1–6.

Holzinger, Andreas (Jan. 2005). "Usability Engineering Methods for Software Developers." In: *Commun. ACM* 48.1, pp. 71–74. ISSN: 0001-0782. DOI: 10.1145/1039539.1039541. URL: http://doi.acm.org/10.1145/1039539.1039541.

Holzinger, Andreas, Martin Brugger, and Wolfgang Slany (2011). "Applying Aspect Oriented Programming in Usability Engineering Processes - On the Example of Tracking Usage Information for Remote

Usability Testing." In: *ICE-B*. Ed. by David A. Marca, Boris Shishkov, and Marten van Sinderen. SciTePress, pp. 53–56. ISBN: 978-989-8425-70-6.

House, James and Quartz Contributors (Sept. 2013). *Quartz: Job Scheduling Library for Java applications*. Terracotta, Inc. URL: `http://www.quartz-scheduler.org/`.

Hussain, Zahid, Wolfgang Slany, and Andreas Holzinger (2009). "Current State of Agile User-Centered Design: A Survey." In: *HCI and Usability for e-Inclusion*. Ed. by Andreas Holzinger and Klaus Miesenberger. Vol. 5889. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 416–427. ISBN: 978-3-642-10307-0. DOI: `10.1007/978-3-642-10308-7_30`. URL: `http://dx.doi.org/10.1007/978-3-642-10308-7_30`.

ICH E6 (R1) Guideline for Good Clinical Practice: Consolidated Guidance (June 1996). International Conference on Harmonisation of Technical Requirements for Registration of Pharmaceuticals for Human Use (ICH).

IEEE Recommended Practice for Software Requirements Specifications (1998). Institute of Electrical and Electronics Engineers (IEEE). DOI: `10.1109/IEEESTD.1998.88286`. URL: `http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=720574`.

Inkscape Contributors (2013). *Inkscape. Draw Freely.* URL: `http://www.inkscape.org/`.

ISO 13120:2013 - Health informatics - Syntax to represent the content of healthcare classification systems - Classification Markup Language (ClaML) (2013). International Organization for Standardization (ISO). URL: `https://www.iso.org/obp/ui/#iso:std:iso:13120:ed-1:v1:en`.

ISO/IEC 9075:1992(E) —Information technology —Database languages —SQL (1992). International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC).

ISO/IEC 9126-1 —Software Engineering —Product Quality (2001). International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC).

Johnson, Rod et al. (2009). *Spring Framework Version 3.0.M3 - Reference Documentation*. URL: `http://static.springsource.org/spring/docs/3.0.0.M3/reference/html/`.

Jong, Jos de (Aug. 2013). *Timeline - CHAP Links Library*. Almende B.V. URL: `http://almende.github.io/chap-links-library/timeline.html`.

Jonsson, J. and B. Kaliski (Feb. 2003). *Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1*. RFC 3447 (Informational). Internet Engineering Task Force. URL: `http://www.ietf.org/rfc/rfc3447.txt`.

Kaliski, B. (Mar. 1998). *PKCS #1: RSA Encryption Version 1.5*. RFC 2313 (Informational). Obsoleted by RFC 2437. Internet Engineering Task Force. URL: `http://www.ietf.org/rfc/rfc2313.txt`.

Kemper, A. and A. Eickler (2011). *Datenbanksysteme: Eine Einführung*. Oldenbourg Wissenschaftsverlag. ISBN: 9783486598346. URL: `http://books.google.at/books?id=xpNefMq5nYwC`.

Khan, Andy (Mar. 2013). *Java Excel API - A Java API to read, write and modify Excel spreadsheets*. URL: `http://www.andykhan.com/jexcelapi/`.

King, Gavin et al. (2004). *HIBERNATE - Relational Persistence for Idiomatic Java, Reference Documentation 3.3.2.GA*. Red Hat Middleware, LLC. URL: `http://docs.jboss.org/hibernate/orm/3.3/reference/en/html/`.

Kurz, Michael, Gerald Müllan, and Martin Marinschek (2009). *JavaServer Faces 2.0*. Dpunkt.Verlag GmbH. ISBN: 3898646068. URL: `http://www.amazon.com/JavaServer-Michael-Gerald-M%C3%83%C2%BCllan-Marinschek/dp/3898646068?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=3898646068`.

Lehmkühler, Andreas and PDFBox Contributors (Apr. 2013). *Apache PDFBox - A Java PDF Library*. The Apache Software Foundation. URL: `http://pdfbox.apache.org/index.html`.

Leitch, Joe et al. (Mar. 2014). *google-gson - A Java library to convert JSON to Java objects and vice-versa*. Google, Inc. URL: `https://code.google.com/p/google-gson/`.

Lemmé, Thomas (Aug. 2012). "Entwurf und Implementierung einer Domain Specific Language zur effektiven Modellierung von Formulardaten im Bereich der medizinischen Dokumentation." Master Thesis. Fakultät für Informatik der Technischen Universität Wien.

Leonello, Chris (Apr. 2013). *jqPlot Charts and Graphs for jQuery*. URL: `http://www.jqplot.com/`.

Levenshtein, Vladimir I. (1966). "Binary Codes Capable of Correcting Deletions, Insertions and Reversals." In: *Soviet Physics Doklady* 10, p. 707.

Levey, Andrew S. (Mar. 1999). "A More Accurate Method To Estimate Glomerular Filtration Rate from Serum Creatinine: A New Prediction Equation." In: *Annals of Internal Medicine* 130.6, p. 461. DOI: 10.7326/0003-4819-130-6-199903160-00002. URL: http://dx.doi.org/10.7326/0003-4819-130-6-199903160-00002.

Levine, David K., Gunnar Teege, and Gerhard Hagerer (2013). *Jarnal - Java Notetaker and PDF Annotator*. URL: http://jarnal.wikispaces.com/.

Li, Ian (2013). *Raphael SketchPad*. URL: http://ianli.com/sketchpad/.

Marasteanu, Alexandru (Apr. 2013). *sprintf for JavaScript*. URL: https://github.com/alexei/sprintf.js.

marketsandmarkets.com (Jan. 2012). *Clinical Trial Management System (CTMS) Market - Site Management, Billing & Patient recruitment, Cloud (SaaS) Global Trends, Opportunities, Challenges and Forecasts till 2016*. marketsandmarkets.com. URL: http://www.marketsandmarkets.com/Market-Reports/clinical-trial-management-systems-market-470.html.

Martins, Samuel (Mar. 2013). *Raphaël IsPointInsidePath problems*. Google Groups. URL: https://groups.google.com/forum/#!msg/raphaeljs/LxQOgxtHzVE/kAG-7wShYKoJ.

Mattsson, Ulf T. (2005). "Database Encryption - How to Balance Security with Performance." In: *SSRN Electronic Journal*. DOI: 10.2139/ssrn.670561. URL: http://dx.doi.org/10.2139/ssrn.670561.

Medical University of Graz (Jan. 2012a). *Corporate Design Manual*. Version 1.5. URL: http://www.meduni-graz.at/images/content/file/themen/av/druckservice/CD_manual.pdf.

Medical University of Graz (Dec. 2012b). *Entwicklungsplan der Medizinischen Universität Graz 2013-2018*. Medical University of Graz. URL: http://www.meduni-graz.at/images/content/file/organisation/grundsatzdokumente/Entwicklungsplan.pdf.

Medizinproduktegesetz (MPG) (Nov. 1996). Bundesgesetzblatt I 657/1996. URL: https://www.ris.bka.gv.at/GeltendeFassung.wxe?Abfrage=Bundesnormen&Gesetzesnummer=10011003.

Momjian, Bruce (Jan. 2012). *PostgreSQL Internals Through Pictures*. EnterpriseDB. URL: http://momjian.us/main/writings/pgsql/internalpics.pdf.

Myers, Eugene W. (1986). "An O(ND) Difference Algorithm and Its Variations." In: *Algorithmica* 1, pp. 251–266.

No Magic (Apr. 2014). *MagicDraw - The multi award-winning UML business process, architecture, software and system modeling tool with teamwork support*. No Magic. URL: http://www.nomagic.com/products/magicdraw.html.

Ofner-Kopeinig, Petra and Maximilian Errath (Sept. 2004). "Randomizer: A Web-based Trial Management and Patient Randomization System for Clinical Trials." In: *Jahrestagung der Deutschen Gesellschaft für Medizinische Informatik, Biometrie und Epidemiologie (gmds)*. Vol. 49. Abstractband. Innsbruck.

OpenClinica, LLC (Mar. 2014). *OpenClinica - Clinical Trial Software for Electronic Data Capture*. OpenClinica, LLC. URL: https://www.openclinica.com.

OpenClinica, LLC and Contributors (Aug. 2013). *OpenClinica 3.1 Technical Documentation*. OpenClinica, LLC. URL: https://docs.openclinica.com/3.1/technical-documents.

Operationen- und Prozedurenschlüssel (2013). Deutsches Institut für Dokumentation und Information (DIMDI). URL: http://www.dimdi.de/static/de/klassi/ops/index.htm.

Oracle America, Inc. (2012). *Mojarra 2.1 - Overview*. Oracle America, Inc. URL: https://javaserverfaces.java.net/nonav/2.1/index.html.

Oracle Corporation (29 2014). *Jersey - RESTful Web Services in Java*. Oracle Corporation. URL: https://jersey.java.net/.

Organisationseinheit für Forschungsinfrastruktur (O-FIS) (Apr. 2014). *Jahresbericht 2011 / 2012 - Organisationseinheit für Forschungsinfrastruktur*. Zentrum für medizinische Forschung (ZMF). URL: http://www.meduni-graz.at/zmf/images/content/file/pdf/jahresbericht_2011+2012_screen.pdf.

O'Rourke, Joseph (1998). *Computational Geometry in C (Cambridge Tracts in Theoretical Computer Science)*. Cambridge University Press, pp. 239–245. ISBN: 0521649765. URL: http://www.amazon.com/Computational-Geometry-Cambridge-Theoretical-Computer/dp/0521649765?SubscriptionId=

0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0521649765.

Österreichische Post AG (May 2013). *Richtig Adressieren - Kriterien für maschinell bearbeitbare Sendungen*. URL: http://www.post.at/downloads/Druckfreigabe_1550_13_Adressieren_2013_comment.pdf?1399751920.

Oswald, Wolfgang et al. (Dec. 2013). "PROCHORN - Prospektives Register von Patientinnen mit Aderhautnävi." In: *Spektrum der Augenheilkunde* 27.6, pp. 299–304. ISSN: 1613-7523. DOI: 10.1007/s00717-013-0192-3. URL: http://dx.doi.org/10.1007/s00717-013-0192-3.

Parr, Terence (2007). *The definitive ANTLR reference: Building domain-specific languages*. Pragmatic Bookshelf.

PDF Reference, sixth edition: Adobe Portable Document Format version 1.7 (2006). Adobe Systems Inc.

Perera, Lakshan (Oct. 2008). *Really Simple Color Picker in jQuery*. URL: http://www.laktek.com/2008/10/27/really-simple-color-picker-in-jquery/.

PIC/S (Sept. 2007). *PIC/S Guidance - Good Practices for Computerised Systems in Regulated "GxP" Environments*. PIC/S Guidance. Pharmaceutical Inspection Convention and Pharmaceutical Inspection Co-operation Scheme (PIC/S).

Poppendieck, Mary and Tom Poppendieck (2003). *Lean Software Development: An Agile Toolkit*. Addison-Wesley Professional. ISBN: 0321150783. URL: http://www.amazon.com/Lean-Software-Development-Agile-Toolkit/dp/0321150783?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0321150783.

Prevoo, M. L. L. et al. (Jan. 1995). "Modified disease activity scores that include twenty-eight-joint counts development and validation in a prospective longitudinal study of patients with rheumatoid arthritis." In: *Arthritis & Rheumatism* 38.1, pp. 44–48. DOI: 10.1002/art.1780380107. URL: http://dx.doi.org/10.1002/art.1780380107.

Prigmore, Martyn (2007). *Introduction to Databases With Web Applications*. Prentice Hall, p. 395. ISBN: 0321263596. URL: http://www.amazon.com/Introduction-Databases-With-Web-Applications/dp/0321263596?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0321263596.

Q-FINITY Quality Management (May 2014). *Electronic Records and Electronic Signatures*. Q-FINITY Quality Management. URL: http://www.q-finity.com/annex-11-part-11.php.

Recommendation E.164: The international public telecommunication numbering plan (Nov. 2010). International Telecommunication Union. URL: https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-E.164-201011-I!!PDF-E&type=items.

Redmill, Felix (2001). "The COTS Debate in Perspective." In: *Computer Safety, Reliability and Security*, pp. 119–129. DOI: 10.1007/3-540-45416-0_13. URL: http://dx.doi.org/10.1007/3-540-45416-0_13.

Reese, G. (2003). *Java Database Best Practices*. O'Reilly Media. ISBN: 9781449365639. URL: http://books.google.at/books?id=TANVAgAAQBAJ.

Research Industrial Systems IT Engineering (RISE) GmbH (Apr. 2013). *SPICS - a documentation platform for medical processes*. Research group for Industrial Software (INSO), Vienna University of Technology Research Industrial Systems Engineering (RISE) - Research-, Development and Large Scale Project Consulting Ltd. URL: http://www.inso.tuwien.ac.at/projects/spics/.

Resig, John and jQuery Contributors (Apr. 2014). *jQuery - Javascript library for DOM-traversal and manipulation, event handling, animation, and Ajax*. The jQuery Foundation. URL: http://www.jquery.com/.

Reynolds, Toby (June 2011). "Clinical trials: can technology solve the problem of low recruitment?" In: *BMJ* 342. DOI: 10.1136/bmj.d3662.

RSA Laboratories (Nov. 1993). *PKCS #5: Password-Based Encryption Standard*. An RSA Laboratories Technical Note Version 1.5. RSA Laboratories. URL: ftp://ftp.rsasecurity.com/pub/pkcs/ps/pkcs-5.ps.

Sasaki, Yu and Kazumaro Aoki (2009). *Finding Preimages in Full MD5 Faster Than Exhaustive Search*. Springer-Verlag, pp. 134–152. DOI: 10.1007/978-3-642-01001-9_8. URL: http://dx.doi.org/10.1007/978-3-642-01001-9_8.

Schumacher, Martin and Gabriele Schulgen (2002). *Methodik klinischer Studien: Methodische Grundlagen der Planung, Durchführung und Auswertung (Statistik und ihre Anwendungen) (German Edition)*. Springer. ISBN: 3540433066. URL: http://www.amazon.com/Methodik-klinischer-Studien-Methodische-Durchf%C3%83%C2%BChrung/dp/3540433066?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=3540433066.

Science Park Graz GmbH (Apr. 2013). *Das war der Ideenwettbewerb 2012 — 2013*. Science Park Graz GmbH. URL: http://sciencepark.at/unser-angebot/ideenwettbewerb/ideenwettbewerb_2012_2013.php.

Sencha Inc. (Mar. 2014). *Sencha ExtJS - JavaScript Framework for Rich Desktop Apps*. Sencha Inc. URL: http://www.sencha.com/products/extjs/.

Sharnoff, David Muir, Tim Pierce, and Perl Contributors (Apr. 2009). *Text::Wrap - line wrapping to form simple paragraphs*. Google, Inc. URL: http://perldoc.perl.org/Text/Wrap.html.

Sibbald, B. and C. Roberts (June 1998). "Understanding controlled trials: Crossover trials." In: *BMJ* 316.7146, pp. 1719–1720. ISSN: 1468-5833. DOI: 10.1136/bmj.316.7146.1719. URL: http://dx.doi.org/10.1136/bmj.316.7146.1719.

Signaturgesetz - SigG (Aug. 1999). Bundesgesetzblatt I 190/1999. URL: http://www.ris.bka.gv.at/GeltendeFassung.wxe?Abfrage=Bundesnormen&Gesetzesnummer=10003685.

Simonic, Klaus M. and Günther Gell (Sept. 2001). *MAGDA-LENA - Datenschutz-Policy für die Kommunikation in Forschung und Lehre*. LKH/Universitätsklinikum Graz. URL: https://www.medunigraz.at/imi/de/projects/DS-Policy-FL-V1-1.pdf.

Singh, Inderjeet et al. (2002). *Designing Enterprise Applications with the J2EE Platform (2nd Edition)*. Addison-Wesley Professional, pp. 160+. ISBN: 0201787903. URL: http://www.amazon.com/Designing-Enterprise-Applications-Platform-Edition/dp/0201787903?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0201787903.

Smidt, Nynke (Apr. 2008). *STAndards for the Reporting of Diagnostic accuracy studies*. Department of Clinical Epidemiology & Biostatistics, Academic Medical Center, University of Amsterdam. URL: http://www.stard-statement.org/.

Spat, Stephan et al. (2012). "A Mobile Android-Based Application for In-hospital Glucose Management in Compliance with the Medical Device Directive for Software." In: *Wireless Mobile Communication and Healthcare*. Ed. by KonstantinaS. Nikita et al. Vol. 83. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer Berlin Heidelberg, pp. 211–216. ISBN: 978-3-642-29733-5. DOI: 10.1007/978-3-642-29734-2_29. URL: http://dx.doi.org/10.1007/978-3-642-29734-2_29.

Statistik Austria (Sept. 2011). *Statistik Austria - Straßen*. Bundesanstalt Statistik Österreich. URL: http://www.statistik.at/web_de/klassifikationen/regionale_gliederungen/strassen/index.html.

Strachan, James et al. (Mar. 2013). *Apache Commons CLI library*. The Apache Software Foundation. URL: http://commons.apache.org/cli.

Sucan, Mihai (2013). *PaintWeb: Online painting application*. URL: http://code.google.com/p/paintweb/.

SurveyMonkey (2013). *SurveyMonkey: Free online survey software & questionnaire tool*. SurveyMonkey. URL: https://www.surveymonkey.com/.

Tedstone, Barry (2010). *Computer Systems Validation - Content of the Audit Trail*. URL: http://www.csv-qa.com/articles/electronic-records/audit-trail-1.

The Episcopal Church (2000). *Book of Common Prayer (1979, Personal Size Economy, Black)*. Oxford University Press, USA. ISBN: 0195287134. URL: http://www.amazon.com/Common-Prayer-Personal-Economy-Black/dp/0195287134?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0195287134.

Tijms, Arjan et al. (Dec. 2013). *Explicit support for dynamic component tree manipulation*. Java Community Process. URL: https://java.net/jira/browse/JAVASERVERFACES_SPEC_PUBLIC-1007.

Torgerson, David J. and Carole J. Torgerson (2008). *Designing Randomised Trials in Health, Education and the Social Sciences: An Introduction*. Palgrave Macmillan. ISBN: 0230537359. URL: http://www.amazon.com/Designing-Randomised-Trials-Education-Sciences/dp/0230537359?

SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0230537359.

Trial By Fire Solutions, LLC (Mar. 2014). *SimpCTMS - Clinical Trial Management System*. Trial By Fire Solutions, LLC. URL: http://www.simplectms.com/ctms-features/.

Truskaller, Thomas (Sept. 2003). "Data Integration into a Gene Expression Database." Master Thesis. Institute of Biomedical Engineering, University of Technology, Graz, Austria.

Varaksin, Oleg et al. (Nov. 2013). *primefaces-extensions (Additional JSF 2 components for PrimeFaces)*. URL: https://github.com/primefaces-extensions.

Walnes, Joe, Jörg Schaible, and XStream Committers (Dec. 2013). *XStream is a simple library to serialize objects to XML and back again*. URL: http://xstream.codehaus.org/.

Wang, Duolao and Ameet Bakhai (2006). *Clinical Trials - A Practical Guide to Design, Analysis, and Reporting*. Remedica Publishing. ISBN: 1901346722. URL: http://www.amazon.com/Clinical-Trials-Practical-Analysis-Reporting/dp/1901346722?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=1901346722.

WebGL Specification Version 1.0.2 (Mar. 2013). Khronos Group. URL: https://www.khronos.org/registry/webgl/specs/1.0/.

Weng, Chunhua et al. (June 2010). "Formal representation of eligibility criteria: A literature review." In: *Journal of Biomedical Informatics* 43.3, pp. 451–467. DOI: 10.1016/j.jbi.2009.12.004. URL: http://dx.doi.org/10.1016/j.jbi.2009.12.004.

W.H.A.T. (Wound Healing Analyzing Tool) Benutzerhandbuch (Oct. 2011). v20. Research Industrial Systems Engineering (RISE) GmbH.

Wikipedia Contributors (Feb. 2013). *Glossary of clinical research*. Wikipedia. URL: http://en.wikipedia.org/wiki/Glossary_of_clinical_research.

Wikipedia Contributors (Jan. 2014a). *Klinische Studie*. Wikipedia. URL: http://de.wikipedia.org/wiki/Klinische_Studie.

Wikipedia Contributors (Apr. 2014b). *Types of observational studies*. Wikipedia. URL: http://en.wikipedia.org/wiki/Observational_study#Types_of_observational_studies.

Winch, Rob (July 2013). *Spring Security Java Config Preview: Method Security*. URL: http://spring.io/blog/2013/07/04/spring-security-java-config-preview-method-security/.

Wrba, Thomas et al. (Jan. 2011). *Workflow-Szenarien zwischen Routine und Wissenschaft*. Zentrum für med. Statistik, Informatik und Intelligente Systeme (CeMSIIS). URL: http://cemsiis-akim.meduniwien.ac.at/fileadmin/msi_akim/AKIM/INT/AKIM_IntegrationRW_Workflows_Teil1.pdf.

Wrba, Thomas (Mar. 2014). *Klinisches Studiensystem (KSS)*. Zentrum für med. Statistik, Informatik und Intelligente Systeme (CeMSIIS). URL: http://cemsiis-akim.meduniwien.ac.at/fl/plattformen/kss/.

Yoder, Dan and Geoffrey McGill (July 2010). *Datejs - an open-source JavaScript Date Library*. Coolite Inc. URL: http://www.datejs.com/.

Zhou, Xin, Fusheng Wang, and Carlo Zaniolo (2006). *Efficient Temporal Coalescing Query Support in Relational Database Systems*. Springer-Verlag, pp. 676–686. DOI: 10.1007/11827405_66. URL: http://dx.doi.org/10.1007/11827405_66.

Zubatch, Michelle (Apr. 2006). *Value of Hosted Clinical Data Environments*. Bio-IT World. URL: http://www.bio-itworld.com/issues/2006/april/cdms/.

All URLs were checked and valid as of May 19th, 2014.

Appendix

Appendix A.

Phoenix CTMS Brochure

Medizinische Universität Graz

Phoenix

Clinical Trial Management System

Studienmanagementsysteme

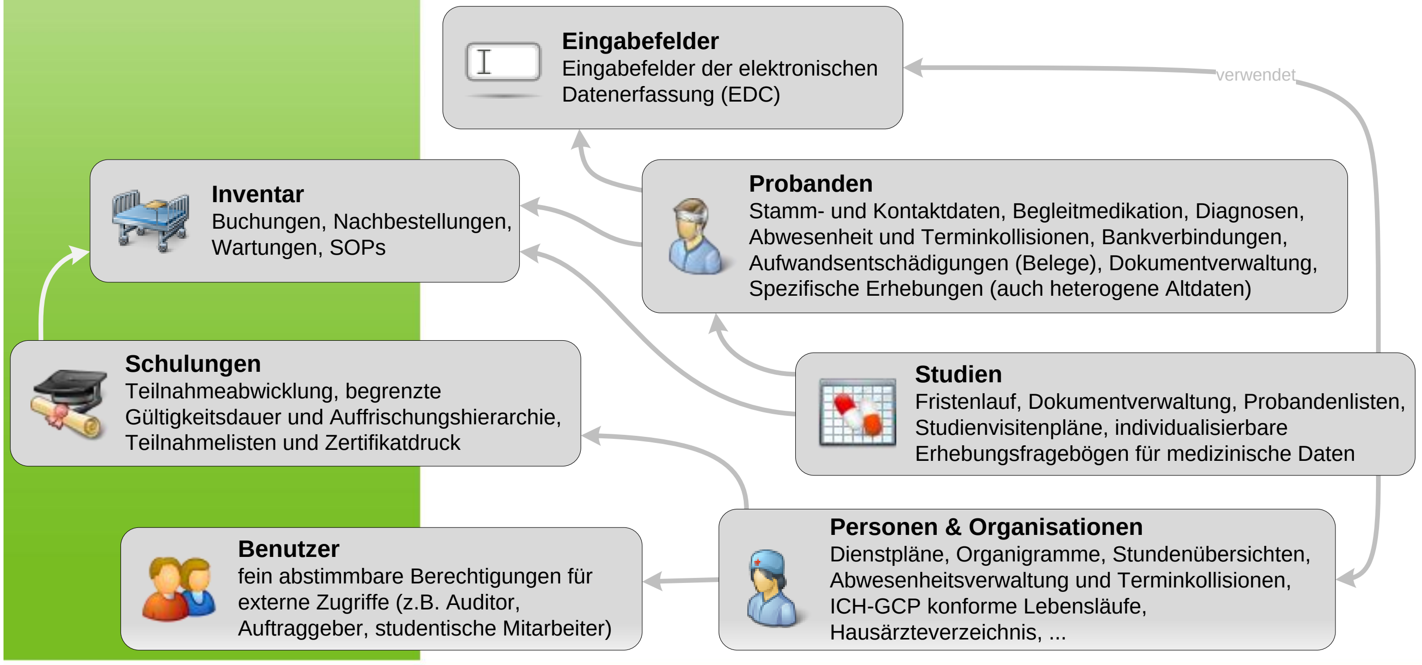
Klinische Prüfungen sind notwendig, um verlässliche neue medizinische Forschungsergebnisse zu gewinnen. Voraussetzung für die Durchführung derselben ist, dass sich die einzelnen Prüfzentren an die gesetzlichen Vorgaben (Arzneimittel- und Medizinproduktegesetz, ICH-GCP) halten.
Um die zahlreichen und komplexen Geschäftsprozesse eines klin. Prüfzentrums abzubilden, kommen klin. Studienmanagementsysteme (CTMS - Clinical Trial Management Systems) zum Einsatz.

Vorstellung Phoenix CTMS

Nach mehrjähriger Entwicklungszeit ist ein neues derartiges Datenbanksystem verfügbar, das die speziellen Anforderungen eines ISO-zertifizierten, österr. Prüfzentrums erfüllt. Einer der Schwerpunkte liegt bei der nachweisbaren Sicherheit abgespeicherter Daten gegen unbefugtes Auslesen (Verschlüsselung von Datensätzen) sowie deren Fälschung (digitale Signaturen für Datensätze). Eine automatische, ICH-GCP-konforme Protokollierung aller Datensatzänderungen stellt die lückenlose Nachvollziehbarkeit sicher. Diese Eigenschaften machen die umgesetzte Lösung auch besonders attraktiv für vergleichbare klin. Prüfzentren in Österreich.

Übersicht

Die Module des „Phoenix CTMS“ bieten vielfältige Verwaltungsfunktionen für die involvierten Ressourcen.



Eingabefelder
Eingabefelder der elektronischen Datenerfassung (EDC)

Inventar
Buchungen, Nachbestellungen, Wartungen, SOPs

Probanden
Stamm- und Kontaktdaten, Begleitmedikation, Diagnosen, Abwesenheit und Terminkollisionen, Bankverbindungen, Aufwandsentschädigungen (Belege), Dokumentverwaltung, Spezifische Erhebungen (auch heterogene Altdaten)

Schulungen
Teilnahmeabwicklung, begrenzte Gültigkeitsdauer und Auffrischungshierarchie, Teilnahmelisten und Zertifikatdruck

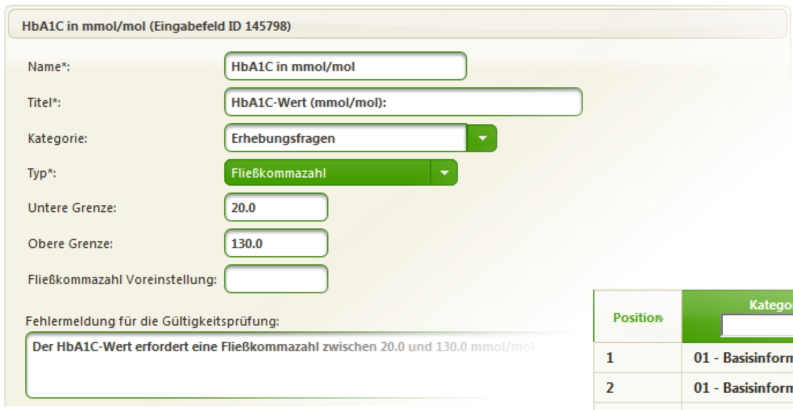
Studien
Fristenlauf, Dokumentverwaltung, Probandenlisten, Studienvisitenpläne, individualisierbare Erhebungsfragebögen für medizinische Daten

Benutzer
fein abstimmbare Berechtigungen für externe Zugriffe (z.B. Auditor, Auftraggeber, studentische Mitarbeiter)

Personen & Organisationen
Dienstpläne, Organigramme, Stundenübersichten, Abwesenheitsverwaltung und Terminkollisionen, ICH-GCP konforme Lebensläufe, Hausärzteverzeichnis, ...

Datenerhebung bei Probanden

Die Eignung eines Kandidaten als Proband für eine klin. Prüfung erfordert im Vorfeld die Erfüllung eines Profils, z.B. Indikatoren, welche die Ausprägung eines bestimmten Krankheitsbildes bemessen.
Während der Rekrutierungsphase ist die Anzahl erfasster, geeigneter Kandidaten eine wesentliche Information. Bereits zuvor bedeutet die Möglichkeit der spontanen Auskunft über das Rekrutierungspotential an anfragende Auftraggeber einen Wettbewerbsvorteil.



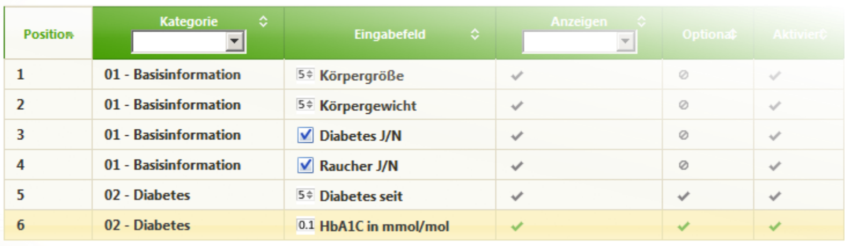
HbA1C in mmol/mol (Eingabefeld ID 145798)

Name*:	HbA1C in mmol/mol
Titel*:	HbA1C-Wert (mmol/mol):
Kategorie:	Erhebungsfragen
Typ*:	Fließkommazahl
Untere Grenze:	20.0
Obere Grenze:	130.0
Fließkommazahl Voreinstellung:	

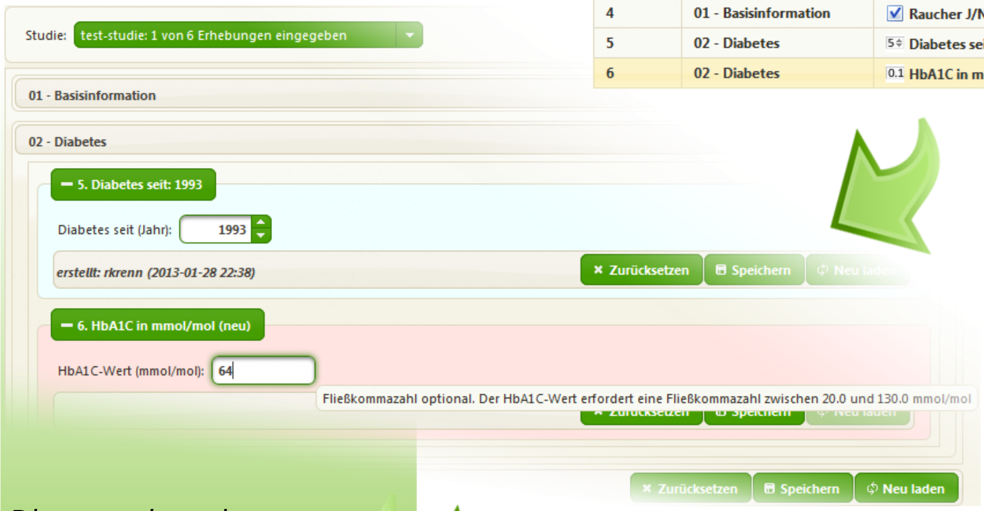
Fehlermeldung für die Gültigkeitsprüfung:
Der HbA1C-Wert erfordert eine Fließkommazahl zwischen 20.0 und 130.0 mmol/mol

Individuelle Eingabefelder (Text-, Zahleneingabe, CheckBox, Auswahl usw.) für zu erfassende Daten werden erstellt…

… um damit die Erhebung für eine Studie zusammenzustellen.



Position	Kategorie	Eingabefeld	Anzeigen	Optional	Aktiviert
1	01 - Basisinformation	Körpergröße	✓	⊘	✓
2	01 - Basisinformation	Körpergewicht	✓	⊘	✓
3	01 - Basisinformation	Diabetes J/N	✓	⊘	✓
4	01 - Basisinformation	Raucher J/N	✓	⊘	✓
5	02 - Diabetes	Diabetes seit	✓	✓	✓
6	02 - Diabetes	HbA1C in mmol/mol	✓	✓	✓



Studie: test-studie: 1 von 6 Erhebungen eingegeben

01 - Basisinformation

02 - Diabetes

5. Diabetes seit: 1993

Diabetes seit (Jahr): 1993

erstellt: rkrenn (2013-01-28 22:38)

6. HbA1C in mmol/mol (neu)

HbA1C-Wert (mmol/mol): 64

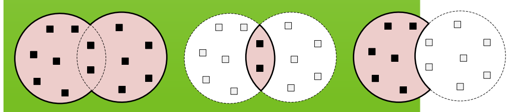
Fließkommazahl optional. Der HbA1C-Wert erfordert eine Fließkommazahl zwischen 20.0 und 130.0 mmol/mol

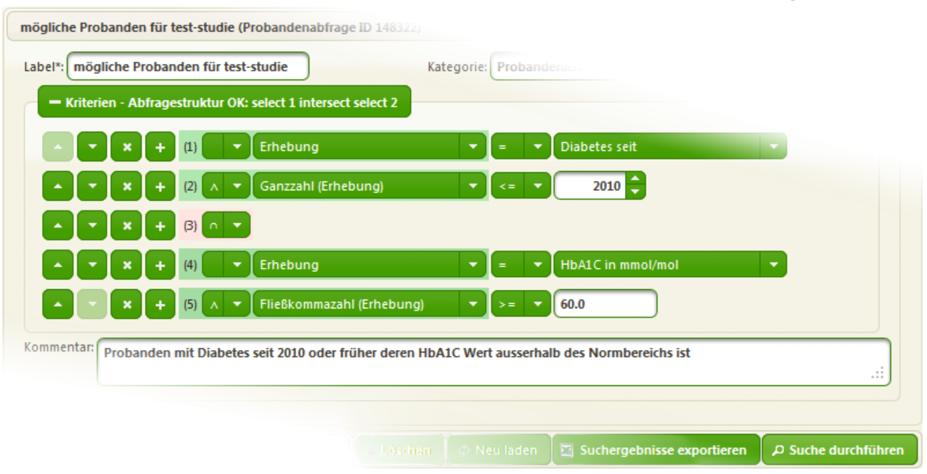
Die Erhebung wird als Fragebogen präsentiert und bei erfassten Kandidaten während der Rekrutierungsphase ausgefüllt.

Neben Erhebungen kommt dieses flexible Konzept auch bei Probandenlisten von Studien zum Einsatz. Dadurch können vorhandene, heterogene Probandendaten in das Phoenix CTMS migriert werden.

Die anwachsende Probandendatenbank kann jederzeit abgefragt werden. Ein Abfrageeditor erlaubt die Verknüpfung beliebiger Kriterien.

Dabei besonders hervorzuheben ist die Unterstützung von Mengenoperationen:





mögliche Probanden für test-studie (Probandenabfrage ID 148322)

Label*: mögliche Probanden für test-studie Kategorie: Probanden…

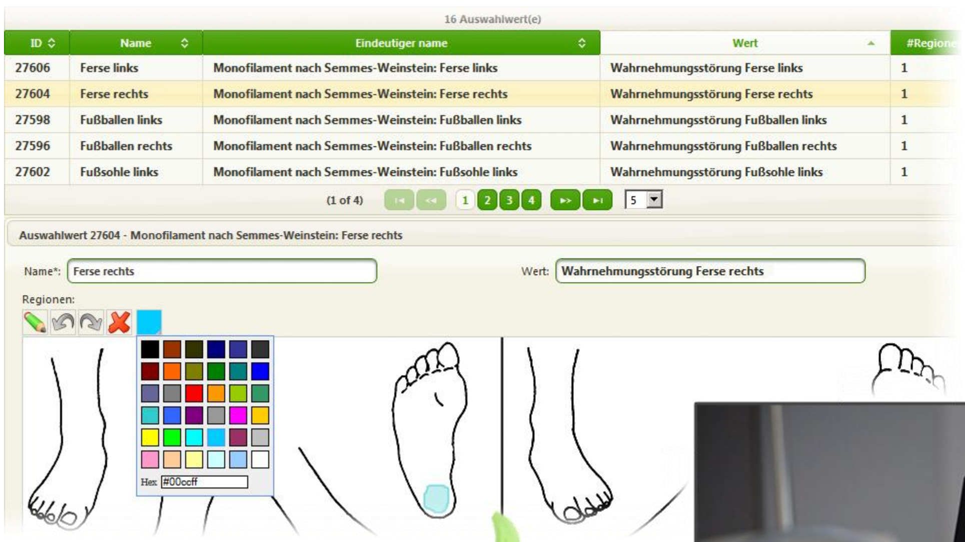
Kriterien - Abfragestruktur OK: select 1 intersect select 2

(1)		Erhebung	=	Diabetes seit
(2)	∧	Ganzzahl (Erhebung)	<=	2010
(3)	∩			
(4)		Erhebung	=	HbA1C in mmol/mol
(5)	∧	Fließkommazahl (Erhebung)	>=	60.0

Kommentar: Probanden mit Diabetes seit 2010 oder früher deren HbA1C Wert ausserhalb des Normbereichs ist

Stifteingaben

Stifteingaben als Eingabefelder ermöglichen Erhebungen, die bislang nur auf Papier denkbar waren.
Um eine Auswertbarkeit bei Abfragen zu erreichen, können ein Hintergrundbild sowie Regionen definiert werden. Durch Ankreuzen werden diese wie in den dargestellten Beispielen dann ausgewählt.



16 Auswahlwert(e)

ID	Name	Eindeutiger name	Wert	#Regione
27606	Ferse links	Monofilament nach Semmes-Weinstein: Ferse links	Wahrnehmungsstörung Ferse links	1
27604	Ferse rechts	Monofilament nach Semmes-Weinstein: Ferse rechts	Wahrnehmungsstörung Ferse rechts	1
27598	Fußballen links	Monofilament nach Semmes-Weinstein: Fußballen links	Wahrnehmungsstörung Fußballen links	1
27596	Fußballen rechts	Monofilament nach Semmes-Weinstein: Fußballen rechts	Wahrnehmungsstörung Fußballen rechts	1
27602	Fußsohle links	Monofilament nach Semmes-Weinstein: Fußsohle links	Wahrnehmungsstörung Fußsohle links	1

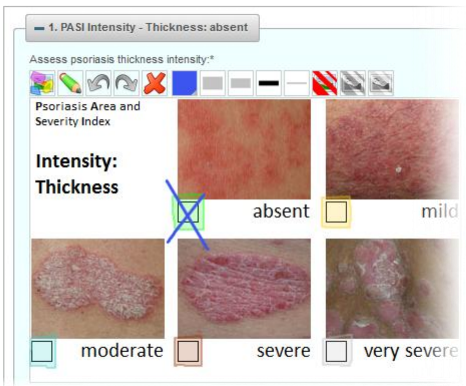
(1 of 4) 1 2 3 4 5

Auswahlwert 27604 - Monofilament nach Semmes-Weinstein: Ferse rechts

Name*: Ferse rechts Wert: Wahrnehmungsstörung Ferse rechts

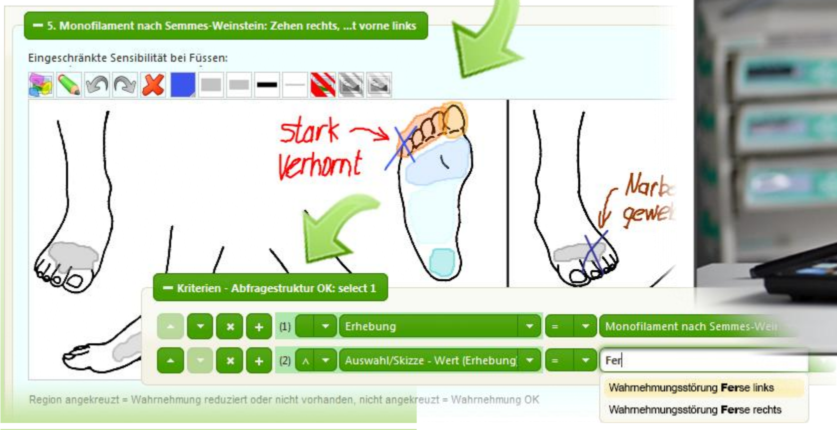
Regionen:

Hex: #00ccff



1. PASI Intensity - Thickness: absent

Assess psoriasis thickness intensity:*



5. Monofilament nach Semmes-Weinstein: Zehen rechts, ...t vorne links

Eingeschränkte Sensibilität bei Füssen:

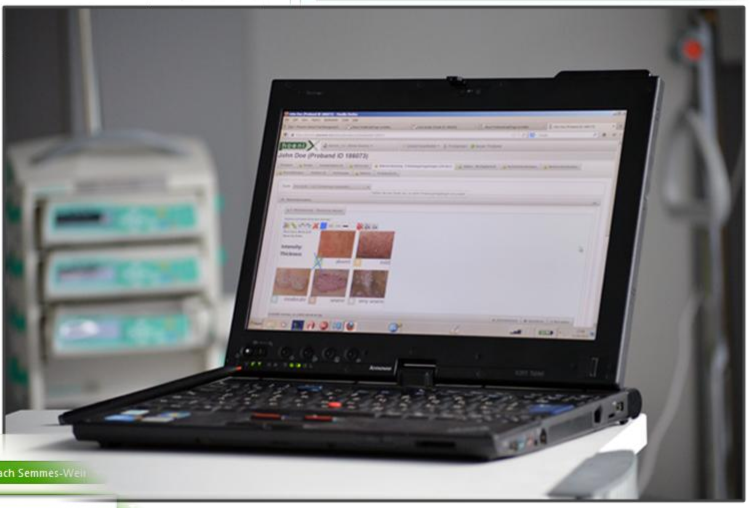
Kriterien - Abfragestruktur OK: select 1

(1) Erhebung = Monofilament nach Semmes-Wei

(2) Auswahl/Skizze - Wert (Erhebung = Fer

Wahrnehmungsstörung Ferse links
Wahrnehmungsstörung Ferse rechts

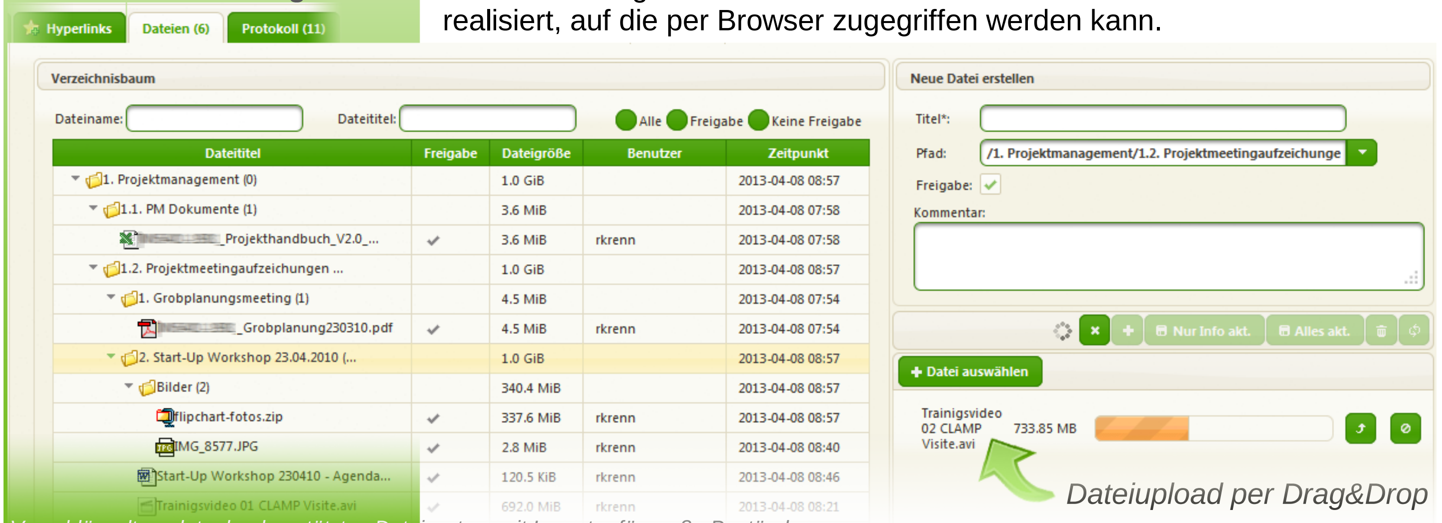
Region angekreuzt = Wahrnehmung reduziert oder nicht vorhanden, nicht angekreuzt = Wahrnehmung OK



Erhebungen durch Einsatz von Notebooks/Tablets mit Digitizer

Dokumentverwaltung

Die Verwaltung von Dokumenten ist durch virtuelle Verzeichnisstrukturen realisiert, auf die per Browser zugegriffen werden kann.



Hyperlinks | Dateien (6) | Protokoll (11)

Verzeichnisbaum

Dateiname: Dateititel: Alle Freigabe Keine Freigabe

Dateititel	Freigabe	Dateigröße	Benutzer	Zeitpunkt
1. Projektmanagement (0)		1.0 GiB		2013-04-08 08:57
1.1. PM Dokumente (1)		3.6 MiB		2013-04-08 07:58
[redacted]_Projekthandbuch_V2.0_...	✓	3.6 MiB	rkrenn	2013-04-08 07:58
1.2. Projektmeetingaufzeichungen ...		1.0 GiB		2013-04-08 08:57
1. Grobplanungsmeeting (1)		4.5 MiB		2013-04-08 07:54
[redacted]_Grobplanung230310.pdf	✓	4.5 MiB	rkrenn	2013-04-08 07:54
2. Start-Up Workshop 23.04.2010 (...		1.0 GiB		2013-04-08 08:57
Bilder (2)		340.4 MiB		2013-04-08 08:57
flipchart-fotos.zip	✓	337.6 MiB	rkrenn	2013-04-08 08:57
IMG_8577.JPG	✓	2.8 MiB	rkrenn	2013-04-08 08:40
Start-Up Workshop 230410 - Agenda...	✓	120.5 KiB	rkrenn	2013-04-08 08:46
Trainigsvideo 01 CLAMP Visite.avi	✓	692.0 MiB	rkrenn	2013-04-08 08:21

Neue Datei erstellen

Titel*:
Pfad: /1. Projektmanagement/1.2. Projektmeetingaufzeichunge
Freigabe: ✓
Kommentar:

Nur Info akt. | Alles akt.

Datei auswählen

Trainigsvideo 02 CLAMP Visite.avi 733.85 MB

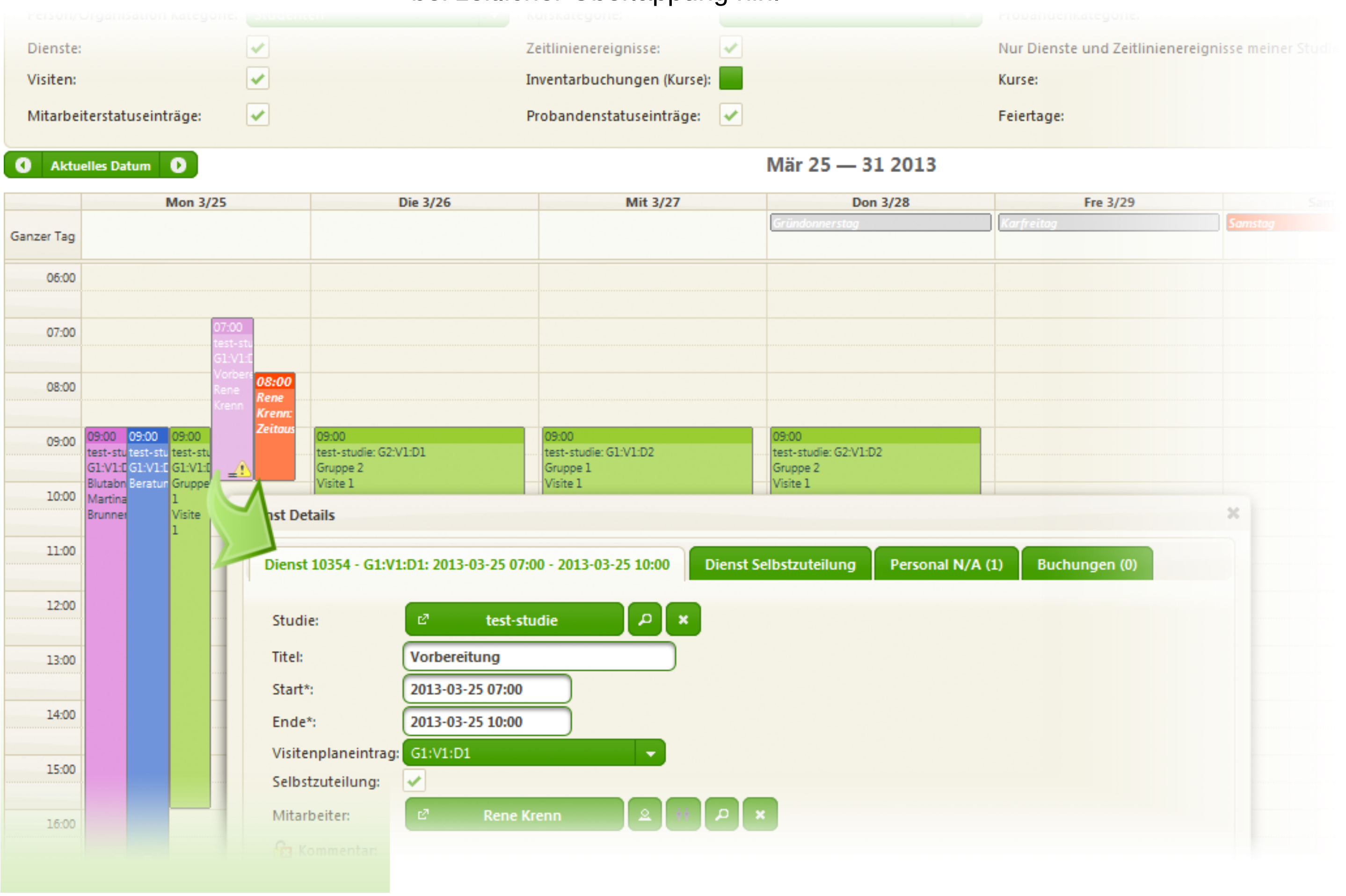
Dateiupload per Drag&Drop

Verschlüsseltes, datenbankgestütztes Dateisystem mit Importer für große Bestände

Medizinische Universität Graz

Kalender - Terminkollisionen

Kalenderansichten stellen Ereignisse filterbar dar und weisen auf Probleme bei zeitlicher Überlappung hin.



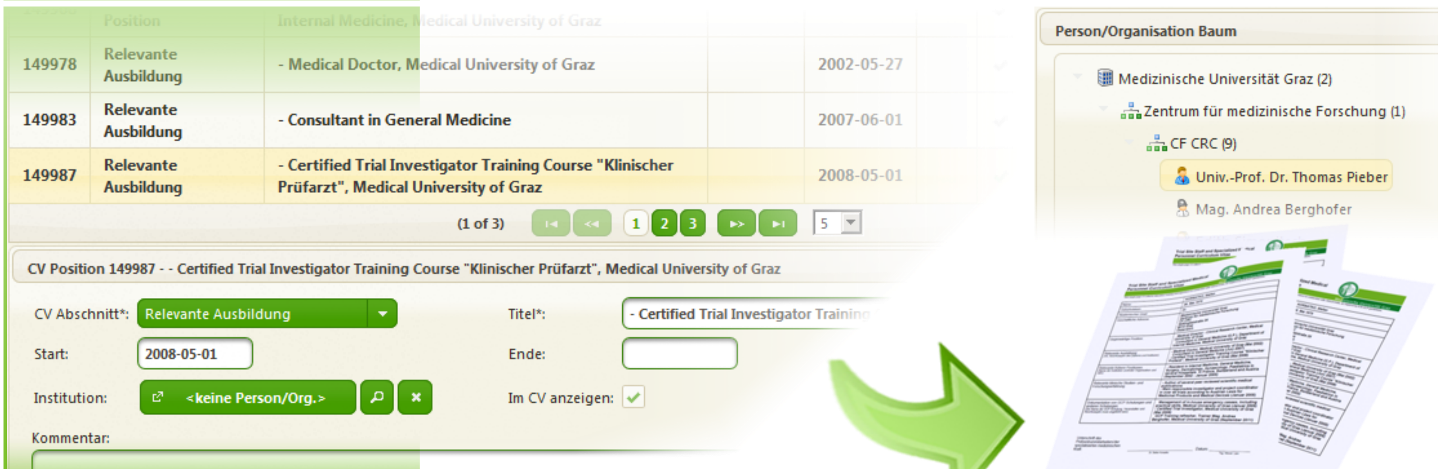
Dienste:	✓	Zeitlinienereignisse:	✓	Nur Dienste und Zeitlinienereignisse meiner Stud
Visiten:	✓	Inventarbuchungen (Kurse):	🟩	Kurse:
Mitarbeiterstatuseinträge:	✓	Probandenstatuseinträge:	✓	Feiertage:

Aktuelles Datum
Mär 25 — 31 2013
Mon 3/25
Die 3/26
Mit 3/27
Don 3/28
Fre 3/29
Ganzer Tag
Gründonnerstag
Karfreitag
Samstag
06:00
07:00
08:00
09:00
10:00
11:00
12:00
13:00
14:00
15:00
16:00
07:00 test-stu G1:V1:D Vorbere Rene Krenn
08:00 Rene Krenn: Zeitaus
09:00 test-stu G1:V1:D Blutabn Martina Brunner
09:00 test-stu G1:V1:D Beratun
09:00 test-stu G1:V1:D Gruppe 1 Visite 1
09:00 test-studie: G2:V1:D1 Gruppe 2 Visite 1
09:00 test-studie: G1:V1:D2 Gruppe 1 Visite 1
09:00 test-studie: G2:V1:D2 Gruppe 2 Visite 1
nst Details
Dienst 10354 - G1:V1:D1: 2013-03-25 07:00 - 2013-03-25 10:00
Dienst Selbstzuteilung
Personal N/A (1)
Buchungen (0)

Studie:	test-studie
Titel:	Vorbereitung
Start*:	2013-03-25 07:00
Ende*:	2013-03-25 10:00
Visitenplaneintrag:	G1:V1:D1
Selbstzuteilung:	✓
Mitarbeiter:	Rene Krenn
Kommentar:	

Einheitliche Lebensläufe

Die Bereitstellung eines standardisierten Lebenslaufs für med. und klin. Personal aller beteiligten Mitarbeiter ist eine weitere Vorgabe (ICH GCP), die im Phoenix CTMS unterstützt wird.



	Position	Internal Medicine, Medical University of Graz	
149978	Relevante Ausbildung	- Medical Doctor, Medical University of Graz	2002-05-27
149983	Relevante Ausbildung	- Consultant in General Medicine	2007-06-01
149987	Relevante Ausbildung	- Certified Trial Investigator Training Course "Klinischer Prüfarzt", Medical University of Graz	2008-05-01

CV Position 149987 - - Certified Trial Investigator Training Course "Klinischer Prüfarzt", Medical University of Graz

CV Abschnitt*:	Relevante Ausbildung	Titel*:	- Certified Trial Investigator Trainin
Start:	2008-05-01	Ende:	
Institution:	<keine Person/Org.>	Im CV anzeigen:	✓
Kommentar:			

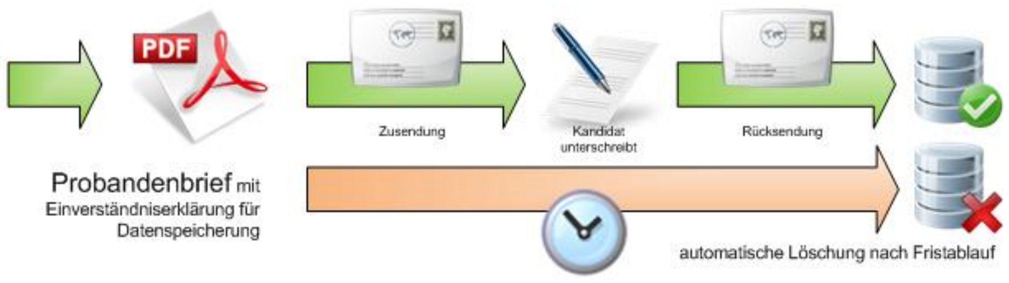
Person/Organisation Baum
- Medizinische Universität Graz (2)
  - Zentrum für medizinische Forschung (1)
    - CF CRC (9)
      - Univ.-Prof. Dr. Thomas Pieber
      - Mag. Andrea Berghofer

Mit einem Mausklick können die aktuellen Lebensläufe aller abgefragten Mitarbeiter als PDF-Datei geöffnet werden.

Hierarchien des Organigramms werden z.B. für die Geschäftsadresse des Mitarbeiters herangezogen.

Datenschutzkonformer Rekrutierungsablauf

Erst durch die schriftliche Einverständniserklärung des Probanden ist die Ablage seiner Daten erlaubt. Ein nach telefonischer Kontaktaufnahme erfasster Kandidat wird deshalb gelöscht, wenn seine unterzeichnete Erklärung nicht rechtzeitig zurückerhalten wird.


PDF
Probandenbrief mit Einverständniserklärung für Datenspeicherung
Zusendung
Kandidat unterschreibt
Rücksendung
automatische Löschung nach Fristablauf

Schutz gegen unbefugtes Einsehen

Durch Spaltenverschlüssellung bleiben die sensiblen personenbezogenen Daten von Probanden auch für technische Betreuer uneinsehbar.

Durch die Haltung eines zufälligen Initialisierungsvektors pro Feldwert verschleiert die symm. Verschlüsselung (AES) auch idente Werte, z.B. bei gleich lautenden Vornamen oder Auswahlangaben wie Geschlecht. Bei Bedarf werden Suchabfragen durch separat geführte Hash-Abbilder ermöglicht.

Datenbanktabelle unverschlüsselt

...	first_name	last_name	...
		...	
	John	**Smith**	
		...	

Datenbanktabelle mit Spaltenverschlüsselung

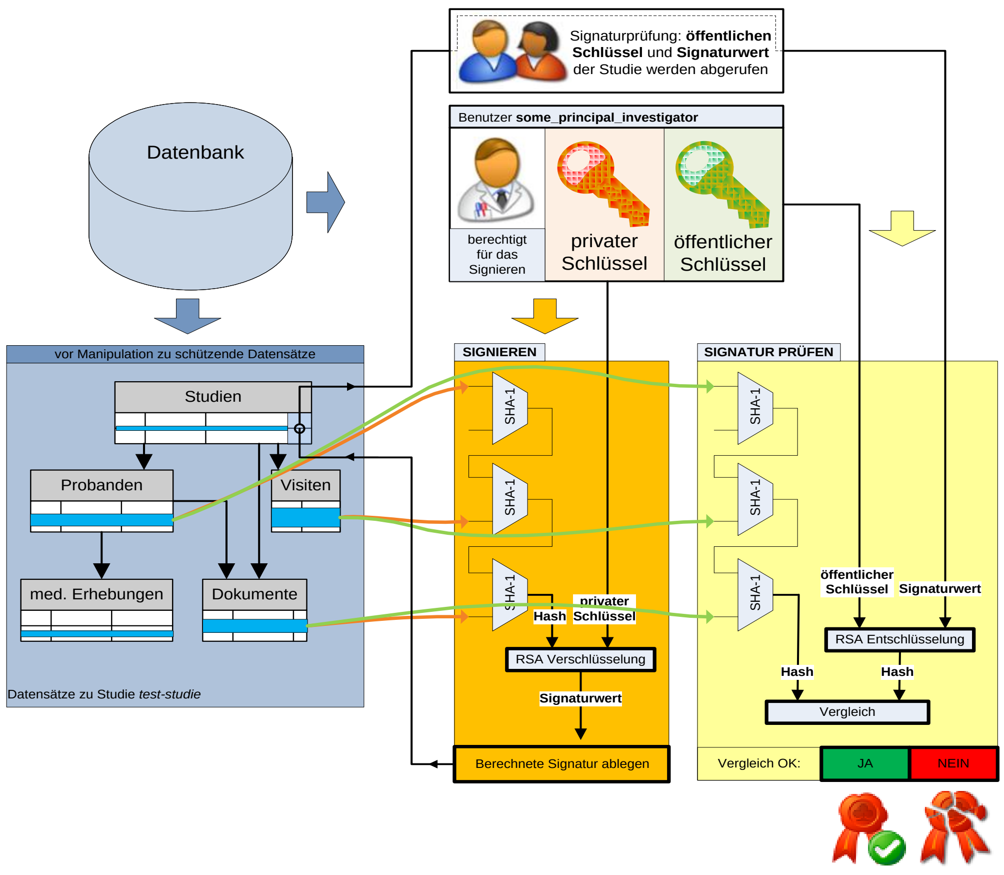
...	first_name hash	last_name **iv**	last_name **encrypted**	last_name **hash**	...
		...	...	...	
	1B3DA01..	**Aü#Zgn5..**	**pttLWvt2..**	**B30C197..**	
		...	...	...	

Schutz gegen Manipulation

Nach Abschluss einer Studie ermöglichen digital signierte Datensätze den Nachweis unveränderter Datenbankinhalte.

Benutzer im Phoenix CTMS erhalten hinter den Kulissen bei ihrer Erstellung ein RSA-Schlüsselpaar. Der private Schlüssel ist durch Spaltenverschlüsselung nur mit Hilfe des Benutzerpassworts einsehbar, und wird bei der Erzeugung einer Signatur benötigt. Der öffentliche Schlüssel wird beim Prüfen einer Signatur durch einen beliebigen Benutzer abgerufen.

Bei der Signaturerstellung werden alle gegen Manipulation abzusichernde Datensätze einbezogen. Mittels Prüfsummen ebenso erfasst sind damit Dokumente, die als verschlüsselte Dateien separat im Dateisystem abgelegt sind.



Appendix B.

Requirement Analysis

StudienManagementSystem

Trial Site Management System

Anforderungsanalyse

Inhaltsverzeichnis

Abbildungen

Tabellen

1 Dokument Kontrolle

1.1 Dokumentversion

Datum	Version	Kommentar	Inhalt
2011-03-18	0.1	Textsatzprogramm Latex, Ausstattungsdatenbank, Personaldatenbank, Benutzerdatenbank, Schulungsdatenbank	rkrenn, skorsatko, aberghofer, jgerdova, mbrunner, hkojzar
2011-03-23	0.5	Latex, Studiendatenbank	rkrenn, sdeller
2011-03-25	0.6	Korrekturen für Studiendatenbank	rkrenn, skorsatko, aberghofer
2011-03-27	0.7	Textsatzprogram Microsoft Word, Korrekturen für Studiendatenbank	rkrenn, skorsatko, sdeller
2011-04-04	0.8	Studiendatenbank: Probandenliste, Dienstplan, Checklisten, Dokumentverwaltung, Ablage und Verwaltung, Änderungen Pre-Screening Fragen Personaldatenbank: Änderungen Ablage und Verwaltung	rkrenn, skorsatko, sdeller, aberghofer, jgerdova, mbrunner
2011-04-06	0.9	Probandendatenbank	rkrenn
2011-04-12	1.0	Erweiterungen für mehrere Abteilungen (Core Facilities)	rkrenn, skorsatko, sdeller

Tabelle 1: Dokumenthistorie

1.2 Empfängerliste

Name	Bereich	Organisation
Dr. Stefan Korsatko	Prüfarzt, Studienleitung	Clinical Research Center
Andrea Berghofer	SOP, Studienverwaltung	Clinical Research Center
Dr. Sigrid Deller	Prüfarzt, Studienleitung	Clinical Research Center
Martina Brunner	Laborleitung	Clinical Research Center
Janka Gerdova	medizinisches Equipment	Clinical Research Center
Harald Kojzar	Studiendurchführung	Clinical Research Center
Thomas Truskaller	techn. Betreuung SMS Entwicklung	Joanneum Research
Bernd Tschapeller	techn. Betreuung SMS Entwicklung	Joanneum Research
Prof. Dr. Andreas Holzinger	Diplomarbeitsbetreuung	TU Graz
Prof. Dr. Frank Kappe	Diplomarbeitsbetreuer	TU Graz
Rene Krenn	Diplomand	TU Graz

Tabelle **2**: Empfängerliste

2 Einleitung

Das CRC (Clinical Research Center) des ZMF (Zentrum für medizinische Forschung) der Medizinischen Universität Graz hat das Projekt namens „Phoenix“ gestartet, mit dem Ziel im 2. Quartal 2012 die Zertifizierung seiner Prozesse abzuschliessen. Das CRC befasst sich mit der Durchführung klinischer Studien mit dem Schwerpunktbereichen Endokrinologie und Stoffwechselerkrankungen, deren ausgedehnte und detaillierte Prozessdokumentation nun automatisiert werden soll.

Das vorliegende Dokument beschreibt die wesentlichen Geschäftsprozesse in Verbindung mit einem Konzept in Textform, wie deren elektronische Abbildung durch ein webbasiertes, datenbankgestütztes Verwaltungssystem (SMS – StudienManagementSystem) erfolgen soll. Die Ausführungen sollen eine gemeinssame Basis der Auffassung der Problemstellung bilden, und ein Bild der angestrebten Lösung schaffen. Davon ausgehend kann das Identifizieren einzelner Anforderungen erfolgen, um die Anforderungsliste aufzustellen.

Die Softwareentwicklung des SMS ist ein Greenfield-Projekt, d.h. abgesehen von der Migration überschaubarer bestehender Datenbestände ist ansonsten keine Vorarbeit involviert.

Das SMS soll klassisch eine zentrale, relationale Datenbank aufbauen. Eine grobe Unterteilung in so weit wie möglich unabhängige Komponenten des Systems wird getroffen:

Ausstattungsdatenbank: Inventar, Wartungen

Personaldatenbank: Mitarbeiter, dynamisch generierte Lebensläufe

Benutzerdatenbank: Benutzer, kryptographische Methoden und Zugriffsberechtigungen

Schulungsdatenbank: Mitarbeiterschulungen, Teilnahmeverwaltung

Studiendatenbank: klinische Studien, Dienstplan, Dokumentverwaltung

Probandendatenbank: Probandendaten, Telefonatprotokollierung

In den nachfolgenden Abschnitten wird jede Komponente im Detail behandelt.

3 Ausstattungsdatenbank

3.1 Übersicht

Sämtliches Equipment, das von der Abteilung für klinische Studien verwaltet wird bzw. zur Verfügung steht soll zentral im StudienManagementSystem (SMS) registriert sein. Grundsätzlich handelt es sich dabei um folgende Objekte:

- Technische Geräte, z.B.: Zentrifugen, Kühlschränke, Computer, TV, Wiedergabegeräte, ...
- Medizinische Geräte, z.B.: Blutzuckermessgeräte, Termometer, Infusionsgeräte, ...
- Weiteres Inventar und reservierbare Ressourcen, z.B.: Betten, Räume

Die Abbildung dieser Informationen soll folgende Zwecke erfüllen:

1. Erinnerungen für Wartungen und Eichungen
2. Wissensdatenbank
3. Reservierungen
4. Ablage und Verwaltung

3.2 Erinnerungen für Wartungen und Eichungen

Ausgewählte Objekte wie medizinische Geräte müssen während der Nutzung im Rahmen von klinischen Studien in nachweisbar einwandfreien Betriebszustand sein. Viele dieser Gerätschaften unterliegen deshalb vorgegebenen und/oder vom Hersteller festgelegten Wartungsintervallen. Messgeräte müssen darüber hinaus regelmässigen Eichungen unterzogen werden.

Für betroffene Geräte soll Information für Wartungen und Eichungen hinterlegbar sein:

- periodische Wartungen, Inspektionen und Eichungen ausgehend von einem Startdatum
- zusätzlich individuelle, einmalige Wartungstermine
- Reminder: der für das Gerät eingetragene Verantwortliche soll in einer spezifizierbaren Zeitdauer vor einem bevorstehenden Wartungstermin darauf aufmerksam gemacht werden
- Standardbezeichnungen des Termins für Wartung, Inspektion oder Eichung oder individueller Text

Kalendertage sind als minimale Auflösung der Zeit ausreichend. Für die Umsetzung der Erinnerung wäre angedacht:

- Erinnerungsfunktion durch ein täglich ab Beginn der Erinnerungs-Zeitspanne versendetes Email
- Plakative Darstellung aller anstehenden Wartungstermine innerhalb ihrer angelaufenen Erinnerungszeitspanne in einer schnell einsehbaren oder permanent sichtbaren („sticky“) Liste der Weboberfläche des SMS
- Wartungsplan: Eine graphische Darstellung eines Kalenderjahres mit eingetragenen Wartungs-, Inspektions und Eichungsterminen soll eine schnelle Übersicht erlauben

Die Funktion kommt im Grunde jener von üblichen Terminverwaltungstools nahe.

3.3 Reservierungen

Der primäre Anwendungsfall, der die Reservierungsfunktion für Ausstattungsobjekte vorsieht, ist die Zuteilung von eingeladenen Probanden zu Betten oder Räumen durch die dafür zuständigen SMS Benutzer.

Betten: Als Beispiel kann bei der Verabreichung von Wirkstoffen im Rahmen einer durchzuführenden Studie zur gesundheitlichen Sicherheit oder zur kontinuierlichen Erfassung von Messdaten eine stationäre Aufnahme des Probanden erforderlich sein. Nun muss ein Bett für die vorgesehene Zeit reserviert sein, damit es nicht zu Mehrfachbelegungen kommt. Für Übersichtsdarstellungen in der Weboberfläche sowie zur lückenlosen Aufzeichnung soll auch festgehalten werden, welcher Proband den vorgesehenen Platz einnimmt.

Räume: Im Falle von ambulanter Verabreichung oder Messung werden für einen kommunizierten Termin zahlreiche Probanden eingeladen. In diesem Szenario erfolgt die Abwicklung in einem Raum, der für diese Zeit reserviert sein muss. Nun ist der Reservierung nicht nur ein Proband zu vermerken, sondern eine ganze Auswahl der für die Studie vorgesehenen Probanden.

Abgesehen von der Probandeninformation enthält eine Reservierung wie bereits angeführt folgende Informationen:

- Reservierungszeitspanne: Beginn-Zeitmarke und Dauer mit einer zeitlichen Auflösung von mind. 15 Minuten
- beantragender Benutzer: Der beantragende Benutzer soll seine eigenen Reservierungen bearbeiten und löschen können. Eine zusätzliche Zugriffsberechtigung soll ausgewählten Benutzern ermöglichen, auch fremde Reservierungen zu verändern.
- Person, für die reseviert wird: Dies ist typischer Weise die mit dem Benutzer verknüpfte Person (Personaldatenbank). Es ist somit auch die Reservierung auf eine andere Person möglich.
- Ein Kommentarfeld kann genutzt werden, um weitere Gründe einzutragen, weshalb eine Ressource eine bestimmte Zeit nicht verfügbar ist, z.B. Raumreservierung für eine bevorstehende Mitarbeiterschulung, oder um im Vorfeld eine längerdauernde Wartung eines Gerätes zu vermerken.
- optionale Referenzen: Es soll nach dem Speichern der bislang aufgelisteten Informationen zur Reservierung die Möglichkeit geboten werden, keine oder mehr der folgenden Datenobjekte des SMS als Referenz zu verknüpfen:
    - Studie: Eine Studie könnte z.B. per DropDown-Feld ausgewählt werden. Dabei wird eine Auswahl aller Studien gelistet, deren eingetragene Zeitspanne (Studienbeginn, -ende) sich mit der angegebenen Reservierungszeitspanne überdeckt. Alternativ könnte man sich auf eine zweistufige Auswahl festlegen (z.B. Kalenderjahr und konkrete Studie aus dem gewählten Kalenderjahr), damit alle aufgezeichneten Studien auswählbar sind (siehe Abbildung 1).

- Proband: Ein Probanden-„Picker“ kann die Auswahl eines Probanden durch unmittelbare Suche und anschliessender Selektion aus der Ergebnisliste ermöglichen. Dadurch kann z.B. ein Proband konkret per Name gesucht und ausgewählt werden. Alternativ wäre auch Auswahl eines Probanden aus der Probandenliste (siehe Abschnitt 7.4) einer zuvor auswählbaren Studie vorstellbar. Eine weitere Variante wäre ein Dropdown-Feld das jene Probanden listet, die im in der Reservierungszeitspanne bereits für eine Visite eingeteilt sind (siehe Abschnitt 7.5).
- Schulung: Räume könnten auch für eine in der Schulungsdatenbank verzeichnete Schulung reserviert werden. Die Auswahl einer zuvor angelegten Schulung erfolgt wiederum per DropDown-Feld mit allen Schulungen des Kalenderjahres des Startdatums der Reservierungszeitspanne oder direkt eines auswählbaren Kalenderjahres.

Für Terminvereinbarungen mit Probanden sollte eine Übersicht wieder in Form einer Kalenderansicht abrufbar sein, um freie Ressourcen schnell zu erkennen (siehe Abbildung 1).

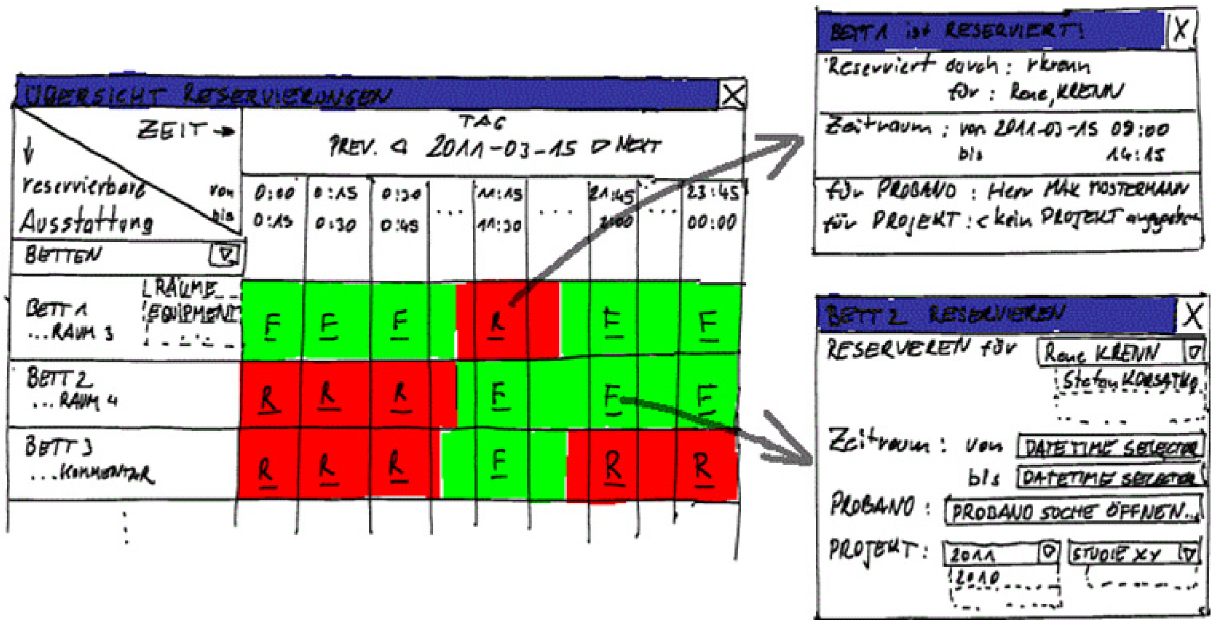


Abbildung 1: Reservierungsansicht

Die Reservierungsfunktion ist als Bestandteil der Ausstattungsdatenbank für sämtliche Ausstattungsobjekte anwendbar. Durch die Verknüpfung einer Reservierung mit einer Studie bietet sich für Auswertungen (Reports) die Möglichkeit, z.B. automatisch eine Auflistung der bei einer Studie eingesetzten Gerätschaften zu erzeugen.

Bemerkenswert für die Reservierung ist, das Ausstattungsobjekte sämtlicher Abteilungen (Abschnitt 4.3) ersichtlich sind. Für Reservierungen ist die Sichbarkeit der Belegungen wichtig, anstatt einer Verschleierung. Optionale Referenzen einer Reservierung (Schulung, Proband, Studie) können jedoch für Benutzer anderer Abteilungen unsichtbar gemacht werden.

3.4 Wissensdatenbank

Um die Funktion eines einfachen Nachschlagewerks einzubringen, können zu den unter dem Abschnitt 3.5 angeführten Attributen für Ausstattungsobjekte weitere vorgesehen werden:

- zusätzliche Liste für PDF-Dokumente, zum Hinterlegen von Bedienungsanleitungen
- Kontakt der Herstellerfirma oder Servicestelle
- SOP Dokumente (Beschreibungen standardisierter Prozesse) als Hyperlink auf Ressourcen eines externen CMS (Content Management System).
- Abbildung des Ausstattungsobjekts als Bilddatei

3.5 Ablage und Verwaltung

Verwaltungsfunktionen umfassen das Anlegen, Einsehen, Bearbeiten und Suchen von Ausstattungsobjekten. Folgende Attribute sind für ein Objekt vorgesehen:

- Kategorie des Ausstattungs-Objekts: Zur Strukturierung in Ansichten der Weboberfläche und für die Suchfunktion werden Kategorien der Objekte mit zumindest einer Hierarchieebene vorgesehen, die in der Datenbank abgebildet sind und somit erweiterbar bleiben (vordefinierte Liste). U.a. gibt es z.B. die Kategorien „Betten“ und „Räume“.
- Wartungs/Eichungsinformation: entspricht der für die beschriebene Wartungserinnerungsfunktion erforderlichen Daten:
  - keine oder mehrere periodische Wartungen/Inspektionen/Eichungen (Label + Startdatum + Interval (u. Laufzeit) + Erinnerungs-Zeitspanne + Verantwortlicher)
  - null oder mehr einmalige Termine (Label + Datum + Erinnerungs-Zeitspanne + Verantwortlicher)
- Verantwortlicher: Verweis auf einen SMS Benutzer, an welche die Wartungsterminerinnerungen gerichtet sind
- Reservierbar: Markierung, ob das Objekt in der Reservierungsübersicht gelistet wird
- Reservierungsinformation: Sollte das Objekt als reservierbar markiert sein, wird der Verlauf der Reservierungen gespeichert. Diese Daten sind über die Reservierungsfunktion darstellbar. Eine Reservierung wird durch folgende Angaben definiert:
  - Beginn-Zeitmarke, Dauer und beantragender Benutzer
  - Verweis auf eine oder mehrere in der Probandendatenbank verzeichnete Personen als Verwendungszweck und/oder
  - Verweis auf eine oder mehrere in der Studiendatenbank vorhandene Studien
- Status: Ein Objekt kann zwischen folgenden Zuständen wechseln:
  - verfügbar
  - verliehen
  - defekt
  - in Reparatur
  - in Wartung/zur Inspektion/zur Eichung
  - end-of-life

Eine Statusinformation enthält zusätzlich eine Beginn-Zeitmarke sowie eine Zeitdauer. Der Status kann jederzeit gesetzt werden wobei dabei die momentane Zeit herangezogen wird sowie eine unbestimmte Zeitdauer. Neben dem Verlauf der Statuswechsel können in weiterer Folge auch zukünftige Zustände im Voraus eingetragen werden (z.B. Gerät ist ab Zeitpunkt x für y Wochen verliehen). Auf diese Information kann dann bei der Reservierungsfunktion Rücksicht genommen werden.

Statusänderungen werden ***immer*** nur manuell durch einen Benutzer durchgeführt, d.h. es sind keine automatischen Statusänderungen vorgesehen (z.B. automatisch in Wartung ab überschrittenem Wartungstermin).

- Historie von (PDF-) Dokumenten: Schriftliche Auftrags- bzw. Durchführungsbestätigungen vom Hersteller oder einer Servicestelle gelten als Beleg einer erfolgten Wartung oder Eichung des Gerätes. Eingescannte Dokumente werden dabei über die SMS Weboberfläche per Datei-Upload eingespielt und in der Datenbank mit Zeitmarke und Vermerk des Benutzers abgelegt.
- Standort: Freitext bzw. vordefinierte Liste oder ein Verweis auf ein vorhandenes Ausstattungsobjekt der Kategorie „Räume“
- Eigentümer: Name einer Person oder Institution als Freitext bzw. vordefinierter Liste. Alternativ evtl. ausgeführt als Personendaten-„Picker“ (vgl. Probanden-„Picker“).
- Abteilung: Referenz auf die Abteilung, der die Ausstattung zuzuordnen ist (Abschnitt 4.3)
- Kommentare: Für allfällige Angaben und Hinweise zum Objekt kann ein Kommentar-Verlauf vorgesehen werden. In Listenansichten der Weboberfläche wäre z.B. immer der aktuellste Kommentar dargestellt.

Das Anlegen bzw. die Bearbeitung dieser Daten von Ausstattungsobjekten sind Benutzern vorbehalten, die über entsprechende Berechtigungen verfügen. Nachdem die Verfügbarkeit bzw. Einteilung von Arbeitskräfte entsprechend der Arbeitslast generell dynamisch ist, könnte für die Zugriffsberechtigungen von Benutzern das Konzept der Zugriffssteuerungslisten angewendet werden (siehe Abschnitt 5.3). Für die Ausstattungsdatenbank werde an dieser Stelle zusammengefasst folgende Berechtigungen vorgesehen, die jeweils pro Benutzer vergeben werden, und daher jederzeit sehr feinkörnig angepasst werden können:

- Neues Austattungsobjekt anlegen sowie statische Daten (Kategorie, Seriennummern, ...) bestehender Einträge bearbeiten erlaubt J/N
- Auswählbarer Verantwortlicher für Gerätewartungen/Eichungen J/N
- Reservierungen eintragen erlaubt J/N
- Reservierungen (anderer Benutzer) löschen erlaubt J/N
- Status ändern erlaubt J/N
- Kommentar hinzufügen erlaubt J/N

Jede dieser Zugriffsberechtigungen kann zur Trennung von Abteilungen hinsichtlich Abschnitt 4.3 weiter unterteilt werden, sofern sich dies dennoch als erforderlich herausstellt:

- Erlaubt nur für Ausstattungsobjekte der Abteilung des Benutzers. Für das Anlegen und Bearbeiten bleibt die Auswahl der Abteilung hierbei auf jene des Benutzers beschränkt.
- Erlaubt für Ausstattungsobjekte sämtlicher Abteilungen

Um den strengen Vorgaben bzgl. Nachvollziehbarkeit und Dokumentation ultimativ Rechnung zu tragen, kann für jedes Ausstattungsobjekt eine Aufzeichnung von bedeutsamen Benutzeraktionen (Change Log) vorgesehen werden. Durch ein Änderungsprotokoll, das auch bei zahlreichen anderen Komponenten des SMS vorgesehen ist, werden Informationen entsprechend folgender Beispiele automatisch aufgezeichnet:

- „Bezeichnung von Ausstattungsobjekt x wurde von ‚Thermometer' auf ‚Brannan Ear Thermometer' geändert. Benutzer rkrenn, Zeitmarke 2011-03-30 13:35:04"
- „Status von Ausstattungsobjekt y wurde von ‚verliehen' auf ‚verfügbar' geändert. Benutzer jgerdova, Zeitmarke 2011-03-31 15:11:37"

3.6 Datenbestand und Ausgangssituation

Bislang wird der wartungsintensive Teil der Ausstattungsgegenstände per Excel-Tabelle organisiert. Es befindet sich ausserdem ein Bestand an eingescannten Bedienungsanleitungen im Aufbau. Es sollte tragbar sein, diese Daten manuell in der Ausstattungsdatenbank des SMS anzulegen.

4 Personaldatenbank

4.1 Übersicht

Das StudienManagementSystem (SMS) soll umfassende Aufzeichnungen über sämtliche, jemals für die Abteilung CRC (Clinical Research Center des ZMF) tätigen Personen erlauben. Das in die Aufgaben des CRCs involvierte Personal fällt unterschieden durch das Anstellungsverhältnis und die jeweilige kostentragende Instituition z.B. in folgende Kategorien:

- Angestellter, geringfügig Angestellter, Freier Mitarbeiter sowie „Externe“ (auf Werksvertragsbasis beauftragte Selbständige bzw. per Personalleasingfirma überlassene Arbeitskräfte)
  - der Medizinischen Universität Graz (MUG)
  - der Steiermärkische Krankenanstalten-gesellschaft m.b.H. (KAGes)
  - der MUG zu x % sowie der KAGes zu y %
- Landesangestellte

Auch darüber hinaus mitwirkendes externes Personal kann verzeichnet sein, jedoch ist ein Eintrag in der SMS Personal Datenbank nicht zwingende Voraussetzung für einen Zugriff auf das System durch Benutzername/Passwort. Dies wird durch die Benutzerdatenbank übernommen, die auch Zugriffsberechtigungen im SMS abbildet.

Ursprünglich war angedacht, Dienstverhältnisse im SMS abzubilden, um die Ressourcenplanung für die durchzuführenden klinischen Studien zu erleichtern.

Neben Mitarbeitern können in der Personaldatenbank auch Firmen oder Institutionen erfasst werden. Diese Daten können dann von anderen SMS Komponenten referenziert werden, z.B. Herstellerfirmen oder Eigentümer bei Ausstattungsobjekten (siehe Abschnitt 3.5). Kontaktdaten sind vereinheitlichter Bestandteil folgender im SMS erfassten Daten:

- Personaleinträge, Firmenkontakte, Institutionen (Personaldatenbank)
- Probandeneinträge (Probandendatenbank)

Die genaue Aufzeichnung der Personalinformation ist in erster Linie für die strengen Dokumentationsvorgaben für klinische Studien erforderlich.

Eine hinzugekommene Anforderung ist die Separation der Daten im SMS für zukünftige Fälle, bei denen das System auch durch andere Abteilungen neben dem CRC genutzt werden können soll. Die Erfassung dieser Abteilungen wird auch als Bestandteil der Personaldatenbank gesehen.

Durch die Personaldatenbank sollen somit zusammenfassend folgende Funktionen abgedeckt werden:

1. einheitlicher Lebenslauf für spezialisiertes medizinisches Personal
2. Abteilungen
3. Ablage und Verwaltung

4.2 Einheitlicher Lebenslauf für spezialisiertes medizinisches Personal

Für das an klinischen Studien beteiligte Personal ist pro Person die Bereitstellung von Daten vorgeschrieben, um deren fachliche Eignung und Kompetenz zu bestätigen. Neben Angaben zur Ausbildung ist auch der Vermerk von zum Teil vorgeschriebenen Schulungen (GCP – Good Clinical Practice Verordnung) notwendig. Beispielsweise sind für eine klinische Studie einmalige oder spezielle Einschulungen vorzunehmen. Details zu besagten Schulungen und Weiterbildungen werden gesondert in der Schulungsdatenbank behandelt. Speziell die Unterstützung für regelmässig zu wiederholende Schulungen ist Zweck der Schulungsdatenbank.

Im SMS wäre es das Ziel, einen einheitlichen Lebenslauf (als PDF Dokument) dynamisch generieren zu können. Abgesehen von den Basis-Personendaten enthält ein Lebenslauf für Personal in Einrichtungen zur Durchführung klinischer Studien bzw. für spezialisiertes medizinisches Personal (Trial Site Staff and Specialised Medical Personnel Curriculum Vitae, kurz CV) folgende Eintragungen:

1. Gegenwärtige Position(en): Bezeichnung der Tätigkeit (Jobbezeichnung) als Freitext sowie Jahr des Eintritts. Vorrangig wird sich hier auf die Positionen innerhalb der Organisation (Abteilung für klinische Studien) bezogen.
2. Bisherige Tätigkeiten: Bezeichnung der Tätigkeit (Jobbezeichnung) als Freitext sowie Jahr des Eintritts. Relevante Tätigkeiten sind hierbei generell Tätigkeiten die fachlich im medizinischen Umfeld liegen.
3. Ausbildungen: Bezeichnung der Ausbildung oder Ausbildungsstätte als Freitext sowie das Abschlussjahr. Relevante Ausbildungen sind wiederum jene aus dem medizinischen Umfeld.
4. Schulungsvermerke: Hierbei gibt es 2 Kategorien:
   (a) Absolvierte Schulungen: Wird die (erfolgreiche) Teilnahme einer Person aus der Personaldatenbank an einer über das SMS verwalteten Schulung als bestätigt freigeben, kann dies automatisch als Schulungsvermerk im CV aufscheinen. Per Schulungsdatenbank würden im wesentliche jene Schulungen und Weiterbildungen abgewickelt werden, die
      - von mehreren Personen zu absolvieren sind, und/oder
      - über eine zeitlich begrenzte Gültigkeitsdauer verfügen bzw. erneuert werden müssen, und/oder
      - zwingend für die Durchführung einer spezifischen klinischen Studie vorgeschrieben sind.

   Die bestätigte Schulung kann dann von der teilnehmenden Person optional mit einer Datei (Bestätigung, Zertifikat als PDF-Dokument) versehen werden. Für eine in der Schulungsdatenbank verwaltete Schulung könnte für diesen Zweck auch eine ausdruckbare Vorlage (PDF-Dokument) generiert werden, die ausgedruckt, unterschrieben, eingescannt und anschliessend wieder hochgeladen wird. Alternativ könnte für diesen Anwendungsfall die besagte Vorlage stattdessen auch digital signiert werden, sofern der Schulungsleiter bzw. die beglaubigende Person selbst über ein Benutzerkonto im SMS verfügt, also Bestandteil der Organisation ist (siehe Benutzerdatenbank). Für Schulungen mit hoher Teilnehmerzahl würde in der Schulungsverwaltung generell die einfachere Möglichkeit bereitstehen, eine eingescannte Unterschriftenliste der jeweiligen Schulung anheften zu können.

Der Vorgang der “Einschreibung” bei einer verwalteten Schulung wird bis auf den Spezialfall von “zur freien Teilnahme” freigegebenen, freiwilligen Schulungen nicht vom Benutzer selbst getätigt, sondern durch den mit der Schulungsverwaltung beauftragten SMS Benutzer.

Eine aus der folgenden, vordefinierten Liste ausgewählte Kategorie entscheidet, unter welchem Abschnitt des erzeugten CV‘s der Person der Vermerk angezeigt werden soll (separat für jede verwaltete Schulung vordefinierbar):

- Relevant Job related Training
- Relevant Clinical Trial and Research Experience including GCP Training
- Certifications and Licensures
- Other Activities Pertinent to Professional Qualifications
- <nicht im CV anzeigen>

(b) Individuelle Schulungen und Weiterbildungen: Falls vorhanden können weitere, für klinische Studien relevante Eintragungen (Bezeichnung, Jahr) gemeinsam mit einem optionalen PDF-Dokument (Bestätigung, Zertifikat) angegeben werden. Auch hier kann die Zuordnung zu einer der Kategorien für die Darstellung im generierten CV erfolgen (s. o.) erfolgen.

CVs können ein erforderlicher Bestandteil der Abgabedokumente einer abgeschlossenen Studie sein. Im SMS soll jederzeit der aktuelle, standardisierte CV zu jeder in der Personaldatenbank verzeichneten Person abrufbar sein. Auch die automatische Erzeugung einer Zusammenfassung über alle an einer Studie aus der Studiendatenbank mitwirkenden Personen wäre in weitere Folge realisierbar.

4.3 Abteilungen

Das StudienManagementSystem ist eine Entwicklung initiiert vom und für das CRC des ZMF. Das Clinical Research Center wird als „Core Facility“ (Abteilung) des ZMF weiterhin die zentrale Institution sein, die mit ihrem Personal und Ressourcen (Ausstattung) klinische Studien durchführt. Darüber hinaus soll durch ein zusätzliches Requirement die Nutzung des SMS auch für andere Core Facilities ermöglicht werden. Dies kann sich grundsätzliche wie folgt äussern:

- Personal und Ausstattung werden vollständig vom CRC gestellt
- Personal z.T. vom CRC, Ausstattung zu 100% vom CRC gestellt
- Personal gänzlich von einer anderen Core Facility, Ausstattung zu 100% vom CRC gestellt

Andere Core Facilitites unterwerfen sich mit der Nutzung der SMS Software dem vorgeschrieben Workflow vollständigt. Eine Voraussetzung dafür ist, dass sämtliche involvierte Daten der fremden Abteilung (Personal und evtl. Benutzer, Ausstattung, Schulungen, Studien und Probanden) im SMS eingepfelgt sind. Nun entsteht eine technische Anforderung in diesem Zusammenhang durch die gegenseitige Abschottung der einsehbaren Daten in den Modulen des SMS, um die Hoheit der Daten zu wahren (vor allem Probandendaten). Realisiert wird dies durch verfeinerte Zugriffsberechtigungen die in den Ausführungen durch folgende allgemeine Abstufungen:

- „Zugriff nur erlaubt für Daten der Abteilung des aktiven Benutzers“
- „Zugriff erlaubt für Daten aller Abteilungen“

In speziellen Fällen reicht dies nicht aus, sodass auf eine ausdrückliche Auswahl befugter Benutzer zurückgegriffen wird (Abschnitt 7.9, Teammitglieder einer Studie).

Als Teil der Personaldatenbank sollen Abteilungen durch Einträge einer Liste zur Referenzierung in den SMS Modulen mit folgenden Attributen hinterlegbar sein:

- Abteilungsname: Name der Institution, Arbeitsgruppe oder Core Facility (Freitext)
- Kürzel
- Leitung: Referenz auf Personaleintrag
- Institution: Referenz auf den zuvor angelegten Personaleintrag der Institution, um Kontaktdaten der juristischen Person zu hinterlegen

Vorerst ist es geplant, die Liste der Abteilungen nur als vordefinierte Liste in der Datenbank vorzubereiten. Dies bedeutet, das das Anlegen neuer Abteilungen nicht durch die SMS Oberfläche vorgesehen ist, sondern nur durch den (minimalen) Eingriff eines Datenbankadministrators.

4.4 Ablage und Verwaltung

Verwaltungsfunktionen umfassen das Anlegen, Einsehen, Bearbeiten und Suchen von Personen in der Personaldatenbank. Diese Funktionen könnten grundsätzlich ähnlich gestaltet werden wie für Probandendaten. Die Suchfunktion stützt sich sinnvoller Weise vorrangig auf Attribute der Kontaktdaten der Person. Diese stellen vollständige Kontaktinformationen dar und werden für das Personal gleich wie für Probanden oder Firmenkontaktinformationen (Gerätehersteller/Servicestellen) eingesetzt und sind daher umfassend ausgestattet:

- Unterscheidungsmarkierung für reale oder juristische Person (bzw. Institution). CVs werden nur für reale Personen unterstützt.
- Name: Felder für Vor- und Nachname bzw. Name des Unternehmens bzw. Organisation bei einer juristischen Person
- Geburtsdatum (reale Person)
- Geschlecht (reale Person)
- Staatsbürgerschaft/Land: Freitextfeld und/oder vordefinierte Liste
- Anrede, Titel: Für einfache Anreden wären z.B. „Herr“, „Frau“, „Firma“ aus einer vordefinierten Liste vorgesehen. Als Titel wird der akademische Grad verstanden, der sich durch Mehrfachauswahl aus einer vordefinierten Liste ergibt. Auf eventuell erzeugten PDF-Dokumenten wäre dann “<Anrede> <Titel1> <Titel2> <Titel3>” zu lesen.

- Adressen: Eine Adresse soll zur Verwendung in einer Anschrift eine vollständige, standardkonforme, postalische Adresse ergeben, zusammengesetzt aus:
  - Land
  - Postleitzahl (ZIP Code)
  - Ortsname (City)
  - Strassenname
  - Hausnummer
  - Stiege (Entrance)
  - Türnummer

  Die Eingabe dieser Felder soll in Form von Freitexten erfolgen, d.h. es ist bis auf weiteres kein vordefinierter Adressdatenstamm vorgesehen, aus dem eine gültige Adresse ausgewählt werden kann. Auf Grund diverser Einschränkungen bei den Nutzungsbedingungen ist auch der Einsatz externer Adressabgleich-Dienste wie z.B. der Google Geocoding API nicht vorgesehen.

  Eine hinzugefügte Adresse kann einer der folgenden, vordefinierten Kategorien zugeordnet sein:

  - Privatadresse
  - geschäftliche Adresse (CV Business Address)

  Für jede Adresse kann zusätzlich eine Bezeichnung inkl. Zustellanweisung als Freitext angegeben werden, z.B.:

  - „Medizinische Universität Graz<Zeilenumbruch>Zentrum für medizinische Forschung“
  - „z. Hd. Herr Max Mustermann“

  Für jede verzeichnete Adresse einer Person ist ausserdem für weiterführende Funktionen des SMS eine Markierung vorgesehen, ob die Adresse aktiv ist bzw. Postsendungen an die Adresse erfolgen sollen. Damit lässt sich die Berücksichtigung der Adresse z.B. bei generierten Serienbrief-PDFs steuern.

- Telefonnummern: Telefonnummer im E.164 Format mit einem auswählbaren Typ aus einer vordefineirten Liste, z.B.:
  - Mobil
  - Home
  - Firma (CV Business Address)
  - Fax
- E-Mail-Adressen: Für E-Mail Adressen wird generell eine Formatprüfung vorgesehen, jedoch kein Bestätigungsmechanismus. Für vom SMS als E-Mail versendete Notifizierungen an SMS Benutzer werden jedoch ausschliesslich die in der Benutzerdatenbank hinterlegten E-Mail Adressen eines Benutzers herangezogen.

Weiters könnte bei diesen allgemeinen Kontaktdaten noch ein Feld für Web URLs und ein Photo-Upload vorgesehn werden. Neben einer Referenz auf diese als Kontaktdaten zusammengefassten Attribute enthält ein Personaleintrag die zuvor beschriebenen Felder für den Lebenslauf.

Im letzten Schritt werden noch Felder festgelegt, um die Ressourcenplanung bei Studien zu unterstützen:

- Kategorie: Eine vordefinierte Liste an Kategorien zur Organisierung der Mitarbeiter ist angelehnt an deren Funktionen bzw. Zuständigkeitsbereichen. Ein Mitarbeiter kann einer oder mehreren Kategorien angehören. Die Zuordnung zu verschiedenen Kategorien wäre z.B. für einen Personaleintrag anzuwenden, wenn mehrere der noch genauer festzulegenden Tätigkeitsfelder durch die betreffende Person bewältigt wird. Die Mitarbeiterkategorien können in anderen Teilsystemen z.B. zur Einteilung zu Schulungen oder der Diensteinteilung zur Auswahl von Mitarbeitergruppen verwendet werden.
- Abteilung: Referenz auf die Abteilung, der der Personaleintrag zuzuordnen ist (Abschnitt 4.3)
- Urlaubstage: Urlaub oder Zeitdauern an denen ein Mitarbeiter absehbar nicht verfügbar ist, werden durch eine Liste von Einträgen abgebildet, jeweils bestehend aus:
  - Zeitintervall durch einschliessende Startzeitmarke und einschliessende Endzeitmarke. Eine unbestimmt lange Zeitdauer ist an sich nicht vorgesehen.
  - Kategorie aus einer vordefinierten Liste:
    - Urlaub
    - Zeitausgleich (ZA), Behördenwege, ...
    - Krankenstand
    - Dienstreise
    - Karenz
  - Optionale Bemerkung als Freitext

  Bei Angaben eine Zeitspanne durch Start- und Stopzeitmarken ergibt sich eine Genauigkeit (Auflösung) von Sekunden.

- Ausgeschiedene Mitarbeiter: Um effektiv abbzubilden, ob ein Mitarbeiter grundsätzlich aktiv ist oder nicht mehr für die Organisation tätig (pensioniert, gekündigt, ausgetreten, ...), gibt es eine übergeordnete Markierung ("retired"). Damit soll die Person von Einteilungen bei Studien und der Schulungsverwaltung gänzlich und dauerhaft ausgenommen werden bzw. in den Ansichten versteckt sein. Für Statusänderungen ist eine Verlaufsaufzeichnung mit Datum der Änderung möglich, um den untypischen Fall abzubilden, bei dem ein ausgeschiedener Mitarbeiter nach einer Zeit wieder angestellt ist.

Das Anlegen und Bearbeiten von Personaldaten ist nur SMS-Benutzern mit entsprechender Berechtigung vorbehalten. Für Zugriffsberechtigungen entsprechend des Konzepts der Zugriffssteuerungslisten gäbe es eine einzelne, abteilungsübergreifende Berechtigung:

- Neuen Personaleintrag anlegen sowie bestehende Einträge in vollem Umfang bearbeiten erlaubt J/N
  - Erlaubt nur für Personaleinträge der Abteilung des Benutzers. Die Auswahl der Abteilung bleibt hierbei auf jene des Benutzers beschränkt.
  - Erlaubt für Personaleinträge sämtlicher Abteilungen

Nach dem Anlegen eines neuen Personaleintrags wäre auch das Anlegen eines SMS Benutzers erforderlich, sofern die Person Zugriff auf das System erhalten soll. Der Benutzer wird anschliessend zu diesem Zweck manuell mit dem zuvor erstellten Personaleintrag verknüpft; es können auch mehrere Personaleinträge an einen (gemeinschaftlichen) Benutzer gebunden sein. Ein SMS Benutzer hat dann die Berechtigung, die mit ihm assoziierten Personaleinträge in vollem Umfang zu bearbeiten.

Um den strengen Vorgaben bzgl. Nachvollziehbarkeit und Dokumentation ultimativ Rechnung zu tragen, kann für jeden Personaleintrag eine Aufzeichnung von bedeutsamen Benutzeraktionen (Change Log) vorgesehen werden. Für Personaleinträge wird dies jedoch eine untergeordnete Rolle spielen.

4.5 Datenbestand und Ausgangssituation

CVs der Mitarbeiter (<20) sind als ausgefüllte Dokumentvorlagen vorhanden, die im wesentlichen von jedem Mitarbeiter selbst verwaltet werden. Nach Aufforderung der Studienleitung werden CVs von den Mitarbeitern aktualisiert und bereitgestellt, um sie bei den Abgabedokumenten abgeschlossener Studien beizufügen.

Personendaten von Mitarbeitern inkl. CV-Informationen sowie Kontaktdaten von relevanten Firmen und Institutionen sind überschaubar und müssen manuell in die Personaldatenbank des SMS eingepflegt werden.

5 Benutzerdatenbank

5.1 Übersicht

Das StudienManagementSystem (SMS) ermöglicht Mitarbeitern der Organisation (Abteilung CRC - Clinical Research Center des ZMF) Zugriff auf sein Webfrontend nach einer Authentifizierung durch Benutzername plus Passwort. Eine solche Benutzeridentität ist mit keinem, einem oderer mehreren Einträgen zu Personen in der Personaldatenbank verknüft. Dieser Ansatz ist der Grund warum die Benutzerverwaltung von der Personaldatenbank ausgegliedert ist, wodurch alle möglichen Situationen abgedeckt werden:

- Benutzer ohne verknüpfte Person: Systembenutzer für Zugriff eines externen Softwaresystems z.B. per Webservices
- Benutzer mit einer verknüften Person: typischer Fall - ein Mitarbeiter kann mit seinem Benutzer einsteigen
- Benutzer mit mehreren verknüften Personen: Mitarbeiter teilen sich einen Benutzernamen (z.B. Nurlese-Benutzer "viewer")
- Personen aus der Personaldatenbank, die mit keinem Benutzer verknüft sind: verzeichnete Mitarbeiter, die z.B. keinen Zugriff (mehr) auf das System haben sollen, oder nur zu Dokumentationszwecken eingetragen wurden.

Die Benutzerdatenbank deckt folgende Funktionen ab:

1. Authentifizierung
2. Autorisierung
3. Digitale Signatur für Datensätze
4. Ablage und Verwaltung

5.2 Authentifizierung

Wie bei vielen Multiuser-System üblich erfolgt die Authentifizierung eines Anwenders durch die Eingabe eines Berechtigungsnachweises bestehend aus:

Benutzername: Ein dem Anwender zugewiesener, i. A. nicht geheimer Benutzername.

Passwort: Ein vom Anwender definierbares, und nur ihm bekanntes Geheimnis in Form einer Zeichenfolge.

Nach erfolgter Anmeldung am System ist eine Sitzung eröffnet. Jede einzelne Interaktion mit der Weboberfläche erfordert grundsätzlich eine Authentifizierung. Sitzungen weisen deshalb eine vordefinierbare Lebensdauer auf und unterstützen damit einen Mechanismus (Session Cookies), der die ständige wiederholte Eingabe Benutzername und Passwort vermeidet. Sofern der Anwender seine Zugangsdaten nicht an andere Personen weitergibt, entspricht eine erfolgreiche Anmeldung am System mit Benutzername und zugehörigem Passwort der Echtheitsbestätigung der Identität des Anwenders.

Das ungewollte Bekanntwerden von Passwörtern ist deshalb durch eine Reihe von Massnahmen zu schützen:

- Schutz gegen Erraten des Passworts (Password Policy):
  - Beim Festelgen von Passwörtern werden Kriterien für die Zeichenfolge vorgegeben, z.B.:
    - minimale Länge
    - minimale Anzahl an Ziffern, Kleinbuchstaben, Großbuchstaben und Symbolen um einen großen Symbolraum zu erzwingen
    - Verbot von offensichtlichen Ähnlichkeiten mit Benutzernamen, Personen- bzw. Kontaktdaten

    Dadurch wird das automatisierte Erraten von Passwörtern mit Hilfe von Wortlisten oder Durchprobieren sämtlicher Variationen der Zeichen (Brute-Force) in die Länge gezogen (Password Strength).

  - Ändern von Passwörtern (Password Change Policy):
    - Ein Anwender soll im Verdachtsfall jederzeit die Möglichkeit haben, sein Passwort zu ändern. Um eine Änderung zu vollziehen, ist immer die Eingabe des aktuellen Passworts erforderlich.
    - Passwörter erhalten eine begrenzte Gültigkeitsdauer. Der Benutzer muss vor Ablauf der Gültigkeitsdauer eine Passwortänderung durchführen. Wird dies verabsäumt, erfolgt die Sperrung des Benutzers. Die Sperrung kann nur von einem entsprechend privilegierten Benutzer aufgehoben werden und zusätzlich muss dabei ein neues Passwort für den betroffenen Benutzer gesetzt werden. Vertrauenswürdige Benutzer mit dieser einflussreichen Berechtigung zum Bearbeiten der Benutzerdatenbank (Administratoren) sind auch zuständig für das Anlegen neuer Benutzer. Bei diesem Vorgang muss gleichermassen ein initales Passwort für einen fremden Benutzer vergeben werden. Passwörter werden in diesem Fall verbal kommuniziert. Diese Passwörter können als Einmalpasswörter ausgeführt werden, die nur für einen einzigen Anmeldevorgang gültig sind.
    - Bei wiederholter Eingabe eines falschen Passworts erfolgt die Sperrung des Benutzers. Die Sperrung kann wiederum nur von einem Adminstrator wieder aufgehoben werden.
- Schutz gegen unbefugtes Auslesen aus der Datenbank: Es könnte den Fall geben, dass der SMS Server in irgend einer Weise kompromittiert wurde, sodass der gesamte Inhalt der Datenbank in Folge für Unbefugte zumindest zeitweise einsehbar war. Auch wenn der Angreifer damit relevante Informationen bereits erhalten hat und Passwörter vielleicht gar nicht mehr benötigt, gilt es, jegliche mögliche Auswirkungen durch technische Massnahmen einzuschränken.

Gleichermassen stellen diese gebräuchlichen Vorkehrungen sicher, das Passwörter auch vor Mitarbeitern geheim bleiben, die mit der technische Betreung des SMS Systems bzw. des Servers oder der Datenbank zu tun haben und ebenfalls den Inhalt der Datenbank einsehen können.

– Anwender neigen oft dazu, Passwörter bei verschiedenen Softwaresystemen der Organisation und darüber hinaus auch bei privat genutzten Diensten (z.B. E-Mail Service) wiederzuverwenden. Das ist und bleibt unkontrollierbar. Ein Angreifer, der die SMS Datenbank und damit auch die Tabellen der Benutzerdatenbank einsehen konnte, darf dabei keinesfalls Passwörter in ihrer unveränderten, blanken Form (Plaintext) vorfinden, die er dann eventuell mit Erfolg bei anderen Systemen bzw. Diensten anwenden könnte. Passwörter müssen jedoch in irgendeiner Form in der Datenbank vorliegen, um sie mit den Eingaben bei einer Anmeldung vergleichen zu können.

  Die Lösung liegt in der Ablage eines berechneten Abbildes (Hash) des Passworts anstatt des blanken Passworts selbst. Von diesem Hash-Wert ist das eigentliche Passwort nicht (in praktikabler Zeit) rekonstruierbar. Der Vergleich des bei der Anmeldung eingegeben Passworts mit dem in der Datenbank hinterlegten Abbild läuft dann auf den Vergleich von Hash-Werten hinaus. Die Berechnung des Abbildes wird durch standardisierte, kryptographische Hashfunktionen ermöglicht. Die Hashfunktion SHA-1 (Secure Hash Algorithm 1) gilt nach wie vor als "sicher", d.h. es gibt bislang keine bekannte, effiziente Methode um das Urbild (das blanke Passwort) aus dem vorliegenden Hash-Wert zu bestimmen. Selbiges gilt bei sicheren Hash-Algortithmen auch für das Auffinden eines alternativen Passworts, das genau denselben Hash-Wert ergeben würde (Kollision).

$$Hash - Wert = SHA1(Passwort)$$

  Für SHA-1 und die ältere Hashfunktion MD5 (Message Digest 5) gibt es inzwischen z.B. Methoden um zumindest Kollisionen für willkürliche Hashwerte zu finden. Für MD5 ist dies sogar mit minimalen Rechenaufwand möglich, sie gilt deshalb als überholt. Die weitaus wichtigere Eigenschaft einer Hashfunktion ist nach wie vor die Sicherstellung der Unmöglichkeit, das Urbild zu berechnen. Dies ist sowohl für SHA-1 und MD5 noch gewahrt, es bleibt also nur die Möglichkeit des Durchprobierens. Um den Rechenaufwand für den Angreifer beim Durchprobieren zu vervielfachen, kann die Hashfunktion wiederholt angewendet werden (z.B. mehrere 1000 mal):

$$Hash - Wert = SHA1(SHA1(\dots(Passwort)\dots))$$

  Im SMS werden systemweit für sicherheitsrelevante Funktionen die SHA-1 oder SHA-2 (SHA-256) Hash-Algorithmen eingesetzt, ansonsten der performantere MD5 Hash-Algorithmus, z.B. für Prüfsummen zur Feststellung der Integrität einer Datei.

Auf Grund der Ablage von Passwörtern in From von Hash-Werten ist es ***nicht*** mehr möglich, bei Passwortänderungen Ähnlichkeiten mit bisherigen Passwörtern des Benutzers durch Vergleich (z.B. Levenshtein Abstand) zu erkennen und zu verbieten (z.B. „Neues Passwort ist einem der alten Passwörter zu ähnlich, weil es sich nur in 2 Zeichen unterscheidet"). Nachdem bei Passwortänderungen sowohl das bisherige als auch das neue Passwort einzugeben ist, könnte man aber zumindest diese beiden Eingaben einem Vergleich unterziehen.

– Eine allgemein mögliche Methode, um von Hash-Werten dennoch zu den blanken Passwörtern zu gelangen, wäre der Einsatz einer vorberechneten Tabelle, die alle möglichen Hash Werte und deren Urbilder enthält um dann darin die gesuchten Einträge zu suchen. Hash-Werte bei SHA-1 haben bei der Darstellung im dualen Zahlensystem 160 Stellen bzw. im dezimalen Zahlensystem 49 Stellen, d.h. die erforderliche Tabelle wäre gigantisch. Abgesehen von der Speicherung dieser Tabelle ist die Berechnung zeitlich nicht tragbar, es müssten mehr als 2^{160} Urbilder durchlaufen werden, um die Tabelle lückenlos zu befüllen. Eine praktikable Form dieses Konzepts der Tabelle sind allerdings Regenbogentabellen, die nur absehbare Urbilder berücksichtigen (z.B. Passwörter mit bis zu 8 alphanumerischen Zeichen). Mithilfe des Verfahrens von Regenbogentabellen wird der immer noch immense Speicherplatzbedarf auf einen bewältigbares Volumen reduziert, wobei dann allerdings auch das Auffinden mit Rechenaufwand verbunden ist, der aber durchaus tragbar ist. Der prinzipielle Vorteil dieser Tabellenverfahren ist erst in ihrem wiederholten Einsatz zu sehen: hat man die Tabelle einmal erzeugt, kann man damit schnell Urbilder weiterer Hash-Werte aufsuchen.

Ein Angreifer, der in Besitz einer Kopie der SMS Datenbank gelangt, hätte damit auch mit einem Schlag die Hash-Werte aller Benutzer, und könnte Regenbogentabellen einsetzen, um damit schneller die zugehörigen Passwörter herauszufinden. Um genau dies zu unterbinden, wird der Hash-Wert aus dem Passwort erweitert mit einer Zufallszeichenkette (Salt) pro Benutzer berechnet.

$$Hash - Wert = SHA1(Salt \circ Passwort)$$

Das Salt wird blank in der Datenbank abgelegt, um es beim Vergleich bei der Anmeldung wieder miteinbeziehen zu können. Eine vorbereitete Regenbogentabelle muss das Salt berücksichtigen. Nachdem jeder Hash-Wert einen unterschiedlichen Salt-Wert miteinbezieht, müsste auch die Regenbogentabelle jeweils neu erzeugt werden, was deren Sinn zunichte macht.

– Spezielle Datensätze oder Felder in der SMS Datenbank sollen ggf. ausschliesslich für den Benutzer abrufbar sein, der diese Daten auch erzeugt hat. Für diese Fälle wird eine symmetrische Verschlüssellung mittels einer Blockchiffre (z.B. AES-128) und einem vom Benutzerpasswort abgeleiteten Schlüssel eingesetzt. Jede Ansicht oder Aktion im SMS Webfrontend, die unverschlüsselte Daten miteinbezieht, wird die Eingabe des Passworts erfordern. Die Verschlüselung ist jedoch nur für sehr spezielle Daten vorgesehen, wie der Ablage des privaten Schlüssels zum digitalen Signieren von Datensätzen (siehe Abschnitt 5.4).

Die Beschriebenen Massnahmen sind Best Practices und oreintieren sich an aktuellen Industriestandards. Sie sollen das Bestmögliche an Datensicherheit bieten, und auch Vorgaben für eine Zertifizierung der SMS Software erfüllen.

5.3 Autorisierung

Gewisse, durch Benutzer durchgeführte Aktionen im SMS sind invasiv, d.h. haben die Änderung der Inhalte der Datenbank zur Folge. Dies muss reglementiert sein, um Probleme in folgenden Bereichen zu vermeiden:

1. Kritische Daten: Diese dürfen ausschliesslich von entsprechenden Verantwortlichen manipuliert werden.
2. Zuständigkeitsbereiche: Benutzer nehmen anderen Änderungen vorweg, wodurch Verwirrung entsteht.
3. Wettlaufbedingungen: Bei Konzepten wie Reservierungssystemen oder der Diensteinteilung ergibt sich prinzipiell ein Wettlaufszenario (der erste beantragende Benutzer gewinnt). Benutzer, die das Reservierungssystem eigentlich gar nicht benötigen könnten Reservierungen dennoch sabotieren oder (unabsichtlich) stören.

Auch nicht invasive Aktionen könnten auf gewisse Benutzer einzuschränken sein, z.B.

- Abrufen bzw. Sichtbarkeit von Daten: Vertraulichkeit
- Auswahlmöglichkeit: Bei Aktionen, die eine Zuweisungen von Benutzern entahlten sollen nur gewisse in der Auswahlliste aufscheinen. Das Kriterium, ob ein Benutzer in der Liste erscheint, kann auch als Berechtigung aufgefasst werden.

Eine Berechtigung bzw. Autorisation ist eine Freigabe der Durchführungen einer bestimmten Aktion im SMS Webfrontend für einen bestimmten Benutzer. Dies wird durch eine Zugriffssteuerungsliste (ACL - Access Control List) in Matrixform abgebildet:

Berechtigung / Benutzer	“Benutzer anlegen, ACL bearbeiten“	“Personendatenbank bearbeiten”	“Studie sperren und signieren”	...
rkrenn	Erlaubt	erlaubt	nicht erlaubt	...
skorsatko	Erlaubt	...	erlaubt	...
...	...	...	...	...

Tabelle 3: Zugriffssteuerungsliste (ACL)

Diese Matrix bleibt für diverse neue Berechtigungen zukünftig implementierter Aktionen erweiterbar. ACLs erlauben die direkte und selektive Einstellung der verfügbaren Berechtigungen. Ein potentieller Nachteil kann die unübersichtliche Verwaltung bei einer sehr großen Anzahl an Berechtigungen sein. Hierfür können im SMS Berechtigungs-Voreinstellungen für eine Reihe von auswählbaren Profilen vorgesehen werden. Das Identifizieren und Festlegen von Benutzerrollen und daraus abgeleitete Profile kann zu einem späteren Zeitpunkt erfolgen. Profile wären bleiben auch jedezeit anpassbar.

Ein Änderungsprotokoll (Change Log) für die ACL möglicht die Aufzeichnung, welcher Benutzer zu einem Zeitpunkt eine Berechtigung hatte.

5.4 Digitale Signatur für Datensätze

Um Daten technisch gegen nicht authorisierte Manipulation zu schützen, kommen digitale Signaturen zum Einsatz. Ein möglicher Anwendungsfall wäre die Signierung von Studiendaten nach deren Fertigstellung. Die Studiendaten mit relevanten Aufzeichnungen (beteiligte Mitarbeiter, abgelegte Case Report Form Dokumente mit Aufzeichnungen zu Visiten usw.) und dem finalen Abschlussdokumenten sind nach Abschluss der Studie vom Verantwortlichen für nachfolgende Veränderungen zu Sperren (Lockdown). Bei diesem Schritt könnten die besagten Daten der Studie inkl. Lockdown-Datum digital signiert werden. Die Signatur ist der unwiederlegbare Nachweis, dass es keine nachfolgenden Veränderungen mehr gegeben hat.

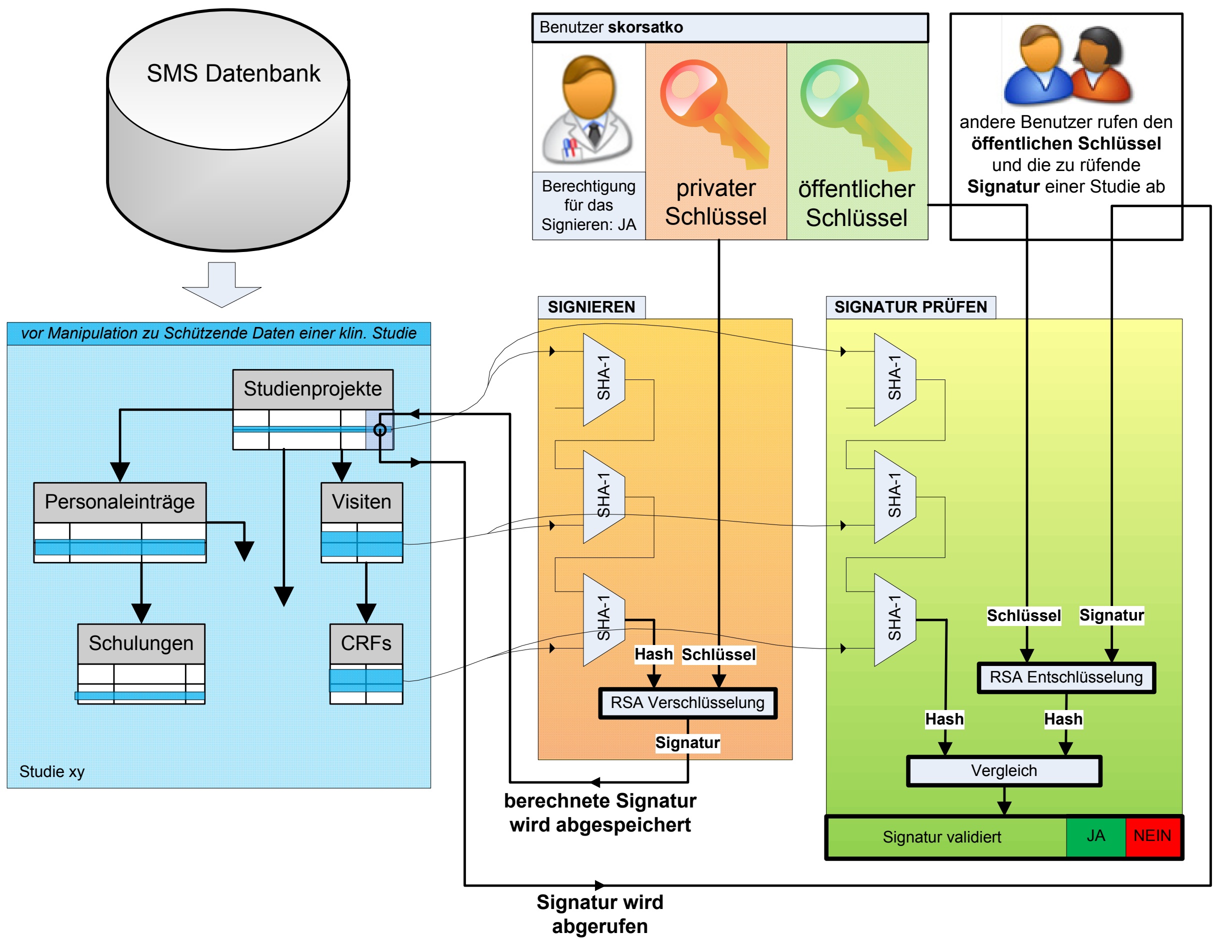


Abbildung 2: Digitale Signatur für Datensätze

Signaturen werden durch asymmetrische Verschlüsselungsverfahren (z.B. RSA - Rivest-Shamir-Adleman) umgesetzt. Diese Verfahren nutzen gegenüber der symmetrischen Verschlüsselung jeweils einen unterschiedlichen Schlüssel für das Verschlüsseln und das Entschlüsseln. Das Schlüsselpaar besteht aus dem geheim gehaltenen “privaten” Schlüssel zum Verschlüsseln und dem “öffentlichen” Schlüssel zum Entschlüsseln. Die Schlüssel stehen zueinander in einer mathematischen Beziehung, können jedoch voneinander nicht in praktikabler Zeit abgeleitet werden. Beim Anlegen des Benutzers wird das RSA Schlüsselpaar einmalig generiert, der private Schlüssel des Benutzers wird durch Verschlüsselung mit seinem Passwort für Andere unlesbar abgespeichert.

Signieren: Beim Erstellen einer digitalen Signatur wird der Hash-Wert über alle zu schützenden Datenfelder berechnet. Dieser Hash-Wert wird mit dem privaten Schlüssel des Benutzers verschlüsselt, der mit seiner Identität somit den Inhalt der Daten bestätigt. Der verschlüsselte Hash-Wert entspricht der Signatur, die dann in Datenbank gespeichert wird (Abbildung 2).

Signaturüberprüfung: Eine zu überprüfende Signatur wird mit dem öffentlichen Schlüssel des Benutzers entschlüsselt, der die Signatur erzeugt hat. Nach der Entschlüssellung wird der ursprüngliche Hash-Wert der involvierten Datenfelder erhalten. Dieser Hash-Wert wird nun zum Vergleich neu berechnet. Sind die Hash-Werte identisch, ist die Validierung positiv (Abbildung 2). In diesem Fall ist bewiesen, dass die Daten nicht manipuliert worden sind. Eine marginale Änderung eines einzelnen Datenfeldes führt zu einem abweichenden Hash-Wert und damit zum Fehlschlagen des Vergleichs. Um Änderungen zu Verschleiern, müsste eine arglistige Partei nach dem Manipulieren der Daten auch die Signatur neu berechnen. Dies ist ohne den privaten Schlüssel des für die Daten verantwortlichen SMS Benutzers und damit ohne dessen Passwort nicht möglich.

Festzuhalten ist, das für die hier präsentierte Signaturlösung ausdrücklich keine Public Key Infrastructure (PKI) miteinbezieht. Dabei verknüpfen kostenpflichtige, extern verwaltete Zertifikate jeweils eine Person mit ihrem öffentlichen Schlüssel. Die Identitätsprüfung entspricht der Überprüfung des Zertifikats und erfolgt durch Abfrage von externen Certificate Authorities (CAs) bei jeder Signaturprüfung.

Sofern diese Funktion in das SMS übernommen wird, wird für das Signieren von Daten eine eigene Berechtigungsstufe vorgesehen (vgl. „Zeichnungsberechtigung“).

5.5 Ablage und Verwaltung

Für die Verwaltung von Benutzerdaten ist das Anlegen, Bearbeiten (z.B. Verknüpfen mit Personaleintrag) sowie eine Auflistung vorgesehen. Für Benutzer werden folgende Felder veranschlagt:

- Benutzername: eine alphanumerischere Zeichenfolge, oft für eine Rolle bezeichnend (“viewer”,”guest”,”dispatcher”,”admin”,”external_system_1”) oder auf die einzelne Person bezogen, für die der User angelegt wird (z.B. erster Buchstabe des Vornamens plus Nachname - “rkrenn”). Nachdem der Benutzername auch der Identifikator eines Benutzers ist (es können nicht zwei gleichlautende Benutzernamen vorkommen), ist das nachträgliche Ändern eines Benutzernamens ausgeschlossen. Möchte man einen personenbezogenen Benutzenamen im Sinne der Konsistenz bzgl. der verknüften Person aus der Personaldatenbank dennoch ändern (z.B. aufgrund einer Namensänderung in Folge einer Eheschliessung), würde dies über den Umweg gehen, einen neuen Benutzer anzulegen und den alten zu deaktivieren.
- E-Mail Adressen: eine Liste von Emailaddressen, an welche automatische Benachrichtigungen für den Benutzer per Email gesendet werden. Für diese Verwendung wäre ein Bestätigungsmechanismus für E-Mail Adressen durchaus sinnvoll. Nachdem aber das Verarbeiten von eingehenden E-Mails einer für das SMS eingerichteten Mailbox nicht weiter vorgesehen ist, ist soweit vorerst nur eine Formatprüfung eingegebener E-Mail Adressen angedacht.
- Telefonnummern: optional können Rufnummern (E.164) angegeben werden, die dann bei Bedarf für Notifizierungen per Kurzmitteilung über ein Kurzmitteilungsgateway herangezogen werden können.

- Kategorie: Eine Kategorie aus einer vordefinierten Liste, um die Ansicht von Benutzern z.B. in einer Baumansicht zu organisieren. Diese Kategorie entspricht auch den in Abschnitt 5.3 beschriebenen Profilen, welche jeweils Voreinstellungen für die verschiedenen Berechtigungen aufweisen. Beim Setzen der Kategorie werden die Berechtigungen auf die Voreinstellungswerte des Profils gesetzt. Beispiele für Kategorien wären:
  - Inaktive Benutzer
  - Readonly Benutzer
  - System - Benutzer für automatisierte Vorgänge und externe Systeme
  - Administratoren mit allumfassenden Berechtigungen - Bearbeiten der Benutzerdatenkank
  - Administratoren für Ausstattungsdatenbank/Personaldatenbank/ Probandendatenbank/Studiendatenbank/...

  Nicht zu vergessen ist, das unabhängig von der Kategorie des Benutzers dessen Berechtigungen durch die Möglichkeit, einzelne Berechtigungen diskret setzen zu können, sehr feinkörnig angepasst werden können.

  Der Verlauf der Kategorie wird nicht aufgezeichnet, weil die detailreichere Aufzeichnung der Verläufe der Berechtigungen vorgesehen ist.

- Referenz auf Einträge aus der Personaldatenbank: Die mit dem Benutzer verknüpften Personen sollten als Verlauf gespeichert werden:
  - Benutzername
  - Änderungszähler: Wird bei einem Bearbeitungsvorgang eines Benutzereintrags die Personenverknüpfung geändert, wird der Änderungszähler inkrementiert. Alle neu verknüpften Personen erhalten dann diesen Zählerstand.
  - Zeitmarke
  - Identifikator für Eintrag aus der Personaldatenbank
- Abteilung: Referenz auf die Abteilung, der der Personaleintrag zuzuordnen ist (Abschnitt 4.3) für den Fall das der Benutzer mit keinem oder mehreren Personaleinträgen verknüpft ist.
- öffentlicher Schlüssel des Benutzers: Um das digitale Signieren von Datensätzen zu unterstützen, wird beim Anlegen des Benutzers einmalig ein unveränderliches Schlüsselpaar (privater und öffentlicher Schlüssel) erzeugt. Der öffentliche Schlüssel wird zum Überprüfen einer Signatur benötigt, eventuell häufig abgerufen und deshalb direkt als Attribut der Benutzerentität gespeichert.
- Bild oder Vektorgraphik der digitalisierten, biometrischen Unterschrift - für den Fall dass der Benutzer eindeutig einer Person zugeordnet ist. Diese Unterschrift kann in generierten PDF-Dokumenten eingefügt werden, die bei der Berechnung einer digitalen Signatur berücksichtigt werden und als druckbare Bestätigung gelten sollen.

Die Zugriffskontrolliste wird als Verlauf der Berechtigungen separat abgebildet:

- Benutzername
- Identifikator der Berechtigung: Die implementierten Berechtigungen sind in einer vordefinierten, erweiterbaren Liste gemeinsam mit benutzerlesbaren Bezeichnungen und Beschreibungen eingetragen. Neben den diversen Zugriffsberechtigungen für spezifische Funktionen werden auch weitere Benutzereigenschaften in die ACL miteinbezogen, z.B.:
    - Benutzer gesperrt: dem Benutzer wird der Zugriff absolut verwehrt; die Anmeldung ist nicht möglich. Dies kann genutzt werden, um Benutzer zu deaktivieren. Diese Markierung wird automatisch gesetzt, wenn ein Benutzer mehrmals ein falsches Passwort eingibt, oder ein Anmeldeversuch nach Ablauf der Passwortgültigkeit versucht wird
    - Zugehörigkeit des Benutzers zu diversen, im System vorkommenden Benutzerauswahllisten (z.B. für Einteilungen, Reservierungen usw.)
- Zeitmarke
- Zähler: sekundengenaue Zeitmarken (z.B. ISO 8601) sind für eine chronologische Sortierung generell nicht ausreichend; es wird die Reihenfolge des Einfügens in die Datenbank durch einen Zähler abgebildet.
- Wert: Ja (erlaubt) oder Nein (nicht erlaubt)

Neben der Zugriffskontrolliste wird nach gleichem Aufbau eine Liste für den Verlauf weiterer Benutzereigenschaften vorgesehen:

- Benutzername, Zeitmarke, Zähler (vgl. Zugriffskontrolliste)
- Identifikator der Eigenschaft: Die Eigenschaften sind in einer vordefinierten, erweiterbaren Liste gemeinsam mit benutzerlesbaren Bezeichnungen und Beschreibungen eingetragen, z.B.:
    - Benutzer gesperrt: dem Benutzer wird der Zugriff absolut verwehrt; die Anmeldung ist nicht möglich. Dies kann genutzt werden, um Benutzer zu deaktivieren. Diese Markierung wird automatisch gesetzt, wenn ein Benutzer mehrmals ein falsches Passwort eingibt, oder ein Anmeldeversuch nach Ablauf der Passwortgültigkeit versucht wird
    - Zugehörigkeit des Benutzers zu diversen, im System vorkommenden Benutzerauswahllisten (z.B. für Einteilungen, Reservierungen usw.)
    - Passwortgültigkeitsdauer in Sekunden
- Wert: generisches Feld (Text), das eine Zahlenwert, eine Markierung (Ja/Nein), einen Identifikator etc. enthalten kann

Im Laufe der Implementierung wird sich neben der Identifizierung aller notwendigen Berechtigungen zeigen, ob ACL und Benutzereigenschaftenverlauf gleich zusammengefasst werden können.

Der letzte, separate Teil der Speicherung der Benutzerdaten ist der Verlauf der Passwörter und zugehöriger Werte zur Unterstützung der diversen kryptographischen Methoden:

- Benutzername
- Zeitmarke
- Zähler
- Gültigkeitsdauer des Passworts: Wird immer ab der Zeitmarke des Passworts gerechnet. Bei Neusetzen des Passwords wird die für den Benutzer hinterlegte Passwortgültigkeitsdauer eingefügt.
- Zähler für erfolgreiche Anmeldungen mit diesem Passwort
- Erlaubte Anzahl an Anmeldungen mit diesem Passwort: Für Einmalpasswörter wäre dieser Wert auf 1 gesetzt. Uneingeschränkte Anzahl an Anmeldungen könnten durch den Wert -1 dargestellt werden. Bei Anmeldevorgängen werden sowohl Gültigkeitsdauer als auch erlaubte Anzahl an Anmeldungen ausgewertet.
- Salt für Passwort
- Hash-Wert von Passwort inkl. Salt
- Zufallswert für Verschlüsselung des privaten Schlüssels
- Verschlüsselter privater Schlüssel: Der private Schlüssel des beim Anlegen des Benutzers erzeugten Schlüsselpaares wird verschlüsselt hinterlegt. Für die hierbei angewendete symmetrische Verschlüsselung wird nun der Hash-Wert des Passworts inkl. obigen Zufallswert verwendet. Diese Massnahme soll Rückschlüsse jeglicher Art auf eventuell gleiche Passwörter des Benutzers verhindern. Jeder Abruf des privaten Schlüssel im Zuge einer Datensatzsignierung wird zu dessen Entschlüsselung somit die Eingabe des aktuellen Passworts erfordern.

Bei einem Passwortwechsel wird neben dem neuen Passwort auch das alte angegeben. Dabei wird wie bei der Anmeldung überprüft, ob das alte Passwort korrekt ist, und gleichzeitigt der private Schlüssel entschlüsselt, um sofort mit dem neuen Passwort verschlüsselt und abgelegt zu werden.

Das Anlegen und Bearbeiten der Benutzerdaten inkl. der Berechtigungen wird selbst als Berechtigung ausgewiesen:

- Neue Benutzer Anlegen, Benutzer bearbeiten (Personalverknüpfungen, Berechtigungen, Passwörter setzen, Sperrungen aufheben...) erlaubt J/N

Benutzer ohne diese Berechtigung sollen eventuell nur ihre E-Mail Adressen und Telefonnummern bearbeiten können. Diese höchste aller Berechtigungen schliesst im Endeffekt alle übrigen ein (der Benutzer kann sich selbst jede weitere Berechtigung erteilen). Ausgehend von einem “root” Benutzer beim Auslieferungszustand der Datenbank kann das sämtliche Benutzer-Setup in weiterer Folge über das SMS Webfrontend durchgeführt werden.

5.6 Datenbestand und Ausgangssituation

Bislang sind keine externen Softwaresysteme bestimmt, die Zugriff auf die SMS Datenbank erhalten sollen. Das Einrichten von Benutzern für CRC Mitarbeiter würde mit dem Anlegen der Personaldaten (Abschnitt 4.5) einhergehen.

6 Schulungsdatenbank

6.1 Übersicht

Mit der einfach gehaltenen Schulungsdatenbank soll das StudienManagementSystem (SMS) die zeitliche Planung sowie die Teilnahmedokumentation von Schulungen und Weiterbildungen für die Mitarbeiter der Organisation (Abteilung CRC - Clinical Research Center des ZMF) ermöglichen. Diese Schulungen fallen unter eine oder mehrere der folgenden Kategorien:

- angebotene, freiwillige Schulungen
- generell vorgeschriebene Schulungen
- Schulungen, die einheitlich im Trial Site Staff and Specialised Medical Personnel Curriculum Vitae (CV) der Mitarbeiter aufscheinen sollen
- Schulungen zu internen Abläufen der Organisation
- Schulungen, die nach einer bestimmten Zeitdauer erneuert werden müssen
- Schulungen, die für Mitarbeiter vorgeschrieben sind, die an einer Studie mitwirken

Die Schulungsdatenbank deckt folgende Funktionen ab:

1. Abwicklung von Schulungen
2. Erinnerungsfunktion für aufzufrischende Schulungen
3. Ablage und Verwaltung

6.2 Abwicklung von Schulungen

Schulungen werden von einem dafür zuständigen (berechtigten) SMS Benutzer, dem Schulungsdatenbank-Adminstrator, verwaltet. Der Vorgang läuft dabei wie folgt ab:

1. Jede einzelne, abzuhaltende Schulung wird als Eintrag in der Schulungsdatenbank angelegt. Für Schulungen mit gleichem Inhalt oder auch jährlich wiederholten Schulungen ist ein Zusatz im Titel empfehlenswert, z.B.
   - “GCP Training 2008”
   - “Ersthelfer, Gruppe 1”
2. Die Teilnehmer werden vom Schulungs-Administrator durch Auswahl der Mitarbeiterkategorien (Personaldatenbank) und/oder einzelne Selektion der Personen festgelegt. Die Einladung (freiwillige Schulungen) bzw. Anmeldung (verpflichtende Schulungen) wird den ausgewählten Benutzern in der SMS Weboberfläche angezeigt sowie durch eine E-Mail Benachrichtigung mitgeteilt. Ist die Schulung nicht verpflichtend, sehen alle Mitarbeiter die Schulung im SMS angekündigt und können sich selbst eintragen, solange eine eventuell angegebene Teilnehmerzahl nicht überschritten wurde.
3. Die Teilnahme eines einzelnen Mitarbeiters durchläuft dann verschiedene Zustände, um einen Bestätigungsvorgang abzubilden.

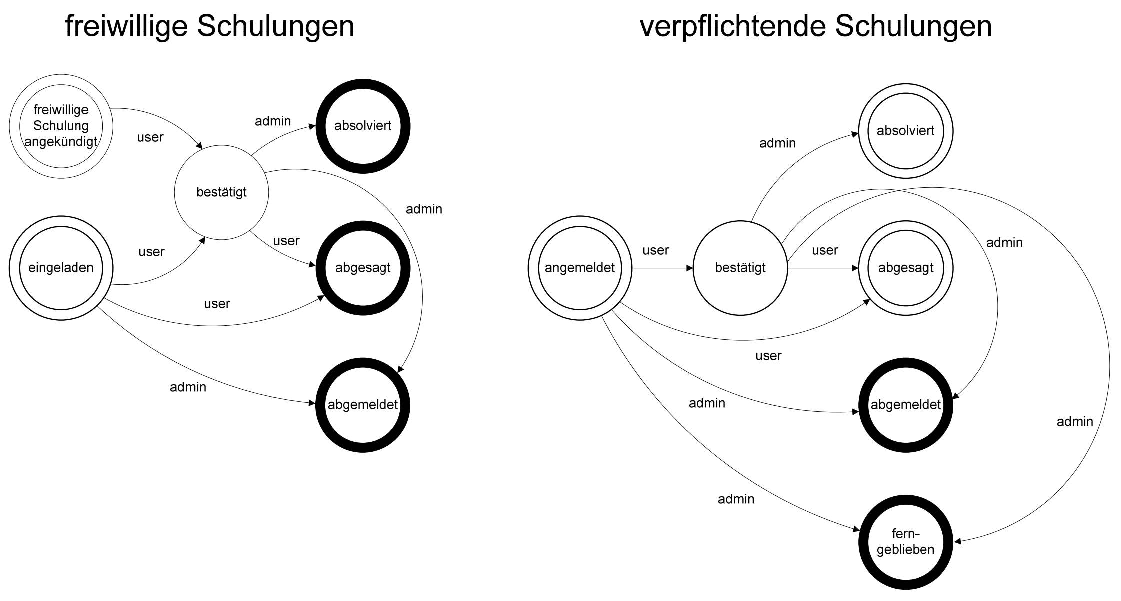


Abbildung 3: Zustände eines Benutzers bei der Teilnahme an Schulungen

Wie in Abbildung 3 dargestellt können auch verpflichtende Schulungen vom angemeldeten Mitarbeiter abgesagt werden (wenn dieser z.B. zum Schulungstermin unter keinen Umständen verfügbar ist). Dem Schulungsadministrator wird dies per SMS Webfrontend in einer Überblicksansicht der Schulungen dementsprechend auffällig, z.B. durch farbliche Hinterlegung dargestellt:

gelb: Die Schulung wurde erst angelegt, es hat noch kein Mitarbeiter die Teilnahme an der verpflichtenden Schulung bestätigt.

orange: Einige der Teilnehmer haben die Teilnahme an der verpflichtenden Schulung bereits bestätigt.

rot: Mindestens einer der Teilnehmer der verpflichtenden Schulung hat abgesagt.

grün: Alle Teilnehmer der verpflichtenden Schulung haben die Teilnahme (respektive den Termin) bestätigt.

Ab dem Beginn des ersten Abhaltungstermins einer Schulung können Teilnehmer von verpflichtenden Schulungen nicht mehr absagen. Der Schulungsadminstrator kann ab diesen Zeitpunkt stattdessen den Status der Teilnehmer auf “absolviert” setzen, wenn er zur Bestätigung sämtliche Unterschriften per Unterschriftenliste eingesammelt hat. Die eingescannte Unterschriftenliste wäre abschliessend der Schulung anzuheften. Im Sinne der genauen Dokumentation der Personaldaten könnte für Teilnehmer einer Schulung eine ausdruckbare Vorlage (PDF-Dokument) einer Teilnahmebestätigung generiert werden, die ausgedruckt, unterschrieben, eingescannt und anschliessend vom Teilnehmer in dessen Bereich der CV-/Personendatenverwaltung wieder hochgeladen und abgespeichert wird. Alternativ könnte für diesen Anwendungsfall die besagte Vorlage beim Ändern des Teilnahmestatus eines Schulungsteilnehmers auf “absolviert” gleichzeitig erzeugt, digital signiert und unter den Schulungszertifikaten (PDF-Dokumente) der Person abgespeichert werden.

Hat ein Teilnehmer unerwarteter Weise doch nicht teilgenommen, oder war er nicht während der gesamten Dauer der verpflichteten Schulung anwesend, kann der Status auf "ferngeblieben" gesetzt werden. Der Administrator hat jederzeit die Möglichkeit, Mitarbeiter abzumelden, um z.B. eine zusätzliche Schulung an einem alternativen Termin anzulegen, und damit eine große Anzahl an Teilnehmern nachträglich aufzuspalten. Für ferngebliebene Mitarbeiter wäre ebenfalls eine neuerliche Schulung zur Nachholung anzusetzen.

Es zeigt sich die Unterscheidung von normalen Benutzern und Schulungsdatenbank-Adminstrator. Diese Rollen können durch zwei Berechtigungen vergeben werden:

- Neue Schulungen anlegen und bearbeiten erlaubt J/N
- An Schulungen teilnehmen möglich J/N

Jede dieser Zugriffsberechtigungen kann zur Trennung von Abteilungen hinsichtlich Abschnitt 4.3 weiter unterteilt werden, sofern erforderlich:

- Erlaubt nur für Schulungen der Abteilung des Benutzers. Für das Anlegen und Bearbeiten bleibt die Auswahl der Abteilung hierbei auf jene des Benutzers beschränkt.
- Erlaubt für Schulungen sämtlicher Abteilungen

Im Normalfall wird ein Benutzer in der Rolle als Schulungsdatenbank-Administrator nur Schulungen für seine Abteilung anlegen und bearbeiten dürfen. Die übrigen Benutzer können an Schulungen teilnehmen, wobei die Teilnahme an Schulungen sämtlicher Abteilungen freigeschaltet wird.

6.3 Erinnerungsfunktion für aufzufrischende Schulungen

Gewisse, typischerweise verpflichtende Schulungen müssen nach einem gewissen Zeitintervall zur Auffrischung oder auf Grund der veralteten Inhalte wiederholt werden. Bei solchen Schulungen kann die Gültigkeitsdauer oft auf Grund von Vorgaben im Vorhinein eingetragen werden. Innerhalb einer weiteren, definierbaren Zeitspanne vor Ablauf der Gültigkeit werden zu erneunernde Schulungen in der Listenansicht von Schulungen farblich hervorgehoben. Auch die "Absolventen" dieser Schulung sehen eine entsprechende Einblendung. Der Schulungsadministrator kann nun zur Auffrischung eine neuerliche Schulung organisieren und diese in der Schulungsdatenbank anlegen, wobei er dabei die abgelaufene Schulung verknüpft. Für den Fall, dass dadurch gleich mehrere obsolet gewordenen Schulungen aufgefrischt werden, ist auch die Verknüpfung mit mehr als einer abgelaufenen Schulung angedacht. Ein erfundenes, anschauliches Beispiel für Verknüpfungen:

1. "Erste-Hilfe-Kurs 2010", Juni 2010, 1 Jahr gültig
2. "Reanimationstraining 2010", September 2010, 1 Jahr gültig
3. "Sanitätskurs 2011", Juni 2011, 1 Jahr gültig: umfasst inhaltlich (1) und (2), wird deshalb verknüpft mit (1) und (2)

Durch diese Verknüpfungen kann die farbliche Hervorhebung von abgelaufenen Schulungen wieder aufgehoben werden.

6.4 Ablage und Verwaltung

Für Einträge in der Schulungsdatenbank werden folgende Attribute vorgesehen:

- Kategorie: Eine übergeordnete Kategorie der Schulung aus einer vordefinierten Liste zur Unterstützung von Abfragen und zur Aufbereitung in graphischen Oberflächen
- Abteilung: Referenz auf die Abteilung, der der Schulungseintrag zuzuordnen ist (Abschnitt 4.3). Einem Schulungsadministrator sollen Filterungen der Schulungen der Abteilungen ermöglicht werden.
- Titel: Zeichenfolge mit dem (internen) Titel der Schulung, inkl. Hinweis auf den Abhaltungszeitpunkt (bei wiederholten Schulungen) oder Teilnehmergruppe (bei Aufspaltung)
- CV-Titel: Bezeichnung, unter der die Schulung/Weiterbildung im CV des Teilnehmers angeführt werden soll
- Beschreibung: Allfällige Beschreibung der Schulung, Abhaltungsmodus, Kurzinformation für Ort und Zeit. Hier wäre noch eine feiner Aufspaltung in Felder möglich, falls erforderlich. Raumreservierungen können über die Reservierungsfunktion der Ausstattungsdatenbank erfolgen.
- verpflichtende Teilnahme: Markierung, ob die Schulung freiwillig oder verpflichtend ist.
- Referenzen auf klinische Studien: Für verpflichtende Teilnahme kann vermerkt werden, für welche laufenden oder bevorstehenden Studien diese Schulung erforderlich ist.
- freie Teilnahme: Markierung, die für freiwillige Schulungen vorgibt, ob sich Benutzer selbst eintragen können oder nicht.
- Teilnehmerzahl: Im Falle der freien Teilnahme die beschränkende Teilnehmerzahl
- Datum der Schulung: Stichtag der Schulung für
  - Gültigkeitsdauer
  - Erinnerungsfunktion
  - Zeugnis/Schulungszertifikat
- Abhaltungszeiträume: Eine Liste von Abhaltungsterminen jeweils bestehend aus:
  - Start- und Endzeitmarke
  - Titel (optional)
- Auffrischung erforderlich: Markierung, die eine Schulung als begrenzt gültig kennzeichnet
- Referenzen auf andere Schulungen, welche durch diese aufgefrischt werden
- Gültigkeit: Zeitspanne (Tage) ab dem Stichtag der Schulung, bis eine Auffrischung erfolgen muss. Alternativ könnte ein Ablaufdatum vorgesehen werden, weil es dadurch leichter fällt bei z.B. Zeitspannen von einem Jahr die gleiche Kalenderwoche des Vorjahres auszuwählen.
- CV-Kategorie: Voreinstellung für den Abschnitt des erzeugten CV‘s des Teilnehmers, unter dem der Vermerk angezeigt werden soll, den der Teilnehmer dann selbst noch abändern kann. Die verfügbaren Abschnitte sind auch im Abschnitt 4.2 zu finden:
  - Relevant Job related Training
  - Relevant Clinical Trial and Research Experience including GCP Training
  - Certifications and Licensures
  - Other Activities Pertinent to Professional Qualifications
  - <nicht im CV anzeigen>

- Referenzen auf die Mitarbeiter-Kategorien: Falls der Schulungsdatenbank-Administrator die Teilnehmerauswahl durch Mitarbeiter-Kategorien vornimmt, werden diese vermerkt. Nachdem dies zur Vorselektion dient, von der ausgehend dann einzelne Teilnehmer noch hinzugefügt oder abgewählt werden können, ist die Speicherung dieser Information nur der Übersicht halber vorgesehen.
- Dokumente zur Schulung: Ein Verlauf von PDF-Dateien wie z.B. Schulungsunterlagen, Unterschriftenlisten zur Teilnahmebestätigung usw.

Für Modifikationen der Schulungsdatenbank ist kein Änderungsprotokoll (Change Log) vorgesehen, abgesehen von den Zustandsänderungen der Teilnahme eines Mitarbeiters.

6.5 Datenbestand und Ausgangssituation

Um CVs der Mitarbeiter vollständig erzeugen zu können, müssten diverse, vergangene Schulungen in die Schulungsdatenbank des SMS nachgetragen werden. Der Aufwand hierfür sollte überschaubar bleiben.

7 Studiendatenbank

7.1 Übersicht

Der zentrale und auwändigste Bestandteil des StudienManagementSystems (SMS) widmet sich der eigentlichen Aufgabe der Organisation (CRC - Clinical Research Center des ZMF), der Abwicklung klinischer Studien. Im Rahmen einer klinischen Studie soll der Einsatz eines neuen Heilmittels, medizinischen Gerätes oder einer neuen Behandlungsmethode/Diagnostik unter kontrollierten Bedingungen an freiwilligen Probanden, die spezifische Kriterien erfüllen, überprüft werden. Entsprechend strenge Auflagen zur Durchführung der Studie sollen die gesundheitliche Sicherheit der Probanden gewährleisten und dem Ergebnis der Studie Aussagekraft verleihen:

- Votum einer Ethikkommission (national)
- EudraCT Registrierung der Studie (EU-weit)
- GCP (Good Clinical Practice) Richtlinen (EU-weit) bzw. GCP Verordnungen (national)
- Deklaration von Helsinki (weltweit)

Das CRC wickelt klinische Studien strukturiert als Projekte ab, deren Phasen bzw. Arbeitsschritte als Prozesse aufgefasst werden können. Die Vorgaben bzgl. der Dokumentation der Studie betreffen nicht nur Abgabedokumente mit Resultaten der Studie, sondern auch die Ausführung aller beteiligten Prozesse. Es muss absolute Nachvollziehbarkeit sämtlicher Abläufe gewährleistet sein.

Studien des CRC werden durch verschiedenste Institutionen (z.B. MUG, KAGes, externe Auftraggeber) beauftragt. Pro Studie werden dabei nicht selten spezifische Bestimmungen, Design der Studie oder Durchführungsanweisungen individuell vom Auftraggeber vorgegeben. Die u.A. davon betroffenen organisatorischen Aufgaben im Rahmen einer Studie können sich auf folgende Teilbereiche erstrecken:

1. Probandenrekrutierung, Probandendatenerhebung, Telefonprotokolle
2. Zusammenstellung von Probandengruppen,
3. Visitenplan
4. Mitarbeiterschulungen
5. Dienstplan
6. Ressourcenverwaltung (Raumreservierung, Bettenreservierung, Küche)
7. Bestellungen, Trial Material Manual (TMM)
8. Medikationsplanung
9. Proben:
   a. Probenentnahme
   b. Probendokumentation
   c. Probenhandhabung/-Lagerung
   d. Laborkoordination
   e. Messwertverwaltung
   f. statistische Auswertung

10. Dokumente:
    a. zu erstellenden Dokumente (teilweise mit Fristenlauf):
        i. Promotion (Inserate)
        ii. Protokoll (Clinical Trial Protocol), sofern für die Studie vom CRC selbst zu verfassen
        iii. Studienhandbuch
        iv. Case Report Form (CRF)
        v. Reports zur Probandeneinschreibung und -auswahl (Screening Log, Enrollment Log)
        vi. Abschlussbericht
    b. Ablage:
        i. Protokoll (Clinical Trial Protocol), sofern für die Studie vom Auftraggeber vorgegeben
        ii. Prüfarztbroschüre (IB - Investigator's Brochure)
        iii. Involvierte SOPs (Standard Operating Procedure Dokumente)
        iv. Votum d. Ethikkommission
        v. eidesstattliche Erklärung
        vi. Lebensläufe der Mitarbeiter
        vii. Datenschutzerklärungen für Probanden
        viii. Einverständniserklärungen der Probanden (IC - Informed Consent)
        ix. Probandentagebücher
    c. zuzuweisende Dokumente:
        i. CRF Review Anweisungen an Mitarbeiter
        ii. CRF Korrekturvermerke durch Mitarbeiter (Note-To-File)
        iii. Protokoll Reviews - Fragen durch Mitarbeiter, Antworten durch Studienauftraggeber
11. Versicherungen, med. Notfallmanagement
12. Probandenentschädigung
13. Handkassaverwaltung
14. Verwaltung der Hotelkosten von Probanden

Grundsätzlich ist davon auszugehen, dass die aufgelisteten Bereiche bzw. Aufgaben für jede Studie nach Vorgabe im Detail unterschiedlich abzuwickeln und abhängig vom Umfang z.T. gar nicht ausgeprägt sind . Gewisse Vorgänge kommen bei jeder Studie vor und sind darüber hinaus ähnlich. Für solche Teilprozesse, die verallgemeinert werden können und auch nicht durch spezielle, andere Softwaresysteme abgedeckt werden, soll die Softwareunterstützung durch die Studiendatenbank umgesetzt werden. Um die Machbarkeit innerhalb der vorgegebenen Zeit im Auge zu behalten, beschränken sich diese Funktionen vorläufig auf folgende Komponenten:

1. Projekt-Timeline
2. Probanden Pre-Screening Fragen und studienspezifische Probandendaten
3. Probandenliste
4. Gruppeneinteilung für Probanden und Visitenplan
5. Dienstplan
6. Checklisten
7. Dokumentverwaltung
8. Ablage und Verwaltung

Abbildung 4 zeigt als Überblick den Ablauf einer klinischen Studie als Sequenzdiagramm, wie er mit der Unterstützung durch das SMS aussehen würde.

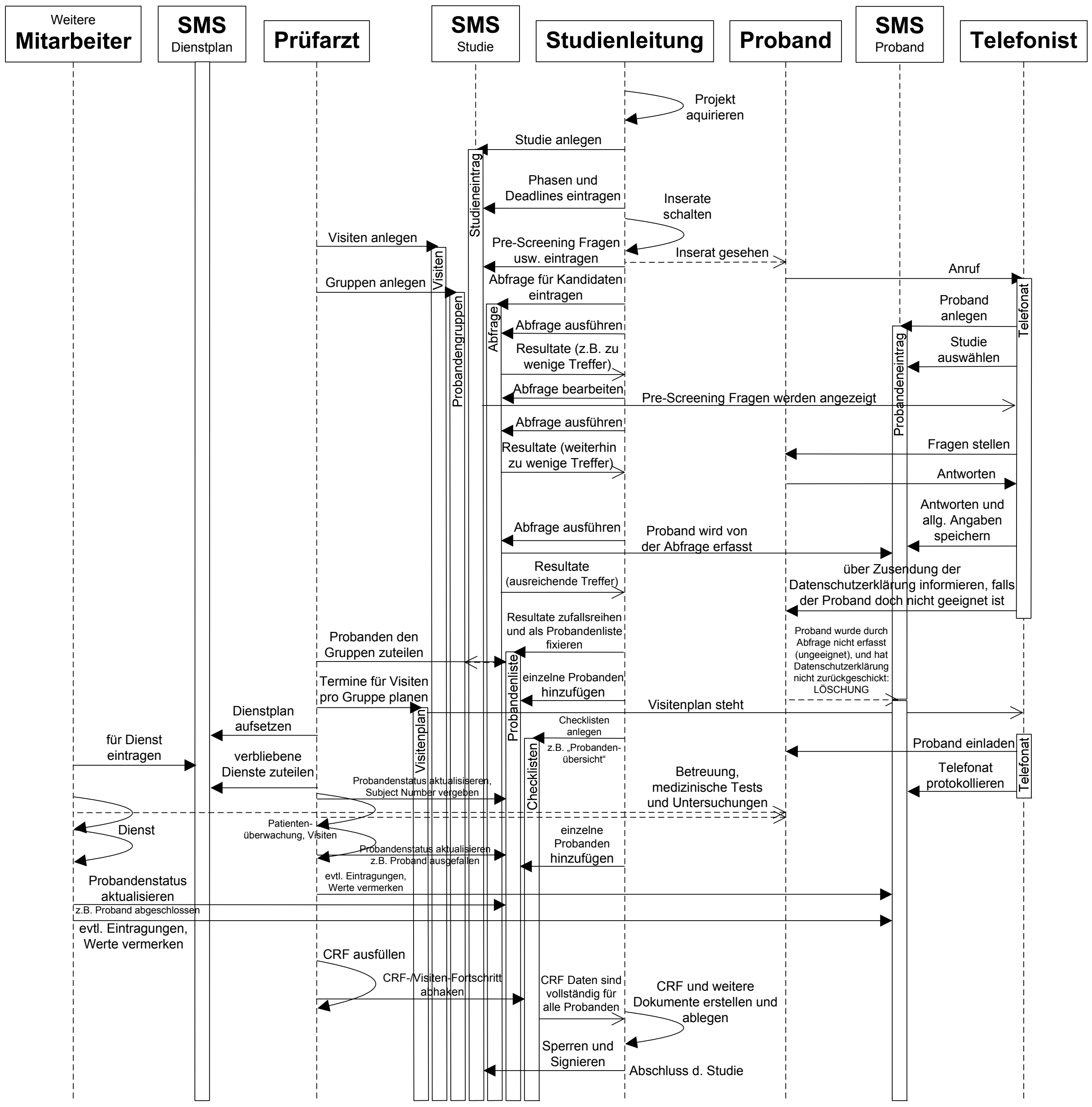

Weitere Mitarbeiter
SMS Dienstplan
Prüfarzt
SMS Studie
Studienleitung
Proband
SMS Proband
Telefonist
Projekt aquirieren
Studie anlegen
Studieneintrag
Phasen und Deadlines eintragen
Inserate schalten
Visiten anlegen
Pre-Screening Fragen usw. eintragen
Inserat gesehen
Anruf
Visiten
Gruppen anlegen
Abfrage für Kandidaten eintragen
Telefonat
Proband anlegen
Abfrage ausführen
Abfrage
Probandengruppen
Studie auswählen
Resultate (z.B. zu wenige Treffer)
Probandeneintrag
Abfrage bearbeiten
Pre-Screening Fragen werden angezeigt
Abfrage ausführen
Resultate (weiterhin zu wenige Treffer)
Fragen stellen
Antworten
Antworten und allg. Angaben speichern
Abfrage ausführen
Proband wird von der Abfrage erfasst
Resultate (ausreichende Treffer)
über Zusendung der Datenschutzerklärung informieren, falls der Proband doch nicht geeignet ist
Resultate zufallsreihen und als Probandenliste fixieren
Probanden den Gruppen zuteilen
Proband wurde durch Abfrage nicht erfasst (ungeeignet), und hat Datenschutzerklärung nicht zurückgeschickt: LÖSCHUNG
Termine für Visiten pro Gruppe planen
einzelne Probanden hinzufügen
Probandenliste
Visitenplan steht
Dienstplan aufsetzen
Visitenplan
Checklisten anlegen
z.B. „Probanden-übersicht“
Proband einladen
für Dienst eintragen
verbliebene Dienste zuteilen
Checklisten
Telefonat protokollieren
Probandenstatus aktualisiseren, Subject Number vergeben
Betreuung, medizinische Tests und Untersuchungen
Dienst
Patienten-überwachung, Visiten
einzelne Probanden hinzufügen
Probandenstatus aktualisieren z.B. Proband ausgefallen
evtl. Eintragungen, Werte vermerken
Probandenstatus aktualisieren
z.B. Proband abgeschlossen
evtl. Eintragungen, Werte vermerken
CRF ausfüllen
CRF-/Visiten-Fortschritt abhaken
CRF Daten sind vollständig für alle Probanden
CRF und weitere Dokumente erstellen und ablegen
Sperren und Signieren
Abschluss d. Studie

Abbildung 4: Abwicklung einer Studie im StudienManagementSystem

7.2 Projekt-Timeline

Betrachtet man eine klinische Studie als Projekt mit definierter Laufzeit, bietet sich die Darstellung als Zeitlinie an. Die Zeitliniendarstellungen stellt alle laufenden Projekte gegenüber. Diese Ansicht hat folgende Eigenschaften:

- dargestellter Zeitbereich (horizontal): Kalenderjahr (Überlegung: 6 Monate, 3 Monate)
- Auflösung: Kalendertage (Kalenderwochen bzw. Monate in der Kopfzeile)
- Ein Projekt (klinische Studie) wird jeweils in einem vertiaklen Bereich als separate Timeline dargestellt (siehe Abbildung 5). Studien werden anhand eines auswählbaren Status der Studie gestaffelt in folgenden Kategorien gegliedert:
  - zeitlich fixiert
  - zeitlich noch nicht fixiert, aber geplant
  - Projekte in Verhandlung
- dargestellte Ereignisse pro Projekt:
  - Startdatum des Projekts
  - Enddatum des Projekts
  - Zeitbereiche innerhalb des Projekts (Phasen): Jeder Zeitbereich erhält eine eigene Timeline unter der Timeline des zugehörigen Studienprojekts (siehe Abbildung 5).
    - Startdatum, Enddatum: einzelne Deadlines werden dargestellt als Phasen mit Startdatum gleich Enddatum
    - Titel: Freitext zur Beschriftung
    - Marke, ob Ereignis in der Timeline eingeblendet werden soll
    - Marke, ob eine Notifizierung per Email über das bevorstehende Ereignis erfolgen soll
    - Erinnerungszeitdauer: Zeitspanne vor dem Startdatum, zu deren Beginn die Erinnerung erfolgen soll. Während der Erinnerungszeitdauer kann wie bei der Erinnerungsfunktion für Wartungen der Ausstattungsdatenbank eine hervorstechende Ankündigung in einem Startseitenbereich der SMS Weboberfläche angezeigt werden.

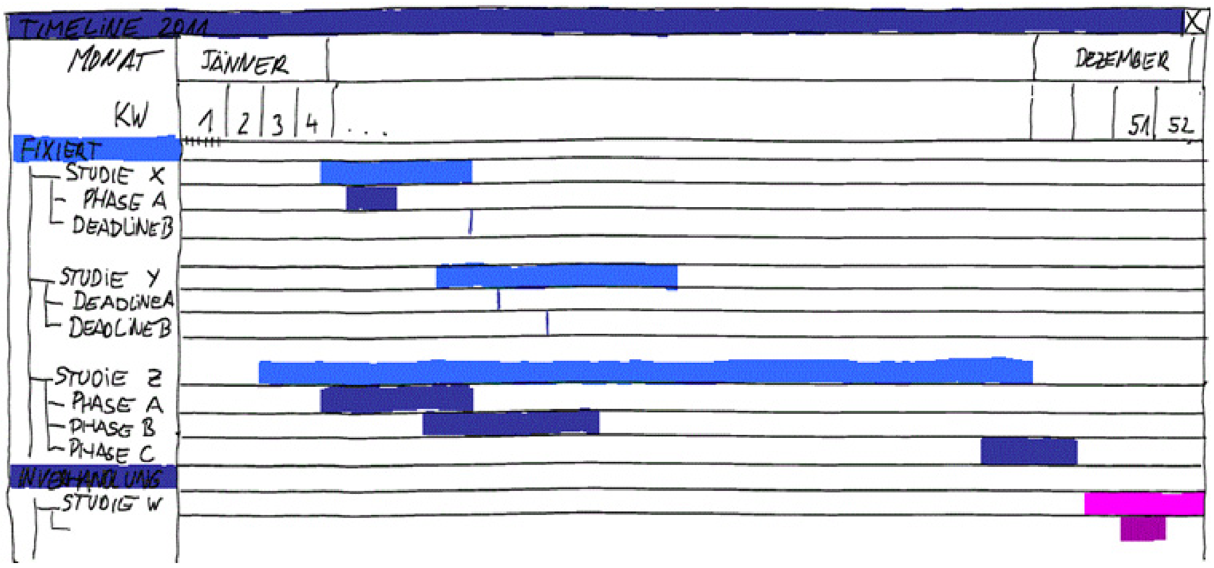


Abbildung 5: Projekt-Timelines

Ein SMS Benutzer mit entsprechender Berechtigung kann in der Timeline-Ansicht die Eigenschaften einer angezeigten Timeline (Projekt, Phase, Deadline) zur Bearbeitung öffnen.

Die Timeline-Ansicht soll die “Auftragslage” auf einen Blick erkennen lassen, sowie die Auslastung und mögliche freie Ressourcen schnell erkennen lassen. Die Erinnerungsfunktion der Phasen einer Studie kann eingesetzt werden, um den Fristenlauf elektronisch zu unterstützen.

Es wäre auch denkbar, Schulungen sowie Urlaubstage von Mitarbeitern in der Timeline-Ansicht einzublenden.

Die folgende, hochrangige Zugriffsberechtigung für Benutzer wird eingeführt:

- Studien anlegen und bearbeiten erlaubt J/N
  - Erlaubt nur für Studien der Abteilung des Benutzers. Für das Anlegen und Bearbeiten bleibt die Auswahl der Abteilung hierbei auf jene des Benutzers beschränkt.
  - Erlaubt für Studien sämtlicher Abteilungen („Supervisor“)

  Es werden dadurch folgene Funktionen kontrolliert:

  - Studiendaten und Details festlegen und bearbeiten (Abschnitt 7.9)
  - Phasen anlegen und bearbeiten

Studien können in der Timeline-Ansicht nach Abteilungen graphisch aufgeschlüsselt werden. Soll die Einsicht eines Benutzers über Studien fremder Abteilungen eingeschränkt werden können, kann dies durch Verfeinerung in Form zusätzlicher Berechtigungen erreicht werden:

- Timelines der Studien sämtlicher Abteilungen anzeigen J/N
- Timelines der Studien der Abteilung des Benutzers anzeigen J/N

7.3 Probanden Pre-Screening Fragen und studienspezifische Probandendaten

Eine frühe und essentielle Phase jeder klinischen Studie ist die Aufstellung einer Liste von teilnehmenden Probanden. Die Probandendatenbank wird ausgehend von einem bereits vorhandenen Probanden-Datensstamm im Laufe der Zeit durch kontinuierliches Anlegen von neuen Probandeneinträgen anwachsen, sodass für eine Studie eine Auswahl aus den existierenden Probanden vorgenommen werden kann. Diese Auswahl entspricht einer Auflistung mit zufälliger Reihenfolge von verzeichneten Probanden, die z.B. entsprechende Abfragekriterien erfüllen.

Es ist offensichtlich, dass für eine gewisse Studie Probanden mit speziellen Eigenschaften bzw. Kriterien benötigt werden, die sich nicht unter den vorhandenen Probandeneinträgen finden lassen werden - triviales Beispiel: “Probanden für die Studie xy dürfen noch nie bei einer anderen Studie teilgenommen haben”. In solchen Fällen enthält die Studie eine eigene, anfängliche Phase - die Rekrutierung von geeigneten, neuen Probanden. Dies geschieht durch Anwerbung z.B. durch in Printmedien geschaltete Anzeigen mit einem Aufruf zur Teilnahme. Daraufhin melden sich dann potentielle Probanden (typischerweise telefonisch), deren Daten daraufhin in die Probandendatenbank aufgenommen werden. Die Anzahl der Treffer bei der Probandenabfrage erhöht sich, bis irgendwann die für die Studie erforderliche Anzahl an Probanden aufgestellt werden kann.

Als “Screening” bezeichnet man die prüfärztliche Feststellung der Eignung eines aufgelisteten Probanden für eine durchzuführende Studie. Dabei wird das gesundheitliche Risiko des Probanden durch mögliche Auswirkungen der Teilnahme an der Studie fachkundig erfasst, um die gesundheitliche Sicherheit des Probanden so weit wie möglich sicherzustellen. Des weiteren wird die Erfüllung der studienspezifischen Kriterien des Probanden im Detail abgeklärt und offiziell bestätigt. Das Screening passiert bei der ersten von mehreren, im Visitenplan abgebildeten Visiten durch die für die Studie verantwortlichen Prüfärzte. Probanden müssen zu diesem Zweck bereits in das CRC eingeladen werden, darüber hinaus entsteht teurer Personalaufwand (Prüfärzte). Deshalb sollten nur Probanden untersucht werden, die mit hoher Wahrscheinlichkeit tatsächlich geeignet sind. Während der erstmaligen telefonischen Kontaktaufnahme in der Rekrutierungsphase sollen die dafür erforderlichen Daten erhoben werden, sodass beim Aufstellen der Probandenliste durch Abfragekriterien ungeeignete Probanden bereits herausgefiltert werden. Diese allererste Selektion mittels Abfragekriterien, die sich auf frühestmöglich erfasste Probandendaten beziehen, wird als “Pre-Screening” bezeichnet.

Die Unterstützung des Pre-Screening Vorgangs in der SMS Software soll durch die studienspezifische Definition von Fragen bzw. zu erfragenden Parametern ermöglicht werden, welche in der Datenerfassung beim Anlegen neuer Einträge für die Probandendatenbank zur Befüllung dargestellt werden. Abfragekriterien für die Probandenliste können in weiterer Folge auf die Antworten bzw. eingetragenen Werte für die Parameter zurückgreifen. Ein SMS Benutzer, der im Rahmen der Rekrutierungsphase Telefonanrufe von Probanden entgegennimmt, soll die Dateneingabe im SMS “live” während des Telefonats vornehmen können. Er legt dabei die grundsätzlichen Personendaten des Probanden (vgl. Personaldatenbank, Probandendatenbank) an und bekommt darüber hinaus die Eingabeformulare für zusammenfassend folgende Datenbündel präsentiert (z.B. graphisch durch Karteireiter unterteilt, siehe Abbildung 6, Abbildung 15):

1. Allgemeine Probandendaten (siehe Probandendatenbank): Eingabemöglichkeiten, um allgemeine Kriterien zu erfassen, die für unabhängige Probandenabfragen/-Suchvorgänge und Probandenabfragen für die Probandenliste für andere laufende oder zukünftige Studien sinnvoll sind.
2. Auswahl der Studie, für die sich der Proband meldet:
   (a) Pre-Screening Fragen für die spezifische Studie
   (b) detailiertere Probandendaten (z.B. diverse Parameter) für die spezifische Studie. Diese können z.B. auch von Mitarbeitern bzw. Prüfärzten während der laufenden Studie ausgefüllt und vervollständigt werden. Es steht offen, inwiefern diese von den Pre-Screening zu trennen wären. Interessant wäre in erster Linie eine Priorisierung der zu erfragenden Angaben.
3. Pre-Screening Fragen und Probandendaten der restlichen Studien:
   (a) momentan laufende Studien
   (b) evtl. vergangene Studien

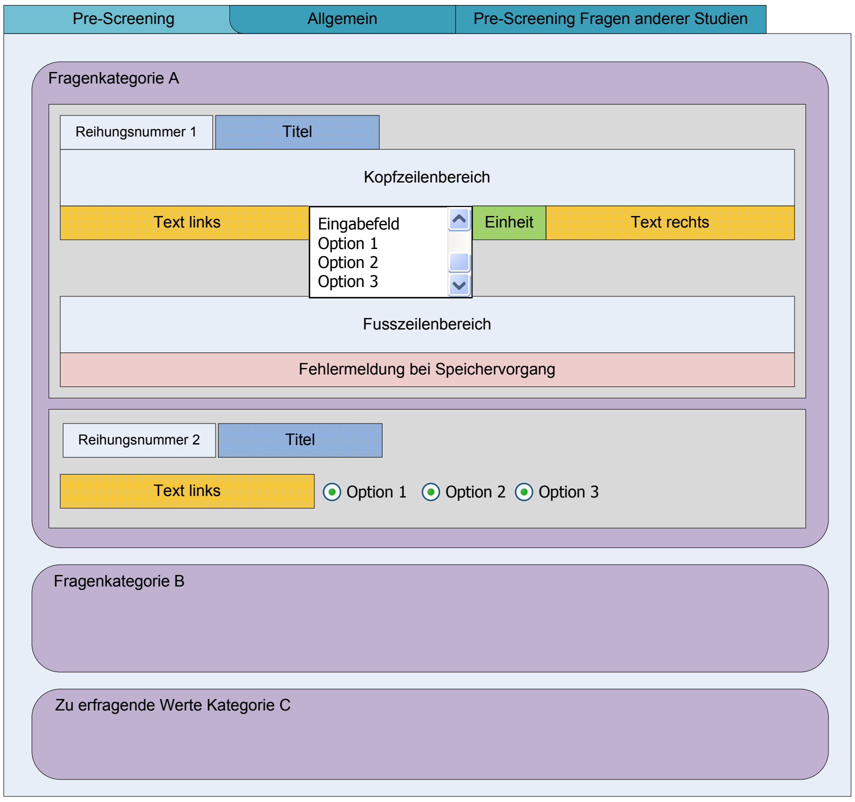


Abbildung 6: Eingabefelder für Fragen und Parameter

Die telefonische Erfassung sensibler, medizinischer Daten potentieller Probanden unterliegen zum Schutz der Privatsphäre strengen Datenschutzbestimmungen. Ein Proband, der das Screening vor Ort (CRC) positiv besteht, wird unmittelbar eine Einverständniserklärung für die Teilnahme an der Studie unterschreiben (Informed Consent), welche auch die Erlaubnis zur Ablage seiner Daten in Datenbanken des CRCs enthält und damit rechtlich absichert. Für Probanden, die sich bereits anhand der Pre-Screening Fragen nach dem telefonischen Erstkontakt für die eine Studie disqualifizieren, können für andere, zukünftige Studien geeignet sein und wünschen sich sogar oft die Teilnahme (inkl. Benachrichtigung) an einer solchen. Auch wenn sich der Proband am Telefon mit der Aufzeichnung seiner Daten ausdrücklich einverstanden erklärt, führt der diskutierte Weg zur rechtlichen Absicherung nicht an einer schriftlichen Bestätigung vorbei. Die Vorgehensweise würde mit der Zusendung einer Einverständniserklärung zur Speicherung der Daten (evtl. als PDF Dokument im SMS zum Ausdruck/E-Mail Versandt generierbar) an den Probanden beginnen. Diese wird vom Probanden unterschrieben und postwendend an das CRC zurückgesendet und dort abgelegt. Schickt der Proband die Erklärung innerhalb einer definierbaren Zeit (z.B. 30 Kalendertage) nicht zurück, ist davon auszugehen, dass er/sie nicht einverstanden ist, und es dürfen keine Aufzeichnungen ausser den Personenkontaktdaten behalten werden. Um dies in der SMS Software durch minimale Benutzerinteraktion zu erfüllen, bleiben die Probandendaten nach der Eingabe zwischenzeitlich gespeichert und könnten durch eine periodisch (in diesem Fall täglich) laufende, automatische Programmausführung (Job) am SMS Server gelöscht werden, sofern die Dateneingabe länger als 30 Tage zurückliegt. Wird eine unterschriebene Einverständniserklärung erhalten, kann ein SMS Benutzer den Eintrag des betroffenen Probanden suchen und öffnen, und als permanent markieren, um die automatische Löschung abzuwenden.

Die Abbildung von dynamsich konfigurierbaren Eingabefeldern für Fragen an Probanden bzw. Parametererfassung kann in der Datenbank durch eine Liste erfolgen, deren Einträge folgende Attribute aufweisen:

- ID: interner Name oder Zahl zur Referenzierung
- aktiv: Markierung, ob dieses Eingabefeld eingeblendet wird um ausgefüllt zu werden. Einträge der Liste für die Eingabefelder dürfen nicht gelöscht werden, stattdessen sind nicht mehr benötigte Felder mittels dieser Markierung zu deaktivieren.
- Kategorie: eine Kategorie als Freitext oder Auswahl aus einer vordefinierten Liste, z.B.:
  - "Basisinformation"
  - "Organsystem"

  Bei Freitexteingaben muss die Schreibweise der Kategorie übereinstimmen, damit Eingabefelder darin zusammengefasst werden.

  Durch Kategorien können auch Tupel von Eingabefeldern gebildet werden, z.B.:

  - Kategorie „Krankengeschichte Eintrag 1":
    - „Behandlungszeitraum 1 von": Datums-Auswahl
    - „Behandlungszeitraum 1 bis": Datum-Auswahl
    - „Behandlung 1:": Textfeld
  - Kategorie „Krankengeschichte Eintrag 2":
    - „Behandlungszeitraum 2 von": Datums-Auswahl
    - „Behandlungszeitraum 2 bis": Datum-Auswahl
    - „Behandlung 2:": Textfeld
  - Kategorie „Krankengeschichte Eintrag 3", Krankengeschichte Eintrag 4", ...

  Bei Bedarf könnte eine zweite Hierarchieebene zur Kategorisierung eingebaut werden.

- Reihungsnummer: Ganzzahl, die für die Reihung der Eingabefelder herangezogen wird. Dadurch wird die Position in der Anzeige innerhalb der Kategorie (siehe Abbildung 6) festgelegt.
- Titel: Titel der Frage bzw. des Parameters für Auflistungen oder versteckbare (collapsable) Anzeigebereiche, z.B.:
  - "Blutdruck systolisch"
  - "Geschlecht"
  - "Raucher"
  - "Eingriffe"
- Text links, rechts: Text links und rechts des Eingabefeldes für den Wortlaut der Fragestellung oder für den Parameternamen, z.B.:
  - "Blutdruck systolisch:"
  - "Geschlecht:"
  - "Sind Sie Raucher?
  - "Welchen operativen Eingriffen wurden Sie bis dato unterzogen?"
- Einheit: Masseinheit für Parameterwert (Zahlenwerteingaben) z.B.: "mmHg", "kg" als Freitext oder Auswahl aus einer vordefinierten Liste

- Konfigurationsdaten für Eingabefelder:
  - Typ:
    - einzeiliges, mehrzeiliges Textfeld
    - Auswahllisten mit Einfachauswahl, mit Mehrfachauswahl, verschachtelten Strukturen (Menü)
    - Checkboxes, Radiobuttons
    - evtl. Datums-Auswahl
  - Auswahloptionen für Auswahllisten, Checkboxen und Radiobuttons, z.B.:
    - “männlich”, “weiblich”
    - “Ja”, “Nein”, “Unbekannt”
  - Repräsentation der Auswahloptionen bei der Speicherung (Mapping), z.B.:
    - Ja = 1, Nein = -1, Unbekannt = 0
  - voreingestellter Wert
  - Feld ausgegraut: Markierung, ob das Eingabefeld unveränderlich ausgegraut ist. Das Feld kann dadurch deaktiviert werden, sodass zwangsweise immer der voreingestellte Wert gespeichert wird.
  - Eingabe optional: Sofern kein voreingestellter Wert festgelegt wird, kann diese Markierung eine Eingabe erzwingen.
  - Überprüfungsausdruck: Die eingegebenen Werte werden prinzipiell als Text gespeichert. Um nun beispielsweise für eine Zahlenwerteingabe sicherzustellen, dass nur Ziffern eingetippt werden, kann die Überprüfung durch Vergleich mit einer Musterzeichenkette erfolgen, die umfassende Platzhalterzeichenangaben enthalten kann. Prinzipiell können mittels solcher regulären Ausdrücke (Regular Expressions) z.B. auch Wertebereiche eingeschränkt werden, was allerdings evtl. etwas umständlich ist. Der folgede reguläre Ausdruck besteht den Vergeich mit Zeichenfolgen, die Ganzzahlen von 0 bis 999 darstellen:

    $$/\hat{}[0-9]\{1{,}3\}\$/$$

    Noch flexibler wäre hierbei ein Überprüfungsausdruck als Codefragment der Programmiersprache der Laufzeitumgebung der SMS Software. Programmiersprachen mit der Fähigkeit zur “Reflektion” können solche Codefragmente ausführen, obwohl diese nicht vordefinierter Bestandteil der Software im Auslieferungszustand sind (d.h. ohne Intervention durch den Programmierer). Diese Variante birgt jedoch auch Risiken bzgl. der Zuverlässigkeit und Datensicherheit, nachdem auf diese Weise grundsätzlich beliebiger Programmcode eingeschleust werden kann.

  - Fehlermeldung: Schlägt der Vergleich mit dem Überprüfungsausdruck fehl, soll der Benutzer beim Speichervorgang eine aussagekräftige Meldung sehen. Diese Fehlermeldung kann als Freitext vordefiniert werden, z.B. (“Zahlenwert zwischen 0 und 999 erforderlich.”).
- Kopf- und Fusszeilenbereich: Freitexte für Hinweise, Beispiele, SOP Hyperlinks zur Frage bzw. dem zu erfragenden Parameter etc.

Nicht berücksichtigt wird z.B. die automatische Berechnung von Kenngrößen wie dem BMI (Body Mass Index) aus eingegebener Körpergröße und Gewicht.

Das Vordefinieren von Eingabefeldern muss durch direkte Eintragungen in der Datenbank erfolgen, wofür die (minimale) Intervention durch einen technischen Datenbankadministrator erforderlich ist. Alternativ wäre die Importmöglichkeit der Eingabefeld-Liste als XML Datei, Excel Datei o.ä. realisierbar.

Nachdem einmal alle benötigten Eingabefelddefinitionen vorbereitet und eingespielt wurden, kann per SMS Webfrontend für eine neu angelegte Studie die Auswahl getroffen werden, welche der verfügbaren Eingabefelder dem Telefonisten am SMS Frontend beim telefonischen Erstkontakt mit Probanden als Pre-Screening Fragen präsentiert werden. Alle weiteren, im Laufe der Visiten zu erfassenden Probandendaten können durch eine weitere Zusammenstellung von verfügbaren Eingabefeldern für die Studie festgelegt werden.

Folgende Zugriffsberechtigungen für Benutzer können identifiziert werden:

- Studien anlegen und bearbeiten erlaubt J/N (vgl. Abschnitt 7.2)
  - Erlaubt nur für Studien der Abteilung des Benutzers, sowie für Studien, deren Projektteam der Benutzer angehört.
  - Erlaubt für Studien sämtlicher Abteilungen („Supervisor“)

  Es werden hierbei folgene Funktionen kontrolliert:

  - Auswahl von Pre-Screening Fragen sowie zu erfassenden Probandendaten festlegen bzw. bearbeiten
- Pre-Screening Fragen des Probandeneintrags ausfüllen erlaubt J/N
- Probandendaten des Probandeneintrags ausfüllen erlaubt J/N

Die letzten beiden Berechtigungen betreffen genaugenommen die Probandendatenbank (Abschnitt 8.3), sind der Übersicht halber an dieser Stelle angeführt. Folgende Verfeinerungen sind dabei vorgesehen:

- Erlaubt nur für Probanden der Abteilung des Benutzers oder für Probanden der Probandenliste (Abschnitt 7.4) der jeweiligen Studie, sofern der Benutzer im Projektteam der Studie (Abschnitt 7.9) vermerkt ist.
- Erlaubt für Probanden sämtlicher Abteilungen

7.4 Probandenliste

Für klinische Studien ist immer die zufällige Auswahl von Probanden vorgeschrieben. Die Auswahl von Probanden erfolgt nach dem im Folgenden erklärten Ablauf.

Für eine im SMS angelegte Studie wird eine Abfrage gegen die Probandendatenbank festgelegt und gespeichert, welche die Probandenkriterien erfasst. Eine Abfrage kann formal als WHERE Klausel von SQL (Structured Query Language) SELECT Statements aufgefasst werden:

$$SELECT \ldots FROM < Tabelle1, Tabelle2, \ldots > WHERE < WHERE - Klausel >$$

Die WHERE Klausel entspricht der bool'schen Verknüpfung von Abfragebedingungen (Termen), die für jede Zeile (Datensatz) der abgefragten Datenbanktabelle erfüllt sein muss, um im Ergebnis der Abfrage (Result Set) gelistet zu werden.

Eine einzelne Abfragebedingung entspricht einer durch Vergleichsoperatoren festgelegte Bedingung, welche die Werte in betroffenen Spalten (Attributen) der Tabellen erfüllen müssen. Folgendes Beispiel zeigt eine Abfrage, bei der männliche Personen über 50 oder Personen mit einem Blutzuckerspiegel über 7.0 mmol/l ausgewählt werden:

$$(< Spalte\_Alter \quad > >= 50\ AND < Spalte\_Geschlecht \quad > = 'male')\ OR \\ < Spalte\_Blutzucker\_Nuechtern \quad > >= 7.0$$

Festzuhalten ist, dass durch diesen Ansatz komplexere Abfragen ausgeschlossen sind, wie z.B.:

- Abfragen die durch temporäre Tabellen bzw. verschachtelten SELECT Statements realisiert werden müssen, z.B.: „Probanden, deren Maximum aller vermerkten Blutzuckerwerte über 7.0 mmol/l liegt“
- Abfragen die Gruppierungsklauseln enthalten (GROUP BY Klausel, HAVING Klausel), z.B.: „Probanden, die bereits an mindestens 3 Studien teilgenommen haben“

Die Idee ist, eine allgemeine Abfrage in dieser Form in eine Datenstruktur zu bringen, um dadurch ein speicherbares Format zu erhalten. Der Aufbau der Abfrage-Datenstruktur kann aus der Darstellung der Bearbeitungsmöglichkeiten einer Abfrage in der SMS Weboberfläche abgelesen werden (Abbildung 7, “SQL Where Clause Builder”).

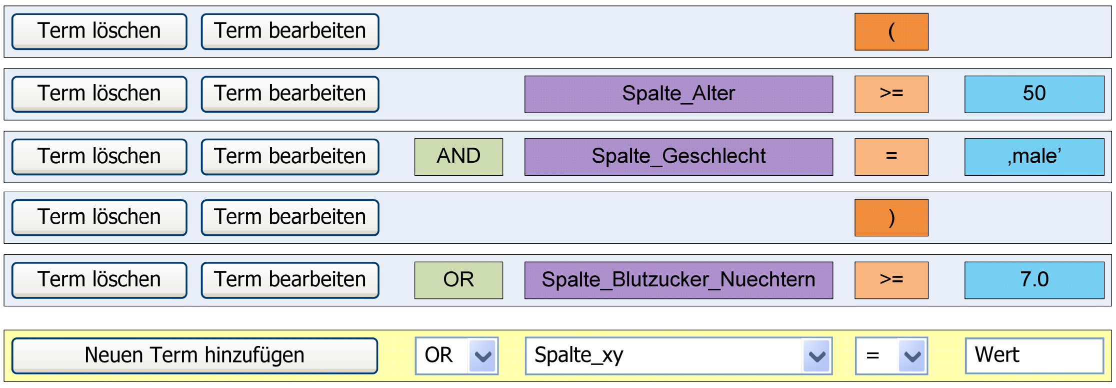

Term löschen | Term bearbeiten | (
Term löschen | Term bearbeiten | Spalte_Alter | >= | 50
Term löschen | Term bearbeiten | AND | Spalte_Geschlecht | = | ‚male’
Term löschen | Term bearbeiten |)
Term löschen | Term bearbeiten | OR | Spalte_Blutzucker_Nuechtern | >= | 7.0
Neuen Term hinzufügen | OR | Spalte_xy | = | Wert

Abbildung 7: Eingabe von Abfragen

Die erforderlichen Attribute für die Probandendatenabfrage sind in der SMS Datenbank jedoch nur zum Teil als statische, diskrete Spalten von Tabellen ausgeführt. Die spezifisch definierten Eingabefelder (Abschnitt 7.3), die bei der Abfrage einbezogen werden sollen sind nicht direkt als Spalten der Tabellen ausgeführt. Der Aufbau der Abfrageeingabe wird zumindest unsichtbar (im Hintergrund) eine getrennte Handhabung erfordern, um die Abfrageattribute einheitlich zu präsentieren. Ebenso wird es immanente, unveränderliche Filterungen geben, wie etwa:

- Einschränkung auf Probanden der Abteilung der Studie, sofern der Benutzer Teil des Projektteams der Studie ist (Abschnitt 7.9). Dies ist ermöglicht überhaupt die einzige Situation, bei der ein abteilungsfremder Benutzer den Probandenstamm der fremden Abteilung abfragen kann (vgl. Abschnitt 8.1)
- Einschränkung auf Probanden, die sich für die Studie gemeldet haben und ihre Bereitschaft angegeben haben, auch an anderen Studien teilzunehmen (Abschnitt 8.3)
- Evtl. Einschränkungen auf Probanden, die nicht auf Probandenlisten anderer, zeitgleicher Studien der Abteilung stehen.

Es wird sich noch im Detail herausstellen, welche Abfragemöglichkeiten der folgenden, möglichen tatsächlich benötigt werden:

- statische Attribute (Tabellenspalten):
  - Personenkontaktdaten (siehe Personaldatenbank, Probandendatenbank): u.a. Informationen wie Alter, Name, Stadt, ...
  - zusätzliche Datenfelder der Probandendatenbank (vgl. Abschnitt 8.3), z.B.:
    - Zeitmarken und Benutzer, der den Probandeneintrag angelegt hat
    - Studie, für die sich ein Proband gemeldet hat
    - Markierung für Bereiterklärung des Probanden zur Teilnahme an anderen Studien
    - zeitliche Verfügbarkeit des Probanden (von – bis)
    - evtl. Interaction Protokoll (Journalaufzeichnungen z.B. über Telefonate mit Freitextkommentaren)
- Referenzierungen von Probandendaten:
  - Probandenlisten von Studien:
    - Probandenstatus bei Studien
    - Gruppenzuteilung
    - vergebene Subject Number, Random Number, Referenznummer für externes Labor
  - Ausstattungsreservierung
- Die eingetragenen Werte aller verfügbaren Eingabefelder (siehe Abschnitt 7.3):
  - Antworten der Pre-Screening Fragen der Studie und aller anderen Studien
  - übrige erfragte bzw. eingetragene Probandendaten der Studie und aller anderen Studien
  - allgemeine eingetragene Probandendaten

Neben spezifischen, für die Studie vorgegebene Kriterien an Probanden (z.B. Mindestalter) wird die Probandenabfrage einer Studie typischer Weise die Filterung nach Pre-Screening Antworten und Parameterwerten der Probandendaten enthalten.

Die der Studie hinterlegte Abfrage kann jederzeit wiederholt ausgeführt werden, um die momentane Größe der Ergebnisliste zu erkennen. Während der Rekrutierungsphase würde diese z.B. von Tag zu Tag ansteigen. Die Ergebnisliste ist eine Auflistung aller Probanden, die die Abfragekriterien erfüllen und damit als Kandidaten für die Studie in Frage kommen. Erhält man zu wenige Treffer, könnte die Abfrage nachträglich bearbeitet werden, um Kriterien abzuschwächen. Ist die Anzahl der gelisteten Kandidaten ausreichend (d.h. nach Abschätzung des Studienleiters z.B. um einen gewissen Prozentsatz über der erforderlichen Anzahl an Probanden), kann durch eine Aktion in der Weboberfläche die Kandidatenliste fixiert werden, wodurch die Probandenliste der Studie erzeugt wird. Dies bedeutet, dass das Ergebnis der Abfrage mit zufälliger Reihenfolge geordnet wird und anschliessend die einzelnen Probanden mit der Studie verknüpft und abgespeichert werden. Einer Vorgabe entsprechend wird auch der Initialisierungsvektor (Seed) für die Instanz des Pseudozufallszahlengenerators der Programmiersprache bzw. Laufzeitumgebung festgehalten, die für die Zufallsreihung erzeugt und verwendet wird. Der Seed-Wert kann beim Fixieren der Probandeliste eingegeben werden, wobei als Standardwert in der SMS Oberfläche z.B. eine Zeichenfolge mit der momentanen Zeitmarke vorgeschlagen wird.

Um Reproduzierbarkeit zu erreichen, müssten für die zufällige Reihung darüber hinaus auf den ersten Blick folgende Kennwerte aufgezeichnet werden:

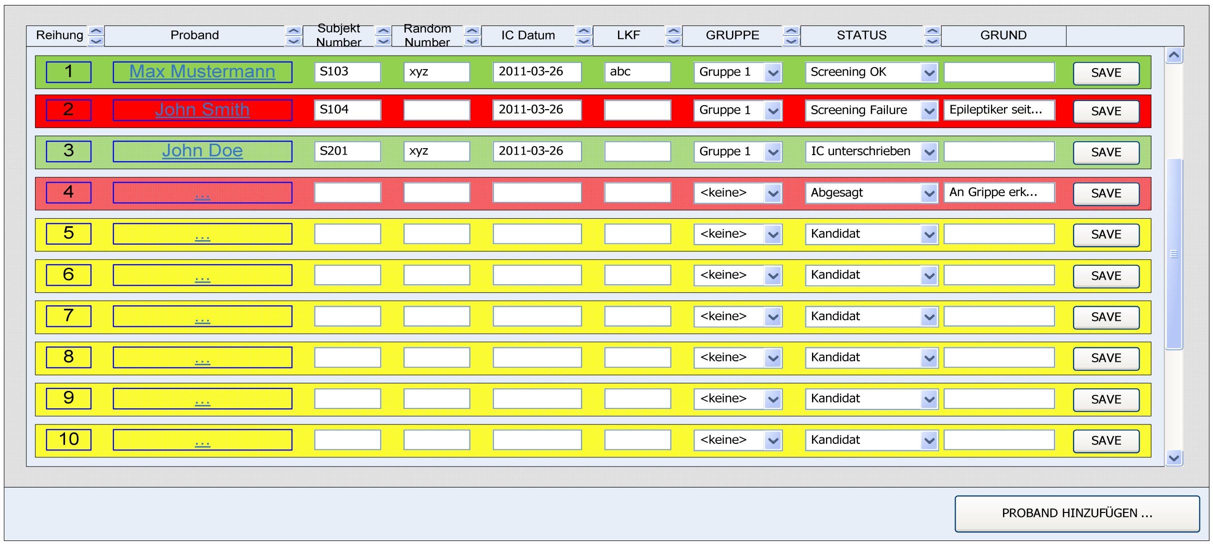
- CPU Archtitektur des SMS Servers (z.B. x64, IA32, IA64 etc.), unterstützte Befehlssätze (z.B. SSEx)
- Version der Laufzeitumgebung (bzw. des Interpreters oder Compilers/Linkers)
- Klassenbibliothekname des eingesetzten Zufallszahlengenerators und evtl. deren separate Version, z.B.:
  - java.util.Random (Java)
  - System.Random - mscorlib.dll, Version 2.x.y.z (Microsoft .NET)
- Namen und Versionen von eventuell eingesetzten Laufzeit-Zusatzmodulen, welche die Funktion des Zufallszahlengenerators durch Überladungen beeinflussen

Auch nach dem Beginn der Probandenuntersuchungen im Rahmen der Visiten durch Prüfärzte kann es zu Änderungen der Teilnehmerzusammenstellung kommen. Ein Proband kann während der laufenden Studie z.B. wegen Unverträglichkeit ausfallen. Vor allem während der ersten Visite der Studie (Screening) werden erst endgültige Probanden aus den fixierten Kandidaten der Probandenliste auserkoren, und es könnten mehr Probanden ausscheiden, als erwartet. Nun könnte der Vorgang der Fixierung der Probandenliste wiederholt werden, jedoch werden dabei bereits getätigte Eingaben wie Probandenstatusänderungen verworfen. Deshalb wird es erforderlich, eine bestehende Probandenliste jederzeit um einzeln aus der Probandendatenbank auswählbare Probanden erweitern zu können (Probanden-„Picker“).

Der Ablauf in Verbindung mit dem SMS nach Fixierung der Probandenliste könnte zusammenfassend wie folgt umrissen werden:

1. Erstellung von Gruppen, Visiten und Visitenplan (Abschnitt 7.5), Zuteilung der Probanden zu Gruppen.
2. Die Unterschrift für die Einverständiserklärung (IC) wird vom Probanden eingeholt.
3. Die engültige Eignung des Probanden wird bei der Screeningvisite durch den Prüfarzt festgestellt. Eine entsprechende Statusänderung des Kandidaten wird in der Probandenliste eingetragen (Abbildung 8).
4. Eine Subject Number wird laut Vorgabe aus dem Clinical Trial Protocol manuell vergeben und in der Probandenliste eingetragen.
5. Das Datum der IC Unterzeichnung jedes Probanden ist für gewisse Studien ausdrücklich zu dokumentieren. Diese Information muss daher durch Eintragung in der Probandenliste erfasst werden.
6. Während Dosierungsvisiten wird dem Probanden das Heilmittel verabreicht. Die vom CRC abgewickelten klinische Studien werden genrell mit „doppelter Verblindung“ durchgeführt. Dies bedeutet, dass sowohl Probanden als auch sämtliche Mitarbeiter während der gesamten Studie nicht wissen, was (z.B. neuer Wirkstoff, Generika oder Referenzwirkstoff, Placebo) welchen Probanden verabreicht wird. Der Auftraggeber der Studie stellt die Medikationen gemeinsam mit dem Trial Medication Manual bereit, wobei Dosen nur durch einen Code identifiziert werden. Der tatsächliche Inhalt der Dosen ist nur dem Auftraggeber bekannt. Den Probanden werden vorbereitete, bereitgestellte „Random Numbers“ vergeben, anhand deren die vorbereiteten Dosen ausgewählt werden. Die Random Number eines Probanden muss in der Probandenliste verzeichnet werden.

7. Ein Proband fällt aus, was durch eine Statusänderung mit Angabe des Grundes vermerkt wird. Ein neuer Proband muss per Probanden-„Picker“ der Probandenliste hinzugefügt werden, um erfasst zu werden.
8. Bei manchen Studien werden Proben zur Analyse an ein externes Labor geschickt (LKF - Laboratorium für Klinische Forschung). In diesem Zusammenhang muss eine Referenznummer (LKF Nummer) für Proben eines Probanden in der Probandenliste vermerkt werden.
9. Studienspezifische Fragen und Parametereingaben können im SMS erfasst bzw. im Laufe der Visiten sukzessive vervollständigt werden.
10. Nach der letzten Visite kann der Status des Probanden auf „abgeschlossen“ gesetzt werden.



Reihung	Proband	Subjekt Number	Random Number	IC Datum	LKF	GRUPPE	STATUS	GRUND	
1	Max Mustermann	S103	xyz	2011-03-26	abc	Gruppe 1	Screening OK		SAVE
2	John Smith	S104		2011-03-26		Gruppe 1	Screening Failure	Epileptiker seit...	SAVE
3	John Doe	S201	xyz	2011-03-26		Gruppe 1	IC unterschrieben		SAVE
4	...					<keine>	Abgesagt	An Grippe erk...	SAVE
5	...					<keine>	Kandidat		SAVE
6	...					<keine>	Kandidat		SAVE
7	...					<keine>	Kandidat		SAVE
8	...					<keine>	Kandidat		SAVE
9	...					<keine>	Kandidat		SAVE
10	...					<keine>	Kandidat		SAVE

Abbildung 8: Probandenliste

Die Datenstruktur hinter der Probandenliste (Abbildung 8) lässt sich somit als Tabelle mit Datensätzen modellieren, die folgende Attribute aufweisen:

- Probanden ID: Referenz auf den Eintrag in der Probandendatenbank
- Gruppen ID: Gruppenzuteilung durch Referenz auf eine zuvor angelegte Gruppe (Abschnitt 7.5)
- Status: Folgende Zustände sind für einen Probanden vorgesehen:
  - Kandidat: Default-Zustand nach Fixieren der Probandenliste
  - Kontaktiert: Telefonischer Kontakt wurde hergestellt
  - Abgesagt: Proband hat abgesagt
  - IC unterschrieben: Proband hat zugesagt, unterzeichneter IC liegt vor
  - Screening OK: Screeningvisite positiv
  - Screening Failure: Screeningvisite negativ, Proband ist nicht für die Studie nicht geeignet
  - Drop Out: Proband ist während der laufenden Studie ausgefallen
  - Abgeschlossen: z.B. nachdem der Proband sämtlicher Visiten unterzogen wurde

Die erlaubten Zustandswechsel sind in Abbildung 8 dargestellt. Die Kanten des gerichteten Graphs beschreiben, welche zusätzlichen Eingaben bei der jeweiligen Transition erfolgen müssen. Zustände können übersprungen werden, sofern ein Pfad vom Anfangs- bis zum Endzustand existiert.

- Datenfelder: Die in Abschnitt 7.3 beschriebene Methode zum Definieren individueller Eingabefelder kann auch für die Probandeliste angewendet werden. Die Zussammenstellung dieser Felder ist somit erweiterbar gestaltet. Folgende Datenfelder werden fest vorbereitet:
  - Subjekt Number: Dem Clinical Trial Protocol entsprechend unmittelbar nach Unterzeichnung des IC's vergebene Zeichenfolge zur Identifikation des Probanden
  - Random Number: Dem Clinical Trial Protocol bzw. Trial Medication Manual entsprechend vergebene Zeichenfolge, um Dosen der vom Auftraggeber vorbereitete Medikation für den Probanden auszuwählen
  - IC Datum: Datum der Unterzeichnung des IC's durch den Probanden
  - LKF Number: Referenznummer (Zeichenfolge) für Proben des Probanden

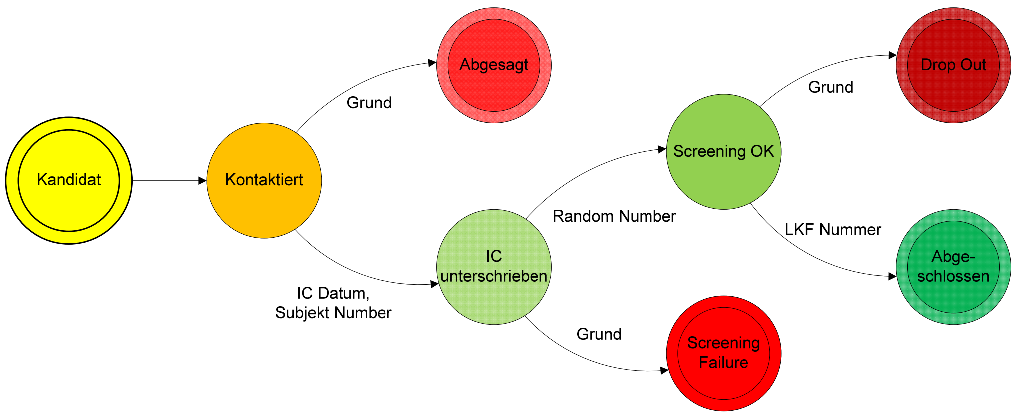


Abbildung 9: Zustände von Probanden der Probandenliste

Um Statuswechsel für die Dokumentation (Enrollment Log, Screening Log) genau aufzuzeichnen wird ein Änderungsprotokoll (Change Log) der Probandenlistentabelle geführt. Neben den zuvor angeführten Attributen der protokollierten Probandenlisten-Datensätze werden ausserdem folgende Felder aufgezeichnet:

- Benutzername, Zeitmarke: Fussabdruck des SMS Benutzers, der die Datenänderung eingegeben hat
- Grund: Freitext zur Eingabe einer Begründung der Änderungen in Textform. Für die in Abbildung 9 mit „Grund" beschrifteten Zustandstransitionen ist die Angabe eines Grundes vorgeschrieben.

Reports dieses Change Logs als Ausgabe in Form einer generierten Excel Datei ergeben die erforderlichen Dokumente:

Enrollment Log: sämtliche Statusänderungen von Probanden auf Zustand „Abgesagt" oder „IC unterschrieben"

Screening Log: sämtliche Statusänderungen von Probanden auf Zustand „Screening Failure" oder „Screeing OK"

Auch die Probandenliste selbst wäre als Excel File exportierbar zu gestalten.

Folgende Zugriffsberechtigungen für Benutzer können identifiziert werden:

- Studien anlegen und bearbeiten erlaubt J/N (vgl. Abschnitt 7.2)
    - Erlaubt nur für Studien der Abteilung des Benutzers, sowie für Studien, deren Projektteam der Benutzer angehört.
    - Erlaubt für Studien sämtlicher Abteilungen („Supervisor“)

    Es werden dadurch folgene Funktionen kontrolliert:

    - Abfrage bearbeiten und ausführen
    - Probandenliste fixieren
    - Einzelnen Proband zu fixierter Probandenliste hinzufügen
    - Gruppenzuteilung bearbeiten
- Probandenstatus ändern erlaubt J/N
    - Gedacht für Studienmitarbeiter, die bei Visiten involviert sind: erlaubt ist die Probandenstatusänderung nur für Probandenlisten von Studien der Abteilung des Benutzers oder für Probandenlisten von Studien, bei denen der Mitarbeiter als Teammitglied vermerkt ist (Abschnitt 7.9).

7.5 Gruppeneinteilung für Probanden und Visitenplan

Die Untersuchungen und Tests an Probanden bei einer klinischen Studie werden unter prüfärztlicher Aufsicht im Rahmen von Visiten durchgeführt. Die Anzahl, Inhalte und Reihenfolge der Visiten einer Studie werden im Protokoll (Clinical Trial Protocol) definiert. Sobald die durchzuführenden Visiten feststehen, können diese für die betreffenden Studie als Liste im SMS eingetragen werden. Pro Visite sind folgende Daten vorgesehen:

- Titel: Bezeichnung der Visite als Freitext, die auf den Inhalt hinweist, z.B.: “Screeningvisite”, “Erste Dosierungsvisite”, “Clamp Visite”, “Abschlussvisite”
- Beschreibung: optionale Beschreibung oder Anmerkung als Freitext, z.B. “Erste Möglichkeit”
- Reihung: eine wählbare Ganzzahl, die zur Reihung der Visiten herangezogen wird. Es wird die um eins erhöhte Anzahl an bislang eingetragenen Visiten voreingestellt.
- Kürzel: kurze Zeichenfolge (z.B.: “V1”), die dann Bestandteil des Visiten-”Tags” im Dienstplan (Abschnitt 7.6) wird. Es wird die um eins erhöhte Anzahl an bislang eingetragenen Visiten mit vorangestelltem Buchstaben “V” voreingestellt.
- Dauer: Vorgesehene Dauer in Stunden
- Kategorie: Kategorie als Freitext, um Visiten zusammenzufassen, z.B. “Screening”,”Erste Behandlungsperiode”,”Abschlussuntersuchung”. Die vergebenen Kategoriebezeichnungen müssen übereinstimmen, damit Visiten unter der gleichen Kategorie aufscheinen.

Probanden werden in Gruppen organisiert. Gruppen haben für eine Studie typischerweise eine einheitliche Größe (z.B. Gruppengröße n=8). Die Gruppenzuteilung eines Probanden erfolgt durch die Probandenliste (Abschnitt 7.4), die Gruppengrößen jedoch nicht berücksichtigt, um flexibler zu bleiben. Pro Gruppe werden dann in weiterer Folge konkrete Termine für die durchzuführenden Visiten festgelegt, und es ergibt sich der Visitenplan. Zunächst können im SMS Gruppen angelegt werden.

Für eine Gruppe können folgende Daten angegeben werden:

- Bezeichnung der Gruppe: Freitext zur Identifikation der Gruppe, z.B. “Gruppe 1”, “Referenzgruppe” usw. Es wird die um eins erhöhte Anzahl an bislang eingetragenen Gruppen mit der vorangestellten Zeichenfolge “Gruppe “ voreingestellt.
- Kürzel: kurze Zeichenfolge (z.B.: “G1”), die dann Bestandteil eines Visiten-”Tags” im Dienstplan wird. Es wird die um eins erhöhte Anzahl an bislang eingetragenen Gruppen mit vorangestelltem Buchstaben “G” voreingestellt.

Sobald Visiten und Gruppen definiert sind, kann der Visitenplan erstellt werden. Jede Gruppe muss dabei alle definierten Visiten durchlaufen. Dies geschieht durch Zuweisen von Terminen pro Gruppenvisite in Form von Zeitspannen. Der Visitenplan entspricht somit einer Liste von Einträgen mit folgenden Attributen:

- Gruppen ID: Referenz auf Probandengruppe
- Visiten ID: Referenz auf Visite
- Start: Startzeitmarke der Visite der Gruppe
- Ende: Endzeitmarke der Visite der Gruppe. Durch die bekannte Dauer der Visite kann die Endzeitmarke automatisch abgeleitet werden. Um Nachbereitungszeiten wie etwa für die Probandenabreise zu berücksichtigen, kann die Endzeitmarke dennoch separat gesetzt werden.

Im Dienstplan wird mit Hilfe des hinterlegten Visitenplans für einen neu angelegenden Dienst die Auswahl der Visite ermöglicht, deren Bestandteil dieser Dienst (u.a.) sein soll. Die Visite einer Gruppe wird dabei in kurzer Form durch den Visiten-„Tag“ dargestellt:

$$< Gruppen - Kürzel >:< Visiten - Kürzel >$$

$$z.B."G1:V1"$$

Nachdem sich ein Dienst im Sinne einer Schicht auf einen Kalendertag bezieht, ist es der bisherigen Praxis entsprechend hilfreich, auf mehrtägige Visiten im Dienstplan hinzuweisen. Der bei einem Dienst eingeblendete Visiten-„Tag“ soll in diesem Fall nicht nur Visiten- und Gruppennummer enthalten, sondern auch den gezählten Kalendertag der mehrtägigen Visite:

$$< Gruppen - Kürzel >:< Visiten - Kürzel >:< Tageszähler >$$

$$z.B.\text{G1:V1:D1}\ (D1 \ldots Day\ 1)$$

Sofern die Gruppeneinteilung der Probanden einer Studie bereits erfolgt ist, kann durch den Visitenplan des Weiteren die Auswahlliste zur Probandenauswahl bei der Raum- und Bettenreservierung (Abschnitt 3.3) für einen gewählten Reservierungszeitraum bestimmt werden.

Folgende Zugriffsberechtigungen für Benutzer können identifiziert werden:

- Studien anlegen und bearbeiten erlaubt J/N (vgl. Abschnitt 7.2):
  - Erlaubt nur für Studien der Abteilung des Benutzers, sowie für Studien, deren Projektteam der Benutzer angehört.
  - Erlaubt für Studien sämtlicher Abteilungen („Supervisor“)

  Es werden dadurch folgene Funktionen kontrolliert:

  - Visiten anlegen und bearbeiten
  - Gruppen anlegen und bearbeiten
  - Visitenplan bearbeiten

7.6 Dienstplan

Der Dienstplan bildet die Einteilung von Mitarbeitern bei klinischen Studien ab. In erster Linie dabei dabei die umfangreichen, in Schichtdiensten organisierten Tätigkeiten im Rahmen von Visiten im Fokus.

Der bislang durch Google Calendar unterstützte Ablauf soll mit der SMS Software wie folgt aussehen:

1. Der Prüfarzt hat folgende Vorkehrungen bereits getroffen:
   - Visiten der Studie definieren
   - Probandengruppen anlegen
   - Zeitlichen Ablauf der durchzuführenden Visiten durch Visitenplan festlegen
2. Für die Tage, über die sich Visiten erstrecken muss überlegt werden, welche Tätigkeiten anfallen, um erforderliche Dienste zu identifizieren. Für jeden Kalendertag werden diese vorgesehen Dienste vom Prüfarzt angelegt. Dies erfolgt für einen bevorstehenden Zeitraum, je nach Planbarkeit z.B. wiederholt jede Woche, monatlich oder einmalig für alle Visiten der Studie.
3. In der Dienstplanansicht sind die vorbereiteten, zu besetzenden Dienste ersichtlich. Benutzer können sich nun selbst für Dienste eintragen (Abbildung 10).
4. Der Prüfarzt kann z.B. vor Beginn einer Visite oder zu Wochenbeginn einsehen, ob alle Dienste zugeteilt sind, und verbliebende Dienste Mitarbeitern ausdrücklich zuteilen. Um die Auswahl eines Mitarbeiters für einen Dienst fair bzw. entsprechend seines Anstellungsverhältnisses zu gestalten, kann eine Übersicht eingeblendet werden, in der bislang geleistete Stunden anhand vergangener Diensteinteilungen summiert dargestellt sind. Die Stundensummen sind dabei für jede laufende Studie getrennt angegeben. Es kann auf folgende, vergangene Zeitträume eingeschränkt werden (Abbildung 10):
   - Kalendertag
   - Kalenderwoche
   - Monat

   Der Prüfarzt bzw. Studienleiter kann dadurch auch Aspekte der „Kostenwidmung“ einfliessen lassen.

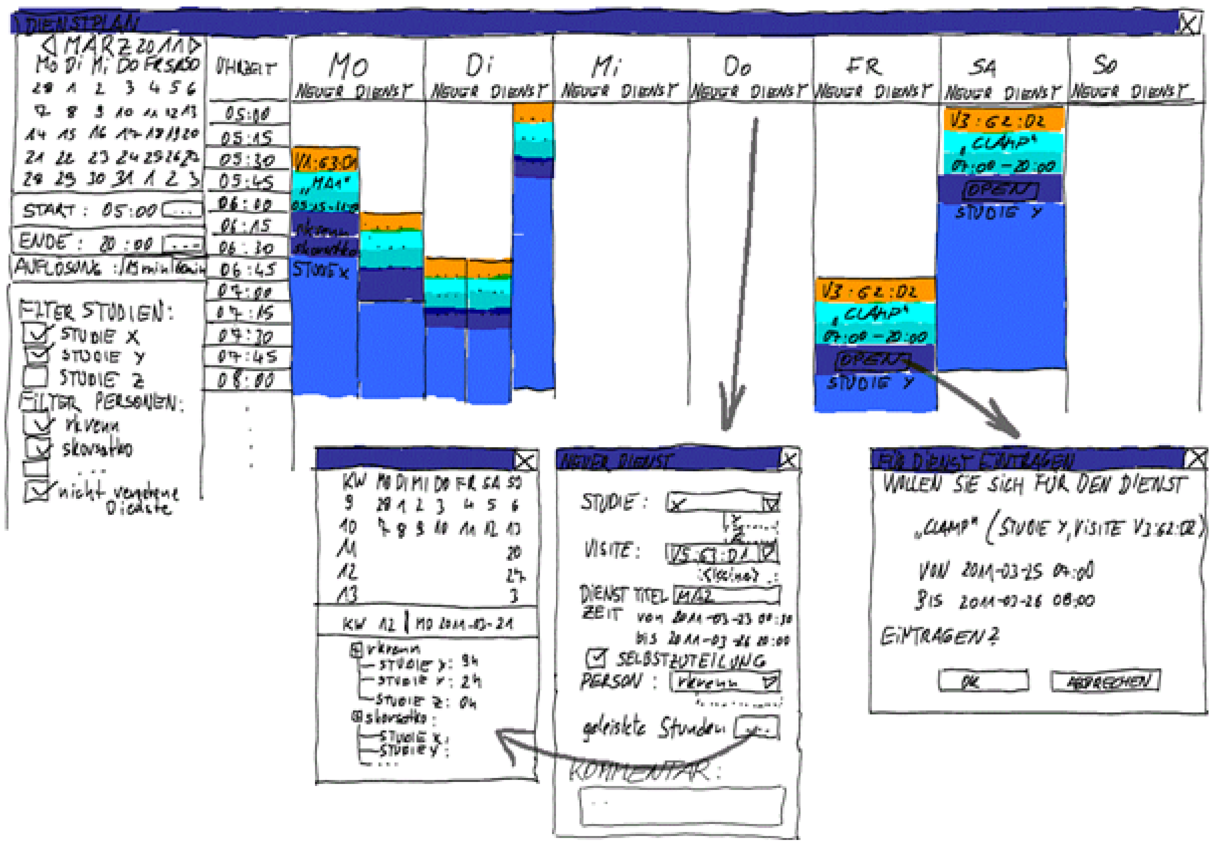


Abbildung 10: Dienstplan

Die graphische Darstellung des Dienstplans ist als Kalenderansicht für eine einzelne Kalenderwoche gedacht:

- Kalendertage horizontal (als Spalten)
- Uhrzeit vertikal, evtl. mit umschaltbarer Auflösung (15 Min., 30 Min., 1h)
- Ein Dienst wird als Zeitspanne eingezeichnet. Zeitlich überlappende Dienste eines Tages werden nebeneinander angeordnet.
- Es kann nach Studien und Benutzern gefiltert werden.

Ein Dienst umfasst die foglenden Attribute:

- Start-, Endzeitmarke: Zeitspanne des Dienstes
- Titel: Freitext zur Bezeichnung des Dienstes bzw. der Schicht. Typischer Weise werden Kurzformen verwendet („MA1“ – „Dienst Mitarbeiter 1“).
- Studien ID: Referenz auf die Studie
- Visiten-„Tag“: Identifikation der Visite, Probandengruppe und laufender Tag der Visite (siehe Abschnitt 7.5)
- Selbstzuteilung möglich: Markierung, ob der Dienst zur Selbstzuteilung durch Benutzer freigegeben ist.
- Kommentar: Freitext mit Erläuterungen oder besonderen Hinweisen zum Dienst.
- Mitarbeiter: Referenz auf Personaleintrag. Damit ist auch die Zuteilung von Personen möglich, die nicht über ein SMS Benutzerkonto verfügen.

- Nacht-/Wochenend-/Feiertagdienst: Diese Markierung berücksichtigt eine besondere Gewichtung des Dienstes durch eine Addition von 3h in berechneten Stundensummen. Beim Anlegen des Dienste wird diese als CheckBox ausgeführte Eingabe entsprechend des Datums voreingestellt. Nachtdienste könnten mit gleichermassen voreingestellter Maske eröffnet werden, sofern die Überlappung der Zeitspanne des Dienstes mit der Zeitspanne von 22:00 bis 06:00 Uhr z.B. mehr als 4h oder 6h ausmacht.

Für die vorgesehen Funktion der Selbstzuteilung von Diensten müssen zwei Zugriffsberechtigungen unterschieden werden:

- Studien anlegen und bearbeiten erlaubt J/N (vgl. Abschnitt 7.2):
  - Erlaubt nur für Studien der Abteilung des Benutzers, sowie für Studien, deren Projektteam der Benutzer angehört.
  - Erlaubt für Studien sämtlicher Abteilungen („Supervisor“)

  Es werden dadurch folgene Funktionen kontrolliert:

  - Dienst anlegen und bearbeiten
- Für Dienst eintragen erlaubt J/N:
  - Benutzer mit dieser Berechtigung sehen die Dienste der Studien ihrer jeweiligen Abteilung sowie Dienste von Studien, deren Projekteam sie angehören (Abschnitt 7.9).

7.7 Checklisten

Um den Fortschritt von diversen weiteren Arbeitsschritten pro Proband einer Studie zu überblicken, an denen verschiedene Mitarbeiter beteiligt sind, sollen im SMS einfache „Checklisten“ vorgesehen werden. Es würden sich dafür z.B. folgende Anwendungsfälle anbieten:

- CRF (Case Report Form) Erstellung: Erst nachdem sämtliche Visiten eines Probanden abgeschlossen sind, soll die Zusammenfassung der Daten des Probanden (Subject) im CRF erfolgen. Generell kann die Vollständigkeit des CRF Dokuments überblickt werden (vgl. „Probandenübersicht“).
- Kontaktaufnahmen, An- und Abreise, IC Unterschriften, Patiententagebücher von Probanden überblicken
- Probenentnahme, Dosierungszeiten, Probenversandt, Messvorgänge überblicken
- Schritte der Abwicklung von Aufwandsentschädigungen und Hotelrechnungen der Probanden überblicken

Nachdem die Probandenliste fixiert wurde, können für eine Studie je nach Bedarf beliebig viele Checklisten angelegt werden. Eine Checkliste stellt die Probanden der Probandenliste frei definierbaren „Tasks“ gegenüber, sodass eine Tabelle entsteht. Jede Zelle der Tabelle kann dann von involvierten Mitarbeitern im Laufe der fortwährende Bearbeitung mit einem (vorläufig) zweiwertigen Wert belegt werden: OK oder NOK (Abbildung 11). Der Standardwert ist NOK, der Wert kann in der Darstellung in der SMS Weboberfläche durch Mausklick umgeschaltet werden. Jede Umschaltung kann per Änderungsprotkoll auch aufgezeichnet werden.

Checkliste „CRF ausgefüllt“	Tasks					
Proband	V1 Screening	V2 Day 1	V2 Day 2	V3 Day 1	V3 Day 2	V4 Abschluss
Max Mustermann	OK	OK	OK	OK	OK	NOK
John Smith	OK	OK	OK	OK	OK	NOK
John Doe	NOK	OK	OK	OK	OK	NOK
...	OK	OK	OK	OK	NOK	NOK
...	OK	OK	OK	OK	NOK	NOK
...	OK	OK	OK	OK	NOK	NOK
...	OK	NOK	OK	OK	NOK	NOK
...	OK	NOK	OK	OK	NOK	NOK
...	OK	OK	OK	OK	NOK	NOK

Checkliste „Preconditions“	Tasks			
Proband	C-Peptid	AB	Endo TP1	Endo TP2
Max Mustermann	OK	OK	OK	OK
John Smith	OK	OK	OK	OK
John Doe	NOK	OK	OK	OK
...	OK	OK	OK	OK
...	OK	OK	OK	OK
...	OK	OK	OK	OK
...	OK	NOK	OK	OK
...	OK	NOK	OK	OK
...	OK	OK	OK	OK

Abbildung 11: Checklisten

Neben den Namen von Probanden könnten in der vertikalen Spalte auch weitere Informationen eingeblendet werden, z.B. Felder der Probandenliste wie Probandengruppe, Probandenstatus oder Subject Number (siehe Abschnitt 7.4). Eine Checkliste umfasst im Wesentlichen folgende Attribute, die beim Anlegen oder nachträglichem Bearbeiten festgelegt werden können:

- Titel (Freitext)
- Tasks: Liste von Freitexten, die abzuhakenden Aufgaben bezeichnen.

Folgende Zugriffsberechtigungen für Benutzer können identifiziert werden:

- Studien anlegen und bearbeiten erlaubt J/N (vgl. Abschnitt 7.9)
  - Erlaubt nur für Studien der Abteilung des Benutzers, sowie für Studien, deren Projektteam der Benutzer angehört.
  - Erlaubt für Studien sämtlicher Abteilungen („Supervisor“)

  Es werden dadurch folgene Funktionen kontrolliert:

  - Checklisten anlegen und bearbeiten
- Checklistenzellen abhaken erlaubt J/N
  - Gedacht für Studienmitarbeiter, die bei diversen Studienprozessen involviert sind: erlaubt ist das Abhaken von Checklisten von Studien der Abteilung des Benutzers oder für Checklisten von Studien, bei denen der Mitarbeiter als Teammitglied vermerkt ist (Abschnitt 7.9).

7.8 Dokumentverwaltung

Im Laufe einer klinischen Studie entsteht eine umfassende Palette an formalen Dokumenten:

- Einreichungen
- Bestätigungen
- Beschreibungen
- Vorlagen
- Logs
- Studienresultate

Diese Dokumente liegen in elektronischer Form als Dateien vor bzw. werden von den Mitarbeitern mit den entsprechenden Tools erstellt (Tabelle 4).

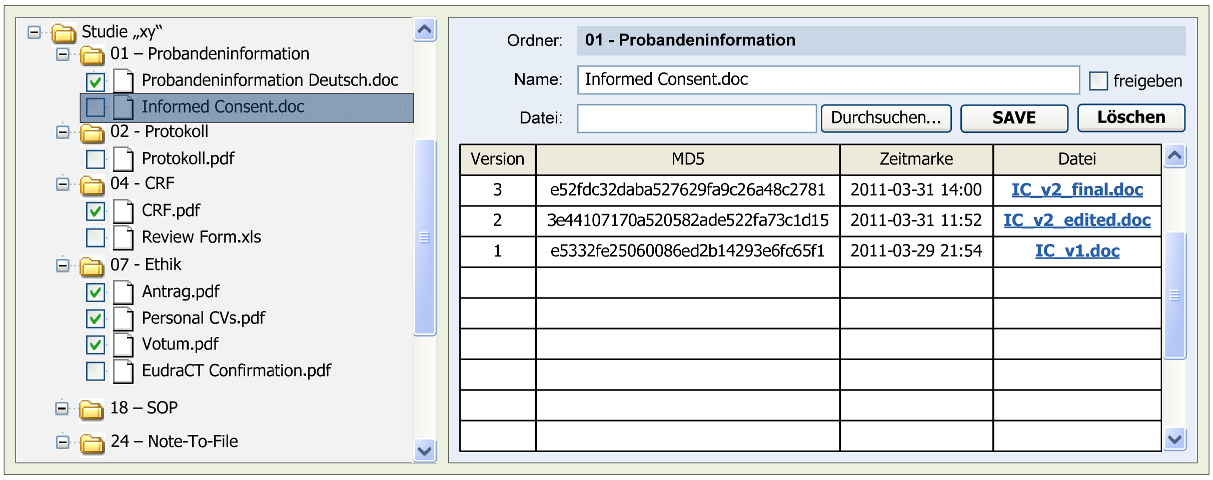
Anwendung	Inhalt der Dateien
Microsoft Excel	application/msexcel, application/vnd.ms-excel, application/vnd.openxmlformats-officedocument.spreadsheetml.sheet, text/csv
Microsoft Word	application/mspowerpoint, application/vnd.ms-powerpoint, application/vnd.openxmlformats-officedocument.presentationml.presentation
Microsoft Powerpoint	application/msword, application/vnd.openxmlformats-officedocument.wordprocessingml.document
Adobe PDF	application/pdf
Bilder/Graphiken	image/jpeg, image/pjpeg, image/bmp, image/gif, image/png, image/tiff

Tabelle 4: Dateitypen zur Dokumentablage

Die Ablage der Dokumente wird per Dateisystem bewerkstelligt, welches als (Microsoft Windows) CIFS Fileshare über das Netzwerk der Organisation den Benutzern mit Windows Benutzerkonten zugänglich gemacht wird. Der Bestand von mehreren hundert Dateien einer Studie wird in eine Ordnerstruktur einsortiert, die für jede Studie eingerichtet wird. Die Ordner sind nummeriert und mit der Bezeichnung des Prozesses beschriftet, dem die Dateien zuzuordnen sind. Die Dokumentversionierung orientiert sich an entsprechenden Beschriftungen in den Dateinamen. Das Problem dieses Ist-Zustandes ist die fehlende Konsistenz der Konventionen, vor allem aber die Unsichheit der Versionierung.

Die Dokumentverwaltung des SMS soll nun eine Möglichkeit bieten, ausgewählte Dateien in der Datenbank abzuspeichern. Dadurch sollen sich gegenüber dem weiterhin bestehen bleibenden Dateisystem folgende Verbesserungen ergeben:

- Benutzer sollen unmissverständlich nur die freigegebene, aktuelle Version von Dokumenten abrufen können
- Dokumente einer abgeschlossenen Studie können als Bestandteil einer digitalen Signatur nicht mehr manipuliert werden, ohne eine neuerliche Signierung durchführen zu müssen, und damit die Zeitmarke zu verändern (Abschnitt 5.4)



- Studie „xy“
  - 01 – Probandeninformation
    - ✓ Probandeninformation Deutsch.doc
    - ☐ Informed Consent.doc
  - 02 - Protokoll
    - ☐ Protokoll.pdf
  - 04 - CRF
    - ✓ CRF.pdf
    - ☐ Review Form.xls
  - 07 - Ethik
    - ✓ Antrag.pdf
    - ✓ Personal CVs.pdf
    - ✓ Votum.pdf
    - ☐ EudraCT Confirmation.pdf
  - 18 – SOP
  - 24 – Note-To-File

Ordner: **01 - Probandeninformation**
Name: Informed Consent.doc ☐ freigeben
Datei:

Version	MD5	Zeitmarke	Datei
3	e52fdc32daba527629fa9c26a48c2781	2011-03-31 14:00	IC_v2_final.doc
2	3e44107170a520582ade522fa73c1d15	2011-03-31 11:52	IC_v2_edited.doc
1	e5332fe25060086ed2b14293e6fc65f1	2011-03-29 21:54	IC_v1.doc

Abbildung 12: Verwaltung für Studiendokumente

Um eine Verwaltung wie in Abbildung 12 zu erreichen, sind für hinterlegte Dateien Datensätze mit folgenden Attributen vorzusehen:

- Studie: Referenz auf den Studieneintrag, zu dem die Datei gehört
- Ordner: Um eine von Dateisystemen bekannte Darstellung zu ermöglichen, erhalten abgelegte Dateien eine Kategorie (z.B. „01-Probandeninformation“) aus einer vordefinierte Liste zugeordnet. Diese Kategorien sollen statisch festgelegt werden und ergeben den Verzeichnisbaum. Verschachtelte (tiefere) Hierarchien sind nicht vorgesehen.
- Name: virtueller Name der Datei, unter dem die Datei den anderen Benutzern zum Download präsentiert wird.
- Freigabe: Markierung zur Steuerung, ob die Datei für Benutzer sichtbar sein soll, die nur über die Leseberechtigung verfügen.
- Dateiinformationen: Details zur Datei werden hinterlegt:
  - Ursprünglicher Dateipfad der Datei auf dem Host-Computer, durch den der Upload erfolgte
  - Größe, MD5 Hash-Wert zur eindeutigen Identifikation des Inhalts
  - Typ des Inhalts (MIME Type)

  Es ist noch abzuwägen, ob der Inhalt der Datei selbst in ein Datenbankfeld kommen soll, oder ob eine Ablage im Dateisystem des SMS Servers günstiger wäre. Die durchschnittliche Größe der Dateien einer Studie bewegt sich zwischen 0.5MB und 2MB.

  Durch den MD5 Hash-Wert der Datei kann für die in Frage kommende Ablage im Dateisystem des Servers zuverlässig ein lokaler Dateiname abgeleitet werden, und mehrfache Speicherung gleicher Inhalte ausgeschlossen werden. Auch das Hochladen eines unveränderten Dokuments kann abgefangen werden.

- Kommentar: optionaler Freitext, vorwiegend als Memo für die Versionierung

Ein Änderungsprotokoll (Change Log) würde auf einfache Weise den angestrebten Versionierungsmechanismus bieten.

Folgende Zugriffsberechtigungen für Benutzer können identifiziert werden:

- Dateien hochladen, löschen, Kommentar und Freigabe bearbeiten erlaubt J/N
- Dateien einsehen erlaubt J/N

Widerum ist jeweils eine Verfeinerung dieser Berechtigungen zur Trennung von Abteilungen vorgesehen:

- Erlaubt nur für Dateien von Studien der Abteilung des Benutzers, sowie für Studien, deren Projektteam der Benutzer angehört.
- Erlaubt für Dateien von Studien sämtlicher Abteilungen („Supervisor“)

7.9 Ablage und Verwaltung

Um die einzelnen, zuvor beschriebenen Komponenten der Studiendatenbank nutzen können, muss eine Studie selbst als übergeordente Datenstruktur dienen, die angelegt und bearbeitet werden kann. Dieser Abschnitt beschreibt entsprechend den übrigen Modulen des SMS die Ablage und Verwaltung einer Studie, die als Studieneintrag abgebildet wird. Dieses komplexe Datenobjekt enthält zunächst im Groben die für Relationen innerhalb der SMS Datenbank relevanten Referenzen (freibleibend):

- Liste von Referenzen auf ausgewählte Pre-Screening Fragen und Probandendateneingabefelder
- Referenz auf eine Probandenabfrage
- Referenzen auf eine fixierte Probandenliste
- Liste von Referenzen auf Visiten der Studie (Abschnitt 7.5)
- Liste von Referenzen auf Probandengruppen der Studie (Abschnitt 7.5)
- Liste von Referenzen auf Gruppenvisitentermine (Abschnitt 7.5)
- Referenzen auf angelegte Checklisten

Einträge der Dokumentverwaltung für Dateien sowie definierte Dienste des Dienstplans (Abschnitt 7.6) referenzieren Studien ihrerseits (freibleibend). Des weiteren sind folgende Attribute für die vorgeshenen Funktionen erforderlich:

- Start- und Enddatum der Studie (Projekt-Timelines, Abschnitt 7.2)
- Abteilung: Referenz auf die Abteilung, der die Studie zuzuordnen ist (Abschnitt 4.3). Verschiedene Zugriffsberechtigungen für Funktionen der Studiendatenbank stützen sich auf die Abteilung der Studie. Im Vordergrund bleibt jedoch die Festlegung auf den jeweiligen Probandenstamm, der für die Probandenlisten herangezogen werden soll (Abschnitt 7.4, 8.1). Nachdem die Probandenliste fixiert ist, sollte die nachträgliche Änderung der Abteilung einer Studie in der SMS Oberfläche ausgeschlossen werden.

- Status: Status des Studienprojektes
  - zeitlich fixiert (Abschnitt 7.2)
  - zeitlich noch nicht fixiert, aber geplant (Abschnitt 7.2)
  - Projekte in Verhandlung (Abschnitt 7.2)
  - Lockdown: Das Projekt ist in der SMS GUI für Bearbeitungen gesperrt. Konkret sind folgende Komponenten für alle Benutzer im readonly-Modus:
    - Probandenliste (inkl. Change Log)
    - Gruppeneinteilung und Visitenplan
    - Checklisten
    - Dokumentverwaltung

    Der Lockdown Status könnte noch weiter unterteilt werden:

    - Studie abgeschlossen
    - Studie abgebrochen
    - Studie unterbrochen
  - Signiert: Die Lockdown Zustände könnten doppelt ausgeführt werden, um dem Administrator der Studiendatenbank noch Spielraum vor dem endgültigen, unwiederruflichen Signieren der Studie einzuräumen:
    - Studie abgeschlossen und signiert
    - Studie abgebrochen und signiert
    - Studie unterbrochen und signiert

    Wechselt ein authorisierter Benutzer den Status der Studie auf einen der „signiert"-Zustände, wird die in Abschnitt 5.4 beschriebene Methode zum Signieren von Datenbankinhalten ausgelöst. Damit die Signatur ihren Zweck erfüllt, muss die Zeitmarke sowie der ausführende Benutzer der Signierung aufgezeichnet und Bestandteil der signierten Daten sein. Diese Daten können durch Führung eines Änderungsprotokolls (Change Log) für den Studienstatus bewerkstelligt werden.

- Signatur: Datenfeld für Signatur des Studieneintrags und referenzierender Datensätze anderer Tabellen, die vor nachträglicher Veränderung geschützt werden müssen (Abbildung 2). In die Signaturberechnung einbezogen werden so weit folgende Inhalte:
  - die beim Lockdown als gesperrt beschriebenen Daten
  - Change Log des Studienstatus
  - Studieneintrag selbst

Zusätzlich kann eine Auswahl grundlegender Angaben zur Studie durch weitere Attribute erfasst werden:

- Projekt-Code: kurze, eindeutige Zeichenfolge zur Identifikation der Studie
- Projekt-Titel (kurz): beschreibende Bezeichnung der Studie als Freitext
- Projekt-Titel: optionaler, erweiterter Titel (Freitext)
- Beschreibung: Freitext zur optionalen Aufnahme einer kurzen Beschreibung der Studie (vgl. „Abstract")
- PI: Principal Investigator (Liste von Refrenzen auf Personal- bzw. Kontaktdateneinträge)
- PAG: Projektauftraggeber (Liste von Referenzen auf Personal- bzw. Kontaktdateneinträge)
- Projektleiter: Liste von Refrenzen auf Personal- bzw. Kontaktdateneinträge

- Verantwortlicher Mediziner: Liste von Refrenzen auf Personal- bzw. Kontaktdateneinträge
- Projektteam: Liste von Refrenzen auf Personal- bzw. Kontaktdateneinträge. Das Projekteam dient neben dem Aufzeichnungzweck der verfeinerten Kontrolle der diversen angeführten Zugriffsberechtigungen der Studiendatenbank. Somit ist die vollständige Anführung aller beteiligten Personen unumgänglich. Das Projektteam muss auch die Personalreferenzen für PI, PAG, Projektleiter und verantwortlicher Mediziner beinhalten, sofern diese Personen einen Zugang zum SMS System mit entsprechenden Berechtigungen nutzen wollen.
- Ethik Nummer: Freitext für Referenznummer der Ethik-Kommission
- Ethik Votum Gültigkeit: Datum, an dem das Votum der Ethik-Kommission abläuft. Nachdem für gewisse Studien kein Ethik-Votum erforderlich ist (z.B. spezielle Gerätetests oder Studien ohne Probanden), wäre die Überlegung, dieses Datum nur als optionale Deadline in der Projekt-Timeline einzutragen.
- EudraCT Nummer: Freitext für Referenznummer der EudraCT Meldung
- IA (Interne Auftragsnummer), ZMF Code: Freitextfelder

Ein Änderungsprotokoll für diese Felder kann prinzipiell in das Change Log des Studienstatus aufgenommen werden. Dies wäre erforderlich, wenn ein weitereres formelles Protokoll als abrufbarer Report im SMS umzusetzen ist, das z.B. Änderungen der Zusammenstellung des Projektteams oder Zeitmarken des Erhalts von Ethik-Votum etc. dokumentiert.

Die Präsentation eines Studieneintrags in der SMS Oberfläche wäre als einseitiges „Studienstammblatt" angedacht. Zusätzlich ist die Bereitstellung der Informationen des Studienstammblatts in druckbarer Form (generiertes PDF-Dokument) gewünscht.

Folgende Zugriffsberechtigungen für Benutzer können identifiziert werden:

- Studien anlegen und bearbeiten erlaubt J/N

Diese Berechtigung schliesst auch Studienstatusänderungen ein und damit implizit das Sperren bzw. Signieren von abgeschlossenen Studien. Dies entspricht somit auch einer „Zeichnungsberechtigung", über die typischerweise nur der Studienleiter verfügen sollte.

7.10 Datenbestand und Ausgangssituation

Es besteht die Möglichkeit, laufende und vergangene klinische Studien in einem gewissen Ausmass in die Studiendatenbank des SMS nachzutragen, um z.B. die Projekt-Timelines des laufenden Kalenderjahres für Planungszwecke bereits sinnvoll nutzen zu können.

Das Einspielen sämtlicher Dokumente vergangener Studien ist hingegen ein aufwändiger Vorgang. Die Idee des einmaligen Imports des Filesystems, das zur Ablage sämtlicher Dateien zu Studien dient, wurde verworfen, weil diese kaum automatisch identifizierbar sind. Zudem sind darunter zahlreiche Dateien, die häufig und von verschiedenen Benutzern editiert werden oder temporäre Arbeitsdokumente darstellen.

Folgende Übersicht zeigt die bisher zur Unterstützung eingesetzten Mittel:

Hilfsmittel/Arbeitsgrundlage	Bisherige Softwareunterstützung	SMS Studiendatenbank
Projekt-Zeitplan	Microsoft Excel Spreadsheet	Projekt-Timelines
Projektübersicht	Microsoft Excel Spreadsheet	Ablage und Verwaltung
Dienstplan	Google Calendar	Dienstplan
Probanden-Übersicht	Microsoft Excel Spreadsheet	Probandenliste
Probanden-Termine	Microsoft Excel Spreadsheet	Visitenpan
Pre-Screening, Protokoll für telefonische Datenerhebung	Microsoft Excel Spreadsheet, Microsoft Word (Formulare / Druckvorlagen)	Probanden Pre-Screening Fragen und studienspezifische Probandendaten
Enrollment Log, Screening Log	Excel Spreadsheets	Änderungsprotokoll der Probandenlsite
Dokumentablage	Windows CIFS Fileshare	Dokumentverwaltung

Tabelle 5: Hilfsmittel und Arbeitsgrundlagen bei der Studienabwicklung

8 Probandendatenbank

8.1 Übersicht

Klinische Studien werden in den meisten Fällen an Personen durchgeführt. Diese freiwilligen Probanden werden ursprünglich im Laufe von Rekrutierungsphasen bei Studien vom CRC angeworben und erfasst. Der Ablauf der Probandenerfassung ist an Einverständniserklärung zur Speicherung von Daten gebunden und in Abbildung 13 umrissen.

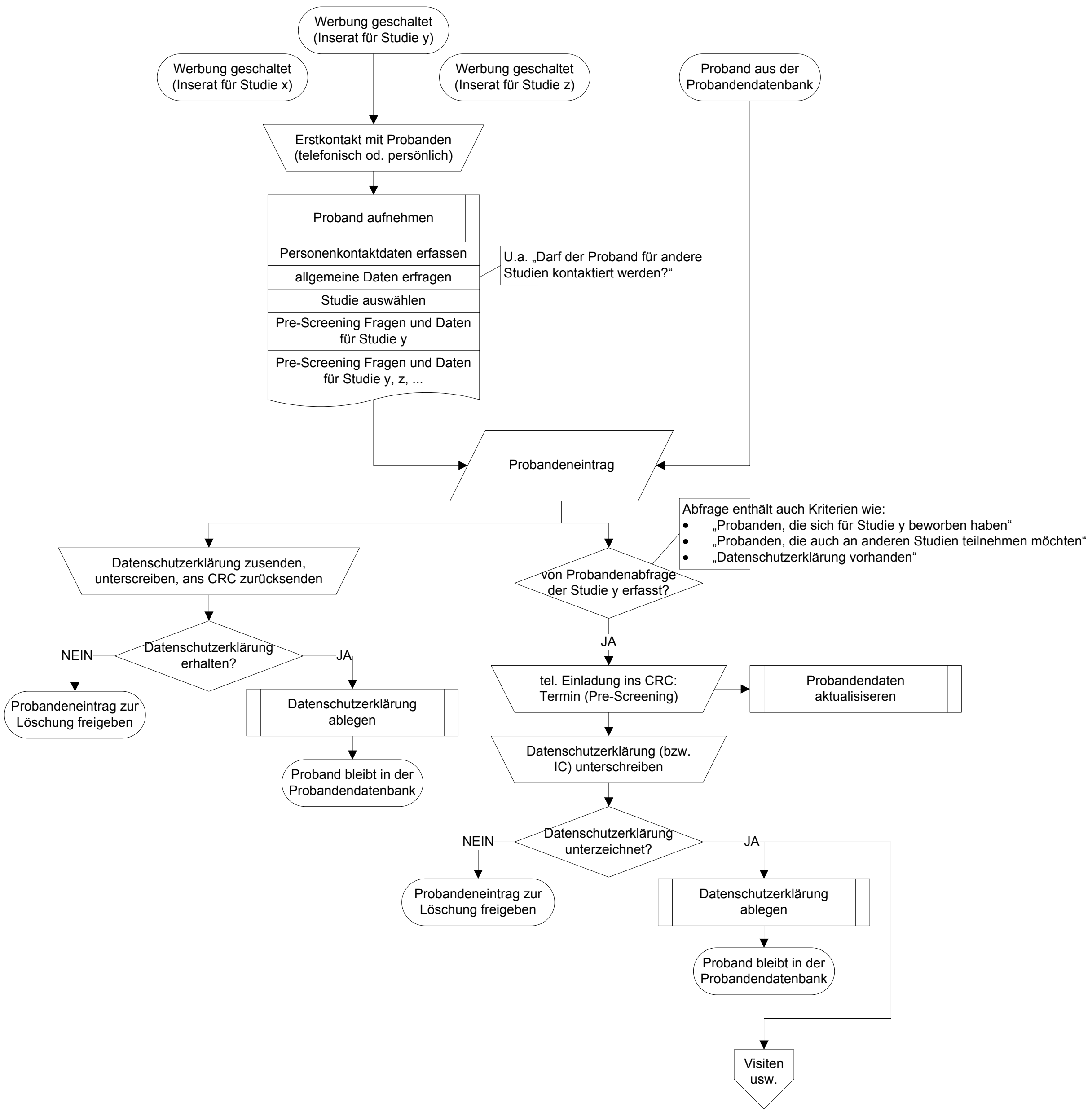

Werbung geschaltet (Inserat für Studie y)
Werbung geschaltet (Inserat für Studie x)
Werbung geschaltet (Inserat für Studie z)
Proband aus der Probandendatenbank
Erstkontakt mit Probanden (telefonisch od. persönlich)
Proband aufnehmen
Personenkontaktdaten erfassen
allgemeine Daten erfragen
Studie auswählen
Pre-Screening Fragen und Daten für Studie y
Pre-Screening Fragen und Daten für Studie y, z, ...
U.a. „Darf der Proband für andere Studien kontaktiert werden?“
Probandeneintrag
Abfrage enthält auch Kriterien wie:
„Probanden, die sich für Studie y beworben haben“
„Probanden, die auch an anderen Studien teilnehmen möchten“
„Datenschutzerklärung vorhanden“
Datenschutzerklärung zusenden, unterscreiben, ans CRC zurücksenden
von Probandenabfrage der Studie y erfasst?
Datenschutzerklärung erhalten?
NEIN
JA
Probandeneintrag zur Löschung freigeben
Datenschutzerklärung ablegen
Proband bleibt in der Probandendatenbank
JA
tel. Einladung ins CRC: Termin (Pre-Screening)
Probandendaten aktualisiseren
Datenschutzerklärung (bzw. IC) unterschreiben
Datenschutzerklärung unterzeichnet?
NEIN
JA
Probandeneintrag zur Löschung freigeben
Datenschutzerklärung ablegen
Proband bleibt in der Probandendatenbank
Visiten usw.

Abbildung 13: Erfassung von Probanden

Das SMS soll durch die Probandendatenbank den entstehenden Probandendatenstamm beherbergen und Benutzern zugänglich machen. Eine wesentliche Verflechtung der Probandendatenbank innerhalb des SMS ist die beim Aufsetzen einer Studie durchgeführte Probandenabfrage gegen die Probandendatenbank mit anschliessender Fixierung der Probandenliste (Abschnitt 7.4).

Bei der Probandenauswahl für Studien wird oft auf Probanden zurückgegriffen, die bereits an Studien teilgenommen haben, entweder wegen der speziellen Eignung von Probanden oder um die für die Studie erforderliche Anzahl zu erreichen. Deshalb ist ein stetiger Zuwachs der Probandeneinträge angestrebt. Nicht zuletzt wird durch einen massiven Probandenstamm die Konkurrenzfähigkeit des CRCs z.B. hinsichtlich des Umfangs bewältigbarer Studienprojekte gesteigert.

Die Probandendatenbank selbst soll folgende Funktionen kapseln, die sich auf die Ablage und Verwaltung der Probandeneinträge beschränken:

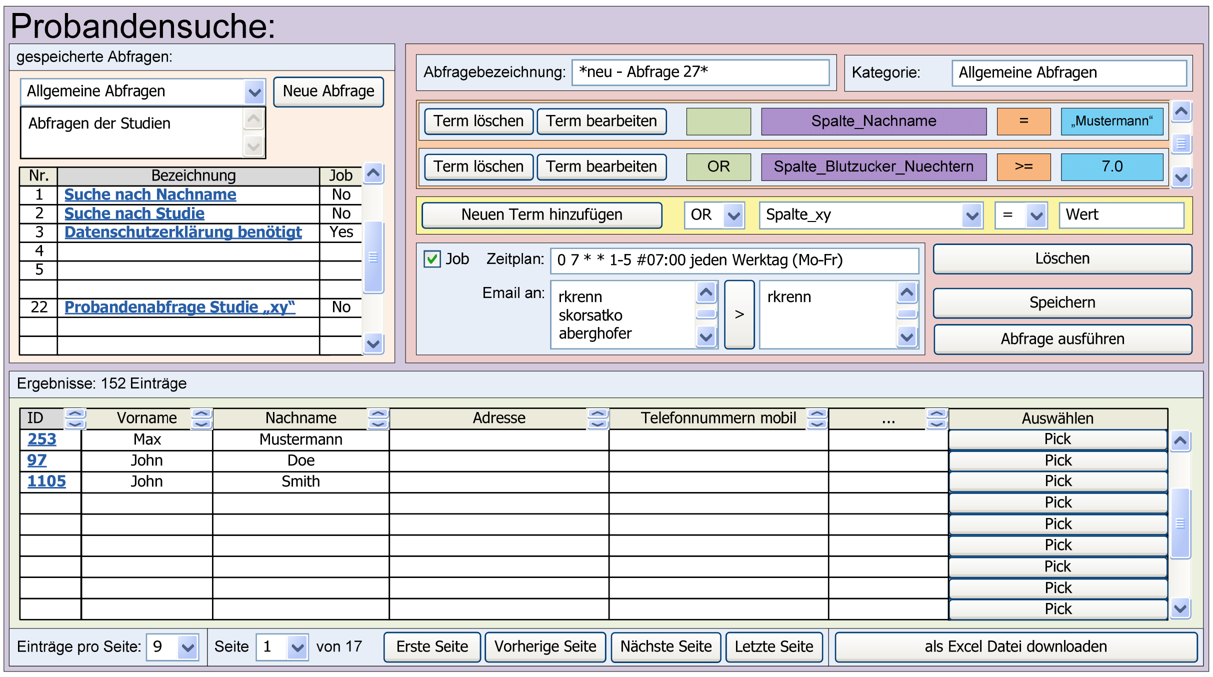
- Probandensuche
- Probandenstammblatt

Der Probandenstamm einer Abteilung soll von denen anderer Abteilungen als getrennt erscheinen, um der Vertraulichkeit und Geheimhaltung dieser Daten ultimativ Rechnung zu tragen. Die Einsicht eines Benutzers in die Probandeninformationen anderer Abteilungen ist somit weitestgehend einzuschränken. Es soll dadurch auch die technisch einfache Möglichkeit bestehen, einen exklusiven SMS Datenbankauszug des Probandenstamms einer Abteilung bereitstellen zu können. Im Endeffekt wird damit auch das mehrfache Vorkommen eines Probanden in der Probandendatenbank in Kauf genommen. Jede Abteilung ist somit auch selbst verantwortlich, für den Ausschluss eines Probanden bei der Rekrutierung wegen zeitgleicher Teilnahme an einer anderen Studie (auch einer anderen Abteilung) zu sorgen.

Nachdem ein Hauptaugenmerk auf die unmittelbare Verwendung der SMS Weboberfläche während der persönlichen oder telefonischen Aufnahme von Probandendaten liegt, wird in den Beschreibungen bereits primär auf die geplante Gestaltung der Oberflächen eingegangen.

8.2 Probandensuche

Ein einzelnes, funktionales Suchformular (Abbildung 14) soll flexible Suchabfragen ermöglichen.



Probandensuche:

gespeicherte Abfragen: Allgemeine Abfragen | Neue Abfrage

Abfragen der Studien

Nr.	Bezeichnung	Job
1	Suche nach Nachname	No
2	Suche nach Studie	No
3	Datenschutzerklärung benötigt	Yes
4		
5		
22	Probandenabfrage Studie „xy“	No

Abfragebezeichnung: *neu - Abfrage 27* Kategorie: Allgemeine Abfragen

Term löschen	Term bearbeiten		Spalte_Nachname	=	„Mustermann“
Term löschen	Term bearbeiten	OR	Spalte_Blutzucker_Nuechtern	>=	7.0
Neuen Term hinzufügen		OR	Spalte_xy	=	Wert

✓ Job Zeitplan: 0 7 * * 1-5 #07:00 jeden Werktag (Mo-Fr)

Email an: rkrenn, skorsatko, aberghofer > rkrenn

Löschen | Speichern | Abfrage ausführen

Ergebnisse: 152 Einträge

ID	Vorname	Nachname	Adresse	Telefonnummern mobil	...	Auswählen
253	Max	Mustermann				Pick
97	John	Doe				Pick
1105	John	Smith				Pick
						Pick
						Pick
						Pick
						Pick
						Pick
						Pick

Einträge pro Seite: 9 | Seite 1 von 17 | Erste Seite | Vorherige Seite | Nächste Seite | Letzte Seite | als Excel Datei downloaden

Abbildung 14: Probandensuche

Wie in Abschnitt 7.3 können sich die Kriterien auch auf Eingaben sämtlicher dynamischer Eingabefelder (Probanden Pre-Screening Fragen und studienspezifischen Probandendaten) beziehen. Zu diesem Zweck kommt wieder das Konzept des "SQL Where Clause Builders" zum Tragen (vgl. Abbildung 7). Nachdem die Eingabe der Abfragekriterien dabei z.B. während eines Telefonats zu langwierig sein wird, können gebräuchliche Abfragen gespeichert bzw. vorbereitet werden. Gespeicherte Abfragen können schnell aufgerufen werden, sodass dann nur mehr das Literal des Kriteriums eingegeben werden muss (z.B. „Mustermann“ im Term für den Nachnamen). Abfragen sehen intern immer eine fixe Filterung nach Probanden der Abteilung des Benutzers vor, der die Suchabfrage ausführt. Dadurch wird die Trennung der Probandenstämme von Abteilungen realisiert.

Nach der Ausführung der Abfrage wird die Ergebnisliste als paginierte Tabelle präsentiert. Es liegt der Wunsch vor, eine Exportmöglichkeit der Abfrageergebnisse als Excel Spreadsheet vorzusehen. Vorerst nicht näher ausgeführt wird eine ebenfalls an dieser Stelle anzusiedelnde Funktion zum Abruf von Serienbriefen an gelistete Probanden als generiertes PDF-Dokument. Die „Pick“-Schaltfläche jedes Eintrags der Ergebnisliste realisiert die Funktion des in Abschnitt 3.3 und 7.4 erwähnten Probanden-„Pickers“.

Ein optionales Feature wäre die automatische Ausführung vordefinierter Abfragen (als „Job“), deren Resultate dann per E-Mail in Form von Excel-Dateiattachments ausgewählten SMS Benutzern zugesendet werden. Der Zeitplan für die Ausführung könnte wie in Abbildung 14 dargestellt im selben Format erfolgen, wie es bei Konfigurationsdateien („crontabs“) des UNIX Ablaufplanungs-Programms „cron“ Anwendung findet. Dadurch ist die einfache und umfassende Angabe von periodisch wiederholten Ausführungen der Abfrage möglich. Einige Anwendungsszenarien für diese Funktion wären z.B.:

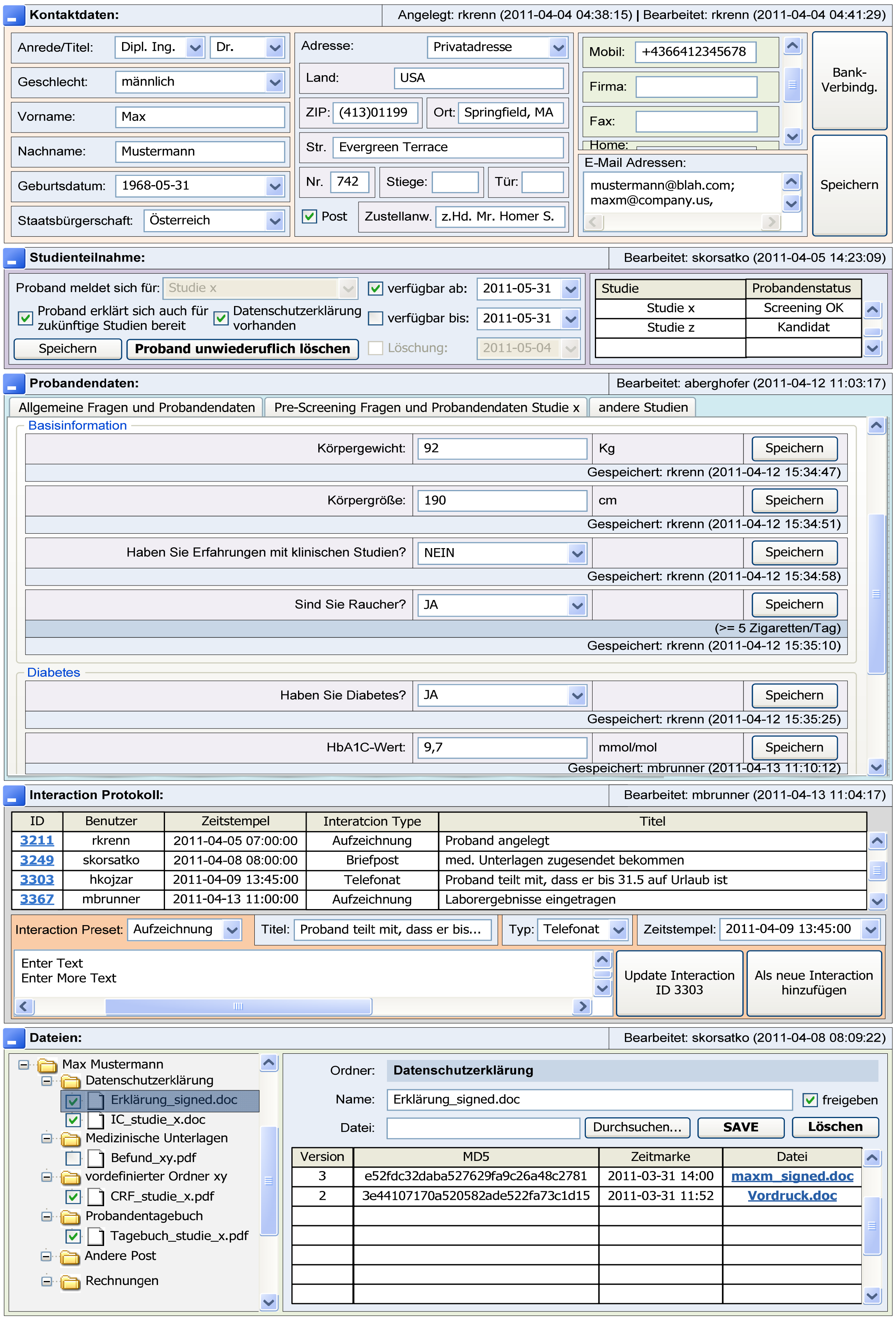
- Tägliche Benachrichtigung über den Fortschritt der Probanderekrutierung einer Studie
- Erinnerungsfunktion für zu kontaktierende Probanden:
    - bevorstehende Löschung von Probandeneinträgen
    - Aussendung von Datenschutz- bzw. Einverständiserklärungen

Folgende Zugriffsberechtigungen für Benutzer können identifiziert werden:

- Abfragen anlegen, bearbeiten und löschen erlaubt J/N
    - Abfragen könnten noch pro Abteilung separiert werden, falls erforderlich
- Abfragen ausführen erlaubt J/N
- Probanden aller Abteilungen anzeigen J/N:
    - Um die Hoheit von Probandendaten einer Abteilung zu wahren, enthält eine Abfrage immer eine fixe Einschränkung auf Probanden der Abteilung des Benutzers. Für spezielle Benutzer (z.B. Systembenutzer der Abfrage-Jobs) könnte dennoch eine Umgehung der Filterung durch diese Zugriffsberechtigung vorsehbar sein.

8.3 Probandenstammblatt

Die Maske der SMS Oberfläche zum Anlegen und Bearbeiten eines Probandeneintrages der Probandendatenbank wird als „Probandenstammblatt“ bezeichnet. Das geplante, detaillierte Design ist in Abbildung 15 skizziert. Möglichst viele Informationen sollen auf einen Blick ersichtlich sein und bearbeitet werden können, um eine Bedienung z.B. während eines Telefonats durch möglichst wenige Mausklicks zu ermöglichen. Eine dialogbasierte Dateneingabe wäre in dieser Hinsicht ein Nachteil. Erfahrungsgemäss ziehen Benutzer im Laufe der Zeit die anfängliche, unübersichtliche Flut an Informationen bzw. Eingabemöglichkeiten mühseliger Navigation durch verschachtelte Strukturen vor. Um bei Dateneingaben trotzdem etappenweise Speicherung zu ermöglichen, sind zusammegehörige Datenbündel mit eigenen „Speichern“-Schaltflächen ausgestattet.



Kontaktdaten: Angelegt: rkrenn (2011-04-04 04:38:15) | Bearbeitet: rkrenn (2011-04-04 04:41:29)

Anrede/Titel: Dipl. Ing. | Dr.
Geschlecht: männlich
Vorname: Max
Nachname: Mustermann
Geburtsdatum: 1968-05-31
Staatsbürgerschaft: Österreich

Adresse: Privatadresse
Land: USA
ZIP: (413)01199 Ort: Springfield, MA
Str. Evergreen Terrace
Nr. 742 Stiege: Tür:
✓ Post Zustellanw. z.Hd. Mr. Homer S.

Mobil: +4366412345678
Firma:
Fax:
Home:
E-Mail Adressen:
mustermann@blah.com;
maxm@company.us,

Bank-Verbindg.
Speichern

Studienteilnahme: Bearbeitet: skorsatko (2011-04-05 14:23:09)

Proband meldet sich für: Studie x
✓ Proband erklärt sich auch für zukünftige Studien bereit
✓ Datenschutzerklärung vorhanden
✓ verfügbar ab: 2011-05-31
verfügbar bis: 2011-05-31
Löschung: 2011-05-04
Speichern | Proband unwiederuflich löschen

Studie	Probandenstatus
Studie x	Screening OK
Studie z	Kandidat

Probandendaten: Bearbeitet: aberghofer (2011-04-12 11:03:17)

Allgemeine Fragen und Probandendaten | Pre-Screening Fragen und Probandendaten Studie x | andere Studien

Basisinformation

Körpergewicht: 92 Kg Speichern
Gespeichert: rkrenn (2011-04-12 15:34:47)
Körpergröße: 190 cm Speichern
Gespeichert: rkrenn (2011-04-12 15:34:51)
Haben Sie Erfahrungen mit klinischen Studien? NEIN Speichern
Gespeichert: rkrenn (2011-04-12 15:34:58)
Sind Sie Raucher? JA Speichern
(>= 5 Zigaretten/Tag)
Gespeichert: rkrenn (2011-04-12 15:35:10)

Diabetes

Haben Sie Diabetes? JA Speichern
Gespeichert: rkrenn (2011-04-12 15:35:25)
HbA1C-Wert: 9,7 mmol/mol Speichern
Gespeichert: mbrunner (2011-04-13 11:10:12)

Interaction Protokoll: Bearbeitet: mbrunner (2011-04-13 11:04:17)

ID	Benutzer	Zeitstempel	Interatcion Type	Titel
3211	rkrenn	2011-04-05 07:00:00	Aufzeichnung	Proband angelegt
3249	skorsatko	2011-04-08 08:00:00	Briefpost	med. Unterlagen zugesendet bekommen
3303	hkojzar	2011-04-09 13:45:00	Telefonat	Proband teilt mit, dass er bis 31.5 auf Urlaub ist
3367	mbrunner	2011-04-13 11:00:00	Aufzeichnung	Laborergebnisse eingetragen

Interaction Preset: Aufzeichnung Titel: Proband teilt mit, dass er bis... Typ: Telefonat Zeitstempel: 2011-04-09 13:45:00

Enter Text
Enter More Text

Update Interaction ID 3303 | Als neue Interaction hinzufügen

Dateien: Bearbeitet: skorsatko (2011-04-08 08:09:22)

- Max Mustermann
  - Datenschutzerklärung
    - ✓ Erklärung_signed.doc
    - ✓ IC_studie_x.doc
  - Medizinische Unterlagen
    - Befund_xy.pdf
  - vordefinierter Ordner xy
    - ✓ CRF_studie_x.pdf
  - Probandentagebuch
    - ✓ Tagebuch_studie_x.pdf
  - Andere Post
  - Rechnungen

Ordner: Datenschutzerklärung
Name: Erklärung_signed.doc ✓ freigeben
Datei: Durchsuchen... SAVE Löschen

Version	MD5	Zeitmarke	Datei
3	e52fdc32daba527629fa9c26a48c2781	2011-03-31 14:00	maxm_signed.doc
2	3e44107170a520582ade522fa73c1d15	2011-03-31 11:52	Vordruck.doc

Abbildung 15: Probandenstammblatt

Das Probandenstammblatt gliedert erfassbare Daten in folgende Bereiche:

- Kontaktdaten:
  - Erfassung der Personenkontaktdaten (Abschnitt 4.5)
  - evtl. Informationen zur Bankverbindung des Probanden.
- Studienteilnahme:
  - Studie, für die sich ein Proband (ursprünglich) gemeldet hat. Dadurch werden die Pre-Screening Fragen und studienspezifischen Probandendaten der betreffenden Studie ausgewählt.
  - Angabe zur Teilnahme und Verfügbarkeit des Probanden, die bei Abfragen für Probandenlisten als Kriterien berücksichtigt werden können (Abschnitt 7.4).
  - Eine Markierung für die Existenz der unterschriebenen Datenschutzerklärung kann explizit gesetzt.
  - *Steuerung der davon abhängigen, automatischen Löschung des Probandeneintrages durch den SMS Benutzer*
  - Statusübersicht der Mitgliedschaft bei Probandenlisten von Studien als Listendarstellung
- Probandendaten: Dies ist im Abschnitt 7.3 umfassend beschrieben.
  - Erfassung von allgemeinen Probandendaten
  - Pre-Screening Fragen und studienspezifischen Probandendaten der Studie, für die sich der Proband gemeldet hat
  - Pre-Screening Fragen und studienspezifischen Probandendaten anderer Studien
- Interaction Protokoll: Eine einfache Möglichkeit zur Aufzeichnung von Interaktionen mit dem Probanden bzw. allgemeinen Vermerken (Journal).
  - Neben Benutzer, Zeitstempel, Titel und ausführlichen Kommentar kann die Art der Kommunikation mit dem Probanden (Interaction Type) aus einer vordefinierten Liste ausgewählt werden, z.B.:
    - Telefonat
    - Postweg
    - allgemeine Aufzeichnung
    - persönlicher Kontakt
    - Journaleintrag: automatisiert eingetragene Vermerke, z.B.:
      - bei Statusänderung des Probanden in der Probandenliste einer Studie – siehe Abschnitt 7.4)
      - Change Log Einträge des Probandenstammblatts selbst
  - Vordefinierte Vorlagen (Interaction Presets) für schnelle bzw. einheitliche Eingabe von Interaction Einträgen
- Dateien zum Probanden: Eine Dokumentverwaltung für anfallende Dokumente des Probanden (z.B. eingescannte, unterzeichnete Schriftstücke) soll die Zuordnung zur betreffenden Person ermöglichen. Für einen über die Jahre anwachsenden Probandenstamm soll damit vor allem ein schnelles Auffinden einzelner Dokumente gewährleistet bleiben. Der Aufbau der Dokumentablage entspricht der in Abschnitt 7.8 beschrieben Dokumentverwaltung für Studiendokumente.

Ein verbleibender Wunsch ist die Möglichkeit zum Abruf des Probandenstammblatts als generiertes PDF-Dokument.

Die Darstellung der Oberfläche in Abbildung 15 zeigt das Erscheinungsbild für einen Benutzer mit sämtlichen der möglichen Zugriffsberechtigungen:

1. Neue Probanden anlegen sowie Kontaktdaten, Studienteilnahmedaten bearbeiten erlaubt J/N
2. Kontaktdaten, Studienteilnahmedaten einsehen erlaubt J/N
   – Fehlt diese Berechtigung, könnte bei Probandensuchvorgängen (Abschnitt 8.2) selbst der Name anonymisiert dargestellt werden („Proband ID xy“).
3. Bankverbindungsdaten bearbeiten erlaubt J/N
4. Bankverbindungsdaten einsehen erlaubt J/N
5. Probanden Löschen *sowie Steuerung der automatischen Löschung bearbeiten* erlaubt J/N
6. Probandendaten bearbeiten erlaubt J/N
7. Probandendaten einsehen erlaubt J/N
8. Interaction Protokoll Einträge hinzufügen erlaubt J/N
9. Interaction Protokoll Einträge bearbeiten erlaubt J/N
10. Interaction Protokoll Einträge einsehen erlaubt J/N
11. Probanden-Dateien hochladen, löschen, Kommentar und Freigabe bearbeiten erlaubt J/N
12. Probanden-Dateien einsehen erlaubt J/N

Ein neuer Proband wird stets unveränderlich jener Abteilung zugeordnet, welcher der Benutzer angehört, der den Probandeneintrag anlegt. Die angeführten Berechtigungen beziehen sich deshalb auf Probandeneinträge der Abteilung des ausführenden Benutzers; andere Probanden sind generell unantastbar. Um konsistent zu bleiben gibt es jedoch die im Abschnitt 7.3 erwähnte Ausnahme, bei der ein Benutzer zumindest Probandendaten (inkl. Pre-Screening Antworten) von Probanden des Probandenstammes einer anderen Abteilung bearbeiten darf. Die folgenden Berechtigungen aus Abschnitt 7.3 entsprechen im Detail den Berechtigungen 6 bis 12 aus der Auflistung zuvor:

- Pre-Screening Fragen des Probandeneintrags ausfüllen erlaubt J/N
- Probandendaten des Probandeneintrags ausfüllen erlaubt J/N

Dies ist aber wie beschrieben auf Probanden der Probandenliste einer Studie eingeschränkt und nur unter der Voraussetzung möglich, dass der bearbeitende Benutzer dem Projektteam der Studie angehört.

8.4 Ablage

Um das Probandenstammblatt in der Datenbank abzubilden, umfassen Probandeneinträge zusammenfassend voraussichtlich folgende Attribute:

- Referenz auf die Abteilung, der die Studie zuzuordnen ist (Abschnitt 4.3), wird ausschliesslich beim Anlegen des Eintrages gesetzt.
- Referenz auf eingegebene Kontaktdaten
- Optionale Referenz auf Bankverbindungsdaten
- Referenz auf die Studie, für die sich der Proband gemeldet hat
- Markierung „Proband erklärt sich auch für zukünftige Studien bereit"
- Markierung „Datenschutzerklärung vorhanden": wird von einem Benutzer gesetzt, sobald die unterschriebene Datenschutzerklärung vorliegt. Durch Setzen dieser Markierung wird die Markierung für die automatische Löschung in der Oberfläche deaktiviert.
- Datum, ab dem der Proband für Studien zeitlich uneingeschränkt längerfristig verfügbar ist: Ist der Proband ab sofort bereit, bleibt das Feld leer.
- Datum, bis zu dem der Proband für Studien zeitlich uneingeschränkt längerfristig verfügbar ist: Ist der Proband weiterhin immer bereit, bleibt das Feld leer.
- Markierung ob die automatische Löschung des Eintrags erfolgen darf: Diese Kontrollmöglichkeit entscheidet zuletzt, ob ein Probandeneintrag ohne vorhandene Datenschutzerklärung wirklich durch den Job für die automatische Löschungen von Probandeneinträgen entfernt werden soll (siehe Abschnitt 7.3).
- Datum der vorgesehenen automatischen Löschung: Dieses Ablaufdatum des Probandeneintrages ist mit dem Datum der Dateneingabe plus z.B. 30 Kalendertage voreingestellt, kann aber individuell gesetzt werden.
- Eingegebene Probandendaten:
  - Referenz auf Probandeneintrag
  - Referenz auf Eingabefeld der jeweiligen Studie
  - Value (Freitext, Wert oder ausgewählte Optionen)
  - Benutzer und Zeitmarke

  Eine Überlegung wäre hierbei, die Probandendatenwerte mit einem Änderungsprotokoll (Change Log) zu versehen, um durch etwaige automatische Auswertungen Korrekturprotokolle (Note-To-File) zu unterstützen.

- Interaction Protokoll: Ein einzelner Eintrag des Interaction Protokolls entspricht einem separaten Datensatz mit folgen Attributen:
  - Referenz auf Probandeneintrag
  - Benutzer, der den Eintrag getätigt hat (inkl. Zeitmarke)
  - Interaction Type (Abschnitt 8.3)
  - Zeitstempel: Zeitmarke für den Zeitpunkt, zu dem das Ereignis tatsächlich stattfand (voreingestellt auf den Zeitpunkt der Eingabe)
  - Titel: Freitext
  - Kommentar: Freitext
- Probanden-Dateien: Einträge für die Dateien des Probanden werden entsprechend Abschnitt 7.8 abgebildet.

8.5 Datenbestand und Ausgangssituation

Natürlich verfügt das CRC bereits über einen vorhanden Probandenstamm. Dieser umfasst insgesamt mehrere Hundert bis Tausend Probanden und liegt verteilt in verschiedenen Formen vor:

- Excel-Files
- Microsoft Access Datenbank

Der Import bzw. die Migration in die SMS Probandendatenbank ist eine ultimative Voraussetzung für die produktive Nutzung des Systems. Bis zum endgültigen Start des StudienManagementSystems muss die vollständige Überspielung der Daten durchgeführt werden. Abgesehen von manuell durchgeführten Eingaben per Probandenstammblatt liegt es nahe, dies so weit es geht automatisiert z.B. mittels einfach gehaltener, gesondert programmierter Konsolenprogramme zu vollziehen. Solche für die einmalige Ausführung vorgesehene Programme lesen Datensätze aus vorbereiteten Dateien (Exports als Textdateien oder CSV – Comma Separated Values Dateien), normalisieren dabei die Datensätze und fügen sie in die SMS Datenbank ein. Die Umsetzung dieser Hilfsprogramme kann z.B. in folgenden Formen erfolgen:

- Java Kommandozeilen-Anwendung (Nutzung vorhandener SMS Webservices zum Einfügen von Datensätzen in die Datenbank)
- Perl-Scripts (direkte Ansteuerung der SMS Datenbank)

Diese automatisierte Migration der bestehenden Daten könnte als getrenntes, kleineres Projekt betrachtet werden.

Appendix C.

RHEL Setup

Clinical Trial Site Management System

Installationsanleitung für RHEL-basierte Linux Server

Inhaltsverzeichnis

1 Dokument Kontrolle

1.1 Dokumentversion

Datum	Version	Kommentar	Inhalt
2012-11-24	0.5	Benutzerberechtigungen	rkrenn
2012-10-03	0.4	DBTool, Backup, Jobs	rkrenn
2012-08-30	0.3	Tomcat JVM Heap Size	rkrenn
2012-06-23	0.2	TODOs erweitertert, Tomcat https Support	rkrenn
2012-06-15	0.1	Initiale Version	rkrenn

Tabelle 1: Dokumenthistorie

1.2 Empfängerliste

Name	Bereich	Organisation
Dr. Stefan Korsatko	Prüfarzt, Studienleitung	Clinical Research Center
Mag. Andrea Berghofer	Monitoring, Studienverwaltung	Clinical Research Center
Dr. Sigrid Deller	Studienleitung, Qualitätskontrolle	Clinical Research Center
Mag. Martina Brunner	Laborleitung	Clinical Research Center
Janka Gerdova	Equipment, Hygiene, Medikation	Clinical Research Center
Harald Kojzar	Studiendurchführung	Clinical Research Center
DI Thomas Truskaller	techn. Betreuung SMS Entwicklung	Joanneum Research
DI Bernd Tschapeller	techn. Betreuung SMS Entwicklung	Joanneum Research
DI Dr. Peter Beck	techn. Betreuung SMS Entwicklung	Joanneum Research
Univ.-Doz. Ing. Mag.rer.nat. Mag.phil. Dr. Andreas Holzinger	Diplomarbeitsbetreuung	TU Graz
Univ.-Prof. DI Dr. Frank Kappe	Diplomarbeitsbetreuer	TU Graz
Univ.Prof. Dr. Barbara Obermayer-Pietsch	Stv. LeiterIn der Klinischen Abteilung	Klinische Abteilung Endokrinologie und Stoffwechsel
Msc. Norbert Tripolt	Klinische Studien	Klinische Abteilung Endokrinologie und Stoffwechsel
Rene Krenn	Diplomand	TU Graz

Tabelle 2: Empfängerliste

2 Linux

Die Beschreibungen dieses Dokuments beziehen sich auf das Aufsetzen des Clinical Trial Site Management Systems (CTSMS) für die (kommerzielle) Linux Distribution

RedHat Enterprise Linux (RHEL) Version 6, Update 2 „Santiago“ x86_64

Ziel ist es, auf einer einzelnen Instanz sowohl Applikations- (Tomcat) als auch RDBMS/Datenbankserver (PostgreSQL) zu integrieren. Ausgangspunkt ist eine Vanilla-Installation mit Standardpaketen. Nachdem die zusätzlich erfoderlichen Softwarepakete separat bzw. nicht per RHEL Quellen durchgeführt wird, dürfte diese Anleitung im Wesentlichen auch für andere auf RedHat basierenden Linuxdistributionen (Centos, Scientific Linux, …) anwendbar sein. Wichtig ist demnach, dass bei der Installation von Linux folgende Pakete nicht ausgewählt sind, da diese bzw. entsprechende Alternativen mit spezifischen Versionen gesondert eingerichtet werden:

- PostgreSQL
- Oracle (Sun) Java bzw. OpenJDK
- Tomcat

Die Reihenfolge der fortfolgend erläuterterten Schritte ist in den meisten Fällen von Bedeutung und sollte eingehalten werden.

3 Linux Benutzerkonten zur Administration

Mindestens ein Benutzerkonto für Administration wird bereits bei der Linux-Installation angelegt. Um diesem Benutzerkonto die Ausführung von Befehlen mit root-Berechtigungen zu ermöglichen, sind folgende Schritte erfoderlich:

1. Benutzer der Benutzergruppe „wheel“ zuweisen
2. Sudo Ausführung für „wheel“ user zulassen:

```
/etc/sudoers                                                    Datei vorhanden
%wheel ALL=(ALL) ALL                                            Zeile auskommentieren
```

Listing 1

4 Installationspakete und Repositories

Um später für alle Fälle jederzeit offline Softwarepakete oder Dependecies nachinstallieren zu können, ist es sinnvoll, den Inhalt des Linux-Installationsmediums als lokale Installationsquelle permanent verfügbar zu machen. Dies geschieht in folgenden Schritten:

1. Verzeichnis mit RPM- (RedHat Package Manger-) Paketen des Installationsmediums in ein Verzeichnis des Dateisystems kopieren, z.B. `/Packages`
2. In diesem Verzeichnis `createrepo` auführen
3. Die vorhandenen Repositories deaktivieren:

/etc/yum.repos.d/*.repo	Datei vorhanden
enabled=0	Zeile bearbeiten

Listing 2

4. Das Paketverwaltungstool von Redhat (`yum`) muss das neue Repository einbinden:

/etc/yum.repos.d/local.repo	Datei erstellen
[RHEL-Repository]	Zeile einfügen
name=RHEL repository	Zeile einfügen
baseurl=file:///Packages	Zeile einfügen
enabled=1	Zeile einfügen
gpgcheck=0	Zeile einfügen

Listing 3

5 PostgreSQL

Der eingesetzte PostgreSQL Datenbankserver muss die spezifische Version 9.1 aufweisen, welche aktueller als die von RHEL mitgelieferte ist. Zumindest eine ältere Version und/oder der zugehörige JDBC Driver hatte Probleme beim Ablegen/Auslesen bzw. Escapen von `bytea` Spaltenwerten[1].

Das PostgreSQL RPM Building Project stellt zur gesonderten Installation praktischer Weise eigene `yum` Repositories bereit, die hinzukommend eingerichtet werden[2]:

1. Vorhandene Repositories sollen die PostgreSQL Pakete ignorieren:

/etc/yum.repos.d/*.repo	Datei vorhanden
exclude=postgresql*	Zeile hinzufügen

Listing 4

/etc/yum/pluginconf.d/rhnplugin.conf	Datei vorhanden
exclude=postgresql*	Zeile hinzufügen – Section "main"

Listing 5

2. Das PostgreSQL Repository selbst wird anschliessend per RPM Paketinstallation hinzugefügt:

http://yum.postgresql.org/9.1/redhat/rhel-6-x86_64/pgdg-redhat91-9.1-5.noarch.rpm

[1] http://jdbc.postgresql.org/changes.html#version_9.0-dev800
[2] http://yum.postgresql.org/howtoyum.php

Die Installation von PostgreSQL 9.1 kann nun beginnen[3]:

1. Binaries installieren:
   - `yum install postgresql-libs`
   - `yum install postgresql`
   - `yum install postgresql-server`
2. Linux-Benutzerkonto `postgres` hinzufügen (falls nicht schon vorhanden):
   - `useradd postgres`
   - Sichers Passwort setzen: `sudo passwd postgres`
3. Cluster initialisieren:
   - `service postgresql-9.1.4 initdb`
4. Runlevel-Konfiguration für den automatischen Start von PostgreSQL beim Systemstart einstellen:
   - `sudo chkconfig --add postgresql-9.1`
   - Schalter pro Runlevel überprüfen mit `chkconfig --list`
   - Evtl. justieren durch:

     ```
     sudo chkconfig --level 2345 postgresql-9.1 on
     ```

5. Linux-Benutzerkonto für PostgreSQL Datenbankuser “ctsms” vorbereiten[4]:
   - Auch der Servlet-Container `tomcat` wird unter diesem User laufen
   - Linux-Benutzergruppe “ctsms” erzeugen: `sudo groupadd ctsms`
   - `sudo useradd -g ctsms ctsms`
   - Sicheres Passwort setzen: `sudo passwd ctsms`
6. PostgreSQL Datenbanken und Rollen für das CTSMS vorbereiten:
   - `su - postgres`
   - `psql template1`
   - Rolle `postgres` wird ausdrücklich mit einem Password versehen, um Zugriff mit z.B. Navicat von externen Hosts zu Verwaltungszwecken zu ermöglichen. Da Externer Zugriff auf die Datenbank nur per SSH Tunnel (gesichert durch Linux Benutzeranmeldung) vorgesehen ist, sind die PostgreSQL Rollenpasswörter prinzipiell nicht zwingend geheimzuhalten:

     ```
     ALTER USER postgres WITH PASSWORD 'postgres';
     ```

   - `CREATE USER ctsms WITH PASSWORD 'ctsms';`
   - `CREATE DATABASE ctsms;`
   - `CREATE DATABASE ctsms_test;`
   - `GRANT ALL PRIVILEGES ON DATABASE ctsms to ctsms;`
   - `GRANT ALL PRIVILEGES ON DATABASE ctsms_test to ctsms;`
   - PostgreSQL-Konsole beenden mit `\q`
   - Test: `psql -d ctsms -U ctsms`

[3] http://yum.postgresql.org/files/PostgreSQL-RPM-Installation-PGDG.pdf
[4] http://www.cyberciti.biz/faq/howto-add-postgresql-user-account/

7. PostgreSQL „host-based authentication” anpassen[5]:
   - verschlüsselte Datenbankverbindungen werden vorerst nicht verwendet

/var/lib/pgsql/9.1/data/pg_hba.conf	Datei vorhanden
# IPv4 local connections: host all all 127.0.0.1/32 **password**	Zeile bearbeiten – method von “ident” auf “password” ändern[6]
# IPv6 local connections: Host all all ::1/128 **password**	Zeile bearbeiten – method von “ident” auf “password” ändern
#**host all all w.x.y.z/32 password**	Optional: Zeile hinzufügen um direkten Zugriff auf die Datenbank von Host w.x.y.z zu erlauben

Listing 6

6 PostgreSQL Datenverzeichnis

Während des Betriebs des CTSMS häufen sich eingebene Daten in der Datenbank an, sodass die Größe der PostgreSQL Datenfiles dahinter sukzessive anwächst. Um die Bootpartition vor dem Vollaufen zu schützen sowie die wertvollen Daten evtl. auf redundanter Festplattenhardware unterzubringen, ist es angebracht das PostgreSQL Datenverzeichnis vom Standardpfad `/var/lib/pgsql/` auf ein anderes Verzeichnis (Mountpoint von alternativem physikalischen Laufwerk, z.b. `/ctsms/`) zu legen. Dazu sind die folgenden Schritte notwendig:

1. PostgreSQL Server deaktivieren
2. Verzeichnis `/var/lib/pgsql/` nach `/ctsms/` verschieben:
3. Berechtigungen sicherstellen: `chown postgres:postgres /ctsms/pgsql –R`
4. PostgreSQL init.d-Script anpassen:

/etc/init.d/postgresql-9.1	Datei vorhanden
PGDATA=**/ctsms/pgsql/9.1/data**	Zeile bearbeiten
PGLOG=**/ctsms/pgsql/9.1/pgstartup.log**	Zeile bearbeiten

Listing 7

Das PostgreSQL Logfile-Verzeichnis `/ctsms/pgsql/9.1/data/pg_log/` befindet sich von Haus aus innerhalb des Datenverzeichnisses und wird durch eine PostgreSQL-eigene Logfile-Rotation begrenzt.

7 PostgreSQL Performance Tuning

TODO: /var/lib/pgsql/9.1/data/postgresql.conf anpassen

[5] http://www.postgresql.org/docs/9.1/static/auth-pg-hba-conf.html
[6] http://www.bobsgear.com/display/bobsgear/Setting+Up+PostgreSQL+for+Confluence

8 Verzeichnis für Dokumentverwaltungsfunktionen

Die Dokumentverwaltungsfunktionen in der CTSMS-Weboberfläche erlauben das Einspielen verschiedenster Dateien in Form von Dateiuploads. Der Inhalt der Dateien wird konfigurationsabhängig ab gewissen Dateigrößen oder generell nicht mehr in Datenbankspalten abgelegt, sondern in einem verwalteten Verzeichnis auf dem Datenträger. Ähnlich wie das Datenverzeichnis der PostgreSQL Datenbank resultiert ein stetiger Zuwachs des Festplattenspeicherbedarfs dieses Verzeichnisses. Um wiederrum das Bootlaufwerk vor Überlauf zu bewaren und/oder redundante physikalische Datenträger einzusetzen, kann das besagte Verzeichnis auf einen entsprechenden Mountpoint gesetzt werden. In der Standardeinstellung ist im CTSMS per .settings-Datei das Verzeichnis auf `/ctsms/external_files` festgelegt. Im Endeffekt muss dafür gesorgt werden, dass dieses Verzeichnis vorgefunden und vom `ctsms` Benutzer beschrieben und gelesen werden kann:

1. `/ctsms/external_files` anlegen
2. Besitzer festlegen: `chown ctsms:ctsms /ctsms/external_files -R`

9 Datenbankverwaltung

Grundlegend wird das Anlegen, Sichern, Löschen und Wiederherstellen der PostgreSQL Datenbank des CTSMS durch die bei jeder PostgreSQL Installation mitgelieferten Command-Line-Tools[7] ermöglicht, welche Textdateien mit SQL-Statements erzeugen bzw. solche in die Datenbank einspielen und auf diese anwenden.

CTSMS Datenbank erstellen/löschen mittels `schema-create.sql` und `schema-drop.sql`:

1. `su – ctsms`
2. Erstellen: `psql -U ctsms ctsms < /ctsms/schema-create.sql`
3. Löschen: `psql -U ctsms ctsms < /ctsms/schema-drop.sql`

Einfaches CTSMS Datenbankbackup (SQL Dump) erzeugen/einspielen:

1. `su – ctsms`
2. Backup erstellen: `pg_dump -U ctsms ctsms > /ctsms/dump.sql`
3. Backup einspielen: `psql -U ctsms ctsms < /ctsms/backup.sql`[8]

Für ein vollständiges Backup müssen zusätzlich die abgelegten Dokumente im Verzeichnis `/ctsms/external_files` gesichert werden. Die Sicherung der Daten durch Kopieren auf andere Speichermedien udgl. ist prinzipiell unbedenklich hinsichtlich des Datenschutzes. In der Datenbank und in Dokumenten enthaltene direkt personenbezogene Daten von Probanden werden durch Verschlüsselung geschützt, die an die Passwörter der Benutzer im CTSMS gebunden ist.

[7] http://www.postgresql.org/docs/9.1/static/backup-dump.html
[8] als Rolle `postgres` ausführen, falls folgendes fehlschlägt: `CREATE EXTENSION IF NOT EXISTS plpgsql WITH SCHEMA pg_catalog`

In Verbindung mit `crontab` zeitgesteuert erstellte Datenbankbackups sind übersichtlich organisierbar bei Verwendung eines erweiterteten Backup-Shellskripts wie z.B. `pg_backup`[9]:

/ctsms/pg_backup.sh	pg_backup.sh Version 1.1.3
`databases=`$PGBINDIR/psql $PARAM_PGHOST -U $PGUSER -q -c "`**`select d.datname as Name from pg_catalog.pg_database d`**`" template1 …`	`Zeile 270 und 272 bearbeiten ("\l" durch "select d.datname as Name from pg_catalog.pg_database d" ersetzen)`

Listing 8

Ein ausgewähltes `pg_backup`-Backup vom Zeitpunkt YYYY-MM-DD HHmm kann wie folgt wiederhergestellt werden, wenn die Datenbank zuvor entfernt wurde:

1. `su – postgres`
2. `bzcat /ctsms/pgsql/9.1/backups/YYYY-MM/pg_db-localhost-ctsms-backup-YYYY-MM-DD-HHmm.bz2 | nice pg_restore --verbose -U postgres -F t -d ctsms`

10 Oracle Java

CTSMS wurde entwickelt und getestet mit Oracle Java SE 6 (1.6.31). Die Installation dieser Version erfolgt durch das Ausführen von http://download.oracle.com/otn/java/jdk/6u31-b04/jdk-6u31-linux-x64-rpm.bin.

11 Tomcat

CTSMS verwendet den Servlet Container Tomcat 6. Auch wenn dieser im Repository der RHEL Distribution enthalten ist, erfolgt seine Installation gesondert von der aktuellen Binary Distribution (Version 6.0.35) des Herstellers. Damit können von einem einheitlichen Ausgangspunkt beginnend die spezifischen Libs (z.B. JDBC Connector) eingepflegt werden, wodurch die Verhältnisse der Entwicklungsumgebung exakt nachgebildet werden[10]:

1. http://tweedo.com/mirror/apache/tomcat/tomcat-6/v6.0.35/bin/apache-tomcat-6.0.35.tar.gz ins Verzeichnis `/opt` kopieren:
2. Entpacken: `tar xzvf apache-tomcat-6.0.35.tar.gz`
3. Logon Script zum setzen erfoderlicher Systemvariablen erstellen:

/etc/profile.d/java.sh	Datei erstellen
`JAVA_HOME=`**`/usr/java/jdk1.6.0_31/bin`**	
`JRE_HOME=`**`/usr/java/jdk1.6.0_31/jre/bin`**	
`CATALINA_HOME=`**`/opt/apache-tomcat-6.0.35`**	
`CATALINA_OPTS="-Xms128m –Xmx1024m"`	
`TOMCAT_USER=`**`ctsms`**	
`export JAVA_HOME JRE_HOME CATALINA_HOME CATALINA_OPTS TOMCAT_USER`	

Listing 9

[9] http://www.bitweaver.org/wiki/pg_backup+PostgreSQL+backup+script
[10] http://wiki.openbluedragon.org/wiki/index.php/Apache_Tomcat_on_CentOS/RedHat

4. Init.d-Script erstellen:

/etc/init.d/tomcat	**Datei erstellen**

`#!/bin/bash`
`#`
`# Init file for Tomcat server`
`#`
`# chkconfig: 2345 55 25`
 	
`# Source function library.`
`. /etc/init.d/functions`	

`TOMCAT_USER=`**`ctsms`**	Siehe /etc/profile.d/java.sh

`CATALINA_OPTS="-Xms128m –Xmx1024m -XX:MaxPermSize=512m`
`-XX:+UseConcMarkSweepGC`
`-XX:+CMSPermGenSweepingEnabled`
`-XX:+CMSClassUnloadingEnabled"`
	JVM Initial und Maximum Heap Size, PermGen Size

`CATALINA_HOME=`**`/opt/apache-tomcat-6.0.35`**	Siehe /etc/profile.d/java.sh Neuerliche Festlegung der beiden Umgebungsvariablen TOMCAT_USER und CATALINA_HOME ist erforderlich, da zum Zeitpunkt der Runlevel-Etablierung noch keine Logon-Scripts (java.sh) ausgeführt wurden.

```
start() {
        echo "Starting Tomcat: "
        if [ "x$USER" != "x$TOMCAT_USER" ]; then
          su - $TOMCAT_USER -c
"$CATALINA_HOME/bin/startup.sh"
        else
          $CATALINA_HOME/bin/startup.sh
        fi
        echo "done."
}
stop() {
        echo "Shutting down Tomcat: "
        if [ "x$USER" != "x$TOMCAT_USER" ]; then
          su - $TOMCAT_USER -c
"$CATALINA_HOME/bin/shutdown.sh"
        else
          $CATALINA_HOME/bin/shutdown.sh
        fi
        echo "done."
}

case "$1" in
  start)
        start
        ;;
  stop)
        stop
        ;;
  restart)
        stop
        sleep 10
        #echo "Hard killing any remaining threads.."
        #kill -9 `cat $CATALINA_HOME/work/catalina.pid`
        start
        ;;
  *)
        echo "Usage: $0 {start|stop|restart}"
esac

exit 0
```

Listing 10

5. Besitzer und Berechtigungen von `/etc/init.d/tomcat` angleichen:
    - `chown root:root /etc/init.d/tomcat`
    - `chmod 755 /etc/init.d/tomcat`

6. Zusätzliche Tomcat Libs im Verzeichnis `/opt/apache-tomcat-6.0.35/lib/` platzieren:
   - JSF Expression Language 2.2: `el-impl-2.2.jar`
   - PostgreSQL ODBC Connector: `postgresql-9.1-901.jdbc4.jar`
7. Keystore mit selbstsigniertem Zertifikat für SSL erstellen[11]:
   - `$JAVA_HOME/bin/keytool -genkey -alias tomcat -keyalg RSA` als User `ctsms` ausführen
8. Character Encoding für URIs auf UTF-8 festlegen und Verbindung (https) aktivieren:
   - Momentan sind sowohl http als auch https Verbindungen aktiviert und erlaubt. Primefaces dürfte inzwischen auch ausschliesslich mit https-Verbindungen auskommen[12].

/opt/apache-tomcat-6.0.35/conf/server.xml	Datei vorhanden

`<Connector port="8080" protocol="HTTP/1.1"`
`connectionTimeout="20000"`
`URIEncoding="UTF-8"`
`redirectPort="8443" />`
	Zeile bearbeiten

`<Connector port="8009" protocol="AJP/1.3"`
`redirectPort="8443"`
`URIEncoding="UTF-8"` `/>`
	Zeile bearbeiten

`<Connector`
`protocol="org.apache.coyote.http11.Http11Protocol"`
`port="8443" maxThreads="200"`
`scheme="https" secure="true" SSLEnabled="true"`
`keystoreFile="${user.home}/.keystore”`
`keystorePass="changeit"`
`clientAuth="false" sslProtocol="TLS"`
`URIEncoding="UTF-8"` **`/>`**
	SSL Konfiguration hinzufügen. Durch protocol=“…“ wird die Auswahl der verwendeten SSL-Implementation erzwungen (hier JSSE). Falls irgendwann ein CA Zertifikat verfügbar ist, könnte auf APR (OpenSSL Libs) gewechselt werden. Keystore Password auf das im vorigen Schritt angegebene anpassen.

Listing 11

9. Optional: Tomcat Manager Role mit sicherem Passwort hinzufügen:

/opt/apache-tomcat-6.0.35/conf/tomcat-users.xml	Datei vorhanden
`<role rolename="manager"/>`	Zeile hinzufügen – Section “<tomcat-users>”
`<user username="tomcat" password="tomcat" roles="manager"/>`	Zeile hinzufügen – Section “<tomcat-users>”

Listing 12

10. Runlevel-Konfiguration für den automatischen Start von Tomcat beim Systemstart einstellen:
    - `chkconfig --add tomcat`
    - Schalter pro Runlevel überprüfen mit `chkconfig --list`
    - Evtl. justieren durch:

```
chkconfig --level 2345 tomcat on
```

11. Firewall: TCP Verbindungen für Tomcat auf Port 8080 zulassen
    - TODO: Port für Tomcat ändern
12. Besitzer des gesamten Tomcat Verzeichnises setzen:
    - `chown ctsms:ctsms /opt/apache-tomcat-6.0.35 –R`

[11] http://tomcat.apache.org/tomcat-6.0-doc/ssl-howto.html
[12] http://code.google.com/p/primefaces/issues/detail?id=609

13. Die Datei `catalina.out` im Tomcat Logfile Verzeichnis `/opt/apache-tomcat-6.0.35/logs/` wächst und muss durch `logrotate` begrenzt werden[13]:

```
/etc/logrotate.d/tomcat                                    Datei erstellen
/opt/apache-tomcat-6.0.35/logs/catalina.out {
  copytruncate
  daily
  rotate 7
  compress
  missingok
  size 5M
}
```

Listing 13

Bis auf weiteres wird kein Apache Webserver einegrichtet, der Servlet-Anfragen per Handler an Tomcat weiterreicht um z.B. statische Resourcen schneller zu liefern oder ein Loadbalancing zu ermöglichen.

12 CTSMS WAR Deployment

Die eigentliche CTSMS Software präsentiert sich als Java Web Archive (WAR) Datei, die abschliessend entpackt in das `/webapps/` Verzeichnis der Tomcat Installation kopiert werden muss:

1. Entfaltete Datei `ctsms-web.war` ins Verzeichnis `/opt/apache-tomcat-6.0.35/webapps/` kopieren
2. Besitzer des gesamten Tomcat Verzeichnises setzen:
   - `chown ctsms:ctsms /opt/apache-tomcat-6.0.35 -R`

13 PDF Vorlagen

Für generierte PDF Dokumente erlauben Templates die Positionierung von Logos, Disclaimer-Zeilen u.ä. in Form eines Vordrucks, um Corporate Design Richtlinien erfüllen zu können. PDF Templates und spezifische True Type Fonts für die generierten PDFs sind per .settings File im CTSMS definierbar und auf die folgende Ordner bzw. Dateinamen:

1. `/ctsms/pdf_templates` erzeugen, Templates und TTF Dateien ablegen:
   - True Type Font Arial, Standard Style: `arial.ttf`
   - True Type Font Arial, Bold Style: `arialbd.ttf`
   - True Type Font Arial, Bold Italic Style: `arialbi.ttf`
   - True Type Font Arial, Italic: `ariali.ttf`
   - Weitere Schriften, z.B. eine monospaced Font
   - PDF Template für Kursteilnahmelisten: `courseparticipantlist_sample_template.pdf`
   - PDF Template für Lebensläufe: `cv_sample_template.pdf`
2. Besitzer setzen: `chown ctsms:ctsms /ctsms/pdf_templates –R`

[13] http://java.dzone.com/articles/how-rotate-tomcat-catalinaout

14 DBTool

Der CTSMS Core Tier (`ctsms-core-VersionXY.jar`) enthält einen ausführbaren Einstiegspunkt ("main"). Das Programm stellt ein Komandozeilentool zur Bewältigung verschiedener Aufgaben dar, welche komplexere bzw. applikationsspezifische Datenbankzugriffe erfordern. Voraussetzung für den Start des Programms ist das Vorliegen sämtlicher jar-Dependencies, wie es z.B. nach dem WAR Deployment im Ordner `/opt/apache-tomcat-6.0.35/webapps/ctsms-web/WEB-INF/lib/` gegeben ist. Um das Programm ohne systemweites Setzen von Classpath-Variablen zu starten, empfiehlt es sich, ein Startscript einzurichten:

```
/ctsms/dbtool.sh                                                Datei erstellen
#!/bin/bash
/usr/java/jdk1.6.0_31/jre/bin/java -Dfile.encoding=Cp1252 -classpath
/opt/apache-tomcat-6.0.35/webapps/ctsms-web/WEB-INF/lib/ctsms-core-1.0-
SNAPSHOT.jar/opt/apache-tomcat-6.0.35/webapps/ctsms-web/WEB-INF/lib/*
at.zmf.crc.ctsms.executable.DBTool $1 $2 $3 $4 $5 $6 $7 $8 $9
```

Listing 14

Somit kann das Datenbank CLI Tool als `dbtool.sh` angesprochen werden:

```
/ctsms/dbtool.sh -h
starting...
Phoenix Clinical Trial Management System
version: 1.0-SNAPSHOT
site: Zentrum f. med. Forschung
task: print help
usage: dbtool [task <arg>] [option1 <arg> option2 <arg> ...]
 -c,--clear                                          task: clear database
 -cct,--clear_criteria_tables                        task: clear criteria tables
 -cd,--create_department                             task: create new department
 -cdi,--create_department_interactive                task: create new department (interactive)
 -cdp,--change_department_password                   task: change department password
 -cdpi,--change_department_password_interactive      task: change department password (interactive)
 -cdpu,--change_department_password_user             task: change department password (user authentication)
 -cu,--create_user                                   task: create new user
 -cui,--create_user_interactive                      task: create new user (interactive)
 -dlk,--department_l10n_key <arg>                    option: department l10n name
 -dm,--delete_missing                                task: scan for missing external files and delete references from DB
 -do,--delete_orphaned                               task: scan for orphaned external files and delete them
 -dp,--department_password <arg>                     option: department password
 -e,--encoding <arg>                                 option: encoding of csv/text file to import
 -epdt,--export_permission_definition_template <arg> task: export permission definition template as csv/text file
 -er,--email_recipients <arg>                        option: send output to email recipients
 -f,--force                                          option: skip confirmation promt
 -h,--help                                           task: print help task: initialize db by insertig required setup records
 -ib,--import_bank <arg>                             task: import bank identifications from csv/text file
 -ic,--import_country <arg>                          task: import country names from csv/text file
 -icf,--import_course_files <arg>                    task: import documents and files for a course entity
 -ict,--init_criteria_tables                         task: initialize criteria tables
 -id,--entity_id <arg>                               option: record id of entity
 -iif,--import_inventory_files <arg>                 task: import documents and files for an inventory entity
 -imc,--import_mime_course <arg>                     task: import course file mime types from csv/text file
 -imi,--import_mime_inventory <arg>                  task: import inventory file mime types from csv/text file
 -imp,--import_mime_proband <arg>                    task: import proband file mime types from csv/text file
 -ims,--import_mime_staff <arg>                      task: import staff file mime types from csv/text file
 -imt,--import_mime_trial <arg>                      task: import trial file mime types from csv/text file
 -ipd,--import_permission_definitions <arg>          task: import permission definitions from csv/text file
 -ipf,--import_proband_files <arg>                   task: import documents and files for a proband entity
 -is,--import_street <arg>                           task: import street names from csv/text file
 -isf,--import_staff_files <arg>                     task: import documents and files for a staff entity
 -it,--import_title <arg>                            task: import titles from csv/text file
 -itf,--import_trial_files <arg>                     task: import documents and files for a trial entity
 -iz,--import_zip <arg>                              task: import zip codes from csv/text file
 -l,--limit <arg>                                    option: limit for number of (processed) records
 -ldd,--load_demo_data                               task: load db with demo data records
 -ndp,--new_department_password <arg>                option: new department password
 -odp,--old_department_password <arg>                option: old department password
 -p,--password <arg>                                 option: user password
 -pn,--prepare_notifications                         task: prepare notifications
 -pp,--permission_profiles <arg>                     option: list of permission profiles
 -rcf,--remove_course_files                          task: remove all files of a course entity
 -rif,--remove_inventory_files                       task: remove all files of an inventory entity
 -rp,--remove_probands                               task: remove probands pending for auto-delete
 -rpf,--remove_proband_files                         task: remove all files of a proband entity
 -rsf,--remove_staff_files                           task: remove all files of a staff entity
 -rtf,--remove_trial_files                           task: remove all files of a trial entity
 -sm,--scan_missing                                  task: scan for missing external files
 -sn,--send_notifications                            task: send notifications
 -so,--scan_orphaned                                 task: scan for orphaned external files
 -u,--username <arg>                                 option: username
```

Listing 15

15 Datenbankinitialisierung und Stammdatenbeladung

Bevor eine Anmeldung per Webfrontend möglich wird, müssen ausgehend von einer leeren Datenbank erste Benutzer erzeugt werden. Neben diesem naheliegenden Vorgang ist es wesentlich, den Initialisierungstask von `dbtool` auszuführen, der eine umfassende, vordefinierte Konfiguration in Form von Datenbankeinträgen erstellt (Auswahlsätze, Berechtigungen, …). Für die Verwendung der Dokumentverwaltungen werden Mimetype-Definitionen[14] für die zugelassenen Dateitypen benötigt. Eine Reihe weiterer Stammdaten kann optional eingespielt werden:

- Titel und akademische Grade
- Kreditinstitutverzeichnis[15]
- Ländernamen[16]
- PLZ/Orte[17]
- Straßenamen[18]

Ein beispielhafter Beladungsvorgang der Datenbank besteht dann aus folgenden Schritten:

Initiale Ladeprozeduren – load.sh

```
#!/bin/bash
#Datenbank erstellen:
#psql -U ctsms ctsms < /ctsms/schema-create.sql

#Datenbank initialisieren:
/ctsms/dbtool.sh –i

#Permissions erstellen
/ctsms/dbtool.sh -ipd /ctsms/master_data_2012_09_17/permission_definitions.csv

#Mimetype-Definitionen pro Dokumentverwaltung einspielen:
/ctsms/dbtool.sh -imi "/ctsms/master_data_2012_09_17/mime.types" -e "ISO-8859-1"
/ctsms/dbtool.sh -ims "/ctsms/master_data_2012_09_17/mime.types" -e "ISO-8859-1"
/ctsms/dbtool.sh -imc "/ctsms/master_data_2012_09_17/mime.types" -e "ISO-8859-1"
/ctsms/dbtool.sh -imt "/ctsms/master_data_2012_09_17/mime.types" -e "ISO-8859-1"
/ctsms/dbtool.sh -imp "/ctsms/master_data_2012_09_17/mime.types" -e "ISO-8859-1"

#Titel/akadem. Grade einspielen:
/ctsms/dbtool.sh -it "/ctsms/master_data_2012_09_17/titles.csv" -e "ISO-8859-1"
#Bankenverzeichnis einspielen:
/ctsms/dbtool.sh -ib "/ctsms/master_data_2012_09_17/kiverzeichnis_gesamt_de_1347893202433.csv" -e "ISO-8859-1"
#Ländernamen einspielen:
/ctsms/dbtool.sh -ic "/ctsms/master_data_2012_09_17/countries.txt" -e "ISO-8859-1"
#PLZ/Orte einspielen:
/ctsms/dbtool.sh -iz "/ctsms/master_data_2012_09_17/streetnames.csv" -e "ISO-8859-1"
#Strassennamen einspielen:
/ctsms/dbtool.sh -is "/ctsms/master_data_2012_09_17/streetnames.csv" -e "ISO-8859-1"

#Testdaten erzeugen (optional):
#/ctsms/dbtool.sh -ldd

#Organisationseinheit "crc" erzeugen:
#/ctsms/dbtool.sh -cd -dlk "crc" -dp "crc department passphrase"

#Benutzer für Organisationseinheit "crc" erzeugen:
#/ctsms/dbtool.sh -cu -dlk "crc" -dp "crc department passphrase" -u "rkrenn" -p "rene"
#/ctsms/dbtool.sh -cu -dlk "crc" -dp "crc department passphrase" -u "skorsatko" -p "stefan"
#/ctsms/dbtool.sh -cu -dlk "crc" -dp "crc department passphrase" -u "aberghofer" -p "andrea"
#/ctsms/dbtool.sh -cu -dlk "crc" -dp "crc department passphrase" -u "sdeller" -p "sigrid"
#/ctsms/dbtool.sh -cu -dlk "crc" -dp "crc department passphrase" -u "mbrunner" -p "martina"
#/ctsms/dbtool.sh -cu -dlk "crc" -dp "crc department passphrase" -u "jgerdova" -p "janka"
#/ctsms/dbtool.sh -cu -dlk "crc" -dp "crc department passphrase" -u "hkojzar" -p "harald"
#/ctsms/dbtool.sh -cu -dlk "crc" -dp "crc department passphrase" -u "ttruskaller" -p "thomas"
#/ctsms/dbtool.sh -cu -dlk "crc" -dp "crc department passphrase" -u "pbeck" -p "peter"
#/ctsms/dbtool.sh -cu -dlk "crc" -dp "crc department passphrase" -u "fkappe" -p "frank"
#/ctsms/dbtool.sh -cu -dlk "crc" -dp "crc department passphrase" -u "aholzinger" -p "andreas"
#/ctsms/dbtool.sh -cu -dlk "crc" -dp "crc department passphrase" -u "ghasenberger" -p "gerhard"
```

Listing 16

[14] http://svn.apache.org/viewvc/httpd/httpd/branches/2.2.x/docs/conf/mime.types?revision=1301896&view=co
[15] http://www.oenb.at/idakilz/kiverzeichnis?action=downloadAllData
[16] http://www.zoll.de/SharedDocs/Downloads/DE/Links-fuer-Inhaltseiten/Fachthemen/Zoelle/Atlas/codeliste_i0300_laenderliste.zip?__blob=publicationFile
[17] http://www.statistik.at/verzeichnis/strassenliste/gemplzstr.zip
[18] http://www.statistik.at/verzeichnis/strassenliste/gemplzstr.zip

16 Einrichtung wiederholt auszuführender Aufgaben

Die vorgesehene Funktionsweise des CTSMS erwartet die wiederkehrende, automatische Ausführung folgender Aufgaben (Jobs):

- Ermittlung von Benachrichtigungen für den aktuellen Tag (z.B. Passwortablauf-Erinnerung)
- Versenden von Benachrichtigungen per Email
- Löschung überfälliger, markierter Probanden

Auch Backups sollen zeitgeplant erstellt werden. Für diese Zwecke wird das auf UNIX-Systemen übliche Ablaufplanungsprogramm `cron` eingesetzt.

Hinsichtlich Backup und der erzeugten Logfiles sind folgende Vorbereitungen zu treffen:

1. Backup-Verzeichnis und Logfile für `pg_backup` erstellen:
   - `mkdir /ctsms/pgsql/9.1/backups`
   - `touch /ctsms/pgsql/9.1/backups/pg_backup.log`
   - `chown postgres:postgres /ctsms/pgsql/9.1/backups/ -R`
   - `chmod 644 /ctsms/pgsql/9.1/backups/ -R`
2. Begrenzung von `/ctsms/pgsql/9.1/backups/pg_backup.log` durch `logrotate`:

/etc/logrotate.d/pg_backup Datei erstellen
```
/ctsms/pgsql/9.1/backups/pg_backup.log {
  weekly
  rotate 7
  missingok
  size 5M
}
```
Listing 17

3. Logfile für `dbtool` erstellen:
   - `touch /ctsms/dbtool.log`
   - `chown ctsms:ctsms /ctsms/dbtool.log`
   - `chmod 644 ctsms/dbtool.log`
4. Begrenzung von `/ctsms/dbtool.log` durch `logrotate`:

/etc/logrotate.d/dbtool Datei erstellen
```
/ctsms/dbtool.log {
  daily
  rotate 7
  missingok
  size 5M
}
```
Listing 18

Abschließend wird ein systemweiter Ablaufplan (crontab) mittels `/etc/crontab` registriert:

```
/etc/crontab (Dateiinhalt bearbeiten)
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=
HOME=/

# For details see man 4 crontabs

# Example of job definition:
# .---------------- minute (0 - 59)
# |  .------------- hour (0 - 23)
# |  |  .---------- day of month (1 - 31)
# |  |  |  .------- month (1 - 12) OR jan,feb,mar,apr ...
# |  |  |  |  .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat
# |  |  |  |  |
# *  *  *  *  * user-name command to be executed

#prepare daily notifications:
00 05 * * * ctsms source /etc/profile ; nice /ctsms/dbtool.sh -pn >> /ctsms/dbtool.log 2>&1
#send notifications via email
*/10 06-21 * * * ctsms source /etc/profile ; nice /ctsms/dbtool.sh -sn >> /ctsms/dbtool.log
2>&1
#proband auto-delete job:
45 23 * * * ctsms source /etc/profile ; nice /ctsms/dbtool.sh -rp -f >> /ctsms/dbtool.log 2>&1

#backup:
00 03 * * * postgres source /etc/profile ; export PGBACKUPDIR=/ctsms/pgsql/9.1/backups ;
export PGLOGDIR=/ctsms/pgsql/9.1/backups/pg_backup.log ; nice /ctsms/pg_backup.sh b >>
/ctsms/pgsql/9.1/backups/pg_backup.log 2>&1

#remove backups older than 1 week except those made on the 1st of month
00 04 * * * postgres source /etc/profile ; find /ctsms/pgsql/9.1/backups/ -mtime +7 -name
"*_db-*[234567890]-0300*" -exec rm {} \;
00 04 * * * postgres source /etc/profile ; find /ctsms/pgsql/9.1/backups/ -mtime +7 -name
"*_db-*[123]1-0300*" -exec rm {} \;

#empty line at end:
```

Listing 19

17 Übersicht Installationsverzeichnis

Zusamenfassend entsteht in dem Verzeichnis /ctsms/ die folgende Dateistruktur:

```
find /ctsms/ -maxdepth 3 -ls
drwxr-xr-x    ctsms    ctsms        4096 /ctsms/
-rwxr-xr-x    ctsms    ctsms         246 /ctsms/dbtool.sh
-rw-rw-r--    ctsms    ctsms       11992 /ctsms/dbtool.log
-rwxrw-rw-    ctsms    ctsms        2427 /ctsms/load.sh
-rw-r--r--    ctsms    ctsms       78842 /ctsms/schema-create.sql
-rw-r--r--    ctsms    ctsms       23656 /ctsms/schema-drop.sql
-rwxrw-rw-    ctsms    ctsms     7964906 /ctsms/dump.sql
-rwxr-xr-x    ctsms    ctsms       19982 /ctsms/pg_backup.sh
drwxr-xr-x    postgres postgres     4096 /ctsms/pgsql
drwxr-xr-x    postgres postgres     4096 /ctsms/pgsql/9.1
-rw-r--r--    postgres postgres       86 /ctsms/pgsql/.bash_profile
drwxr-xr-x    ctsms    ctsms        4096 /ctsms/external_files
drwxrwxrwx    ctsms    ctsms        4096 /ctsms/pdf_templates
-rw-r--r--    ctsms    ctsms       64196 /ctsms/pdf_templates/comsc.ttf
-rw-r--r--    ctsms    ctsms       36890 /ctsms/pdf_templates/courseparticipantlist_sample_template.docx
-rw-r--r--    ctsms    ctsms       36715 /ctsms/pdf_templates/cv_sample_template.docx
-rw-r--r--    ctsms    ctsms      778552 /ctsms/pdf_templates/arial.ttf
-rw-r--r--    ctsms    ctsms       63615 /ctsms/pdf_templates/cv_sample_template.pdf
-rw-r--r--    ctsms    ctsms      749004 /ctsms/pdf_templates/arialbd.ttf
-rw-r--r--    ctsms    ctsms       65062 /ctsms/pdf_templates/courseparticipantlist_sample_template.pdf
-rw-r--r--    ctsms    ctsms      555884 /ctsms/pdf_templates/ariali.ttf
-rw-r--r--    ctsms    ctsms      561924 /ctsms/pdf_templates/arialbi.ttf
drwxrwxrwx    rkrenn   wheel        4096 /ctsms/master_data_2012_09_17
-rw-r--r--    rkrenn   wheel     1321954 /ctsms/master_data_2012_09_17/kiverzeichnis_gesamt_de_1347893202433.csv
-rw-r--r--    rkrenn   wheel       49815 /ctsms/master_data_2012_09_17/mime.types
-rw-r--r--    rkrenn   wheel             /ctsms/master_data_2012_09_17/permission_definitions.csv
-rw-r--r--    rkrenn   wheel     5647673 /ctsms/master_data_2012_09_17/streetnames.csv
-rw-r--r--    rkrenn   wheel         419 /ctsms/master_data_2012_09_17/titles.csv
-rw-r--r--    rkrenn   wheel        3638 /ctsms/master_data_2012_09_17/countries.txt
```

Appendix D.

Finance Module Concept

Erweiterung „Finanzplanungsmodul“

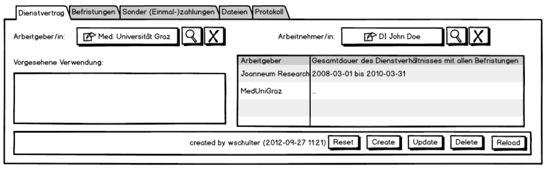
Geplant als zusätzliches Modul, das sich in die Infrastruktur des Phoenix CTMS einfügt:

- Keine gesonderte IT-Infrastruktur notwendig
- Ansprechende, moderne Weboberfläche
- Benutzer & Zugriffsberechtigungen

Nutzung vorhandener Daten:

- Personen/Organisationen
- Studien
- Probanden (Zahlungen)

Erfassung von Dienstverhältnissen



Arbeitgeber/in: Med. Universität Graz

Arbeitnehmer/in: DI John Doe

Vorgesehene Verwendung:

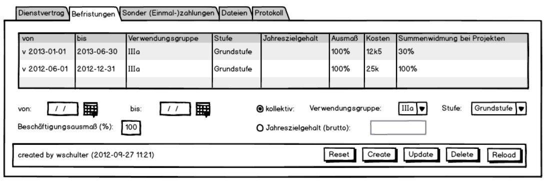
Arbeitgeber	Gesamtdauer des Dienstverhältnisses mit allen Befristungen
Joanneum Research	2008-03-01 bis 2010-03-31
MedUniGraz	...

created by wschulter (2012-09-27 11:21)

Abbildung von Dienstverträgen (und deren Befristungen) ist zur Personalkostenberechnung erforderlich

- Festgelegt durch Arbeitgeber+Arbeitnehmer
- Dokumente wie gescannte Arbeitsverträge können als Dateien in der Datenbank abgelegt werden.

Dienstvertrag - Befristungen



von	bis	Verwendungsgruppe	Stufe	Jahreszielgehalt	Ausmaß	Kosten	Summenwidmung bei Projekten
v 2013-01-01	2013-06-30	IIIa	Grundstufe		100%	12k5	30%
v 2012-06-01	2012-12-31	IIIa	Grundstufe		100%	25k	100%

von: / / bis: / /

● kollektiv: Verwendungsgruppe: IIIa Stufe: Grundstufe

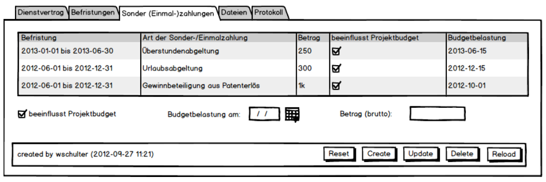
Beschäftigungsausmaß (%): 100

○ Jahreszielgehalt (brutto):

created by wschulter (2012-09-27 11:21)

- Dienstvertrag besteht i.A. aus einer folge fortgesetzter Verlängerungen
- Hinterlegung der Kollektivlohn-Bruttogehälter sowie Auszahlungszeitpunkte erforderlich (aufwändig, veränderlich)

Dienstvertrag - Einmalzahlungen



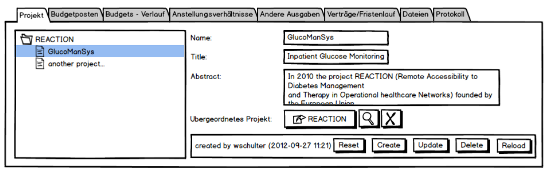
Befristung	Art der Sonder-/Einmalzahlung	Betrag	beeinflusst Projektbudget	Budgetbelastung
2013-01-01 bis 2013-06-30	Überstundenabgeltung	250	✓	2013-06-15
2012-06-01 bis 2012-12-31	Urlaubsabgeltung	300	✓	2012-12-15
2012-06-01 bis 2012-12-31	Gewinnbeteiligung aus Patenterlös	1k	✓	2012-10-01

✓ beeinflusst Projektbudget Budgetbelastung am: / / Betrag (brutto):

created by wschulter (2012-09-27 11:21)

- Sonderzahlungen wie 13.-14. Gehalt werden automatisch berechnet
- Gesondert (manuell) einzutragen sind in diesem Karteireiter jedoch Einmalzahlungen

Erfassung von Projekten



Name: GlucoManSys

Title: Inpatient Glucose Monitoring

Abstract: In 2010 the project REACTION (Remote Accessibility to Diabetes Management and Therapy in Operational healthcare Networks) founded by the European Union

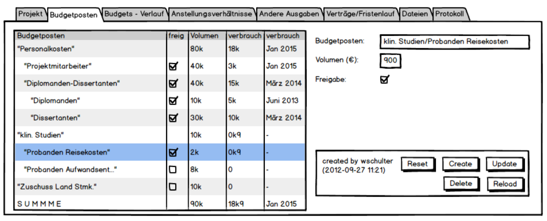
Ubergeordnetes Projekt: REACTION

created by wschulter (2012-09-27 11:21)

Hierarchische Ordnung von Projekten (Baumstruktur):

- eigenes, unabhängiges Budget pro Projekt
- Dokumentablage
- Status und Fristerinnerungen ähnlich bei Studienmodul

Projekt - Budget



Budgetposten	freig	Volumen	verbrauch	verbrauch
"Personalkosten"		80k	18k	Jan 2015
"Projektmitarbeiter"	✓	40k	3k	Jan 2015
"Diplomanden-Dissertanten"	✓	40k	15k	März 2014
"Diplomanden"	✓	10k	5k	Juni 2013
"Dissertanten"	✓	30k	10k	März 2014
"klin. Studien"		10k	0k9	-
"Probanden Reisekosten"	✓	2k	0k9	-
"Probanden Aufwandsent..."	☐	8k	0	-
"Zuschuss Land Stmk."	☐	10k	0	-
S U M M M E		90k	18k9	Jan 2015

Budgetposten: klin. Studien/Probanden Reisekosten

Volumen (€): 900

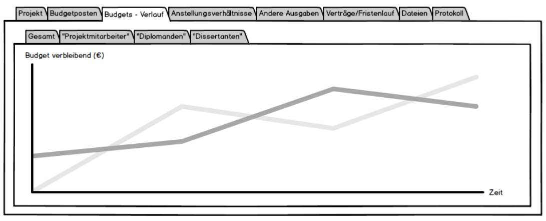
Freigabe: ✓

created by wschulter (2012-09-27 11:21)

- Budget besteht aus einzelnen Posten (Baumstruktur) mit Kontingenten
- Übersicht verbrauchter Beträge u. verbleibende Zeit bei gebuchten, zeitabhängigen Kosten (z.B. Personal)

hoeni **Clinical Trial Management System**

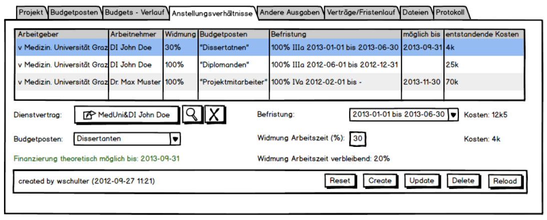
Projekt - Kostenverlauf



- Verlauf zeitabhängiger Kosten (z.B. Personal) als Liniendiagramm
- Start-Endzeitpunkte werden von DV-Befristungen des mitwirkenden Personals abgeleitet

hoeni **Clinical Trial Management System**

Projekt – Dienstverhältnisse mitwirkender Personen



Arbeitgeber	Arbeitnehmer	Widmung	Budgetposten	Befristung	möglich bis	entstandende Kosten
v Medizin. Universität Graz	DI John Doe	30%	"Dissertatnen"	100% IIIa 2013-01-01 bis 2013-06-30	2013-09-31	4k
v Medizin. Universität Graz	DI John Doe	100%	"Diplomanden"	100% IIIa 2012-06-01 bis 2012-12-31		25k
v Medizin. Universität Graz	Dr. Max Muster	100%	"Projektmitarbeiter"	100% IVa 2012-02-01 bis -	2013-11-30	70k

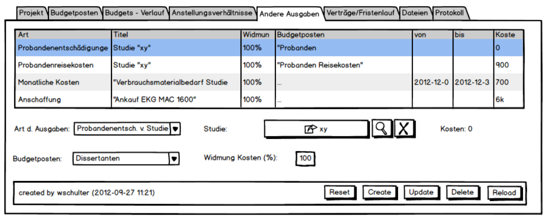
Dienstvertrag: MedUni&DI John Doe
Befristung: 2013-01-01 bis 2013-06-30
Kosten: 12k5
Budgetposten: Dissertanten
Widmung Arbeitszeit (%): 30
Kosten: 4k
Finanzierung theoretisch möglich bis: 2013-09-31
Widmung Arbeitszeit verbleibend: 20%
created by wschulter (2012-09-27 11:21)

Liste von Dienstvertrag-Befristungen:

- Zuweisung einer DV Befristung an Budgetposten
- Anteilmäßige Widmung der Personalkosten

hoeni **Clinical Trial Management System**

Projekt – Übrige Ausgaben



Art	Titel	Widmun	Budgetposten	von	bis	Koste
Probandenentschädigunge	Studie "xy"	100%	"Probanden			0
Probandenreisekosten	Studie "xy"	100%	"Probanden Reisekosten"			900
Monatliche Kosten	"Verbrauchsmaterialbedarf Studie	100%	...	2012-12-0	2012-12-3	700
Anschaffung	"Ankauf EKG MAC 1600"	100%	...			6k

Art d. Ausgaben: Probandenentsch. v. Studie
Studie: xy
Kosten: 0
Budgetposten: Dissertanten
Widmung Kosten (%): 100
created by wschulter (2012-09-27 11:21)

- Erfasste Auszahlungen (Entschädigungen, Reisekosten) aus dem Probandenmodul
- Einmalausgaben
- Monatlich laufende Kosten (mit Angabe des Zeitraums)