Masterarbeit

Entwurf und Realisierung eines Wissensmanagement Tools zur Unterstützung eines Entwickler Teams

Christoph Scherr

Betreuer

Ass.Prof. Univ.-Doz. Dipl.-Ing. Dr.techn. Denis Helic

Institut für Wissensmanagement, Technische Universität Graz

Kooperationspartner

Siemens AG Österreich, CT DC I IAS CEE

Dipl.-Ing. Johann Schinnerl

Graz, Wintersemester 2010

Kurzfassung

Wissensmanagement stellt einen entscheidenden Faktor in Unternehmen dar. Auch zur Unterstützung in Entwicklungsprojekten gewinnt Wissensmanagement zunehmend an Bedeutung. Im Projektumfeld der betreuenden Mitarbeiter der Firma Siemens wurde Verbesserungspotential im Umgang mit Wissen im Projekt identifiziert.

Es ist ein bekanntes Problem, dass Wissen im Unternehmen oft sehr verteilt und isoliert vorhanden ist, sei es in Köpfen der Entwickler, in Dokumenten die manchmal schwer zugänglich sind, oder in verschiedensten Tools, die Information in unterschiedlichsten Dateiformaten ablegen oder teilweise gar keine geeigneten Exportfunktionen anbieten. Es ist äußerst komplex dieses Wissen einheitlich zu erfassen, zu strukturieren oder anderen zugänglich zu machen.

Zur Unterstützung in diesem Bereich sollte ein geeignetes Wissensmanagement-Tool entwickelt werden. Dazu wurde eine Analyse der Ausgangssituation durchgeführt, um anschließend Anforderungen an das zu entwickelnde System zu definieren. Eine grundlegende Literaturrecherche liefert die Basis für die Umsetzung dieses Vorhabens.

Um das Wissen im Projektumfeld entsprechend zu strukturieren und abzubilden, wurde ein geeignetes Datenmodell definiert, welches als Grundlage für die Ablage von Information dient. Die Kernidee war, die Daten bzw. das Wissen in eine semantische Struktur zu bringen, die beliebig in andere semantische Formate transformiert werden kann.

Das Wissensmanagement-Tool selbst wurde als Client/Server Anwendung umgesetzt. Besonderes Augenmerk galt der Definition einer reichhaltigen Auswahl an Schnittstellen, um diversen Clients Zugriff auf die Daten bzw. das Wissen zu ermöglichen. Weiters wurden Import/Export Formate und Schnittstellen implementiert, um die Datenbestände aus verschiedenen Tools einbinden zu können.

Einen entscheidenden Faktor im Projektumfeld der betreuenden Firma Siemens stellt das Generieren von Reports und Dokumenten dar. Dazu wurde mittels XSLT Transformationen und geeigneter Schnittstellen eine praktikable Lösung entwickelt, um aus den gespeicherten Informationen automatisiert Dokumente in den etablierten Formaten Microsoft Word oder PDF generieren zu können. Es zeigte sich, dass das konzipierte Datenmodell den Anforderungen entspricht und die gespeicherten Informationen beliebig in andere Formate transformierbar sind.

Den Abschluss der Arbeit bildet eine Zusammenfassung der wichtigsten vom entwickelten Wissensmanagement-Tool angebotenen Features, sowie ein kurzer Ausblick und Empfehlungen, welche Weiterentwicklungen oder Verbesserungen möglich wären. Mögliche nächste Schritte wären die automatisierte Transformation der Inhalte in Formate des Semantic Web Stacks und Ausnutzung der dort etablierten Technologien zur Abfrage oder Konsistenzprüfung des gesammelten Wissens, oder die Entwicklung weiterer, spezialisierter Clients, um das große Potential dieses Wissensmanagement-Tools weiter auszuschöpfen.

Abstract

Knowledge Management is a critical factor for the success and competitiveness of a company. It is essential for the support of development projects. A project team at Siemens identified room for improvement regarding the treatment of domain knowledge for their projects.

It is a common problem that domain knowledge in companies is often distributed and isolated. Know-how is stored in minds of developers, documents which are difficult to access or various tools that store data in different file formats and do not support useful export functionality. It is a challenging task to gather, structure and share this knowledge in a consistent way.

A suitable knowledge management tool should be developed to support the project staff in these areas. For that purpose the initial situation was analyzed and requirements for the system to be developed have been defined. A comprehensive literature research was done before designing the tool.

To structure and map knowledge in the project environment an applicable data model was defined which provides a base for storing information. The main idea is to store data and knowledge into a semantic structure which can be transformed to any other semantic data format.

The knowledge management tool was implemented by utilizing a Client/Server Architecture. Special attention was paid to the definition of a rich set of interfaces, in order to provide a wide variety of clients with an efficient means of accessing the information. Furthermore import/export formats were defined and interfaces were implemented to integrate data from other tools.

For developer projects at Siemens it is desirable to have accurate and consistent printed documentation, at least for important project milestones. Therefore XSLT transformations and suitable interfaces were used to automatically generate documents in established formats like Microsoft Word or PDF. A representative set of use cases were implemented and it demonstrated that the designed data model fulfills all requirements and that the stored information could be transformed to any other data format or structure.

The concluding chapter of this master thesis gives a short overview of the most important features of the developed knowledge management tool. The chapter future work contains several recommendations for future development. To utilize more of the potential of the tool, features like automatic transformations to standard formats of the Semantic Web Stack or support for specialized clients can be implemented.

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, Jänner 2011 ***Christoph Scherr***

EIDESSTATTLICHE ERKLÄRUNG

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 entnommene Stellen als solche kenntlich gemacht habe.

Graz, Jänner 2011 ***Christoph Scherr***

Danksagung

An dieser Stelle bedanke ich mich bei Denis Helic, der jederzeit bei Fragen oder Problemen zur Verfügung stand, für die Betreuung meiner Masterarbeit.

Besonderer Dank gilt meinem Betreuer bei der Firma Siemens, Johann Schinnerl, der als Ideengeber und Initiator dieser Arbeit diente und während der gesamten Umsetzung eine große Unterstützung in jeder Hinsicht war.

Nicht zuletzt bedanke ich mich bei meinen Eltern, die mich während meiner gesamten Studienzeit unterstützten.

Inhaltsverzeichnis

1 Einleitung

Wissen bzw. Wissensmanagement bekommt eine immer größere Bedeutung in der inner- und überbetrieblichen Leistungserstellung. Daher ist die Gestaltung der Wissenskreation, der Wissensflüsse und der Wissensnutzung ein zunehmender Wettbewerbsfaktor. [Riempp, 2004]

Auch die betreuenden Mitarbeiter der Firma Siemens erkannten in einem gemeinsamen Workshop Optimierungspotential im Umgang mit Wissen im Projekt. Wissen ist teilweise nur in den Köpfen einzelner Entwickler vorhanden, allgemeine Projektinformation oder Verfahrensanleitungen sind nicht entsprechend dokumentiert, weiters befindet sich viel Information in unterschiedlichsten Einzeldokumenten oder Tools die teils schwer zugänglich und/oder kombinierbar sind.

Zur Unterstützung der Entwickler in diesen Bereichen sollte ein geeignetes unterstützendes Wissensmanagement-Tool entwickelt werden, welches zusätzlich auch den Dokumentationsprozess unterstützen sollte, indem sich automatisch Dokumente oder Reports aus dem gespeicherten Wissen generieren lassen.

Dazu wurde eingangs eine Literaturrecherche durchgeführt, um den theoretischen Hintergrund von Wissensmanagement, sowie unterstützende Tools wie Wissensmanagementsysteme und deren Teilaspekte zu beleuchten. Vor allem das Datenmodell bzw. verwenden von Technologien wie XML, RDF, etc. stellt einen wesentlichen Faktor dar.

Anschließend wurde anhand der vorliegenden Rechercheergebnisse eine Analyse durchgeführt und Anforderungen an das zu entwickelnde Wissensmanagement-Tool definiert.

Der praktische Teil dieser Arbeit ergibt sich aus der Umsetzung dieses Wissensmanagement-Tools, das den Projektnamen *Infotool* erhielt. Das entwickelte Infotool sollte nicht nur den definierten Anforderungen entsprechen, sondern auch auf die zuvor erhaltenen Erkenntnisse der Literaturrecherche, betreffend Technologie, Architektur oder Datenmodell, aufbauen. Exemplarisch finden sich weiters einige Erklärungen und Erläuterungen zum entwickelten Wissensmanagement-Tool in dieser Arbeit.

Abschließend finden sich ein Ausblick und Empfehlungen, in welche Richtung Weiterentwicklungen des implementierten Wissensmanagement-Tools gehen sollten.

2 Theoretische Grundlagen

Zu beachten ist, dass diese Arbeit sich nicht direkt mit der Einführung von Wissensmanagement in der Firma Siemens beschäftigt, sondern ein Wissensmanagement-Tool bzw. –System entwickelt werden soll, welches zur softwaretechnischen Unterstützung des Wissensmanagements dient. Als Einstieg folgt dennoch ein kurzer Überblick über Wissensmanagement im Allgemeinen. Im Speziellen wird nachfolgend dann auf Wissensmanagementsysteme und weitere Teilaspekte eingegangen.

2.1 Wissensmanagement

Es existieren mehrere Definitionen die das Verständnis der Ziele und Aufgaben des Wissensmanagements darlegen sollen. Eine klare, einheitliche Definition von Wissensmanagement die als Grundlage dient, gibt es nicht, allerdings weisen mögliche Definitionen allesamt Ähnlichkeiten auf.

Grundsätzlich ist der Begriff Wissensmanagement anwendungsorientiert und multidisziplinär zu verstehen, es geht schlicht und einfach um die Handhabung von Wissen. Somit kann man es als Komplex von Steuerungsaufgaben verstehen, der alle Prozesse, Methoden und Strukturen einer Organisation umfasst, welche sich mit Wissen befassen. Folglich stellt die planvolle und effiziente Bewirtschaftung des Produktionsfaktors Wissen ein übergeordnetes Ziel des Wissensmanagements in der Organisation dar. [Brücher, 2004]

Da wie gerade angesprochen unterschiedlichste Definitionen von Wissensmanagement existieren, sind zwei weitere, auch zeitlich auseinander liegende Definitionen des Begriffs nachfolgend angeführt.

Das Wissensmanagement umfasst das Management der Daten-, Informations- und Wissensverarbeitung im Unternehmen (es ist nicht zu verwechseln mit der Wissensverarbeitung im Management, das die Wissensproblematik von Managementaufgaben behandelt, und auch nicht mit dem Symbolic Management, das das Führen und Steuern mit Hilfe bestimmter Symbole zum Inhalt hat). Wissen und Informationen werden dabei als grundsätzlich handhabbare Objekte angesehen, die direkt oder indirekt über Wissens- bzw. Informationsträger in materieller (Daten-)Form vorliegen. Wissensmanagement beschränkt sich jedoch nicht nur auf den technischen Problemkreis, wie das traditionelle Daten- und Informationsmanagement, sondern es verwaltet insbesondere auch die personellen und institutionellen Wissenspotentiale und deren Verarbeitung. Es übernimmt damit spezielle Funktionen des Personalmanagements. [Kleinhans, 1989]

Wissensmanagement meint die Gesamtheit organisationaler Strategien zur Schaffung einer „intelligenten“ Organisation. Mit Blick auf Personen geht es um das organisationsweite Niveau der Kompetenzen, Ausbildung und Lernfähigkeit der Mitglieder; bezüglich der Organisation als System steht die Schaffung, Nutzung und Entwicklung der kollektiven Intelligenz und des „collective mind“ in Frage; und hinsichtlich der technologischen Infrastruktur geht es vor allem darum, ob, wie und wie effizient die Organisation eine zu ihrer

Operationsweise kongeniale Kommunikations- und Informationsinfrastruktur nutzt. [Willke, 2001]

Somit stellt Wissensmanagement einen wichtigen Faktor im Unternehmen dar, bedeutet aber auch ein hohes Maß an Aufwand für die Unternehmensorganisation.

Generell lassen sich zwei Grundausrichtungen des Wissensmanagements identifizieren, welche nachfolgend kurz erklärt werden.

2.1.1 Humanorientierter Ansatz des Wissensmanagements

Unter dem humanorientierten Ansatz des Wissensmanagements versteht man jenen Ansatz, der die Person bzw. das Individuum im Mittelpunkt, und damit als zentralen Wissensträger sieht. Wobei das Wissensmanagement nicht ausgenützte Potentiale wecken soll, sowie die kognitiven Fähigkeiten des Einzelnen unterstützen soll. Somit ist der humanorientierte Ansatz von psychologischen und soziologischen Erkenntnissen geprägt. Dieser Ansatz steht daher in einem Nahverhältnis zum Personalmanagement und beschäftigt sich damit, wie Individuen motiviert werden können, um ihr persönliches Wissen mit anderen in der Organisation zu teilen, sowie aktiv am Lernprozess teilzunehmen. Als weiterer Teilaspekt gilt der Aufbau von Kontakten und Netzwerken, also sozialer Themen. Der Schwerpunkt liegt auf jeden Fall bei der Einrichtung eines ganzheitlichen Konzepts für das Wissensmanagement und nicht auf der technischen Umsetzung, wie sich Wissen personenunabhängig speichern lässt, oder der Verarbeitung des Wissens in Form einer organisatorischen Wissensbasis. [Lehner, 2009]

2.1.2 Technologieorientierter Ansatz des Wissensmanagements

Beim technologieorientierten Ansatz findet eine starke Konzentration auf Aspekte der computergestützten Informationsverarbeitung statt. Hauptziel ist die Unterstützung der Organisationsmitglieder, Wissen aus ihrem jeweiligen Aufgabengebiet zu erfassen, sammeln, verteilen oder einfach abzurufen. Dadurch soll eine organisatorische Wissensbasis aufgebaut werden, um das Wissen der Organisation zu erfassen, zu erweitern, zu nutzen, zu speichern und zu verteilen. Der technologieorientierte Ansatz sieht die organisatorische Wissensbasis oft sehr eingeschränkt auf den technischen Aspekt, zum Beispiel als Datenbank, Wissensportal oder Expertensystem. Somit befasst sich der technologieorientierte Ansatz vorwiegend mit der Problemstellung, die beim Entwurf solcher Systeme auftreten, während der humanorientierte Aspekt vernachlässigt wird. [Lehner, 2009]

2.2 Wissensmanagementsysteme

Die Aufgabe von Wissensmanagementsystemen ist die informationstechnologische Unterstützung (IT-Unterstützung) des Wissensmanagements. Es ist ein softwaretechnisches System, welches idealerweise Funktionen zur Unterstützung der Identifikation, des Erwerbs, der Entwicklung, Verteilung, Bewahrung und Bewertung von Wissen bereitstellen sollte. Mit Hilfe dieser Funktionen sollte organisatorisches Lernen, sowie die organisatorische Effektivität unterstützt werden. Wissensmanagementsysteme können als „Enabler“ für eine Wissensmanagement-Kultur oder auch als „Support“ für Wissensmanagement-Prozesse

bezeichnet werden, sie bilden sozusagen das Rückgrat des Wissensmanagements. [Brücher, 2004] [Lehner, 2009]

Als grundlegende Teilbereiche eines Wissensmanagementsystems lassen sich unten angeführte Funktionen identifizieren. [Lehner, 2009]

- Kommunikation: Funktionen der synchronen und asynchronen Kommunikation, zum Beispiel: E-Mail, Workflowmanagement oder Newsgroups
- Inhaltsmanagement: Funktionen zum Umgang mit Inhalten, zum Beispiel: Bildern, Dokumenten und Lernobjekten
- Entscheidungsunterstützung: Funktionen zur Auswertung und Zusammenfassung von Informationen
- Suche: Funktionen zum Finden von Inhalten und Experten in unterschiedlichen Quellen, zum Beispiel: Datenbanken, Dateiserver, Dokumentmanagementsysteme
- Visualisierung: Funktionen zur Präsentation von Informations- und Wissenselementen sowie der Struktur dieser.

2.3 Architektur von Wissensmanagementsystemen

Ein wesentlicher Punkt bei der Entwicklung eines Wissensmanagementsystems spielt die ausgewählte Architektur. Einige Aspekte sowie Grundlagen zur üblichsten Architektur, der Client – Server Architektur, werden in diesem Kapitel behandelt.

2.3.1 Zentrale vs. Dezentrale Architektur

Die wohl entscheidenste Frage ist, ob man eine zentrale, oder dezentrale Architektur einsetzt. Während bei der zentralen Architektur ein (oder mehrere) Server Dienste für unterschiedlichste Clients zur Verfügung stellt, gibt es auch einen dezentralen Ansatz, bei welchem gleichrangige Peers (Paare) miteinander kommunizieren und interagieren. Wobei es hier zwei unterschiedliche Ausprägungen gibt. Die Architektur kann vollkommen Peer-to-Peer basiert sein, oder hybrid gestaltet sein. Beim hybriden Ansatz existieren neben den gleichrangigen Peers sogenannte „Super-Peers“ welche Koordinations- und Verwaltungsaufgaben übernehmen. Das Prinzip basiert darauf, dass jeder Peer seine individuelle Wissensbasis hält, die auch auf Basis der individuellen Ontologie des Benutzers verwaltet wird. Diese Wissensbasis kann dann mit anderen Peers geteilt werden, also veröffentlicht werden (public), oder intern bleiben, also anderen Peers oder Nutzern den Zugriff verweigern (private). [Lehner, 2009]

Laut [Lehner, 2009] eignet sich der Einsatz einer zentralen Architektur eher zur Umsetzung von technologieorientierten Wissensmanagements, wogegen dezentrale Architekturen bei humanorientiertem Wissensmanagement angewandt werden sollten. Tabelle 1 stellt die beiden unterschiedlichen Architekturansätze nochmals gegenüber.

Merkmal	Zentrale Architektur	Dezentrale Architektur
Wissenstypus	stabil, objektiv, bewährt, allgemeines Wissen	ad-hoc, subjektiv, unsicher, spezielles Wissen
Wissensspeicher	zentraler Wissensspeicher mit organisationsweiter Ontologie	dezentraler Wissensspeicher mit gruppenweiten oder individuellen Ontologien
Inhalt	Best Practices, Wissensprodukte, gesichertes Wissen, Ideen, Erfahrungen, individuelle Inhalte	individuelle Inhalte, Ideen, Ergebnisse von Gruppensitzungen, Erfahrungen, Lessons learned, Good Practices
Steuerung	zentral	dezentral
Ziele	Sichern und Wiederverwenden von organisatorischem Wissen	Verwalten von an den individuellen Anforderungen ausgerichtetem Wissen
Wissensmanagementansatz	technologieorientiert	humanorientiert
Organisatorische Struktur	Hierarchie, Abteilungen	Projektteams, Communities, Ad-hoc Gruppen

Tabelle 1: Vergleich von zentraler und dezentraler Architektur (Quellen: [Lehner, 2009], [Maier, Hädrich, & Peinl, 2005])

2.3.2 Client-Server Architektur

Das Client-Server Modell hat sich als übliches Architektur Modell durchgesetzt. Wie der Name schon sagt setzt man hier auf eine zentrale Architektur, bei welcher der Anwender den Client betreibt, welcher auf Ressourcen eines Servers zugreift. Diese Ressourcen werden zentral am Server verwaltet. Abbildung 1 veranschaulicht dieses Architektur Modell, welches auf einem einfachen Anfrage-Antwort-Schema basiert. Das heißt, es müssen nicht sämtliche Daten übertragen oder ausgetauscht werden: Der Client sendet eine Anfrage, der Server verarbeitet diese, zum Beispiel greift der Server häufig auf eine dahinter liegende relationale Datenbank zu, führt eine Abfrage für den Client durch und sendet das entsprechende Ergebnis dieser Anfrage an den Client zurück. [Vogel, et al., 2008]

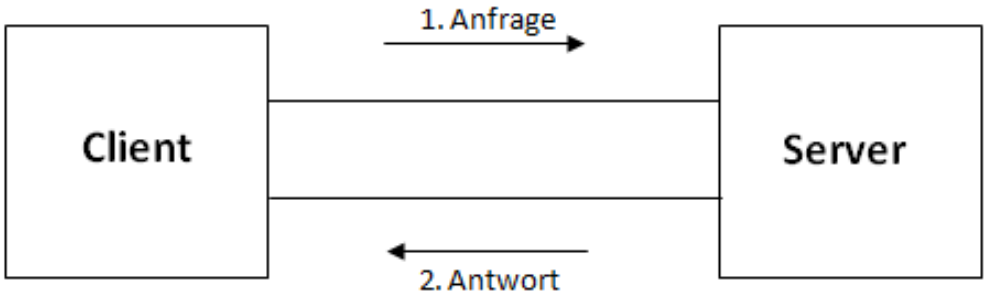


Abbildung 1: Client-Server Modell [Vogel, et al., 2008]

Betrachtet man dieses Modell im Kontext von Wissensmanagementsystemen, heißt das, die organisatorische Wissensbasis wird am Server gespeichert und verwaltet, und die Anwender greifen über Clients auf dieses Wissen zu. Somit kann man vom technologieorientierten Wissensmanagementansatz sprechen, der durch eine zentrale Architektur unterstützt wird.

2.3.3 n-Tier Architektur

Das im vorigen Kapitel beschriebene Client-Server Modell basiert auf einer so genannten 2-Tier Architektur. Das bedeutet, ein zentraler Server bedient mehrere Clients die als Benutzerschnittstelle dienen. Beispielsweise stellt ein Datenbank-Server Stored Procedures oder Datenbank-Trigger zur Verfügung, um auf der Datenbank Berechnungen oder Abfragen durchzuführen. Dies stellt einen häufigen Anwendungsfall von 2-Tier oder n-Tier Architekturen dar, es muss sich aber nicht immer zwingend um einen Datenbank-Server handeln. Ein Nachteil von 2-Tier Architekturen liegt darin, dass es bei einer sehr großen Anzahl gleichzeitiger Anfragen von Clients zu rapiden Performanceeinbrüchen kommen kann. Grundsätzlich funktioniert dieses Modell bis zu einer bestimmten Anzahl von Clients allerdings sehr gut. Ein weiterer Nachteil liegt in der Abhängigkeit vom Hersteller der Datenbank. Da bestimmte Funktionen oder Prozeduren vom jeweiligen Datenbank Hersteller abhängig sind, ergibt sich bei einem etwaigen Wechsel der Datenbank ein hoher Aufwand. [Vogel, et al., 2008]

Eine Lösung für oben genannte Nachteile der 2-Tier Architektur, ist die 3-Tier Architektur. Es wird eine weitere Schicht eingeführt, welche zwischen Client und Datenbank-Server liegt. Diese Zwischenschicht behandelt die Anfragen der Clients. Diese Anfragen werden beispielsweise gereiht, entsprechend für den Datenbank Server aufgearbeitet oder priorisiert. Gegebenenfalls kann eine Anfrage auch gleich zurückgewiesen werden, ohne den Datenbank Server zu beanspruchen. Die Zwischenschicht ist weiters für zentrale Aufgaben der Anwendungslogik zuständig. Der Vorteil dieser zusätzlichen Schicht liegt in einer Verbesserung der Performance, vor allem wenn viele Clients gleichzeitig zugreifen. Ein weiterer großer Vorteil ist die bessere Flexibilität die man durch diese zusätzliche Schicht erreicht, da es durch einheitliche Schnittstellen einfacher ist beispielsweise die Datenbank auszutauschen. Abbildung 2 zeigt eine schematische Darstellung einer typischen 3-Tier Architektur. [Vogel, et al., 2008]

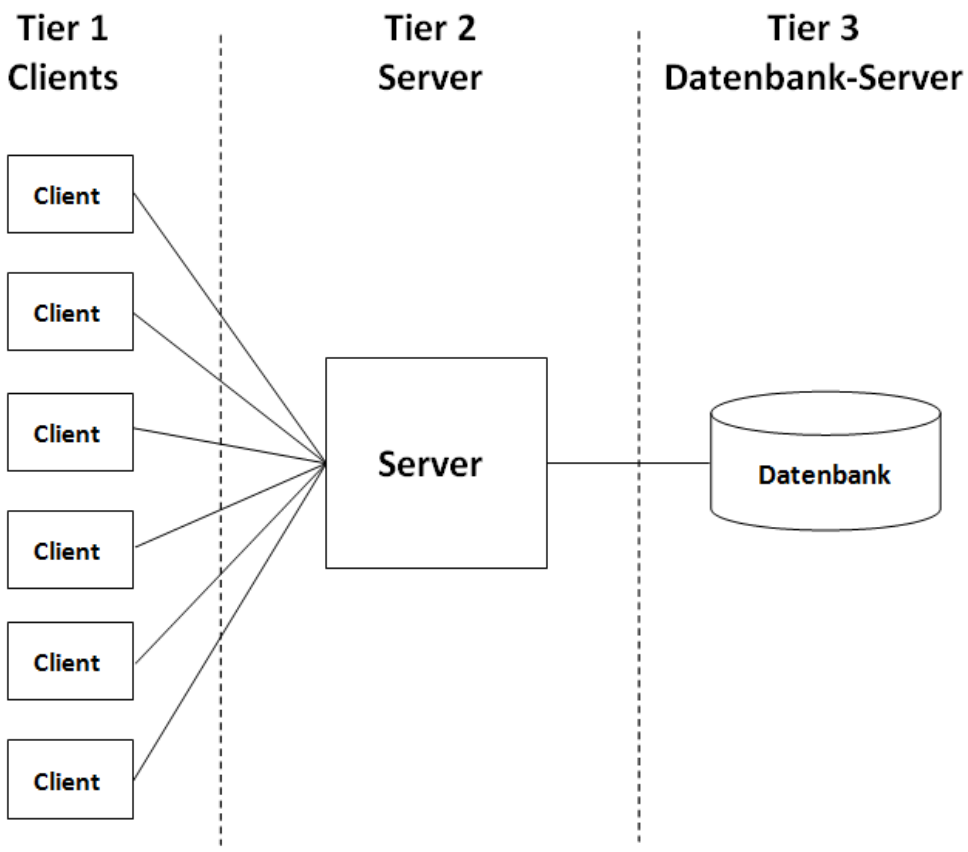


Abbildung 2: Schematische Darstellung einer 3-Tier Architektur [Vogel, et al., 2008]

2-Tier Architektur und 3-Tier Architektur sind spezielle Ausprägungen einer n-Tier Architektur. Es existieren auch Architekturen die mehr als zwei oder drei Tier (Schichten) haben. Dies liegt immer an entsprechenden Anforderungen wie beispielsweise hohe Lastzahlen, bestimmte Qualitätsanforderungen, zusätzlichen Sicherheitsanforderungen oder speziellen Ausfallssicherheitsansprüche. Ein Beispiel wäre, die Server der Zwischenschicht redundant anzulegen, um höhere Performance, bessere Ausfallssicherheit und auch bessere Lastverteilung bei gleichzeitigen Client Anfragen zu erreichen. Dazu bräuchte man aber eine weitere Schicht, welche sich um die Verteilung der eingehenden Anfragen an die redundanten Server kümmert. Dies wäre ein Beispiel für eine 4-Tier Architektur. Es gibt auch den Fall, dass 3-Tier Architekturen miteinander kombiniert oder verbunden werden, wobei dadurch ebenfalls n-Tier Architekturen höherer Ordnungszahl entstehen. Meist ist dabei ein Server einer 3-Tier Architektur selbst Client einer anderen 3-Tier Architektur. Aus Sicht des eigenen Clients ergäbe die Gesamtarchitektur dann eine 4-Tier Architektur. [Vogel, et al., 2008]

2.3.4 Metadaten

Metadaten kann man kurz und knapp als Daten über Daten bezeichnen. Der Ursprung des Wortes „*meta*“ stammt aus dem Griechischen, was soviel bedeutet wie unter, neben oder danach. [Anahory & Murray, 1997]

Das wohl bekannteste und am weitest verbreitete Metadaten Format / Schema ist der sogenannte Dublin Core. Ein Teil davon wurde anschließend für die Umsetzung des entwickelten Wissensmanagement-Tools ausgewählt und angewandt (siehe Kapitel 5.3.9).

Der Dublin Core entstand ca. im Jahre 1995 in Zusammenarbeit von Wissenschaftlern und Bibliothekaren, mit dem Ziel einen allgemeinen Kern bibliographischer Attribute festzulegen. Es sollen gemeinsame Metadaten, unabhängig von spezifischen Anwendungsgebieten definiert werden, durch die der Nachweis, die Erfassung, die Indexierung und die Nutzung digitaler Publikationen erleichtert werden. Diese Metadaten sollten von Autoren und Herausgebern gemeinsam mit der jeweiligen Publikation veröffentlicht werden. Idealerweise sollten Autorenwerkzeuge bereits entsprechende Mechanismen zur Bereitstellung der erforderlichen Metadaten anbieten. Dieser allgemeine Kern an Metadaten könnte je nach Anwendungsgebiet nachträglich durch entsprechende fachliche, spezielle Metadaten erweitert werden. Der Dublin Core bestand anfänglich aus fünfzehn Feldern, welche in Tabelle 2 angeführt sind. Es ist zu beachten, dass es sich primär um intrinsische Eigenschaften der jeweiligen Publikation handelt, während extrinsisch motivierte Parameter wie Nutzungskosten, Zugriffsbeschränkungen etc. nicht vom Dublin Core abgedeckt werden, sondern jeweils durch anwendungsspezifische oder nutzungsabhängige Erweiterungen zu implementieren sind. [Will, 2002]

Man kann die fünfzehn in Tabelle 2 abgebildeten verschiedenen Felder des allgemeinen Kerns des Dublin Core grob in drei Kategorien teilen [Will, 2002]:

- solche, die sich auf den Inhalt einer Ressource beziehen: TITLE, SUBJECT, DESCRIPTION, SOURCE, LANGUAGE, RELATION, COVERAGE
- solche, die sich auf ein abstraktes Werk im Sinne eines intellektuellen Besitzes beziehen: CREATOR, PUBLISHER, CONTRIBUTOR, RIGHTS diese Attribute beziehen sich auf Instanzen oder Institutionen, die Rechte an einer Ressource besitzen oder diese verwalten;
- solche, die sich auf eine konkrete Instanz einer Ressource beziehen, nämlich DATE, TYPE, FORMAT, IDENTIFIER

Bezeichner	**Bedeutung**
TITLE	Titel des Dokuments oder der Ressource
CREATOR	Autor, Verfasser der Ressource, verantwortlich für deren Inhalt
SUBJECT	Thema, Fachgebiet, Schlag- und Stichwörter
DESCRIPTION	Beschreibung, kurze Inhaltsangabe, Zusammenfassung
PUBLISHER	Verlag, veröffentlichende Instanz oder Institution
CONTRIBUTORS	weitere an der Veröffentlichung beteiligte Autoren oder Mitarbeiter

DATE	Zeitpunkt der Veröffentlichung, der Herausgabe, der Gültigkeit
TYPE	Art der Publikation
FORMAT	technisches Format, beispielsweise MIME
IDENTIFIER	weltweit eindeutige Kennung, beispielsweise ISBN oder DOI
SOURCE	Quelle abgeleiteter Publikationen (beispielsweise den Originaltitel des Buchs zum Film)
LANGUAGE	Sprache, in der die Publikation oder deren Inhalt vorliegt
RELATION	logische Einordnung zu anderen Ressourcen
COVERAGE	zeitliche und räumliche Bezugnahme zu anderen Ressourcen
RIGHTS	Verweis auf die Nutzungsbedingungen eines Mediums

Tabelle 2: Dublin Core Elemente und deren Bedeutung [Will, 2002]

2.3.5 Daten

Wie im vorhergehenden Kapitel erwähnt, kann man Metadaten als Daten über Daten bezeichnen. Der wesentliche Punkt sind allerdings die Daten selbst. Im Falle von Wissensmanagement stellen die Daten schließlich das zu verwaltende Wissen dar.

Allgemein betrachtet sind Dokumente Container für Daten [Chen, Snyman, & Sewdass, 2007]. Mangels geeigneter Werkzeuge war Wissensmanagement im in dieser Arbeit untersuchten Projektumfeld eher dokumentzentriert (archivieren, annotieren, …).

Metadaten (speziell Dublin Core) beschreiben traditionell eher allgemeine Eigenschaften von Dokumenten (Autor, Erscheinungsjahr, etc.) und nicht deren Inhalt. Wenn, dann geschieht dies eher oberflächlich mittels Tags, Schlagwörter oder Abstract, jedoch nicht im Detail. Weiters gibt es oft Probleme mit Schlagworten (Dublin Core SUBJECT), da es oft nicht möglich ist mittels einfacher Text-Strings Begriffe eindeutig zu identifizieren. Dieses Problem trifft auf unterschiedliche Begriffe, die dieselbe Schreibung aufweisen, zu. Man spricht dabei von Homonymen. Ein Beispiel wäre das Wort „Apache“, welches mehrere unterschiedliche Bedeutungen aufweisen kann. Es könnte der Apache Webserver gemeint sein, ein Apache Hubschrauber oder auch ein Indianer vom Stamm der Apachen. Ein Mechanismus um solche Mehrdeutigkeiten zu eliminieren und Daten eindeutig zu identifizieren sind sogenannte Identifier. In Kapitel 2.3.6 wird dieser Mechanismus genauer beschrieben.

Um zurück zu den Dokumenten zu kommen: Die benötigten Daten sind jene Informationen, die im Dokument gespeichert sind, diese möchte man verarbeiten und verwalten. Teilweise liegen diese Daten schon sehr strukturiert vor. Als Beispiel wäre hier ein im Projekt verwendetes *SystemFMEA* Dokument zu nennen, welches aus einer vorliegenden Microsoft Access Datenbank generiert wird. Hier ist es einfach diese Daten wie gewünscht zu strukturieren und abzuspeichern. Bei eher unstrukturierten Informationen (z.B. Handbücher, Prozessbeschreibungen, etc.) muss die Information zumindest in Form von elementaren, in

sich abgeschlossenen Einheiten behandelt werden (dafür wurde der Begriff *Topic* verwendet, der in späteren Kapiteln immer wieder auftauchen wird). Diese Topics sollen dann noch mit entsprechenden Metadaten angereichert werden.

Einer der wichtigsten Aspekte eines Wissensmanagementsystems ist die Datenspeicherung. Wobei hier auf die Punkte Offenheit, Verfügbarkeit und Zuverlässigkeit geachtet werden muss. Es sollte Wert auf Offenheit gelegt werden, das heißt Datenstrukturen sollen klar definiert sein, sowie geeignete Schnittstellen für den Zugriff zur Verfügung gestellt sein. Eine unabhängige Zugriffsschicht auf alle integrierten Datenbestände sollte ebenfalls zur Verfügung stehen. Verfügbarkeit und Zuverlässigkeit sind zwei weitere wichtige Bestandteile die betrachtet werden müssen. Als Grundlage dienen ein hochverfügbarer Datenspeicher, sowie regelmäßige (mindestens tägliche) Backups. [Hüttenegger, 2006]

2.3.6 Identifier (URI Schema)

Wie schon in vorigen Kapiteln erwähnt, muss man Dinge die man beschreibt auch eindeutig benennen können. Eine Lösungsmöglichkeit bietet die Verwendung von Uniform Resource Identifier (URI). Diese bieten eine eindeutige Bezeichnung für Ressourcen. Ein URI stellt eine Zeichenfolge dar, die einfach erzeugt werden kann und sowohl abstrakte, als auch physikalische Ressourcen bezeichnet. URIs werden üblicherweise im World Wide Web (WWW) benutzt um Webseiten oder Dateien zu bezeichnen. Die Idee von URIs muss aber nicht zwingenderweise im Kontext mit dem WWW stehen, sondern kann auch losgelöst davon als genereller Mechanismus zur Erzeugung eindeutiger Bezeichner genutzt werden. [Hitzler, Krötzsch, Rudolph, & York, 2007]

Anfänglich existierte eine Trennung in zwei Untermengen, URLs und URNs, diese Trennung ist heutzutage nicht mehr so strikt und es existieren URIs die sich keiner der beiden Untermengen eindeutig zuordnen lassen. [Hitzler, Krötzsch, Rudolph, & York, 2007]

- Uniform Resource Locators (URLs) werden zur Identifizierung von web-adressierbaren Ressourcen benützt. Dies betrifft alle Ressourcen, die im Web über Zugriffsmechanismen wie HTTP (HyperText Transfer Protocol) oder FTP (File Transfer Protocol) verfügbar sind.

- Uniform Resource Names (URNs) werden verwendet, um Ressourcen mittels eines vorhandenen oder frei zu vergebenden Namens zu identifizieren. Die ursprüngliche Idee war, dass URNs für weltweit eindeutige und persistente Bezeichner wie z.B. ISBN (Internationale Standardbuchnummer) verwendet werden.

Syntaktisch bestehen URIs aus mehreren Teilen. Der erste Teil gibt den Typ, oder auch Schema spezifischer Teil genannt, einer URI an und wird durch einen Doppelpunkt vom nachfolgenden Teil getrennt. Bekannte Vertreter sind wie schon erwähnt Internetadressen. Das *http* in http://www.tugraz.at gibt den Typ bzw. das Schema dieser URI an, während der nachfolgende Teil den Bezeichner darstellt. [Hitzler, Krötzsch, Rudolph, & York, 2007]

Bei URNs wird üblicherweise noch ein Namensraum Identifier verwendet, so dass URNs im Format

```
urn: <Namensraum Identifier> : <Bezeichner>
```

dargestellt werden. Die Verwendung von Namensräumen ist Grundvoraussetzung für jede Terminologie. Ein konkretes Beispiel aus dem Projektumfeld wäre *Lastspannungsversorgung*. Diese wird für Simatic-Baugruppen *ET200s 4/8F-DI*, aber auch *ET200m 24F-DI* oder für Dutzend andere verwendet. Daher muss die Informationseinheit für *Lastspannungsversorgung* eindeutig benennbar sein. Dies sollte über URNs geschehen:

```
urn:infotool:et200m:fdi24:powersupply
```

```
urn:infotool:et200s:fdi48:powersupply
```

Dadurch lassen sich die Informationseinheiten zu *Lastspannungsversorgung* schnell und eindeutig identifizieren.

Der URI Mechanismus bzw. das URN Schema wird im in dieser Arbeit entwickelten Wissensmanagement-Tool verwendet, um Ressourcen eindeutig identifizieren zu können.

2.4 Datenaustausch und Datentransformation mit XML

Einer der wichtigsten Aspekte von Informationssystemen bzw. Wissensmanagementsystemen ist der einfache Austausch von strukturierten Daten, sowie die Transformation dieser strukturierten Daten. Als typisches Beispiel lassen sich hier B2B (Business-to-Business) Transaktionen nennen, wo bei einer Online Bestellung Einkäufer auf Angebote des Anbieters zugreifen wollen, diese entsprechend verarbeiten und interpretieren, sowie automatisierte Bestellungen durchführen wollen. Dabei stellt sich das Problem, dass häufig keine einheitlich definierten Industriestandards existieren, so dass es zu unterschiedlichen Datenbeschreibungen kommen kann. Dasselbe Problem tritt auch häufig innerhalb größerer Unternehmen auf, wo verschieden Abteilungen unterschiedliche Konventionen, Namensgebungen oder Definitionen von Daten verwenden. [Vogel, et al., 2008]

Da dieses Problem häufig auftritt, und auch schon seit Langem besteht, wurden mittlerweile mehrere unterschiedliche Lösungsansätze entwickelt. Einer davon wäre der verbreitete EDI-Standard (Electronic Data Interchange), welcher den elektronischen Austausch strukturierter Daten zwischen Unternehmen ermöglicht. Da der EDI-Standard allerdings schon etwas in die Jahre gekommen ist, ergeben sich teilweise erhebliche Nachteile, da er relativ komplex, schwer erweiterbar und auch eine relativ teure Netzwerkinfrastruktur benötigte. Er entspricht daher nicht mehr den heutigen Anforderungen für Datenaustausch. [Vogel, et al., 2008]

Als heutiger Standard für den Austausch von strukturierten Daten lässt sich das XML Format nennen, unterstützt durch beispielsweise XML-Schema und/oder XSLT Transformationen. Folgend werden diese Standards bzw. Technologien beschrieben.

2.4.1 XML

XML (Extensible Markup Language) wurde mit dem Ziel entwickelt einen flexiblen, erweiterbaren und einfachen Standard für strukturierten Datenaustausch im Internet zur Verfügung zu stellen. Wobei genau betrachtet XML selbst kein Standard für oben genannten Datenaustausch von Unternehmen darstellt, sondern zur Definition von XML-basierten Austauschformaten dient. Weiters ist es möglich, proprietäre Datenformate in das XML Format zu transformieren und auch umgekehrt. Dies wird häufig bei der Ein- bzw. Anbindung von Alt-Systemen oder alten Datenbeständen verwendet. Das XML Format hat mittlerweile eine große Verbreitung auf vielen unterschiedlichen Plattformen, Sprachen und Systemen, was die Integration erleichtert und somit einen großen Vorteil darstellt. [Vogel, et al., 2008]

Es lassen sich zwei Haupteinsatzgebiete von XML definieren. Einerseits die layoutunabhängige Beschreibung von Dokumenten, man spricht vom Single-Source-Publishing, andererseits den oben erwähnten Einsatz als universelles Datenaustauschformat. Ein Beispiel für Single-Source-Publishing wäre, Dokumente für mehrere unterschiedliche Medien oder Zielgruppen bereitzustellen. Anstatt für unterschiedliche Adressaten jeweils eigene Dokumente zu erstellen, sollten Inhalt, Struktur und Layout getrennt erstellt werden. Der Inhalt wird nicht wie sonst üblich mit Formatierungsinformationen (Layout) abgespeichert, sondern enthält nur Strukturinformationen. Somit definiert man einmalig den Inhalt, und je nach Anwendung unterschiedliche Layouts, mit Hilfe derer man anschließend Dokumente erstellt. [Becher, 2009]

2.4.2 XML Schema

XML Schema ermöglicht das Validieren von XML Dateien. Sie gelten als Nachfolger von DTD (Document Type Definition), welche zur Strukturdefinition für eine Klasse von Dokumenten eingesetzt werden. Es wird nicht nur die Wohlgeformtheit, das heißt die syntaktische Korrektheit, eines XML-Dokuments validiert, sondern es kann mit Hilfe von XML Schema festgelegt werden, welche Elemente innerhalb eines XML-Dokuments erlaubt sind, wie diese verschachtelt werden können und welche Attribute für diese Elemente verwendet werden können oder müssen. XML Schemata werden selbst in XML ausgedrückt und sind den Code betreffend umfangreicher als DTD. Viele moderne XML Editoren ermöglichen daher eine grafische Visualisierung zusätzlich zur herkömmlichen Code Ansicht. [Becher, 2009]

Abbildung 3 und Abbildung 4 zeigen eine beispielhafte XML Schema Datei (XSD Datei) in Code Ansicht, bzw. der grafischen Visualisierung des XML Editors Oxygen.

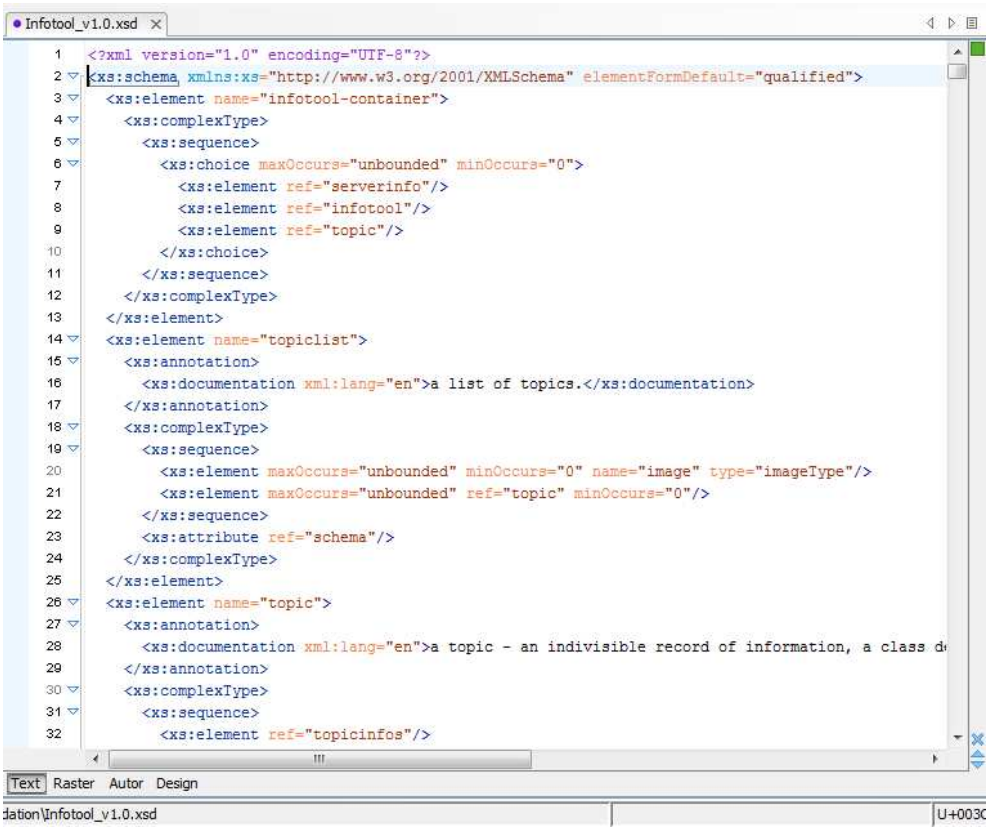


Abbildung 3: Code Ansicht eines XML Schemas - Oxygen XML Editor

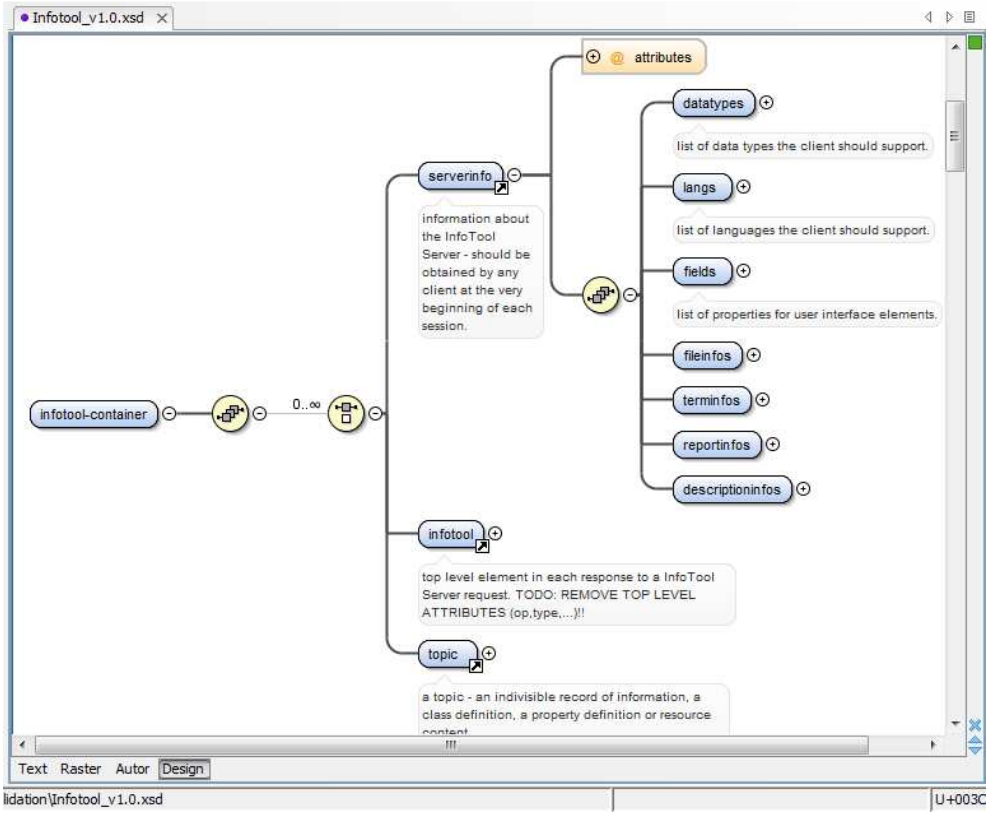


Abbildung 4: Grafische Visualisierung eines XML Schemas - Oxygen XML Editor

In Kapitel 5.3.2 werden XML Schemata verwendet, um das zugrunde liegende Datenmodell zu beschreiben und die Schnittstellen zum Server zu definieren.

2.4.3 XSLT Transformation

XSLT (eXtensible Stylesheet Language Transformations) dient zur Transformation von XML Daten in andere Formen, wie Text oder HTML, oder ein anderes XML Format. Ein geeigneter XSLT-Prozessor führt mit Hilfe von einem oder mehreren XSLT-Stylesheets solche Format-Transformationen durch. [Burke, 2001]

Zu beachten ist, dass sowohl der Begriff XSLT, als auch XSLT Transformationen verwendet wird, obwohl das „T“ in XSLT für Transformations steht. In der Literatur werden beide Begriffe verwendet, hier in dieser Arbeit wird grundsätzlich XSLT Transformationen verwendet.

Nicht zu verwechseln ist XSLT mit XSL-Formatierungsobjekten (XSL-FO), welches wie XSLT Transformationen eine ergänzende Technologie der Extensible Stylesheet Language (XSL) darstellt. XSL-FO ist eine Sprache zur Definition von Formatierungen, wie beispielsweise Fonts oder Seitenlayout. [Burke, 2001]

Ein XSLT-Prozessor ist eine Anwendung, die ein oder mehrere XSLT-Stylesheets auf eine XML-Datenquelle anwendet. Es werden jedoch nicht die Originaldaten modifiziert, sondern das Ergebnis in einen sogenannten Ereignisbaum kopiert. Anschließend existieren mehrere Möglichkeiten diesen Ereignisbaum weiterzuverarbeiten: Dieser kann in eine statische Datei ausgegeben, als Output-Stream verwendet werden oder für eine weitere Transformation an einen XSLT-Prozessor übergeben werden. [Burke, 2001]

Abbildung 5 veranschaulicht das oben beschriebene Prinzip.

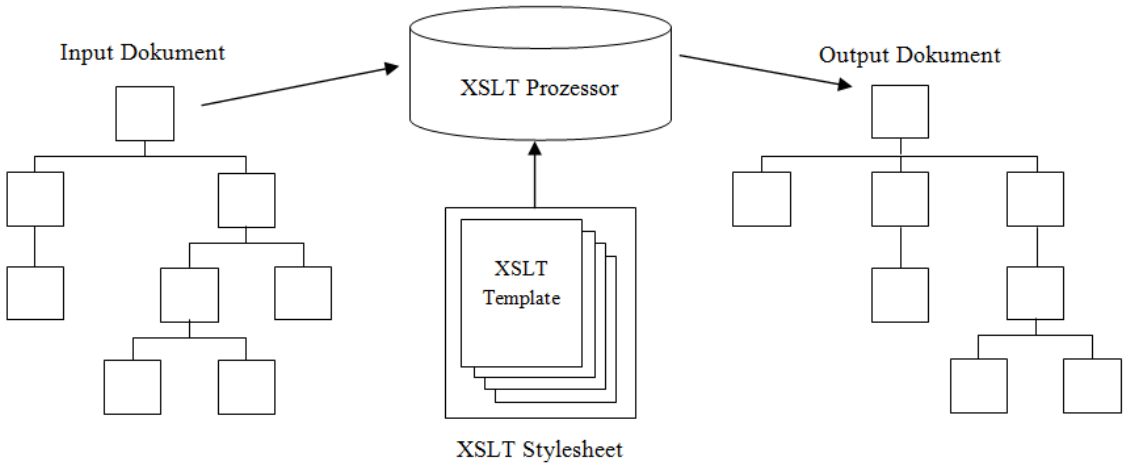


Abbildung 5: XSLT Transformation [Cagle, Corning, & Diamond, 2001]

2.5 JSON

JSON (JavaScript Object Notation) ist ein einfaches Datenaustauschformat. Die Vorteile liegen darin, dass es nicht nur für Menschen einfach zu verstehen und lesbar ist, sondern auch für Maschinen interpretierbar ist. Das heißt zum Analysieren von Datenstrukturen Parsen, sowie Generieren. JSON ist ein Textformat und komplett unabhängig von Programmiersprachen. Es basiert jedoch auf einer Untermenge der JavaScript Programmiersprache. [Json.org, 2011]

Grundsätzlich baut JSON auf zwei Strukturen auf: Einerseits auf Name/Wert Paare, welche beispielsweise mit einer Hash-Tabelle aus Java vergleichbar sind, andererseits auf geordnete Listen von Werten, vergleichbar mit einem Vector aus Java. [Json.org, 2011]

JSON unterstützt grundlegende Datentypen wie String, Number, Boolean und ist sehr stark verbreitet. Es gibt mittlerweile Bibliotheken in vielen gängigen Programmiersprachen. Da JSON auf der nativen JavaScript Notation basiert, ist es ideal für Clients im Web-Browser und wird daher auch für den in dieser Arbeit entwickelten Client verwendet.

Abbildung 6 zeigt ein beispielhaftes JSON Objekt. Das übergeordnete Objekt *serverinfo* enthält weitere Objekte und Arrays. *serverName*, *serverVersion* sind Beispiele für oben genannte Name/Wert Paare, während *langs* oder *datatypes* Beispiele für geordnete Listen von Werten darstellen.

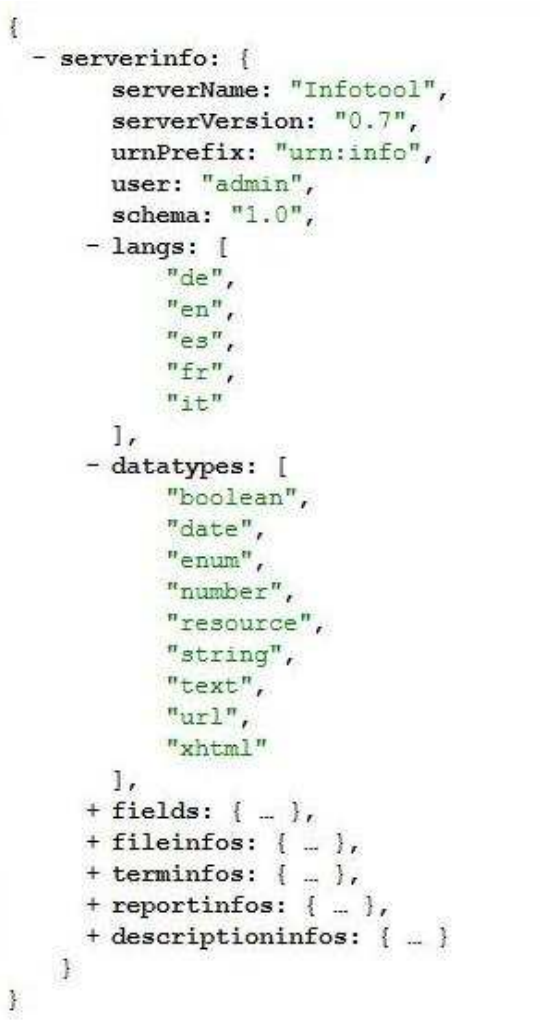


Abbildung 6: Beispiel für ein JSON Objekt

2.6 RDF

RDF (Resource Description Framework) ist eine vielseitige Technologie, die in den unterschiedlichsten Anwendungsfällen eingesetzt werden kann. RDF wird oft in Zusammenhang mit dem Semantic Web verwendet. Laut dem W3C (World Wide Web Consortium) ist RDF als Sprache zur Unterstützung des Semantic Web gedacht, vergleichbar mit HTML als Sprache zur Unterstützung des „ursprünglichen“ Webs. RDF dient als formale Sprache zur Beschreibung von Ressourcen oder Metadaten bzw. strukturierter Information. Es dient hauptsächlich zum Austausch von (Meta-)Daten im Web oder zwischen Anwendungen. Der Unterschied zu HTML oder XML besteht darin, dass mit Hilfe von RDF nicht die Darstellung von Information oder Dokumenten unterstützt werden soll, sondern die beinhalteten Informationen zur Kombination und Weiterverarbeitung strukturiert werden sollen. RDF basiert auf einfachen in RDF Tripeln abgelegten Informationen, die als graph-orientiertes Datenschema angesehen werden können. [Powers, 2003] [Hitzler, Krötzsch, Rudolph, & York, 2007]

2.6.1 RDF Tripel

Unter RDF Tripel versteht man das grundlegende Konzept mit dem in RDF Information strukturiert wird.

```
<subject> HAS <predicate> <object>
```

Das folgende Beispiel veranschaulicht das Konzept auf einfache Weise:

Das Buch „Practical RDF“ hat die ISBN 0-596-00263

Mit RDF kann man dieses Statement als RDF Tripel darstellen, eine einfache Technik dazu ist folgende, wobei *Practical RDF* das Subjekt darstellt, die *ISBN* ein Prädikat und *„0-596-00263“*. das Objekt:

```
<subject> <predicate> <object> → <Practical RDF> <isbn> <”0-596-
00263”>
```

Diese Repräsentation eines RDF Tripels ist nur eine von mehreren Möglichkeiten wie RDF Daten dargestellt werden. Eine häufig angewendete Alternative ist über gerichtete Graphen (siehe nächstes Kapitel). Es können jedoch 4 grundsätzliche Regeln definiert werden:

- Jedes RDF Tripel besteht aus Subjekt, Prädikat und Objekt
- Jedes RDF Tripel ist vollständig und eindeutig
- Ein RDF Tripel ist ein 3-tuple, welches aus Subjekt, Prädikat und Objekt besteht, wobei ein Subjekt ein *uriref* oder *blank node* darstellen soll, ein Prädikat ein *uriref* und ein Objekt ein *uriref*, *blank node* oder *literal*. Ein *uriref node* besteht aus einem Uniform Resource Identifier (URI) was eine spezifische eindeutige Identifikation des Knoten darstellt. Als *blank node* wird ein Knoten verstanden, der keinen URI hat. Ein *literal* besteht aus 3 Teilen – einem String, einem optionalen Language Tag und einem Datentyp. Literale stellen immer nur Objekte dar, keine Subjekte oder Prädikate.
- Jedes RDF Tripel kann mit anderen RDF Tripel kombiniert oder zusammengefügt werden, allerdings muss es seine einzigartige Bedeutung behalten.

Der letzte Punkt bedeutet, dass egal wie komplex ein RDF Graph ist, er sich nur aus einer Kombination eindeutiger, einfacher RDF Tripel ergibt, die alle aus einem Subjekt, Prädikat und Objekt bestehen. [Powers, 2003]

2.6.2 RDF Graph

Eine andere Darstellung für RDF Tripel als

```
<subject> <predicate> <object> → <Practical RDF> <isbn> <”0-596-
00263”>
```

sind RDF Graphen. Nimmt man obiges Beispiel ließe sich das RDF Tripel auch als einfacher gerichteter Graph darstellen. Abbildung 7 veranschaulicht dies. [Hitzler, Krötzsch, Rudolph, & York, 2007]

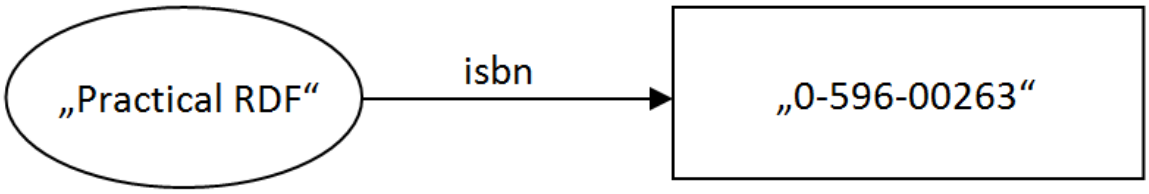


Abbildung 7: Beschreibung einer Beziehung zwischen Subjekt, Prädikat, Objekt als Graph

Wie bereits erläutert basiert RDF auf einem einfachen graph-orientierten Datenschema. Somit beschreibt ein typisches RDF Dokument einen gerichteten Graphen. Dieser besteht aus Knoten und gerichteten Kanten („Pfeile“). Knoten und Kanten müssen mit eindeutigen Bezeichnern definiert werden. [Hitzler, Krötzsch, Rudolph, & York, 2007]

Dadurch lassen sich Zusammenhänge aber keine Klassenhierarchien modellieren. Annahme: Man hat die Information das *Informatik I* vom Typ *Lehrbuch* ist. Man möchte nun abbilden, dass *Informatik I* auch vom Typ *Buch* ist. Mit Hilfe von RDF lässt sich diese generelle Information nicht modellieren. Erst das RDFS-Vokabular (siehe nächstes Kapitel) beinhaltet vordefinierte Möglichkeiten um derartige Informationen abzubilden. Konkret für diesen Fall über *rdfs:subClassOf*. Dazu definiert man einfach das RDF Tripel

```
<Lehrbuch> <rdfs:subClassOf> <Buch>
```

[Hitzler, Krötzsch, Rudolph, & York, 2007]

Ein kurzer Überblick zu RDFS befindet sich im nächsten Kapitel.

2.6.3 RDFS

Wie im vorhergehenden Kapitel erwähnt, lassen sich mit RDF alleine zwar Beziehungen, aber keine Klassenhierarchien modellieren. Dazu bietet das RDF Schema (RDFS) Vokabular einige vordefinierte Möglichkeiten um derartige Informationen abzubilden. Man spricht von sogenannten terminologischen Wissen oder auch Schemawissen. Wobei RDFS selbst auf einem RDF Vokabular basiert, somit jedes RDFS Dokument ein syntaktisch korrektes RDF Dokument darstellt. [Hitzler, Krötzsch, Rudolph, & York, 2007]

RDFS wird durch die Möglichkeit Schemawissen zu spezifizieren auch als Wissensrepräsentations- oder Ontologiesprache eingeordnet. Dadurch ist es möglich, eine Vielzahl von semantischen Abhängigkeiten welche in einer Domäne vorkommen zu beschreiben. [Hitzler, Krötzsch, Rudolph, & York, 2007]

Durch Verwendung von *rdfs:subClassOf* lassen sich ganze Klassenhierarchien aufbauen wie zum Beispiel:

```
<Buch> <rdfs:subClassOf> <Printmedium>

<Zeitschrift> <rdfs:subClassOf> <Printmedium>

<Lehrbuch> <rdfs:subClassOf> <Buch>
```

Durch die festgelegten Tripel lassen sich somit Beziehungen herstellen. Zum Beispiel kann man automatisch schlussfolgern, dass Lehrbuch eine Unterklasse von Printmedium ist, ohne dass man dies per eigenen Tripel festlegen muss. Definiert man nun einige Tripel lässt sich dann ein RDF Graph aufbauen, der sich einfach abfragen und durchsuchen lässt bzw. aus dem man Schlussfolgerungen ableiten kann. Diese Art der Wissensmodellierung ist vergleichsweise intuitiv und entspricht sehr nahe dem menschlichen Denken in Begriffen, lässt sich aber wie erwähnt einfach automatisch auswerten und durchsuchen. [Hitzler, Krötzsch, Rudolph, & York, 2007]

Weitere Sprachelemente (neben *rdfs:subClassOf*) von RDFS sind [Allemang & Hendler, 2008]

- *rdfs:subPropertyOf* legt fest, dass eine Eigenschaften für ein SubProperty auch für das SuperProperty gilt. Aus

  ```
  <Scherr> <hat Werkvertrag mit> <Siemens AG>
  ```

  und

  ```
  <hat Werkvertrag mit> <rdfs:subPropertyOf> <arbeitet für>
  ```

  kann man schließen:

  ```
  <Scherr> <arbeitet für> <Siemens AG>
  ```

- *rdfs:domain* legt Klasse(n) des Subjekts in einer Beziehung fest, aus

  ```
  <Scherr> <arbeitet für> <Siemens AG>
  ```

  und

  ```
  <arbeitet für> <rdfs:domain> <Firma>
  ```

  kann man schließen

  ```
  <Siemens AG> <rdf:type> <Firma>
  ```

- *rdfs:range*, legt Klasse oder Datentyp des Objekts in einer Beziehung fest, aus

  `<arbeitet für> <rdfs:range> <Lohnempfänger>`

  und

  `<Scherr> <arbeitet für> <Siemens AG>`

  kann man schliessen:

  `<Scherr> <rdf:type> <Lohnempfänger>`

- *rdfs:label*, Standard Property für einen Namen bzw. Bezeichner
- *rdfs:comment*, Standard Property für einen Kommentar oder Kurzbeschreibung

3 Analyse der Ausgangssituation

Die betreuenden Mitarbeiter der Firma Siemens arbeiten in einem Projekt der Entwicklungsabteilung I IA AS R&D DH A in Amberg (Deutschland) in welchem Safety-Firmware für fehlersichere Peripherie-Baugruppen des Simatic Automatisierungssystems entwickelt wird.

Bei einem gemeinsamen Workshop aller Entwickler ergab sich Optimierungspotential im Umgang mit Wissen bzw. Know-How im Projekt. Im Detail wurden folgende Sachverhalte identifiziert:

- Wissen ist teilweise nur in den Köpfen Einzelner vorhanden
- Wissen wird oft nur zwischen einzelnen Entwicklern (z.B. zwischen Hardware und Firmware Entwickler) ausgetauscht ohne dies zu dokumentieren
- Allgemeine Projektinformationen oder Verfahrensanleitungen sind manchmal nicht entsprechend dokumentiert
- Es befindet sich sehr viel Information in unterschiedlichsten Einzeldokumenten
- Es befindet sich sehr viel Information in unterschiedlichsten Tools die teils schwer zugänglich und/oder kombinierbar sind

Zur Behebung dieser Schwächen sollte ein Wissensmanagement-Tool entwickelt werden.

Weiters sollte dieses Tool den Dokumentationsprozess insofern unterstützen, indem sich zum Beispiel Dokumente oder Reports aus den Informationen generieren lassen.

3.1 Problemstellung

Ein Beispiel für Wissensmanagement ist das Requirement Engineering in Software Projekten. Es ist heutzutage immer noch mehrheitlich üblich die Anforderungsanalyse als Word-Dokument zu verfassen. Einzelne Anforderungen (Requirements) sind darin beispielsweise als einzelner numerierter Abschnitt oder auch als eine Tabellenzeile enthalten. Eine automatisierte Verarbeitung ist somit nicht mehr einfach, oder auch gar nicht mehr möglich.

Angenommen man hätte auch ein Architektur-Dokument, welches im Detail ein Software Design beschreibt. Möchte man nun wissen, welche Stellen im Design eine bestimmte Anforderung abdecken, müsste man manuell die Dokumente untersuchen, ob irgendwo textuelle Verweise von Architektur- auf das Anforderungsdokument existieren. In diesem Fall würden auch keine Metadaten auf Dokumentebene weiterhelfen können.

Eine Verbesserung der Situation lässt sich durch den Einsatz eines Requirements-Tools erzielen, welches jede einzelne Anforderung als eindeutig identifizierbaren Datensatz enthält. Für das Design der Software verwendet man gleichermaßen ein Tool, das jedes Artefakt mit einem eindeutigen Bezeichner versieht und womöglich auch mit einem Requirement (z.B. durch Angabe einer ID) in Beziehung setzen kann. Einzelne Daten liegen in maschinenlesbarer Form vor und können somit verarbeitet werden.

Die Antwort auf die vorher formulierte Frage, welche Stellen im Design welche Anforderungen abdecken, könnte somit auf Knopfdruck durch die automatisierte Generierung eines Reports erhalten werden. Ein weiteres Problem mag sein, dass die Informationen (Daten) in den verwendeten Tools nicht einfach extrahierbar (oder z.B. wieder nur als Word-Dokument), oder die extrahierten Informationen nicht einfach kombinierbar sind.

3.2 Informationsquellen

Zu Beginn wurden alle im Projekt verfügbaren Informations- bzw. Know How Quellen identifiziert und analysiert.

3.2.1 Entwickler

Wie schon eingangs erwähnt, existiert viel Information nur in den Köpfen der Entwickler. Eine große Herausforderung wird darin liegen, diese Informationen strukturiert im entwickelten Wissensmanagement-Tool abzulegen. Aber auch manche Ablagen sind als langlebige Projektinformation weniger geeignet. Beispiele könnten sein:

- Hardware-Verschaltungen mit Bleistift auf Papier skizziert
- Blockschaltbilder in Microsoft VISIO erstellt und irgendwo am Projektlaufwerk abgelegt
- Konzepte für Software-Tests mit Microsoft Word verfasst und auf der eigenen Festplatte abgelegt

3.2.2 Microsoft Word

Projektinformation die in Word Dokumenten liegt ist derzeit kaum dokumentiert oder wiederauffindbar. Weitere Nachteile des Word Formats: Keine einheitliche Struktur, sehr viele Formatelemente und kaum Möglichkeiten die Informationen vernünftig zu extrahieren. Abbildung 8 zeigt eine Feature Spezifikation in Microsoft Word. Die vorliegende Information lässt sich schwer extrahieren, da keine einheitliche Struktur vorgegeben ist.

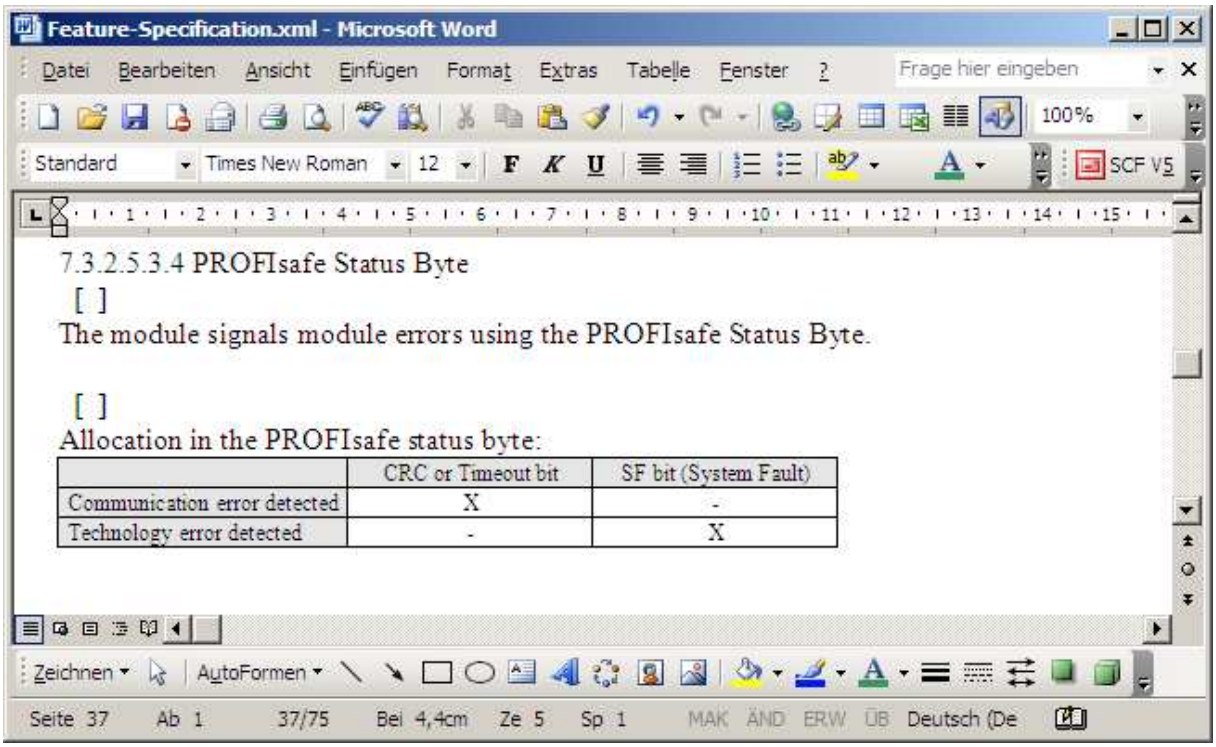


Abbildung 8: Feature Spezifikation in Microsoft Word

3.2.3 Microsoft Excel

In Excel vorliegende Information lässt sich in gängige Formate (CSV, XML) extrahieren. Hier sollte es keine Probleme geben diese zu extrahieren und in ein einheitliches Format zu bringen. Abbildung 9 zeigt eine Unit Test Spezifikation in Microsoft Excel. Daraus werden zum Beispiel Debugger Test Skripte generiert.

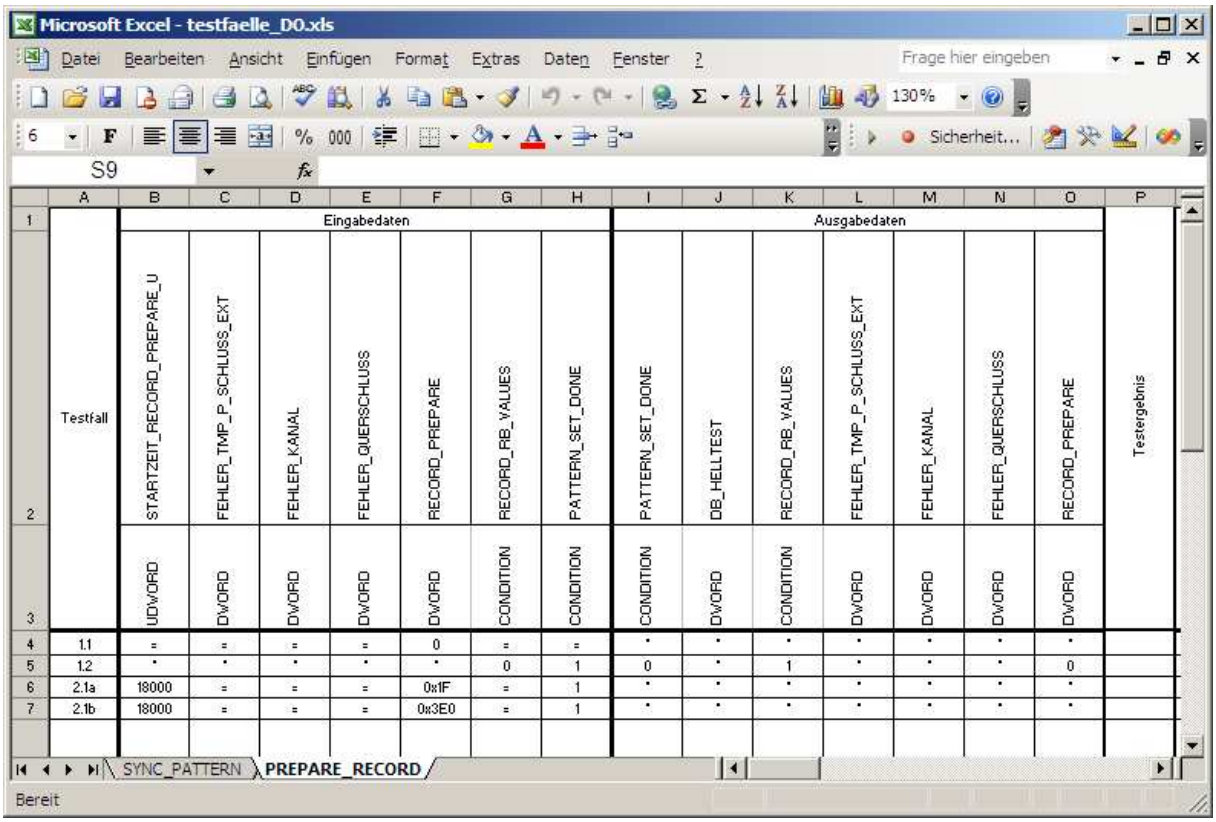


Abbildung 9: Unit Test Spezifikation in Microsoft Excel

3.2.4 Microsoft Access

Ein Teil der Projektinformationen wurde mit Microsoft Access abgebildet. Access bietet einige Vorteile wie SQL Abfragen oder XML Export. Es ist aber auch nur für spezielle Information möglich, für die sich ein geeignetes Datenbank Schema definieren lässt. Zudem entsteht hoher Aufwand für die Entwicklung, Einrichtung und Wartung der Datenbank, Formulare und Schnittstellen. Es existiert ein eigens entwickeltes Tool für Failure Modes und Effects Analysis (SystemFMEA) von sicherheitsrelevanten Systemen als Datenbankanwendung (siehe Abbildung 10). Eine rudimentäre Benutzeroberfläche konnte noch mit Mitteln des Tools definiert werden. Die Ausgabe der Information als Report, dessen Inhalt und Layout den Vorstellungen der Entwickler entsprochen hätte, war schon nicht mehr einfach möglich. Letztendlich musste die Information aus der Access Datenbank nach XML exportiert, mittels XSLT Transformation nach XHTML und nachfolgend mittels PDF Konvertierung in ein Dokument transformiert werden.

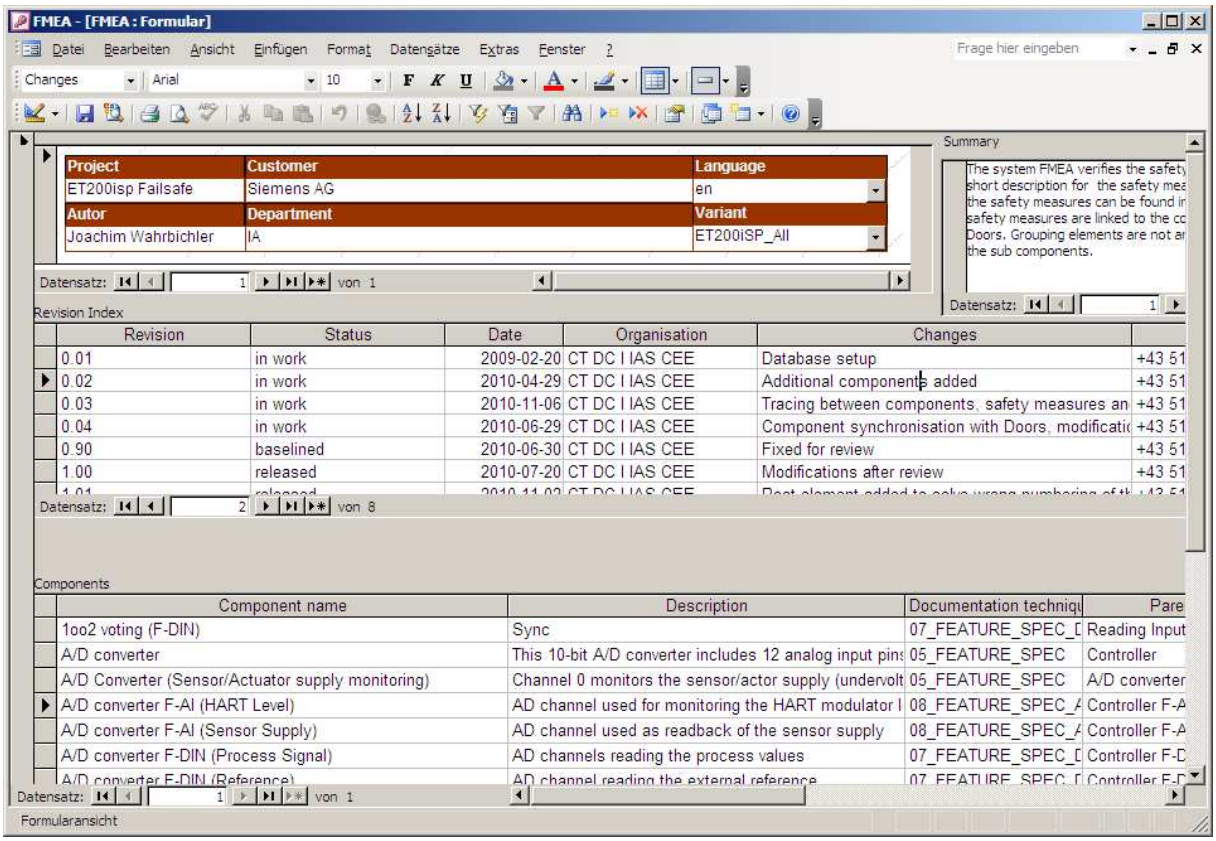


Abbildung 10: System FMEA Tool in Microsoft Access

3.2.5 Telelogic DOORS

Mit DOORS wird bereits ein bekanntes, hochspezialisiertes Tool zum Requirements Engineering eingesetzt. Der große Nachteil ist allerdings, dass es keinen brauchbaren Import / Export bietet und kaum anpassbar ist.

Um dennoch eine vernünftige Dokumentation zur Weitergabe an Zertifizierungs-Behörden anbieten zu können, musste ein recht komplizierter Prozess (halbautomatisiert) durchlaufen werden:

- zunächst wurden die einzelnen Module (geteilt in Requirements-, Feature- und Testcase-Specification) aus DOORS im Format Microsoft Word exportiert
- danach mussten die Word Dokumente mittels Skript nach XML konvertiert werden
- und am Ende konnten die Informationen mit einer XSLT Transformation in eine konsistente Dokumentation transformiert werden. Auch die geforderte Traceability zwischen Requirements, Features und Testcases wurde durch automatisches Generieren von Hyperlinks in diesem Schritt machbar

Der letzte Schritt der XSLT Transformation ermöglichte es sogar recht einfach weitere Information von anderen Tools, wie zum Beispiel Information im XML Format zur SystemFMEA, zu integrieren.

Insgesamt war aber der Aufwand sowohl bei der Eingabe von Information in die diversen Tools als auch beim abschließenden Extrahieren und Zusammenfassen der Information unangemessen hoch.

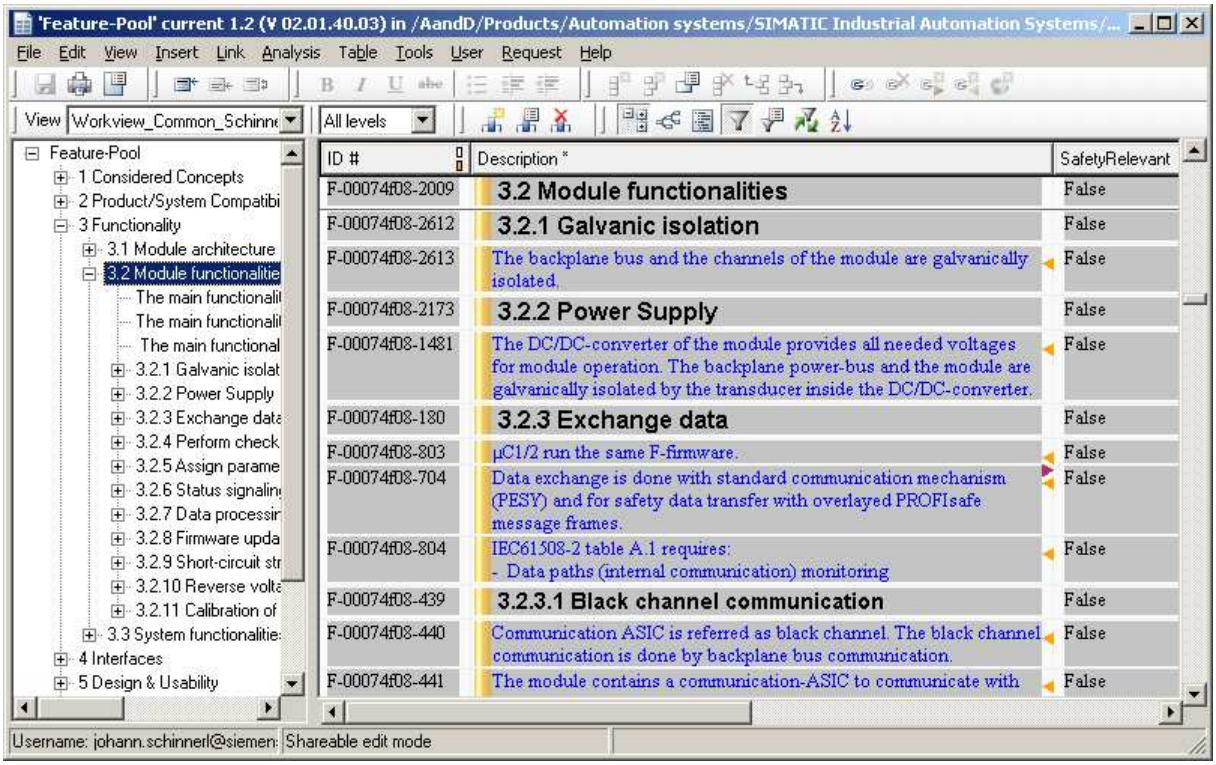


Abbildung 11: Requirements Engineering in DOORS

3.2.6 Weitere

Als Software Design Tools werden Enterprise Architect [Enterprise Architect, 2011] und Telelogic (IBM) Statemate verwendet. Da es sich um ein UML Tool handelt, ist bei Enterprise Architect ein XML Export in der Form des XMI Standards (XML Metadata Interchange) einfach möglich. Bei Statemate konnte erst ein selbst geschriebener Parser des proprietären Dateiformats dem Tool Information entlocken.

Teilweise wurden Tools zur Ablage einfacher Information auch von den Entwicklern zur Gänze selbst geschrieben. Zum Beispiel existieren kleinere Web-Applikationen für die Eingabe von Datensätzen für die Änderungsdokumentationen von Software-Modellen und für die Eingabe von Testergebnissen zu Integrationstests. Hier wurde generell darauf geachtet als Informationsablage am Server das XML Format zu wählen damit die Informationen einfach verarbeitet werden können.

4 Anforderungen an ein Wissensmanagement-Tool

Dieses Kapitel beschäftigt sich allgemein mit Anforderungen an ein Wissensmanagement-Tool. Im Speziellen mit aufgrund der Literaturrecherche und Analyse der Ausgangssituation erhaltenen Erkenntnisse definierte Anforderungen an das zu entwickelnde Wissensmanagement-Tool.

Die Anforderungen bzw. die Umsetzung des Wissensmanagement-Tools wurde in drei Teilbereiche gegliedert:

4.1 Tool zum Management von Begriffsdefinitionen, Informationseinheiten und Beziehungen

In der ersten Phase sollte eine gewisse Grundfunktionalität definiert werden, die das Eingeben, Abfragen und Verknüpfen von Informationen und Beziehungen ermöglicht.

4.1.1 Technologie

Der technologische Aspekt wird in dieser Phase noch nicht genau beleuchtet. Es sollten grundlegende Anforderungen definiert werden, einzige Voraussetzung ist, dass eine zentrale Stelle (Server) existiert, an der sämtliche Daten gespeichert, verwaltet und abgefragt werden können. Der Zugriff soll über einzelne verteilte Clients möglich sein. Daraus ergibt sich schon die Wahl einer Client-Server Architektur. Genaueres zur technologischen Umsetzung findet sich dann ab Kapitel 5.

4.1.2 Eingabe von Informationen

Die Eingabe von Informationen soll über einen standardkonformen Web-Browser möglich sein, ohne das zusätzliche technische Hilfsmittel benötigt werden. Die wesentliche Grundlage für das Wissensmanagement im Projekt ist die Erfassung möglichst vieler Begriffe, welche im Projekt verwendet werden, sowie die Ablage dieser Begriffe an zentraler Stelle (Server), um sie später zu finden und/oder zu verknüpfen.

Jede Informationseinheit bzw. Begriff soll über eine eindeutige ID verfügen, um eindeutig identifiziert werden zu können, bzw. eindeutig darauf verweisen zu können. Dabei sollte man sich an das URI-Schema orientieren (siehe Kapitel 2.3.6). Weiters sind entsprechende Metadaten notwendig, wie eine Bezeichnung, Kurzbeschreibung, oder beliebig viele Schlagworte (Tags). Weitere mögliche Metadaten wären Ersteller, Letztbearbeiter, oder Zugriffszeitpunkte (erstellt, zuletzt bearbeitet).

Es sollte weiters möglich sein, beschreibenden Text mit Formatierung in einem Format zu erfassen, welches auf jeden Fall von einem XML-Parser verarbeitet werden kann. Vor allem die Möglichkeit zur Eingabe von (auch komplexeren) Tabellen und Bildern sollte existieren. Dies sollte nach dem Prinzip WYSIWYG (What You See Is What You Get) ähnlich wie in einem herkömmlichen Textverarbeitungsprogramm möglich sein. Eine unterstützte Texteingabe mit Formatierungskürzel wie beispielsweise in einem gängigen Wiki-System erfüllt diese Anforderung nicht und ist daher von vornherein auszuschließen. Diese Anforderung ergibt sich aus der späteren Phase zur Generierung von Reports und/oder

Dokumenten. Als Beispiel sei hier das in Abbildung 12 gezeigte Firefox Plugin XStandard [XStandard, 2010] erwähnt, welches sich bereits in früheren Projekten bewährt hat und standardkonformes XHTML 1.1 erzeugt.

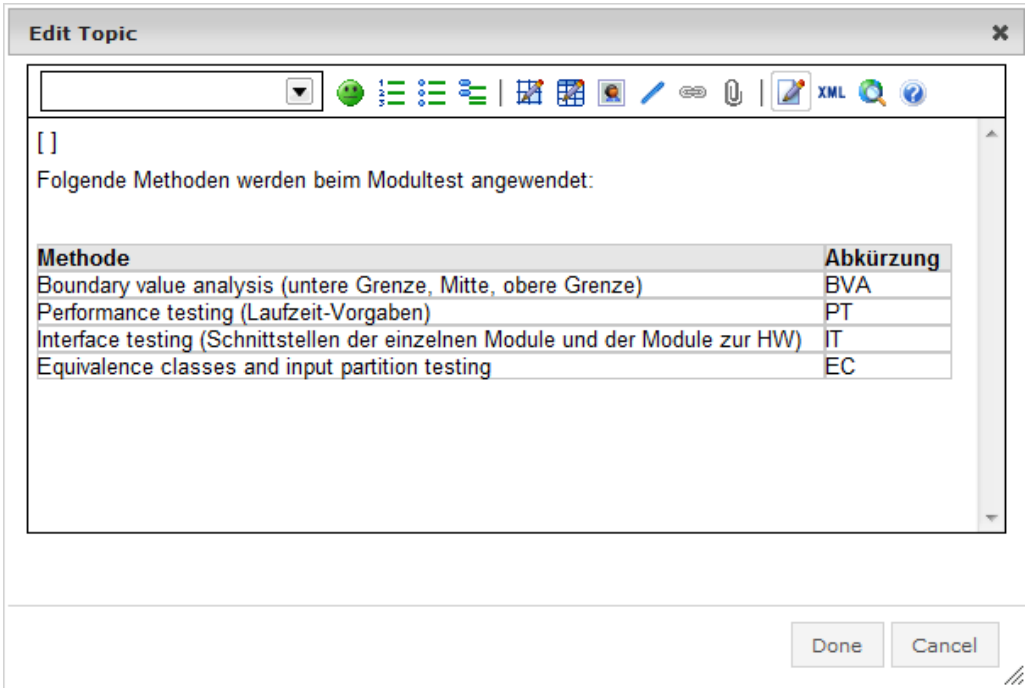


Abbildung 12: XStandard Editor Plugin

4.1.3 Verknüpfen von Informationen

Sämtliche Informationen sollen mit Hilfe von Beziehungen verknüpfbar sein. Das angewandte Datenmodell sollte an RDF (Resource Description Framework, siehe Kapitel 2.6) angelehnt sein. Das heißt, das Datenmodell basiert auf Tripel (Subjekt, Prädikat, Objekt), wobei Subjekte über ein Prädikat mit Objekten verknüpft werden. Objekte können wiederum selbst eine Informationseinheit darstellen, oder einen einfachen Wert wie Text, String, Zahl oder Datum annehmen.

Weiters soll es analog zu RDFS (siehe Kapitel 2.6.3) möglich sein, Informationseinheiten in Klassen einzuordnen. Dadurch lassen sich Informationen einfach kategorisieren und gewisse Sätze von Eigenschaften, die die jeweils verknüpften Klassen besitzen einfach zuordnen. Ein gewisser Satz an allgemeinen Beziehungen sollte bereits definiert werden, wie Domain/Range (hat/ist) Beziehungen. Das heißt, Beziehungen sollten teilweise typisiert sein, ähnlich wie die *rdfs:domain* oder *rdfs:range* Mechanismen in RDFS. Dadurch ist es möglich beim Editieren von Information bereits eine Auswahl passender Alternativen zu präsentieren.

Eine Verwendung der RDF-Syntax oder vollständige Implementierung des Standards ist jedoch nicht nötig, sinnvolle Erweiterungen sind aber erwünscht. Als Beispiel wären hier Unterstützung von Default-Werten oder Enumerations (Aufzählungen) zu nennen.

Analog zur Kategorisierung durch Klassen, sollten Informationseinheiten auch bestimmten Namensräumen (Namespaces) oder Arbeitsräumen (Workspaces) zugeordnet werden können, um eine Zuordnung zu gewissen Domänen oder Projekten zu ermöglichen.

4.1.4 Abfrage von Information

Information soll einfach wieder zu finden sein. Einerseits über eine „Browse-Ansicht“, welche alle verfügbaren Informationen gegliedert zum Beispiel nach Namensräumen, Domänen oder ähnlichen anzeigt. Andererseits sollte über die Eingabe eines Suchbegriffs Information eindeutig zu finden sein. Dies kann über die eindeutige ID einer Informationseinheit geschehen, oder mit Hilfe der vergebenen Metadaten wie Bezeichnung, Kurzbeschreibung oder Schlagworten (Tags) geschehen. Die Suchergebnisse sollten sich dann als Liste der gefundenen Informationseinheiten präsentieren, inklusive der beschreibenden Kurztexte.

Die verknüpften Beziehungen sollten ebenfalls in die Suche einbezogen werden können, gegebenenfalls auch sofort dargestellt werden können.

Die Suchmöglichkeit muss besonders einfach gestaltet werden, da nicht nur Entwickler im Projekt, sondern auch ein erweiterter Personenkreis (Projektleiter, Marketing, Systemtest, etc.) auf das Wissensmanagement-Tool zugreifen.

Eine weitere Anforderung ist die kontextabhängige Suche, indem das gerade bearbeitete Projekt, ausgewählte Namensräume oder Domänen, sowie die Rolle des jeweils Suchenden in die Suchanfrage einfließen, bzw. die Suchergebnisse entsprechend einschränken.

Es sollten geeignete Filter für die Suche eingesetzt werden, um diese Anforderungen zu erfüllen.

4.1.5 Weitere wesentliche Anforderungen

Es müssen unterschiedliche Sprachen unterstützt werden, da Siemens als global agierendes Unternehmen derzeit hauptsächlich Deutsch und/oder Englisch in Entwicklungsprojekten verwendet.

Bilder und externe Dateien wie PDF Datenblätter oder Microsoft Office Dokumente sollten in das Wissensmanagement-Tool integriert werden können. Es sollten dann ebenfalls entsprechende Metadaten teilweise automatisch angelegt und auch vergeben werden können. Als Beispiel wäre hier ebenfalls das Versehen mit Schlagworten (Tags) zu nennen, dadurch lassen sich Dokumente einfacher im System wieder finden.

Es muss eine Schnittstelle geben, um externe Information die sich in anderen Tools befindet (zum Beispiel DOORS) zu importieren. Dazu sollte ein einheitliches XML-Schema definiert werden, in das diese Daten gebracht werden, um sie anschließend in das Wissensmanagement-Tool zu importieren.

Weiters muss eine einfache Zugriffskontrolle implementiert sein, um den Zugriff von nicht autorisierten Personen zu verhindern.

4.2 Integration projekt-spezifischer Information

Aufbauend auf das allgemein verwendbare Tool des vorhergehenden Kapitels soll projekt-spezifische Information in das Wissensmanagement-Tool integriert werden.

Konkret soll für das Safety - Projekt „Entwicklung sicherheitsgerichteter Firmware für SIMATIC Peripheriebaugruppen“ eine projektspezifische Erweiterung der bis jetzt allgemein gehaltenen Begriffsdefinitionen, Beziehungen und Klassen erfolgen.

Das heißt im Speziellen, dass mit Hilfe der in Phase 1 genannten Anforderungen projektspezifische Information angelegt werden muss, und somit die Erfassung von Entwickler-Information auf das Wissensmanagement-Tool umgestellt wird.

Als Beispiele wären hier genannt:

- Failure Modes und Effects Analysis (FMEA) wird derzeit als Microsoft Access Datenbank Anwendung geführt
- Änderungsdokumentation für Software wird derzeit über XForms Interface am Browser eingegeben und als XML-Dateien im Dateisystem abgelegt
- Einfache Projektinformation wird über XHTML-Formulare eingegeben und ebenfalls als XML-Dateien im Dateisystem abgelegt

Durch Erweiterung des in Phase 1 definierten XML-Schemas soll eine einheitliche Schnittstelle für den Import dieser Daten verfügbar sein, gleichzeitig natürlich auch als Export um gegebenenfalls im Wissensmanagement-Tool erstellte oder bearbeitete Information wieder in die vorhandenen Tools zu integrieren. Um die Daten in das gewünschte XML-Schema zu bringen, sollten geeignete XSLT Transformationen bereitgestellt werden.

4.3 Reporting, Dokumentgenerierung und Konsistenzprüfung

Interaktives Reporting am Client sollte möglich sein. Dazu sollten geeignete Schnittstellen zur Verfügung gestellt werden, mit Hilfe deren man mit Information aus dem Wissensmanagement-Tool einfach Reports oder Dokumente generieren kann.

Als Unterstützung sollten wieder XSLT Transformationen eingesetzt werden, es soll für versierte User oder Entwickler auch möglich sein, eigene XSLT Transformationen anzuwenden. Das heißt, es soll Information via Export Schnittstelle in ein geeignetes XML-Format gebracht werden, welches dann wiederum mittels XSLT Transformation in ein gewünschtes generiertes Dokument oder Report transformiert wird.

Dokumente, welche für die Zertifizierung der Firmware durch den TÜV bisher mit unterschiedlichsten XSLT Transformationen aufbauend auf die verschiedenen verwendeten Tools generiert wurden, sollten zu einem möglichst großen Teil mit Hilfe des neuen Wissensmanagement-Tools generiert werden.

4.4 Anwendungsfälle

Anhand der oben genannten Anforderungen, ergeben sich einige typische Anwendungsfälle (Use Cases), die nachfolgend beschrieben werden. Die Anwendungsfälle sind sehr allgemein und einfach gehalten und sollen noch keine Richtung in technischer Sicht vorgeben, sondern nur veranschaulichen, welche Vorgänge durch das Wissensmanagement-Tool abzudecken sind.

4.4.1 Akteure

Für die allgemein definierten Anwendungsfälle, wird hier auch ein allgemeiner Akteur definiert, genannt Benutzer. Dieser stellt einen typischen Benutzer des Wissensmanagement-Tools dar, und kann beispielsweise ein Entwickler, Projektleiter, oder Architekt sein. Eine Unterscheidung ist in dieser Phase aber noch nicht nötig, da es für die nachfolgend angeführten Anwendungsfälle keinen Unterschied macht, ob der Benutzer beispielsweise ein Entwickler oder Projektleiter ist.

4.4.2 Anlegen von Information

Use Case	Anlegen von Information
Akteure	Benutzer
Vorbedingungen	-
Interaktionsfolge	Der Benutzer legt eine neue Information an Geeignete Metadaten wie Name, Kurzbeschreibung, werden vergeben Weitere Daten die für diese Information notwendig sind, werden eingegeben
Nachbedingungen	Der Information ist angelegt und im System gespeichert
Beispiel	Ein Entwickler erstellt eine Informationseinheit zu einem Bauteil, vergibt geeignete Metadaten, sowie eine Beschreibung und technische Details für dieses Bauteil

4.4.3 Anlegen eines Begriffs/Terms

Use Case	Anlegen eines Begriffs/Terms
Akteure	Benutzer
Vorbedingungen	-

Interaktionsfolge	Der Benutzer legt einen neuen Begriff an Geeignete Metadaten wie Name, Kurzbeschreibung, werden vergeben Weitere Daten die diesen Begriff/Term beschreiben werden eingegeben (Zuordnung zu einem bestimmten Bereich, hierarchische Einordnung, etc.)
Nachbedingungen	Der Begriff/Term ist angelegt und im System gespeichert
Beispiel	Der Projektleiter legt den Simatic Term „Automation Systems“ im System an, ordnet diesem der Begriffsgruppe „Simatic“ zu und beschreibt diesen Term.

4.4.4 Anlegen von mehrsprachiger Information

Use Case	Anlegen von mehrsprachiger Information
Akteure	Benutzer
Vorbedingungen	Information ist im System vorhanden
Interaktionsfolge	Der Benutzer wählt die Information im System aus Vergabe eines Namens, sowie Kurzbeschreibung über diese Information einer weiteren Sprache Weitere Daten die diese Information beschreiben werden in einer weiteren Sprache angelegt
Nachbedingungen	Die Information ist mehrsprachig im System angelegt
Beispiel	Der Projektleiter wählt den Begriff „Automation Systems“ und legt zusätzlich zur englischen Kurzbeschreibung, auch eine deutsche Kurzbeschreibung an

4.4.5 Eingabe von strukturierten Text (XHTML)

Use Case	Eingabe von strukturierten Text (XHTML)
Akteure	Benutzer
Vorbedingungen	Information ist im System vorhanden
Interaktionsfolge	Der Benutzer wählt die Information im System aus Zusätzliche Beschreibung zu dieser Information wird über einen geeigneten XHTML Editor eingegeben und gespeichert

Nachbedingungen	Der strukturierte Text ist für die Information im System gespeichert
Beispiel	Ein Entwickler wählt ein Datenblatt zu einer bestimmten Bauteil aus und gibt eine Beschreibung mit Hilfe eines XHTML Editors ein

4.4.6 Verlinkung von Termen in strukturiertem Text (XHTML)

Use Case	Verlinkung von Termen in strukturiertem Text (XHTML)
Akteure	Benutzer
Vorbedingungen	Information ist im System vorhanden Strukturierter Text zu dieser Information ist vorhanden Terme sind im System vorhanden
Interaktionsfolge	Der Benutzer wählt die Information im System aus Der Benutzer editiert den strukturierten Text Der Benutzer markiert den Textbereich der einen Term repräsentiert Das System schlägt passende Terme vor Der Benutzer wählt den gewünschten Term aus Das System verlinkt den vom Benutzer gewünschten Term mit dem im strukturierten Text gewünschten Textbereich
Nachbedingungen	Der ausgewählte Term ist mit dem strukturierten Text verlinkt
Beispiel	Ein Entwickler wählt eine Beschreibung zu einer Information in dem der Term „LED“ vorkommt – er editiert diese Beschreibung, wählt dazu aus einer geeigneten Liste vorgeschlagener Terme den Term „LED“ aus – anschließend ist der im Text vorkommende Begriff „LED“ mit dem Term „LED“ verlinkt

4.4.7 Gliedern von Informationen

Use Case	Gliedern von Informationen
Akteure	Benutzer
Vorbedingungen	Information ist im System vorhanden Geeignete Hierarchie ist definiert
Interaktionsfolge	Der Benutzer wählt die Information im System aus Der Benutzer weist eine hierarchische Abhängigkeit zu
Nachbedingungen	Die Information ist entsprechend in der Hierarchie eingeordnet
Beispiel	Ein Entwickler wählt die Informationseinheit „Manual“ aus und ordnet sie dem Oberbegriff „Documentation“ zu

4.4.8 Verlinken von Informationen

Use Case	Verlinken von Informationen
Akteure	Benutzer
Vorbedingungen	Beide Informationen sind im System vorhanden
Interaktionsfolge	Der Benutzer wählt die Information im System aus Der Benutzer verlinkt diese Information mit einer anderen, indem er einen Eintrag zu einer Liste von Beziehungen hinzufügt
Nachbedingungen	Die Information Verlinkung ist im System gespeichert
Beispiel	Ein Projektleiter wählt die Informationseinheit „Mitarbeiter X“ aus und verlinkt diese mit der Information über „Projekt Y“, so dass eine Zuordnung „Mitarbeiter X“ ist Entwickler in „Projekt Y“ gegeben ist

4.4.9 Ordnen von Informationen

Use Case	Ordnen von Informationen
Akteure	Benutzer
Vorbedingungen	Informationen sind im System vorhanden
Interaktionsfolge	Der Benutzer wählt eine Information im System Der Benutzer ändert die Position einer Informationseinheit im

	eingebetteten Kontext
Nachbedingungen	Die neue Sortierung ist gespeichert
Beispiel	Ein Tester wählt die Informationseinheit „Testcases“ und sortiert die der Informationseinheit „Testcases“ zugeordneten Informationen „Testcase1“, „Testcase2“ und „Testcase3“ entsprechend

4.4.10 Suchen von Informationen

Use Case	Suchen von Informationen
Akteure	Benutzer
Vorbedingungen	Informationen sind im System vorhanden
Interaktionsfolge	Der Benutzer gibt einen passenden Suchbegriff ein Der Benutzer schränkt die Suche durch geeignete Filter ein Suchergebnisse werden angezeigt Der Benutzer wählt die gewünschte Information aus
Nachbedingungen	-
Beispiel	Ein Entwickler sucht nach einer Information, welche ein anderer Entwickler erstellt hat, durch entsprechende Filter schränkt er die Suche ein, so dass nur Informationen angezeigt werden, die von diesem Entwickler stammen. Mit einem entsprechenden Suchbegriff erhält er nun nur Informationen, die vom zuvor gewählten Entwickler stammen oder aus einem Namespace der ein konkretes Projekt repräsentiert

4.4.11 Ablegen von extern Dokumenten

Use Case	Ablegen von extern vorhandenen Dokumenten
Akteure	Benutzer
Vorbedingungen	-
Interaktionsfolge	Der Benutzer lädt ein Dokument hoch Es wird für dieses Dokument eine Informationseinheit abgelegt, die mit der physischen Datei verknüpft ist

	Der Benutzer gibt geeignete Metadaten (Name, Kurzbeschreibung, Tags, …) zu diesem Dokument ein Vom System werden weitere Metadaten (Dateiname, Content-Type, File-Url, …) angelegt
Nachbedingungen	Die Datei ist im System, Information über diese Datei ist abgespeichert
Beispiel	Ein Entwickler lädt ein Diagramm als Bilddatei hoch, versieht dies mit geeigneten Metadaten, einer Kurzbeschreibung und vergibt Tags, so dass diese Information später einfach gefunden werden kann

4.4.12 Suchen nach Dokumenten

Use Case	Suchen nach Dokumenten
Akteure	Benutzer
Vorbedingungen	Das Dokument befindet sich im System Geeignete Metadaten zum Dokument sind vorhanden
Interaktionsfolge	Der Benutzer wählt einen geeigneten Suchbegriff Der Benutzer wählt passende Filter indem er beispielsweise den Zeitraum einschränkt Das System sucht das Dokument anhand des Suchbegriffs und Filters, Metadaten wie Name, Kurzbeschreibung oder Tags werden berücksichtigt Der Benutzer bekommt die Suchergebnisse präsentiert Der Benutzer wählt das gewünschte Dokument aus
Nachbedingungen	-
Beispiel	Ein Entwickler sucht ein Schaltbild, welches in den letzten beiden Monaten erstellt worden sein müsste. Mit Hilfe eines geeigneten Suchbegriffs, sowie der Einschränkungen auf die letzten beiden Monate, sollte wenn das Dokument beim hochladen mit geeigneten Metadaten versehen wurde, das richtige Dokument gefunden werden können

4.4.13 Generierung von Reports

Use Case	Generierung von Reports
Akteure	Benutzer
Vorbedingungen	Informationseinheiten die für den Report benötigt werden befinden sich im System Eine entsprechende Konfiguration für den Report ist vorhanden
Interaktionsfolge	Der Benutzer wählt eine entsprechende Konfiguration Das System sammelt die Informationseinheiten und generiert mit Hilfe der Konfiguration ein Dokument Für das generierte Dokument wird im System eine eigene Information mit Metadaten abgelegt Der Benutzer ruft das generierte Dokument ab
Nachbedingungen	Das generierte Dokument ist im System gespeichert
Beispiel	Ein Entwickler erstellt einen Glossar aus den Termen, die mit beschreibenden Text verlinkt sind

4.4.14 Import von Daten

Use Case	Import von Daten
Akteure	Benutzer
Vorbedingungen	Die Daten liegen in einem für den Server verarbeitbaren XML Format vor
Interaktionsfolge	Der Benutzer lädt die zu importierende Daten mit Hilfe einer XML Datei hoch Das System verarbeitet diese im XML Format vorliegenden Daten und speichert diese in der Datenbank
Nachbedingungen	Die zu importierenden Daten sind im System gespeichert
Beispiel	Ein Entwickler exportiert Daten aus Telelogic DOORS und transformiert diese in das vom System erwartete XML Format. Die entstandene XML Datei wird anschließend hochgeladen und ins System importiert, wo die Daten danach in die Datenbank gespeichert werden

4.4.15 Export von Daten

Use Case	Export von Daten
Akteure	Benutzer
Vorbedingungen	Die Daten sind im System gespeichert
Interaktionsfolge	Der Benutzer wählt die zu exportierenden Daten aus Der Server bringt die Daten aus der Datenbank in ein geeignetes XML Format Der Benutzer lädt die Daten per zur Verfügung gestellten Download herunter
Nachbedingungen	-
Beispiel	Ein Entwickler exportiert Daten aus dem Wissensmanagement-Tool, diese Daten liegen dann im XML Format vor und können zum Beispiel in ein anderes Tool importiert werden oder als Backup für einen speziellen Entwicklungsmeilenstein dienen.

5 Praktische Umsetzung / Entwicklung

Dieses Kapitel beschreibt die praktische Umsetzung und Entwicklung des in dieser Arbeit vorgestellten Wissensmanagement-Tools, genannt *Infotool*.

5.1 Architektur

Es wird eine einfache 3-Tier Client-Server Architektur angewandt. Ein Client schickt Anfragen, die Servlets im Container nehmen die Anfragen entgegen. Je nachdem welches Format der Client verlangt, wird entsprechend ein XML oder JSON Format zurückgeliefert. Dazu wurden sogenannte Builder entwickelt, die mit der Datenbank Schnittstelle kommunizieren. Auch dazu wurde eine eigene Klasse entwickelt, die sich um die Kommunikation mit der Datenbank kümmert. Eine typische Anfrage wird somit wie oben erwähnt vom Servlet entgegengenommen, an einen geeigneten Builder weitergeleitet, dieser baut mit Hilfe der von der Datenbank Klasse erhaltenen Daten einen entsprechenden Response zusammen, der vom Servlet an den Client zurückgegeben wird. Abbildung 13 veranschaulicht die angewandte Architektur.

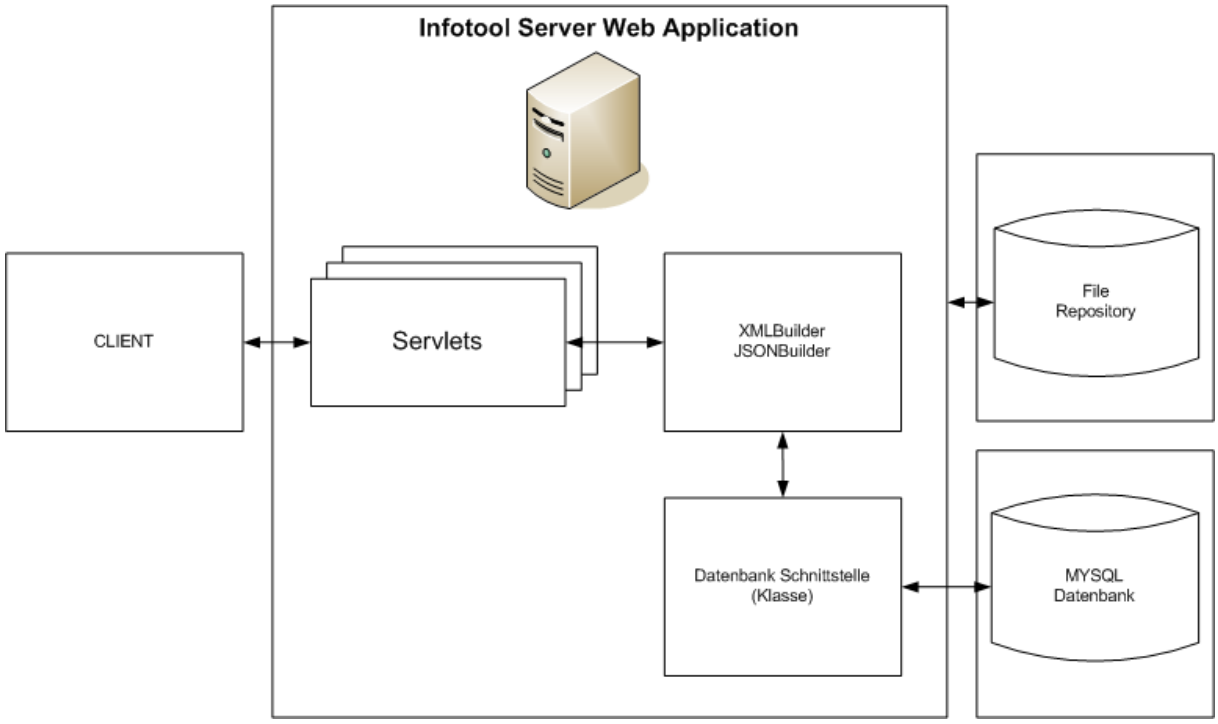


Abbildung 13: Vereinfachte Skizze zur Client Server Architektur des Infotools

5.2 Clients

Das Konzept ist so ausgelegt, dass mehrere und auch unterschiedliche Clients auf den Server zugreifen können. Exemplarisch wurde für diese Arbeit ein allgemeiner Client entwickelt, der auf JQuery [JQuery.com, 2011] basiert. Dieser allgemeine Client benützt die im Anhang A definierten Schnittstellen, um Operationen am Server durchzuführen. Der Client ist eher als Testumgebung zu verstehen, mit dem sich die entwickelten Konzepte testen lassen. Die Idee ist aber (siehe auch kurz im Kapitel Ausblick), dass man spezialisierte Clients entwickelt, die

unterschiedliche Sichten für den Benutzer bieten. Eine konkrete Idee wäre, dass ein Entwickler nur eine Ansicht eines Projekts erhält, an dem er gerade beteiligt ist und die Informationen in diesem Projekt präsentiert bekommt, sowie diese verwalten kann.

Um unterschiedlichste Clients bedienen zu können, unterstützen ein Großteil der Schnittstellen nicht nur XML Format, sondern auch das in Kapitel 2.5 vorgestellte JSON Format. Auch der oben erwähnte allgemeine Client nützt das JSON Format, da dies mit Hilfe von JavaScript weit einfacher zu verarbeiten ist als das XML Format.

Es ist zu beachten, dass der Fokus dieser Arbeit darin lag, ein geeignetes Datenmodell zu entwickeln, sowie einen Server zur Verfügung zu stellen, der diese verschiedenen, teils spezialisierten Clients bedient, die aber natürlich auf dieselbe, gemeinsame Datenbasis zugreifen.

5.3 Datenmodell

Um das Wissen abzubilden wurde ein an RDF angelehntes Datenmodell angewendet. Das heißt es wurden sogenannte Tripel verwendet welche aus einem Subjekt, Prädikat und Objekt besteht. Damit lassen sich generelle Informationen und Beziehungen abbilden.

5.3.1 Begriffserklärungen und Beispiele

Um die Informationen abzubilden wurden einige Begriffe definiert. Diese Begriffe werden in den folgenden Kapiteln verwendet.

Eine Information bzw. Informationseinheit wird *topic* genannt (im Folgenden auch Topic genannt) – ein Topic hat bestimmte Metadaten, die allgemeine Informationen zum Topic angeben.

Diese Benennung und Gliederung in Informationseinheiten wurde von der Darwin Information Typing Architectur (DITA) inspiriert. [Oasis DITA, 2011]. Dieser Ansatz definiert ein Topic folgendermaßen:

Ein Topic ist eine Informationseinheit, die durch Titel und Inhalt bestimmt wird. Diese Einheit muss knapp genug sein, um ein einzelnes Thema zu behandeln oder eine einzige Frage zu beantworten. Sie muss jedoch auch ausreichend genug sein, um alleine sinnvoll stehen zu können und dabei alleine weiterentwickelt zu werden. [Oasis DITA, 2011]

Das Infotool unterstützt Hierarchien und generierte Reports (siehe spätere Kapitel), diese würden den nach dem DITA Ansatz definierten *Maps* entsprechen. Wobei *Maps* Dokumente sind, in denen einzelne Referenzen zu Topics sinnvoll gesammelt und organisiert sind.

Ein Topic ist von einem bestimmten Typ – dazu wurden die Begriffe *class*, *property* und *resource* gewählt – die entsprechend an das RDF Konzept *Subjekt*, *Prädikat* und *Objekt* bzw. *rdf:type* angelehnt sind. Damit lassen sich Informationen wie Tabelle 3 und Tabelle 4 zeigen folgendermaßen verknüpfen und abbilden.

Klasse	Property
Person	vorname
Person	nachname
Person	geburtstag
Projekt	diplomand
Projekt	status

Tabelle 3: Beispielhafte Zuordnung Klasse – Property

Klasse	Ressource
Person	Scherr
Projekt	Masterarbeit

Tabelle 4: Beispielhafte Zuordnung Klasse - Resource

Aufgrund der Verknüpfung Klasse ←→ Property und Klasse ←→ Ressource kann man nun Informationen gemäß dem Schema Subjekt – Prädikat – Objekt darstellen. Das heißt – die Ressource *Scherr* ist vom Typ (Klasse) *Person*, das Projekt *Masterarbeit* ist vom Typ (Klasse) *Projekt*. Die Ressourcen instanzieren sozusagen eine Klasse und die Properties werden mit Daten gefüllt (siehe Tabelle 5)

Subjekt	Prädikat	Objekt
Scherr	vorname	„Christoph“
Scherr	nachname	„Scherr“
Scherr	geburtstag	1985-05-18
Masterarbeit	diplomand	Scherr
Masterarbeit	status	„in Entwicklung“

Tabelle 5: Beispielhaftes Tripel Subjekt-Prädikat-Objekt

Ein Objekt kann nun wie oben ein String, Datum oder sonstiger Datentyp sein – oder eine bestimmte Instanz einer Klasse. Abbildung 14 und Abbildung 15 zeigen das Topic *Scherr* im Infotool:

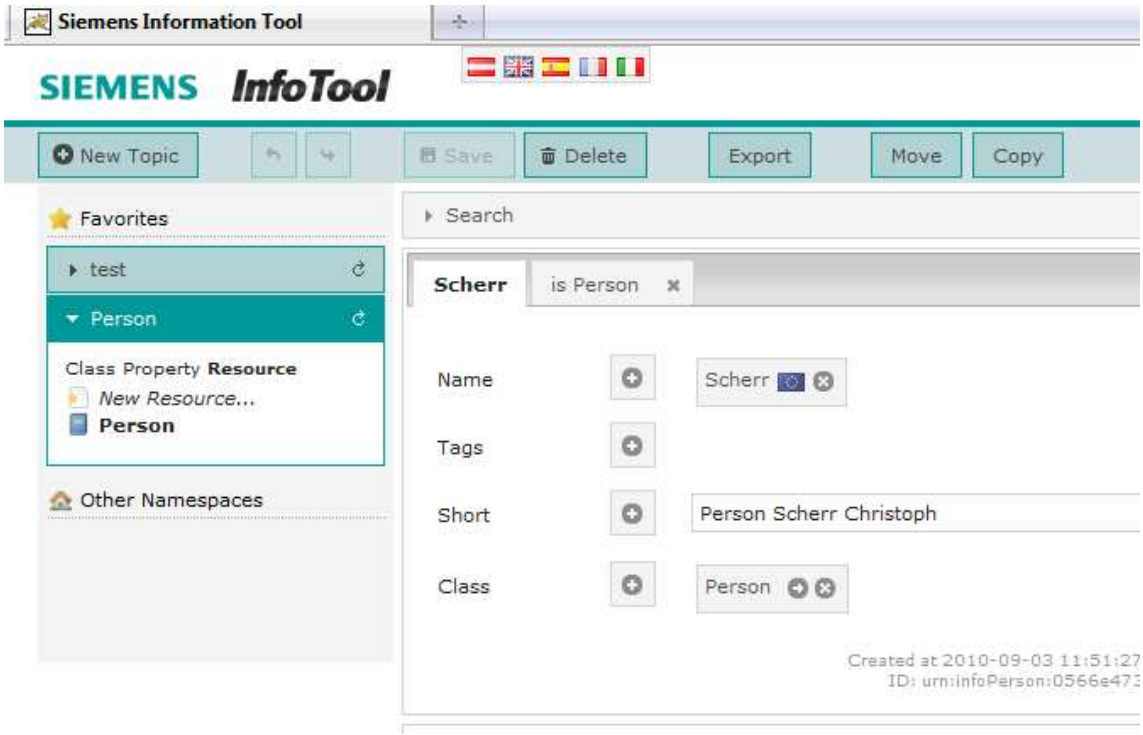


Abbildung 14: Topic *Scherr* im Infotool

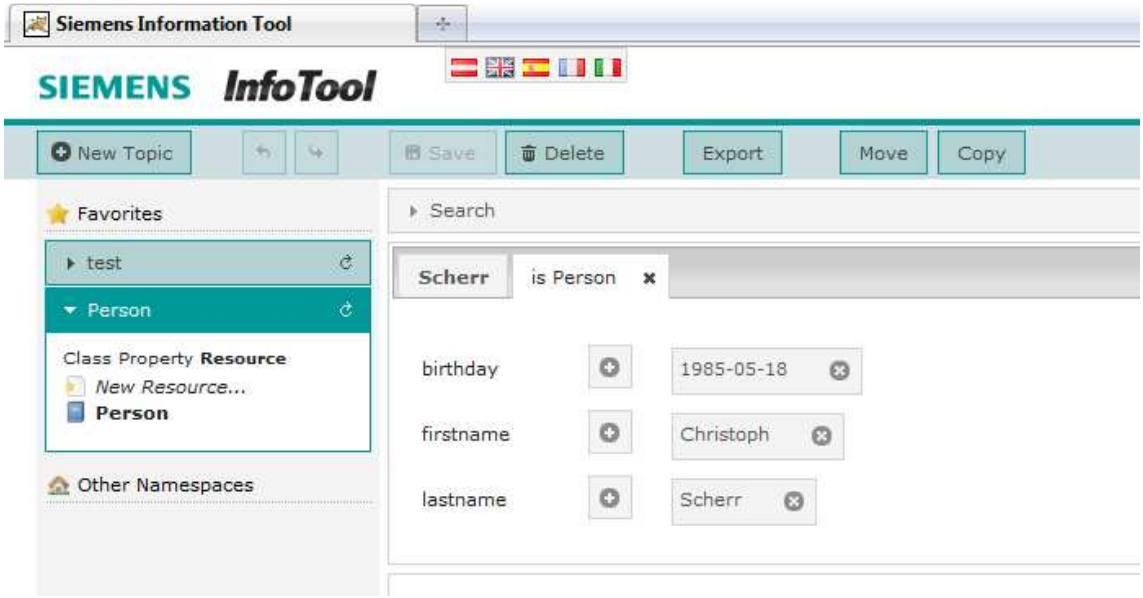


Abbildung 15: Properties der Klasse Person für Topic *Scherr*

5.3.2 XML Schema

Es wurde ein umfassendes XML Schema definiert, um die Informationen bzw. Daten die im Wissensmanagement-Tool gespeichert werden sollen, in eine gewisse Struktur zu bringen. Dieses Schema dient einerseits um ein XML Import/Export Format zu definieren, andererseits werden sämtliche Serverschnittstellen mit Hilfe dieses XML Schemas beschrieben. Da die Informationen serverseitig auch in einer Datenbank gespeichert werden, baut das zu Grunde liegende Datenbankschema auf diesem XML Schema auf, es stellt jedoch nicht das

serverseitige Speicherformat selbst dar. Weiters zu beachten ist, dass das vorhandene XML Schema bzw. Datenmodell nicht mit dem Format RDF/XML [W3.org RDF/XML, 2011] verwechselt werden sollte. Das hier verwendete Schema orientiert sich nur in gewissen Teilbereichen an RDF und hat nichts mit dem Format RDF/XML zur Serialisierung von RDF zu tun.

Folgend werden exemplarisch die wichtigsten Bestandteile dieses Schemas dargestellt. Zur Erstellung und grafischen Visualisierung wurde der Oxygen XML Editor [Oxygen XML Editor, 2010] verwendet.

Abbildung 16 zeigt die Definition eines Topics (Informationseinheit). Das heißt, generelle Attribute wie *id*, *type* (*class*, *property*, *resource*) sowie dem zugeordneten *namespace* und weitere Standardattribute wie *creator*, *modifier*, *created* und *modified*, welche Auskunft über die letzte Bearbeitung bzw. die Erstellung dieses Topics geben. Weiters besteht ein Topic verpflichtend aus hier genannten *topicinfos* welche einen Namen, eine Kurzbeschreibung, sowie Tags beinhalten (*name*, *short*, *tags*). Die eben beschriebenen Attribute, sowie das *topicinfos* Element stellen die Metadaten über diese Information bzw. Topic dar. Daher besitzt jedes Topic, egal welchen Typs, diese grundlegenden Attribute.

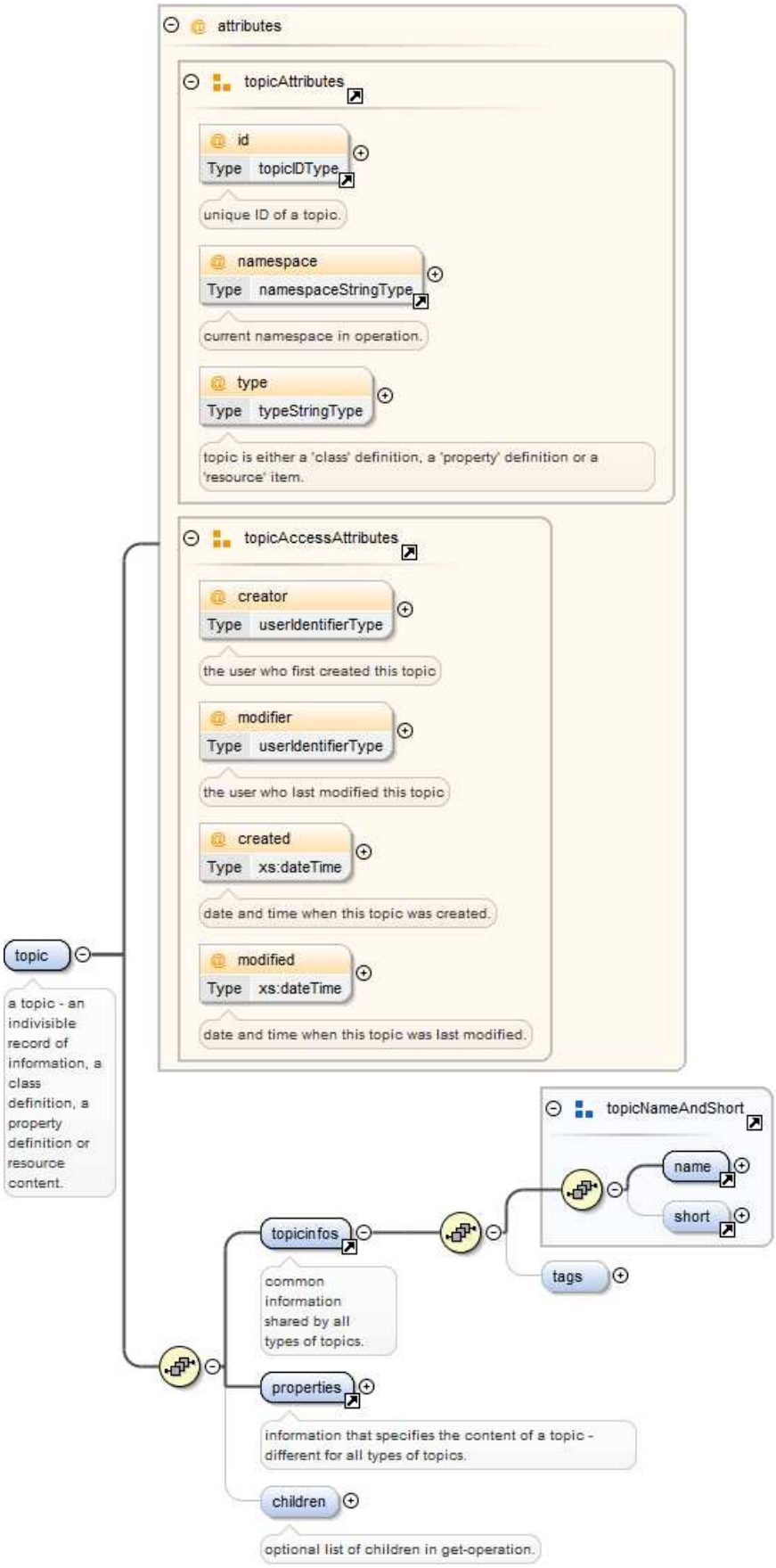


Abbildung 16: Allgemeine Topic Attribute und Metadaten

Um Generalisierung und Wiederverwendbarkeit zu gewährleisten, wurden viele geeignete Typen, Elementgruppen und Attributgruppen definiert, die sich im Schema wiederholt finden und für die unterschiedlichsten Schnittstellen einheitlich verwenden lassen. Abbildung 17 zeigt die Elementgruppierung *TopicNameAndShort*. Diese besteht wie der Name schon sagt aus den Elementen *name* und *short* und wird zum Beispiel wie oben erwähnt für Metadaten (*topicinfos*) über ein Topic verwendet. Um die eigentlichen Daten abzubilden, wurden ebenfalls eigene Elemente und Typen definiert. Dazu dient das allgemeine Datenelement *item*, welches an mehreren Stellen des Schemas mit unterschiedlichem Typ auftritt. Ein Element *item* vom Typ *nameItemWithLangType* besteht aus dem Attribut *lang*, welches wie der Name schon sagt Auskunft über die Sprache gibt, sowie dem eigentlichen Text, in diesem Fall vom Typ *nameStringType*, welcher vom XML Schema Standardtyp *xs:token* ist. Man hätte dies auch etwas einfacher, ohne vordefinierten eigenen Typ und mit einem einfachen *xs:string* Typ darstellen können. Der Sinn hinter den eigens definierten Typen ergibt sich darin, daß man damit bereits Einschränkungen vorgeben kann – wie zum Beispiel beim Typ *nameStringType*, dass dieser eine maximale Länge von 255 Zeichen haben darf, sowie der Unterschied zwischen *xs:token* und *xs:string*, dass *xs:token* nicht jeden beliebigen String zulässt, sondern durch ein Leerzeichen getrennte Tokens erwartet.

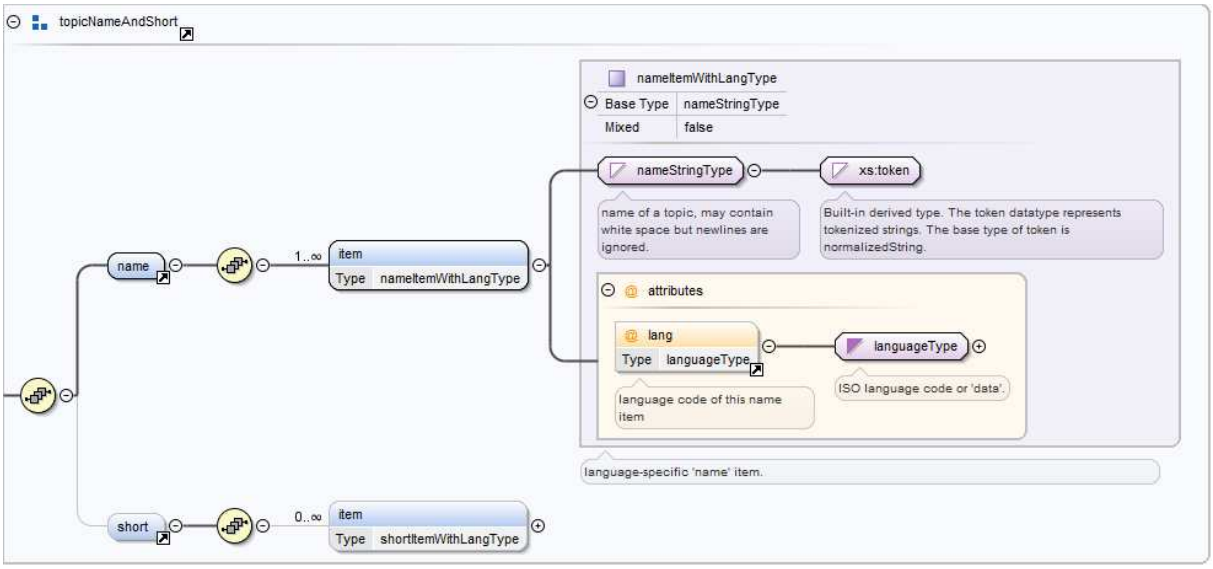


Abbildung 17: Elementgruppe TopicNameAndShort

Wie schon eingangs erwähnt, hat ein Topic einen bestimmten Typ (*class*, *property*, *resource*). Dementsprechend gibt es nun drei verschiedene Möglichkeiten, wie der Inhalt eines *properties* Elements eines Topics aussehen kann. Abbildung 18 zeigt, auch hier wurden wieder Elementgruppen definiert, sowie auf bereits global definierte Typen verlinkt. Die definierten Elementgruppen für *class* und *property* sind in Abbildung 19 und Abbildung 20 zu sehen, eine Beschreibung für *resource* Properties folgt weiter unten.

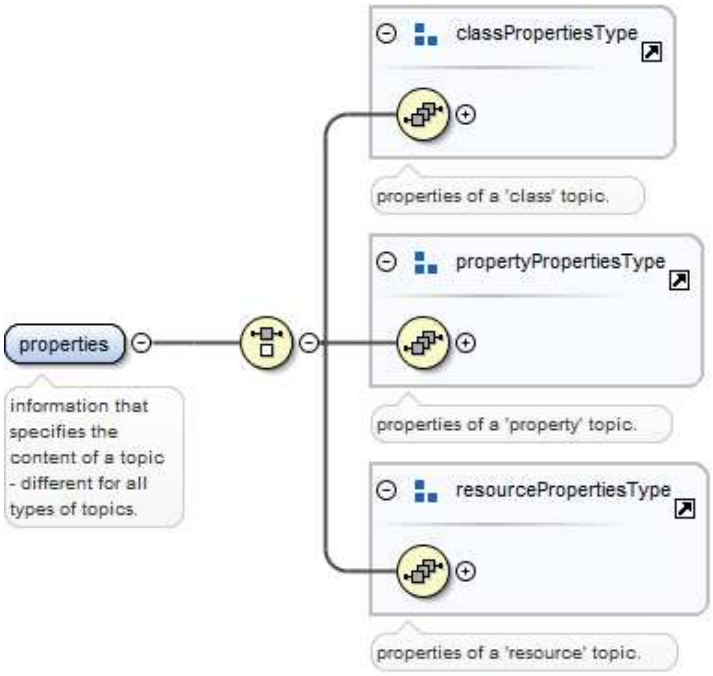


Abbildung 18: Properties Element eines Topics

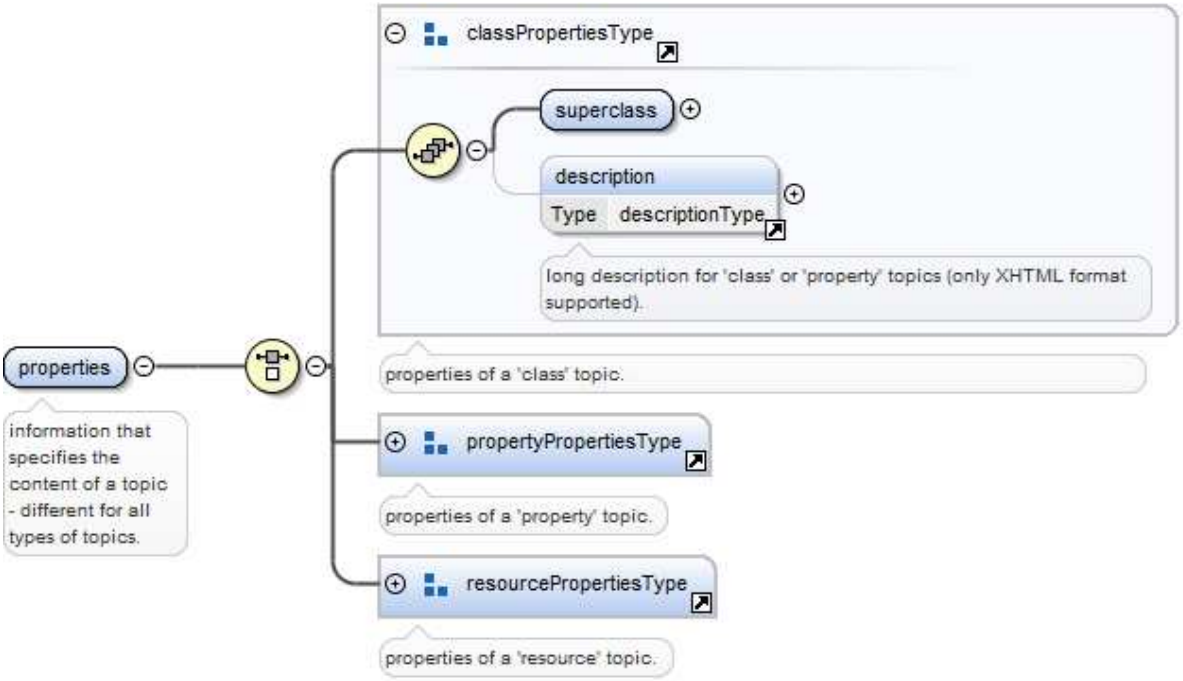


Abbildung 19: Properties Element eines Topics vom Typ class

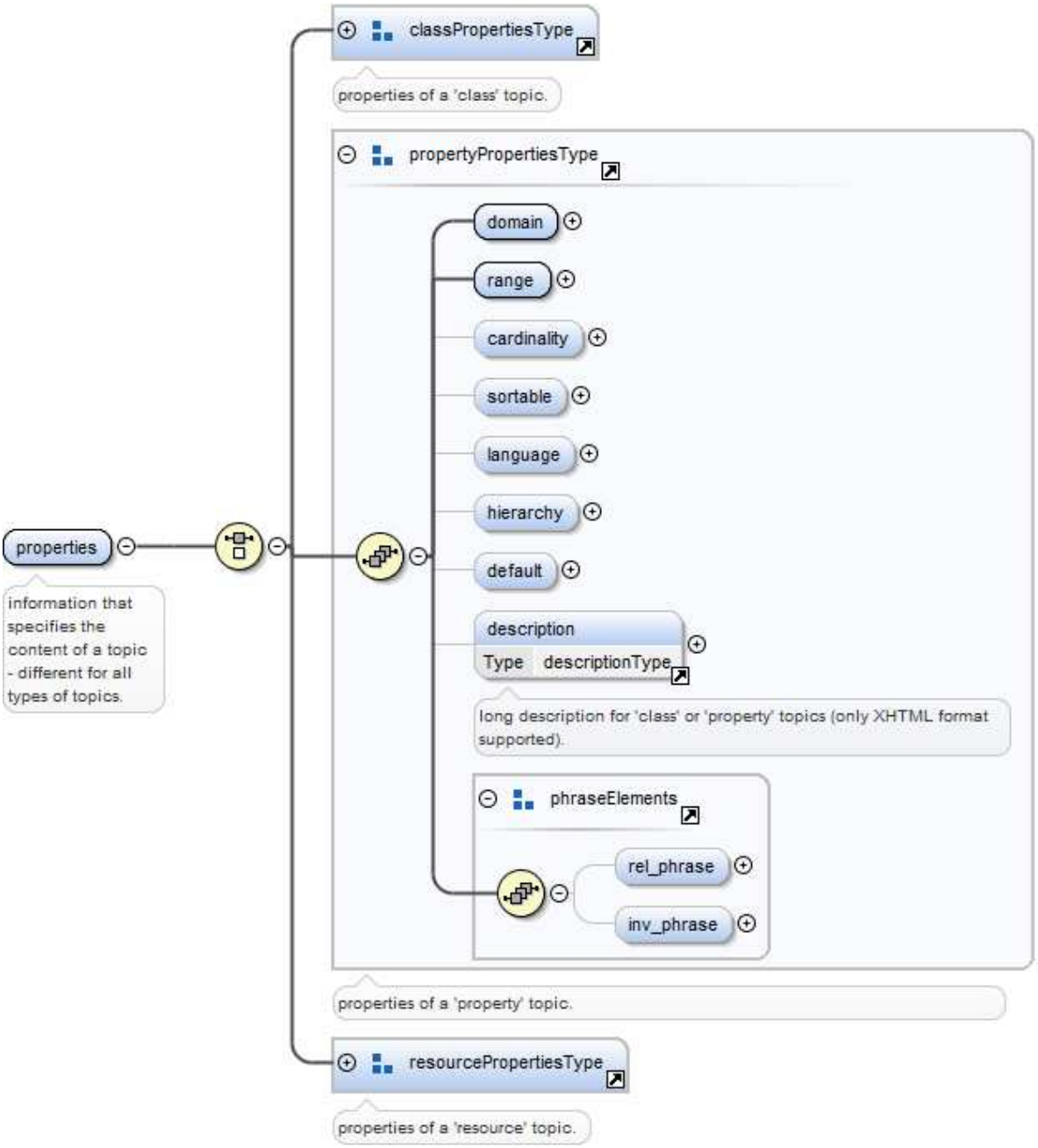


Abbildung 20: Properties Element eines Topics vom Typ property

Tabelle 6 zeigt eine kurze Übersicht, sowie Beschreibung der einzelnen Eigenschaften der jeweiligen Topics.

Typ	Property	Beschreibung
class *(classPropertiesType)*	*superclass*	Liste aller Superklassen dieser Klasse
	description	Ausführliche Beschreibung der Klasse
property *(propertyPropertiesType)*	*domain*	Liste aller Klassen, die in einer Beziehung mit diesem Property als Subjekt stehen können
	range	Liste aller Klassen, die in einer Beziehung mit diesem Property als Objekt stehen können „Datentyp“ des jeweiligen Properties (String, Enum, …, siehe Kapitel 5.3.4) oder eine Klasse – was soviel heißt dass dieses Property vom Datentyp dieser Klasse ist. Als Beispiel: Für das Property *projektleiter* wird als Range die Klasse *Person* angegeben
	cardinality	Gibt die Kardinalität des Properties an, Beispiele wären 0-1, 1, *, +, (siehe Kapitel 5.3.5)
	sortable	Gibt an, ob die Werte dieses Properties sortiert werden können
	language	Wird für String Properties verwendet – gibt an ob eine Sprache ausgewählt werden kann
	hierarchy	Für spezielle Properties die eine Hierarchy definieren
	default	Defaultwert
	description	Ausführliche Beschreibung des Property
	rel_phrase	Relation Phrase – Phrase um Beziehungen zwischen Subjekt und Objekt zu beschreiben. Zum Beispiel *<Projekt> wird geleitet von <Scherr>* - *wird geleitet von* ist die Relation Phrase
	inv_phrase	Inverse Relation Phrase – siehe obige Relation Phrase, ist die Umkehrung: Beispiel: *<Scherr> leitet <Projekt> – leitet* ist hier die Inverse Relation Phrase

resource *(resourcePropertiesType)*	*class*	Eine Liste aller Klassen, der die Resource angehört. Diese *class* Elemente beinhalten dann auch die Daten, eine genauere Beschreibung folgt

Tabelle 6: Übersicht der Eigenschaften eines Topics

Wie in Tabelle 6 bereits erwähnt, befinden sich die Daten innerhalb der *class* Elemente eines Topics vom Typ *resource*. Abbildung 21 zeigt den Inhalt eines typischen *class* Elements, wobei sich die Daten wieder in den *prop* Elementen befinden. Ein *prop* Element beinhaltet als Attribut eine Referenz auf ein Property, mit zugehörigen Informationen zu diesem Property, wobei hier zu beachten ist, dass diese dem Property zugehörigen Informationen redundant sind und für Clients gedacht sind, damit diese keinen zusätzlichen Serverzugriff auf dieses Property durchführen müssen. Die Werte selbst sind entweder in den *item* Elementen, oder in den *topicref* Elementen. Diese Unterscheidung ergibt sich aus dem jeweiligen Typ des Properties. Ist der Typ eines Properties von einer bestimmten Klasse (z. B. Person), dann befindet sich die Information in einem *topicref* Element – was wie der Name schon sagt eine Referenz auf ein anderes Topic darstellt – und somit als Wert nur die ID des jeweiligen referenzierten Topics beinhaltet. Andere Datentypen wie String, Number etc. werden in *item* Elementen gespeichert, hier ergeben sich noch kleine Unterscheidungen, zum Beispiel das optionale *lang* Attribut oder für den Spezialfall eines Enumeration Datentyps (Aufzählung) sind die jeweiligen Optionen im *options* Element gespeichert.

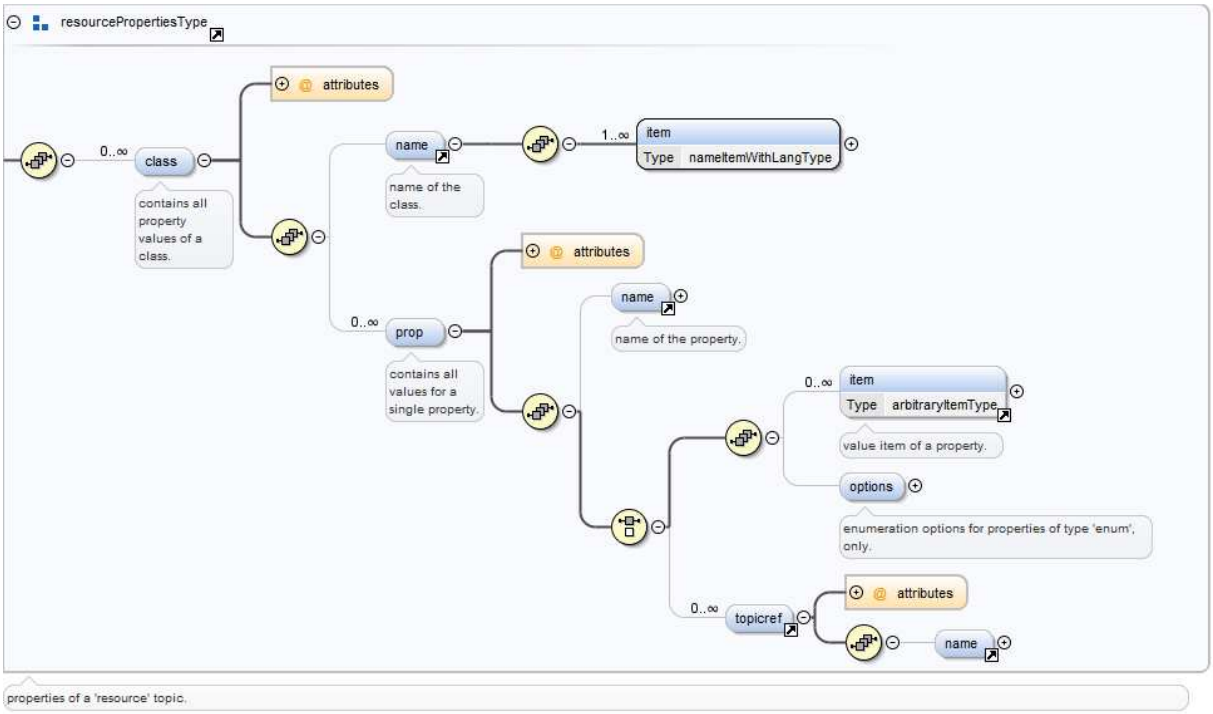


Abbildung 21: class Elemente eines Topics vom Typ resource

5.3.3 Unterschiede zu RDF

Die Unterschiede zu RDF ergeben sich dadurch, dass nicht strikt alles über Tripel definiert wurde, sondern gewisse Eigenschaften fix vorgegeben wurden. Das heißt durch die fix vorgegebene Zuordnung zwischen Klassen und Properties lassen sich auch Properties vererben, indem man eine Unterklasse hat und eine Ressource diese Unterklasse instanziert. Das ist natürlich auch mit RDF möglich, über *rdf:subClassOf* usw., allerdings wird in dem hier verwendeten Datenmodell eine direkte Möglichkeit dafür geschaffen. Dasselbe gilt für die teilweise fix vorgegebenen Metadaten die ein Topic beschreiben, wie *creator*, *modified, namespace* etc., auch diese hätte man einfach über Tripel definieren können. Die Idee bei dieser Arbeit war aber, dass man ein gewisses fix vorgegebenes Grundgerüst verwendet, und diese Informationen dann mit zum Beispiel projektspezifischen Informationen verknüpft, das Grundgerüst aber immer dasselbe bleiben sollte.

Weiters zu beachten ist, dass unterschiedlichste RDF Notationen existieren. Die in dieser Arbeit angewendeten Konzepte orientieren sich jedoch nur teilweise an diesen Notationen, es wurde aber nicht daran gedacht eine dieser Notationen genau umzusetzen. Der Grund dafür liegt darin, dass man eine gewisse Flexibilität erhält, um auch eventuell auf projektspezifische Anforderungen reagieren zu können.

Was die Vererbung betrifft, ist der in diesem Datenmodell verwendete Ansatz eher objektorientiert, durch die Definition von Klassen, Superklassen etc., während bei RDF eher über Tripel geschlußfolgert wird.

5.3.4 Datentypen

Nach einer eingehenden Analyse der verfügbaren Informationen, die in das Wissensmanagement-Tool integriert werden sollen, haben sich folgende, in Tabelle 7 angeführten Datentypen die unterstützt werden müssen ergeben.

Datentyp	Beschreibung
string	einfacher String, Leerzeichen sind nicht signifikant (normalisiert)
number	Zahl, kein Unterschied zwischen *int* und *float*
date	Datum, Format JJJJ-MM-DD, Beispiel: 1985-05-18
boolean	boolscher Wert, *true* (1) oder *false* (0)
text	Für Kurzbeschreibungen sollte es möglich sein mehrzeiligen Text einzufügen, deshalb wurde zwischen den Datentypen *string*, welcher vorzugsweise für einzeilige, kurze Texte verwendet wird, und dem Datentyp *text* (für längere Texte) unterschieden: Weiters sind bei *text* im Gegensatz zu *string* Leerzeichen und Zeilenumbrüche signifikant

xhtml	Es sollte vor allem für beschreibende Texte möglich sein, XHTML-Texte einzufügen. Wie in den Anforderungen in Kapitel 4.1.2 beschrieben, geschieht dies vorzugsweise über einen vom Client zur Verfügung gestellten Editor. Teilweise liegen Informationen in anderen Tools auch im XHTML-Format vor – auch diese müssen integriert bzw. importiert werden können
url	Intern besteht kein Unterschied zum Datentyp *string* – dennoch sollte es einen eigenen Datentyp *url* geben – der anzeigt dass der beinhaltete Text sicher eine *url* darstellt
enum	*enum* steht für Enumeration, das heißt es gibt eine Aufzählung möglicher Werte, aus der ein Wert selektiert werden kann. Dies soll für Properties wie zum Beispiel „Wochentag“ verwendet werden, um eine Auswahlmöglichkeit vorgeben zu können

Tabelle 7: Liste der unterstützten Datentypen im Infotool

5.3.5 Kardinalitäten

Um die Eigenschaften eines Property näher zu beschreiben, wurden Kardinalitäten definiert. Das heißt, es gibt bei einem speziellen Subjekt, Prädikat, Objekt Tripel an, wie viele unterschiedliche Objektwerte von einer Subjekt-Prädikat Kombination angenommen werden können. Das besondere am hier gewählten Konzept liegt darin, dass Mehrsprachigkeit unterstützt wird. Das heißt, Kardinalität „1“ bedeutet, dass pro Sprache ein Attributwert erlaubt ist. Zur Veranschaulichung: Das Property *titel* hat die Kardinalität „1“, das bedeutet, dass pro Sprache ein Titel vergeben werden kann. Entscheidend ist dabei das Attribut *language* eines Properties, welches angibt, ob dieses Property mehrsprachige Werte zulässt. Für ein Property *nachname* macht es keinen Sinn mehrsprachige Werte zu vergeben, hier wäre das *language* Attribut eines Properties auf *false* gesetzt. Tabelle 8 zeigt eine Übersicht der vom Infotool unterstützten Kardinalitäten.

Kardinalität	**Beschreibung**	**Beispiel**
1	genau ein Attributwert	Nachname „Scherr“
+	mindestens einen, oder mehrere Attributwerte	Vorname „Christoph“, „Josef“
*	beliebige viele Attributwerte (auch 0)	akademischer Titel „BSc“
0-1	optionales Attribut, maximal eines	Ehepartner, <ohne Wert>

Tabelle 8: Unterstützte Kardinalitäten

5.3.6 Namespaces

Um Topics eindeutig und thematisch zuordnen zu können wurden Namespaces eingeführt. Ein Namespace kann beispielsweise ein bestimmtes Projekt darstellen. Es existieren auch allgemein gültige Namespaces, wie zum Beispiel der *System* Namespace, der einige grundlegende Klassen und Properties beinhaltet. Ein möglicher Anwendungsfall wäre, dass man einen allgemeinen *Mitarbeiter* Namespace einführt, indem für jeden Mitarbeiter ein Topic angelegt wird und diesem Namespace zugeordnet wird. Ein Topic kann übrigens immer nur genau einem Namespace zugeordnet werden. Der Namespace ist auch im Identifier des Topics enthalten.

5.3.7 Topic Identifier

Wie bereits eingangs erwähnt, besitzt jedes Topic eine eigene ID. Dabei wurde das in Kapitel 2.3.6 vorgestellte URN Schema angewandt.

```
urn:info:<namespace>:bezeichner
```

Der *urn:info* Teil ist das für das Infotool verwendete Präfix, als *bezeichner* kann ein eigener Name oder auch ein zufällig generierter Wert verwendet werden. Beispielhafte Topic Identifier wären:

- `urn:info:System:Description`
- `urn:info:Infotool:List_Namespaces`
- `urn:info:Safety:43ea748c-e6e9-4f20-b864-fdd0b5b4df4e`
- `urn:info:Simatic:Failsafe:ET200iSP:d0e3698`

Wie man erkennen kann, existieren auch Namespaces die mehrere „*:*“ enthalten. Dies dient für eventuelle spätere Erweiterungen, um nicht nur Hierarchien zwischen Topics (siehe nachfolgendes Kapitel), sondern auch Hierarchien zwischen Namespaces aufzubauen. Als Beispiel wäre das letzte angeführte Beispiel zu nennen: *Simatic:Failsafe:ET200iSP*. Dabei könnte *Simatic* der Oberbegriff bzw. Namespace sein, und *Failsafe* sowie *ET200iSP* Verfeinerungen. Intern werden *Simatic*, sowie *Simatic:Failsafe* als getrennte Namespaces verstanden, es wäre aber für spezielle Clients einfach möglich, hier eine Hierarchie zu erkennen und entsprechend darzustellen.

5.3.8 Hierarchien

Es wurde ein Mechanismus implementiert der es gestattet einfache hierarchische Strukturen unter Topics derselben Klasse zu definieren. Zum Aufbau einer Hierarchy werden Properties herangezogen, die als Domain und Range dieselbe Klasse aufweisen und Kardinalität „0-1“ beträgt. Sie eignen sich somit zur Darstellung einer Nachfolger → Vorgänger Beziehung. Dies lässt sich am Einfachsten mittels eines konkreten Beispiels erklären.

Man definiert eine Klasse *Testcase Object*, sowie ein Property *details Testcase*. Wie man in Abbildung 22 erkennen kann, weist das Property *details Testcase* sowohl in Domain, als auch in Range ein *Testcase Object* auf. Weiters wurde die Eigenschaft *hierarchy* gesetzt.

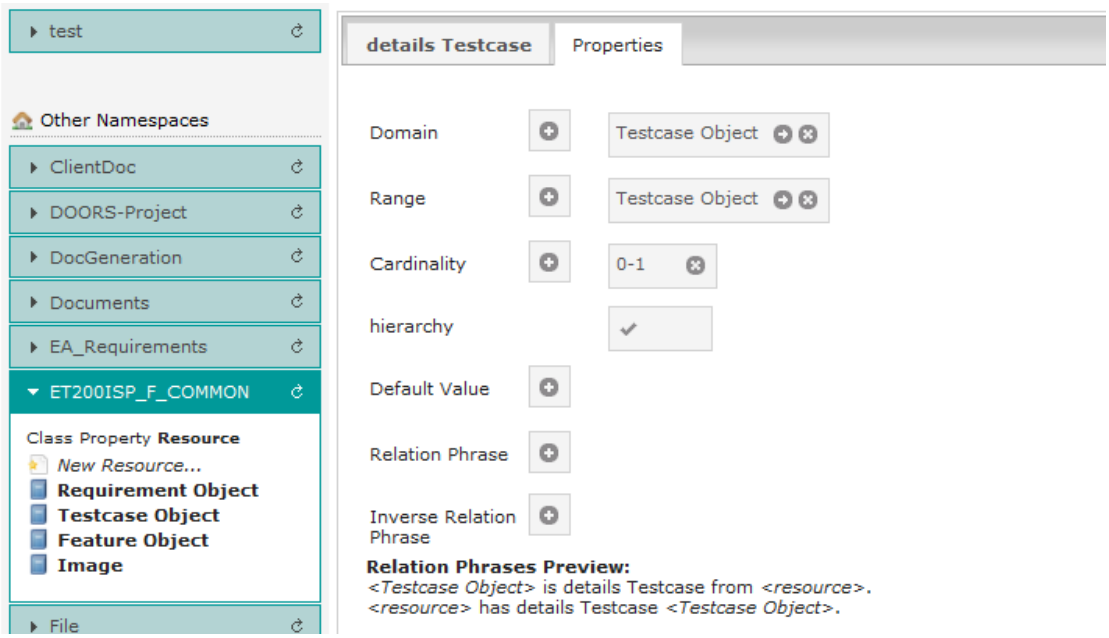


Abbildung 22: Property "details Testcase"

Erstellt man nun Ressourcen vom Typ *Testcase Object*, wählt man für das Property *details Testcase* eine geeignete, in der Hierarchie übergeordnete, Ressource vom Typ *Testcase Object* aus. Abbildung 23 zeigt die Ressource *Methodik*, die als übergeordnete Ressource *Modultest* detailliert. Auf der linken Seite der Abbildung 23 sieht man die komplette Hierarchie der in diesem Namespace definierten *Testcase Objects*.

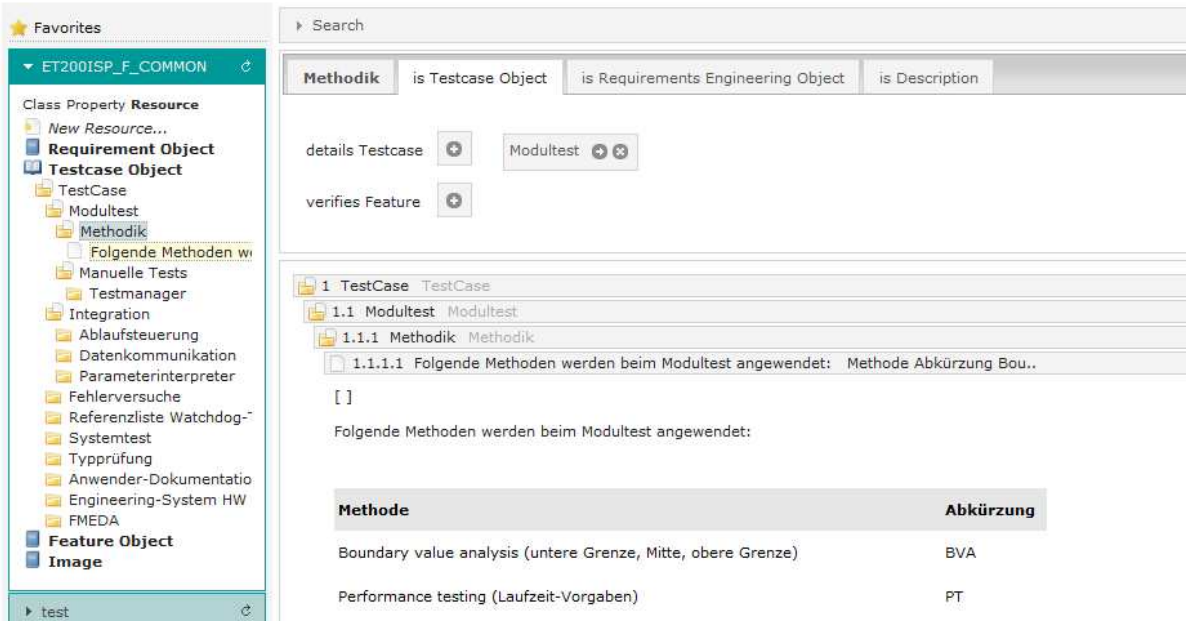


Abbildung 23: Abbildung einer hierarchischen Struktur im Infotool

5.3.9 Metadaten

Zusätzlich zu den als RDF Tripel abgebildeten Informationen, hat jede Informationseinheit (topic) einige am Dublin Core (siehe Kapitel 2.3.4) orientierte Metadaten. Tabelle 9 zeigt die ausgewählte Zuordnung und eine kurze Beschreibung.

Dublin Core	**Beschreibung**	**Infotool**	**Datentyp (Kardinalität, mehrsprachig)**
TITLE	Titel des Dokuments – in diesem Fall Titel des Topics	name	string (1, ja)
CREATOR	Autor oder Verfasser des Dokuments bzw. der Publikation	creator	string (1, nein)
SUBJECT	Versehen des Inhalts mit Schlagwörtern (Keywords) – hier: Tags, um die Suche zu unterstützen	tags	string (*, ja)
DESCRIPTION	Kurzbeschreibung oder Kurzzusammenfassung des Inhalts	short	text (1, nein)
PUBLISHER	Name der veröffentlichenden Instanz, wird im Infotool nicht benötigt	-	
CONTRIBUTORS	Weitere Personen die an der Erstellung mitgewirkt haben – dies wird im Infotool nur anhand des *modifier* abgespeichert, also der letzten Person die an diesem Dokument gearbeitet hat	modifier	string (1, nein)
DATE	Zeitpunkt der Veröffentlichung, oder auch der letzten Bearbeitung	modified	date (1, nein)
TYPE	Art des Dokuments	-	
FORMAT	Formatangabe des Dokuments, wird nicht direkt verwendet, der Server kann aber mit Hilfe des Schnittstellen Parameters *format* (z.B.: *format=xml*, *format=json*) die Information entweder in *json* oder *xml* Format ausgeben.	-	
IDENTIFIER	Eindeutige Identifizierung nach einem Identifier	id	uri (1, nein)

SOURCE	Verweist auf abgeleitete Dokumente, ist anderweitig über Klassen / Superklassen Beziehungen realisiert	-	
LANGUAGE	Sprache des Dokumenteninhalts; Für ausgewählte Attribute der Informations-einheit ist ein *lang* Attribut vorhanden – ein allgemeines *lang* Attribut gibt es allerdings nicht.	lang	
RELATION	Verweist auf in Beziehung stehende Dokumente, ist anderweitig über eigens definierte Eigenschaften realisiert	-	
COVERAGE	Zeitliche und räumliche Eingrenzung – die zeitliche Eingrenzung wird durch eine *created* bzw. *modified* Information abgedeckt	created / modified	date (1, nein)
RIGHTS	Information zur Klarstellung der Rechte, wird im Infotool nicht benötigt	-	

Tabelle 9: Vergleich Metadaten nach Dublin Core und Infotool

5.4 Datenspeicherung / Datenbank

Bezüglich der Datenspeicherung war die erste zu treffende Entscheidung, ob man eine relationale Datenbank (wie MySQL) verwendet, die Daten in XML bzw. einer XML Datenbank, oder auch gleich mit Hilfe einer RDF Datenbank abspeichert. Die Entscheidung fiel aus mehreren Gründen auf die Verwendung einer relationalen Datenbank. Der Hauptgrund ist sicherlich jener, dass eine Datenbank wie MySQL mit Hilfe eines geeigneten Datenmodells die referentielle Integrität am Effizientesten gewährleisten kann. Auch was die Suche durch Queries angeht, hat eine Datenbank sicherlich einen Geschwindigkeitsvorteil gegenüber Abfragen die direkt auf XML oder RDF zugreifen. Ein weiterer Grund war, dass das verwendete Datenmodell nicht direkt in RDF spezifiziert sein sollte, sondern nur daran orientiert sein sollte. Gewisse Teilbereiche sollten vereinfacht sein (Vererbung) oder eingeschränkt werden. Somit ist man etwas flexibler und das Datenmodell ist erweiterbar.

Eine Datenbank kann man vereinfacht betrachtet als Sammlung von Daten sehen. Beispielsweise wäre eine Bücherei als nichtelektronische Datenbank zu verstehen, welche Bücher, Zeitschriften und andere Dokumente sammelt. Um bestimmte Daten zu finden, benützt man Karteikarten, Zeitschriften-Index oder befragt den Bibliothekar. Eine Bücherei wäre allerdings auch ohne diese Karteikarten, Zeitschriften-Index oder Bibliothekar eine Datenbank, die Verwendung wäre aber äußerst erschwert. Daher lässt sich sagen, dass eine Datenbank eine Art von Organisation benötigt, um verwendet werden zu können. Deshalb könnte man die eingangs erwähnte Definition dahingehend erweitern, indem man davon

spricht, dass eine Datenbank eine organisierte Sammlung von Daten darstellt. [Reese, Randy, & King, 2002]

MySQL selbst ist noch keine Datenbank. MySQL ist eine Software, die Funktionen zur Verwaltung von elektronischen Datenbanken zur Verfügung stellt. Dies umfasst Aufgaben wie Datenbanken anlegen, pflegen und verwalten. Man spricht von einem so genannten Datenbank-Management-System (DBMS). Ein DBMS stellt einen „Makler“ (Broker) zwischen der physikalischen Datenbank und den Benutzern dieser Datenbank dar. Ein spezieller Datenbank-Typ ist eine relationale Datenbank, welche Daten in Tabellen organisiert, sowie Relationen (Beziehungen) zwischen Tabellen beschreibt. Mit Hilfe dieser Beziehungen lassen sich Daten von mehreren Tabellen kombinieren oder zusammenfügen. [Reese, Randy, & King, 2002]

5.4.1 MySQL

MySQL ist ein relationales Datenbanksystem. Die Meinungen über MySQL liegen weit auseinander, während starke Befürworter von MySQL meinen, MySQL sei schneller, zuverlässiger und billiger als jedes andere Datenbanksystem, gibt es auch Meinungen, dass MySQL gar kein vollwertiges relationales Datenbanksystem sei. Betrachtet man die Fakten, erkennt man, dass es eine steigende Zahl von MySQL Anwendern gibt, die auch zu großen Teilen zufrieden sind, sowie die Nachteile, dass MySQL noch manche Eigenschaften fehlen, welche bei anderen Datenbanksystemen Standard sind. [Kofler, 2005]

MySQL ist unter der GPL (GNU General Public License) Lizenz verfügbar [GNU License, 2011]. Dies bedeutet vereinfacht gesagt, dass MySQL als Open Source Datenbank frei verwendet werden kann, sowie Abänderung von Quellcodes oder auch die Weitergabe erlaubt ist. Einschränkungen gibt es für rein kommerzielle Nutzung.

Das gesamte MySQL Datenbank System baut auf einer Client-Server Architektur auf, wobei das mitgelieferte *mysqld* den Server darstellt. Der Server dient als Schnittstelle zu den Datenbanken, der die Datenbanken verwaltet und bearbeitet. Client Programme schicken Anfragen, meist in Form von SQL (Structured Query Language) Befehlen. [DuBois, 2003]

Betrachtet man dies aus Sicht des entwickelten Infotools, stellt der Infotool Server den Client für den Datenbank Server dar.

5.4.2 Auswahl von MySQL

Es gibt mehrere Gründe die für die Auswahl von MySQL sprechen. Einer der Gründe liegt auch sicherlich darin, dass der Entwickler dieser Arbeit, am geschultesten im Umgang mit MySQL ist bzw. die größte Erfahrung mit diesem Datenbank System aufweisen konnte. Dennoch wurden anhand einer Literaturrecherche Gründe für MySQL erarbeitet. Exemplarisch sei hier folgendes Fazit aus [Kofler, 2005] zitiert:

MySQL ist ein sehr leistungsfähiges, relationales Client/Server-Datenbanksystem. Es ist für sehr viele Anwendungen ausreichend sicher und stabil und bietet dabei ein exzellentes Preis-Leistungs-Verhältnis (nicht nur, weil MySQL an sich kostenlos ist, sondern auch, weil es

vergleichsweise geringe Anforderungen an die Hardware stellt). MySQL hat sich deswegen im Bereich von Internet-Datenbanken zu einem Quasi-Standard entwickelt. [Kofler, 2005]

Es wurde die zum Beginn der Arbeit aktuellste Version von MySQL, nämlich MySQL 5.1 ausgewählt.

5.4.3 Datenbank Schema

Die Daten des Wissensmanagement-Tools werden in einer MySQL Datenbank abgelegt. Gleichzeitig wurde ein XML Schema definiert, sodass die Daten importiert/exportiert werden können, bzw. auch in XML Form abgelegt werden können.

Nach grundlegender Analyse ergab sich in Abbildung 24 gezeigtes Datenbankschema. Einige nicht relevante Beziehungen wurden aus Übersichtlichkeitsgründen ausgeblendet. Zum Beispiel die Beziehung mit der *lang* Tabelle, da diese aufgrund der Unterstützung für die Mehrsprachigkeit in vielen anderen Tabellen vorkommt. Die Umsetzung orientiert sich an dem in Kapitel 5.3 vorgestellten Datenmodell bzw. XML Schema. Wie sich zeigt, ergeben sich durch die Auflösung vieler *n:m* Beziehungen einige zusätzliche Tabellen. Weiters mussten für die unterschiedlichen unterstützten Datentypen auch eigene Tabellen angelegt werden. Das Kernstück des Models befindet sich in der *topic* Tabelle, welches eine Informationseinheit samt zugehöriger Metadaten darstellt. Die weiteren *topic* Tabellen wie *topic_names, topic_short* wurden aufgrund der Mehrsprachigkeit angelegt, da ja ein Topic beispielsweise einen englischen, sowie einen deutschen Namen oder Kurzbeschreibung haben kann. Die eigentlichen Daten befinden sich somit in den *resource_property_** Tabellen, welche aus einer Zuordnung der jeweiligen *resource_id* und *property_id* sowie dem zugehörigen Wert bestehen.

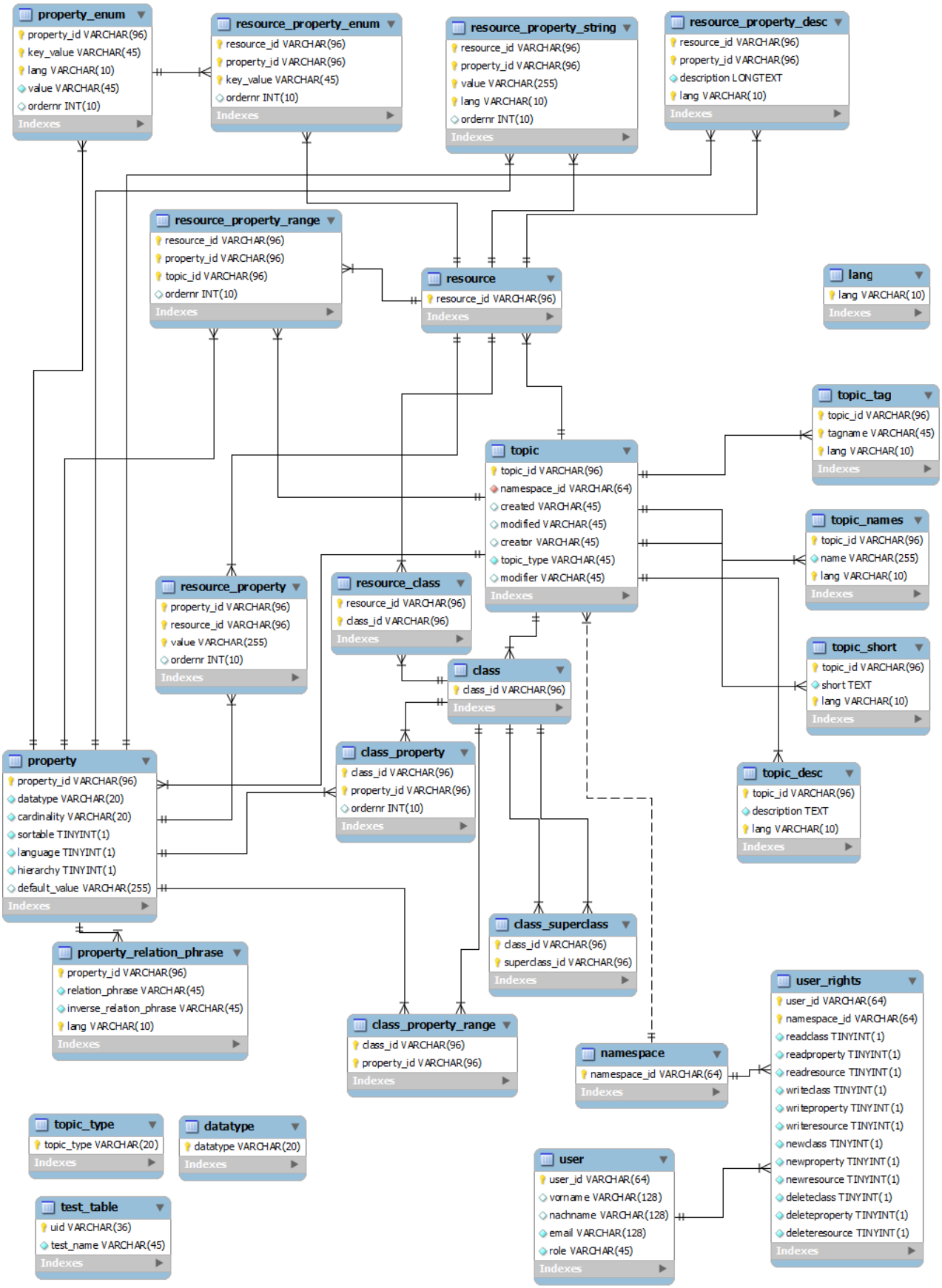


Abbildung 24: Infotool Datenbank Schema

5.5 Dateirepository

Eine weitere Anforderung war, dass Dokumente einfach in das Informationssystem integriert, verlinkt und verwaltet werden können. Dazu wurde ein einfacher Upload Mechanismus implementiert. Entsprechend dem spezifizierten Datenmodell wurde ein Topic *File* definiert mit bestimmten Eigenschaften (*filename*, *content_type*, etc.) welches bei einem Dateiupload automatisch im System angelegt wird. Auch wenn Dateien aus dem System heraus generiert werden wird automatisch eine Instanz (Ressource) im System angelegt. Diese angelegte Ressource kann dann einfach verlinkt werden, oder auch nur im System abgelegt sein um später danach zu suchen. Ein Download ist ebenfalls möglich. Abbildung 25 und Abbildung 26 zeigen ein beispielhaftes *File* Topic bzw. die dazugehörigen Eigenschaften (*filename*, *content_type*, …)

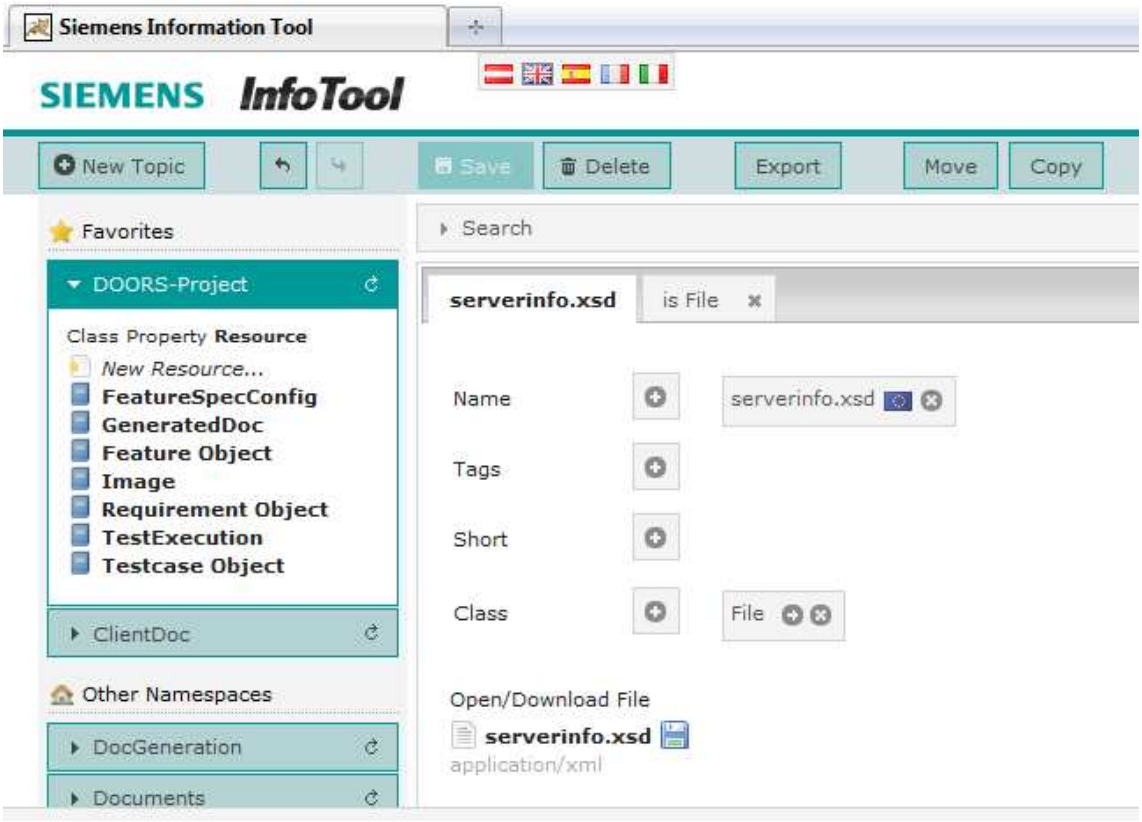


Abbildung 25: File Topic

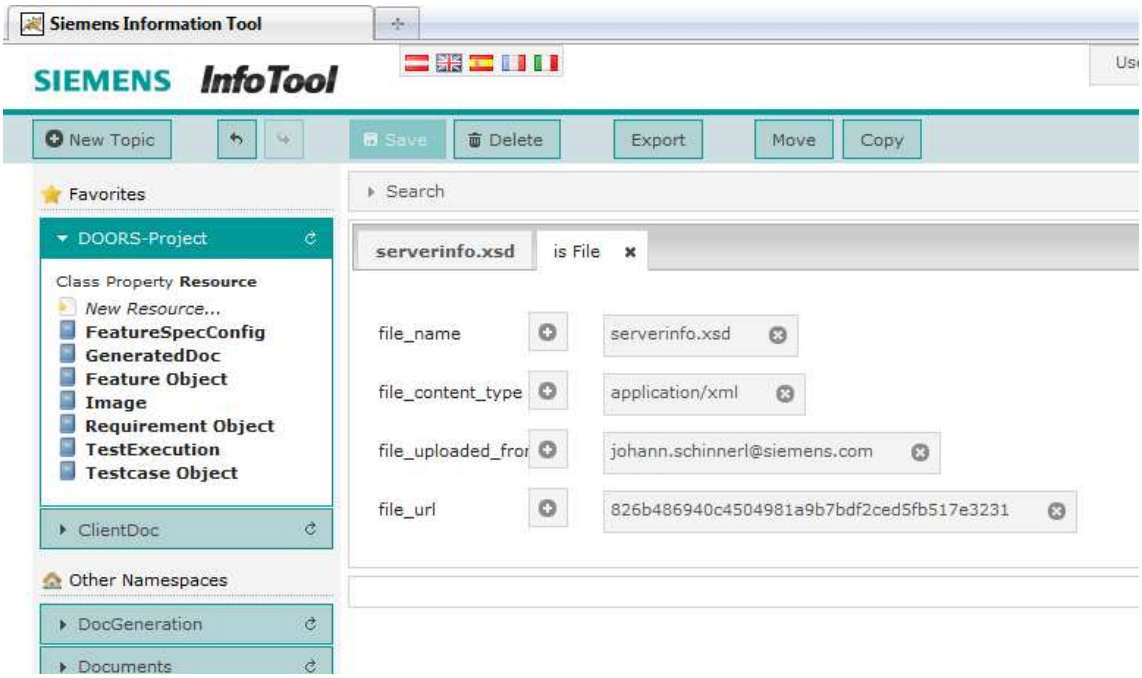


Abbildung 26: File Topic Eigenschaften

Die Datei selbst wird am Server abgelegt. Dazu gibt es ein eigenes Verzeichnis bzw. Repository. Um das mehrmalige Uploaden der selben Datei zu verhindern, wird beim Ablegen der Datei ein Hash-Wert über den Inhalt errechnet, die Datei umbenannt in *<hash>.<dateiendung>* und im Repository Verzeichnis des Servers abgelegt. Da es unter Microsoft Windows unter Umständen Probleme geben kann, wenn sehr viele Dateien in einem Ordner liegen, wurde ein Verfahren angewandt, welches unter anderem beim Apache Jackrabbit Projekt [Apache Jackrabbit, 2011] verwendet wird, indem man die Dateien nach deren Hash-Wert in entsprechenden Ordnern ablegt. Unten angeführtes Beispiel erläutert bezugnehmend auf Abbildung 27 dieses Verfahren:

Datei: *serverinfo.xsd*

Hash-Wert: *826b486940c4504981a9b7bdf2ced5fb517e323*

Ordner 1: *82*

Ordner 2: *6b*

Ordner 3: *48*

Dateiname: *826b486940c4504981a9b7bdf2ced5fb517e323.xsd*

Dateipfad: *<repository>/content/82/6b/48/826b486940c4504981a9b7bdf2ced5fb517e323.xsd*

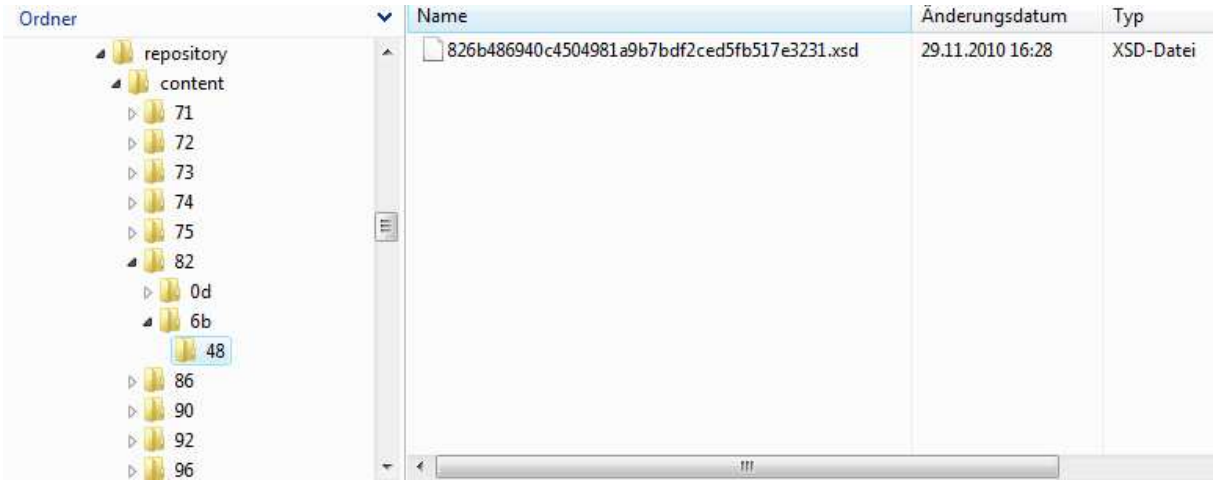


Abbildung 27: Beispiel für Dateiablage im Server Repository

5.6 Anwendungsfälle

Dieses Kapitel soll einige Anwendungsfälle für das Infotool darstellen. Es zeigt wie ein Client die vom Infotool zur Verfügung gestellten Informationen darstellen könnte.

5.6.1 Anlegen von Information (Anlegen eines Topics)

Das Anlegen eines neuen Topics läuft wie in Abbildung 28 gezeigt ab. Durch einen Klick auf *New Topic* gelangt man zur Eingabemaske eines neuen Topics. In diese Eingabemaske werden die grundlegenden Informationen zu einem Topic angelegt, das heißt ein Name, ein zugehöriger Namespace, sowie die zugehörige Klasse. Das System schlägt automatisch per Auto-Vervollständigung geeignete Klassen vor. In diesem Beispiel die Klasse *Person*. Es besteht auch noch die Möglichkeit eine eigene ID zu vergeben, oder eine generieren zu lassen. Mit Klick auf *Create* landet man in der Topic Ansicht. Diese ist in Laschen (Tabs) organisiert. Das erste Tab beinhaltet immer generelle Topic Informationen bzw. Metadaten. Man kann einen Namen, geeignete Tags oder eine Kurzbeschreibung (Short) vergeben. Im markierten unteren Bereich befindet sich leicht gräulich dargestellt weitere Metadaten Information, wie *created, creator, id*. Mit Klick auf die Lasche Person, werden die Properties die der Klasse *Person* zugeordnet sind angezeigt, hier kann man dann entsprechende Werte vergeben.

Dieselbe Topicansicht wird auch für das Bearbeiten von Informationen verwendet. Das heißt man kann die bereits eingetragenen Werte, verändern, löschen, oder neue hinzufügen.

Anzumerken ist weiters, dass der Client beim Speichern des Topics eine entsprechende JSON Repräsentation dieses Topics erstellt. Ein auf XML basierter Client würde ein dem XML Schema entsprechendes Topic erstellen und an den Server senden.

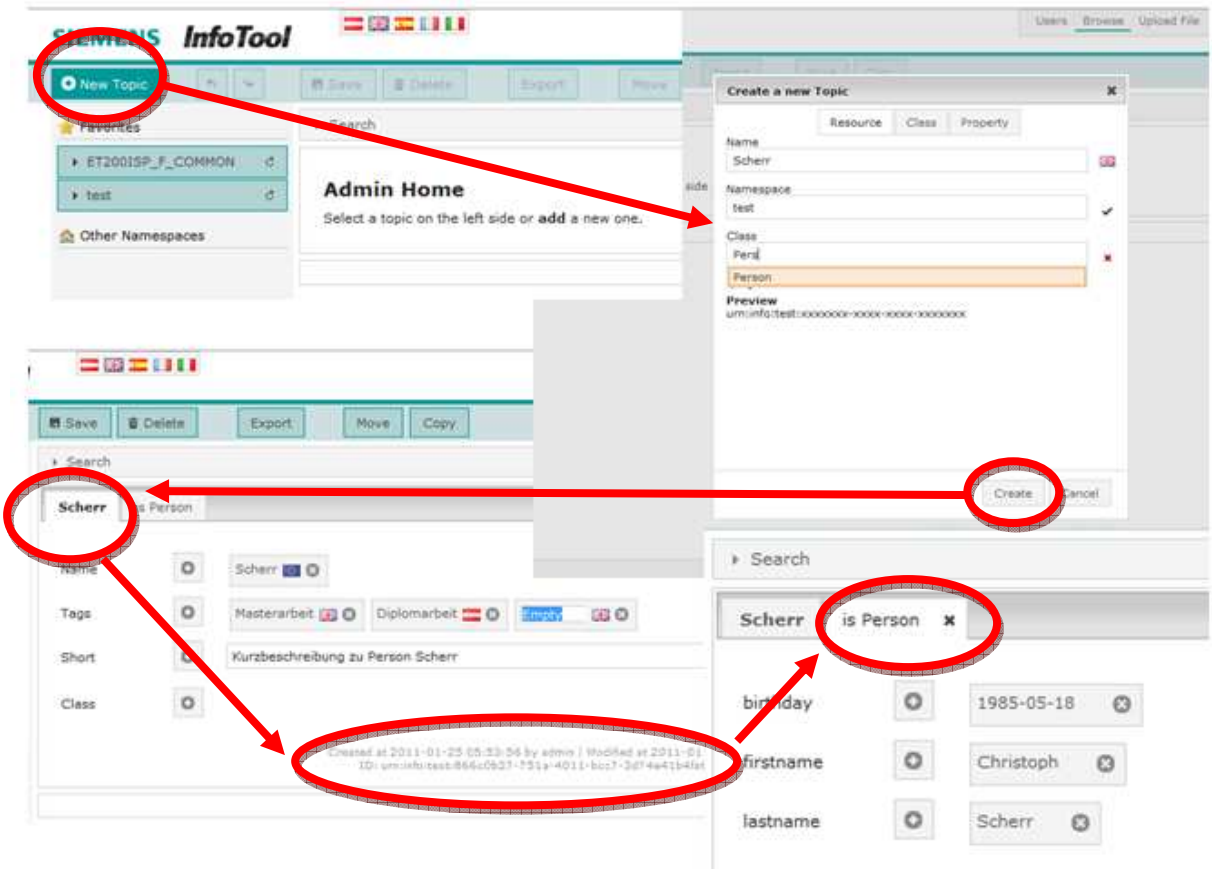


Abbildung 28: Anlegen eines Topics

5.6.2 Suchen von Information

Die Suchfunktion unterstützt mehrere unterschiedliche Features. Abbildung 29 zeigt die Suchmaske mit den Suchbegriffen *Feature* und *Block*. Grundsätzlich sind die Suchbegriffe UND-Verknüpft. Das heißt, es wird nach Informationen (Topics) gesucht, die beide Suchbegriffe beinhalten. Standardmäßig wird in den für jedes Topic definierten Metadaten Name, Kurzbeschreibung und Tags gesucht, dies kann aber eingeschränkt werden, so dass der Benutzer in bestimmten Fällen nur nach dem Namen oder Kurzbeschreibung suchen könnte.

Auf der rechten Seite befinden sich entsprechende Filter, um die Suchergebnisse einschränken zu können. Auch eine zeitliche Einschränkung ist möglich. Die Suche schlägt auch automatisch Klassen und Properties vor, die mit dem Suchbegriff in Verbindung stehen könnten. In diesem Beispiel schlägt das System die Klassen *FeatureSpecConfig*, sowie *Feature Object* und die Properties *details Feature*, sowie *verifies Feature* vor. Der Benutzer könnte nun einen oder mehrere dieser Begriffe wählen. Ein Anwendungsfall wäre wenn der Benutzer *Feature Objects* suchen möchte, dann wählt er diese *Feature Objects* aus und sucht auch nur in *Feature Objects*.

Die eigentlichen Suchergebnisse werden nach zugehörigen Namespace sortiert ausgegeben. In diesem Fall wurde ein Ergebnis im *Simatic* Namespace gefunden. Die gefunden Topics werden mit ihrem Namen, sowie ihrer Kurzbeschreibung angezeigt.

Wie schon erwähnt, schlägt das System geeignete Klassen oder Properties vor. Es präsentiert gefundene Topics in den Suchergebnissen, die diesen Klassen angehören bzw. mit diesen Properties in Beziehung stehen. Diese werden in der Liste für *relatedClassResources* bzw. *relatedPropertyResources* angeführt.

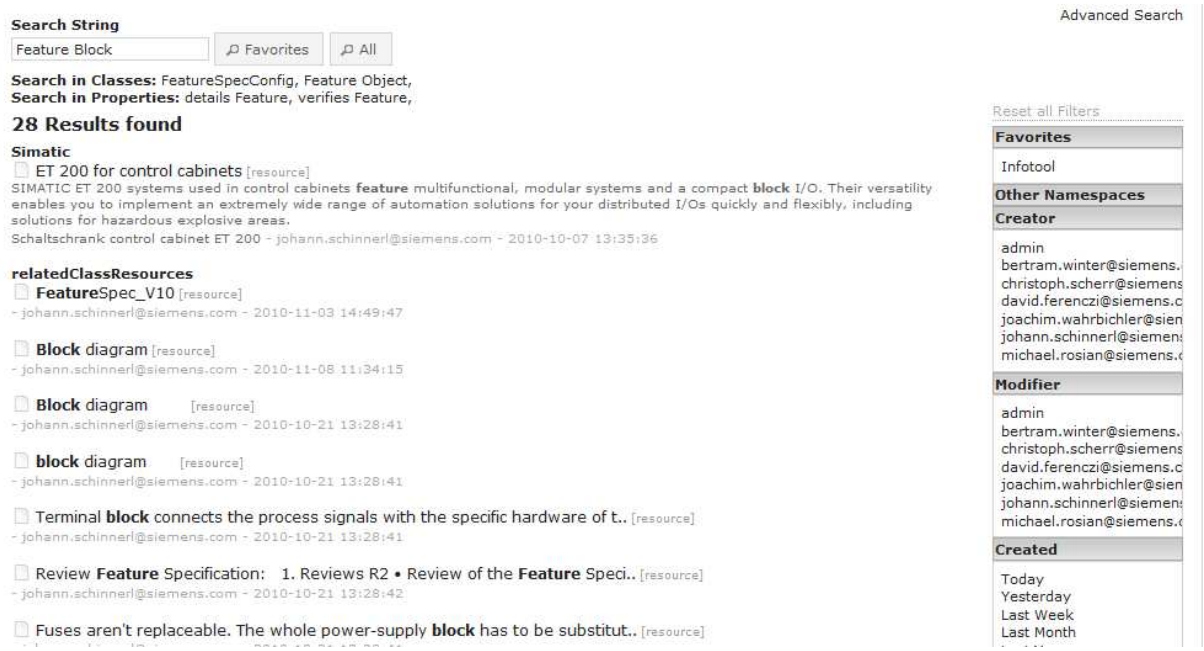


Abbildung 29: Infotool-Suchmaske

Die entsprechende Schnittstellen Dokumentation zur Suche befindet sich in Anhang A.

5.6.3 Darstellung der Information am Client

Um die im Infotool gespeicherten Informationseinheiten bzw. Topics in eine Struktur zu bringen, werden hier exemplarisch einige mögliche Darstellungsformen der Information am Client angeführt. Wie eingangs erwähnt, sind Topics gewissen Namespaces zugeordnet – der Client stellt diese auf der linken Seite dar, man kann diese auf- und zuklappen, sortieren und sich Favoriten definieren indem man einen Namespace nach oben Richtung Favoriten zieht. In diesem Fall sind die Namespaces *Infotool* und *System* in den Favoriten zu finden, die anderen Namespaces könnten weggeblendet werden.

Die angeführten Namespaces können aufgeklappt werden, um eine Übersicht der diesen Namespaces zugeordneten Topics zu erhalten. Diese kann nach Klassen, Properties und Ressourcen geordnet werden. Abbildung 30 zeigt den aufgeklappten *System* Namespace, samt einer Liste der dem *System* Namespace zugeordneten Klassen. Weiters ist der Namespace *ET200ISP_F_COMMON* geöffnet, welcher ein Projekt darstellt. Dieser beinhaltet hierarchisch angeordnete Ressourcen, standardmäßig sind die Ressourcen nach ihrer Klasse geordnet, mit einem Klick öffnet sich eine Baumansicht, wie in Abbildung 31 für Ressourcen der Klasse *Testcase Object.*

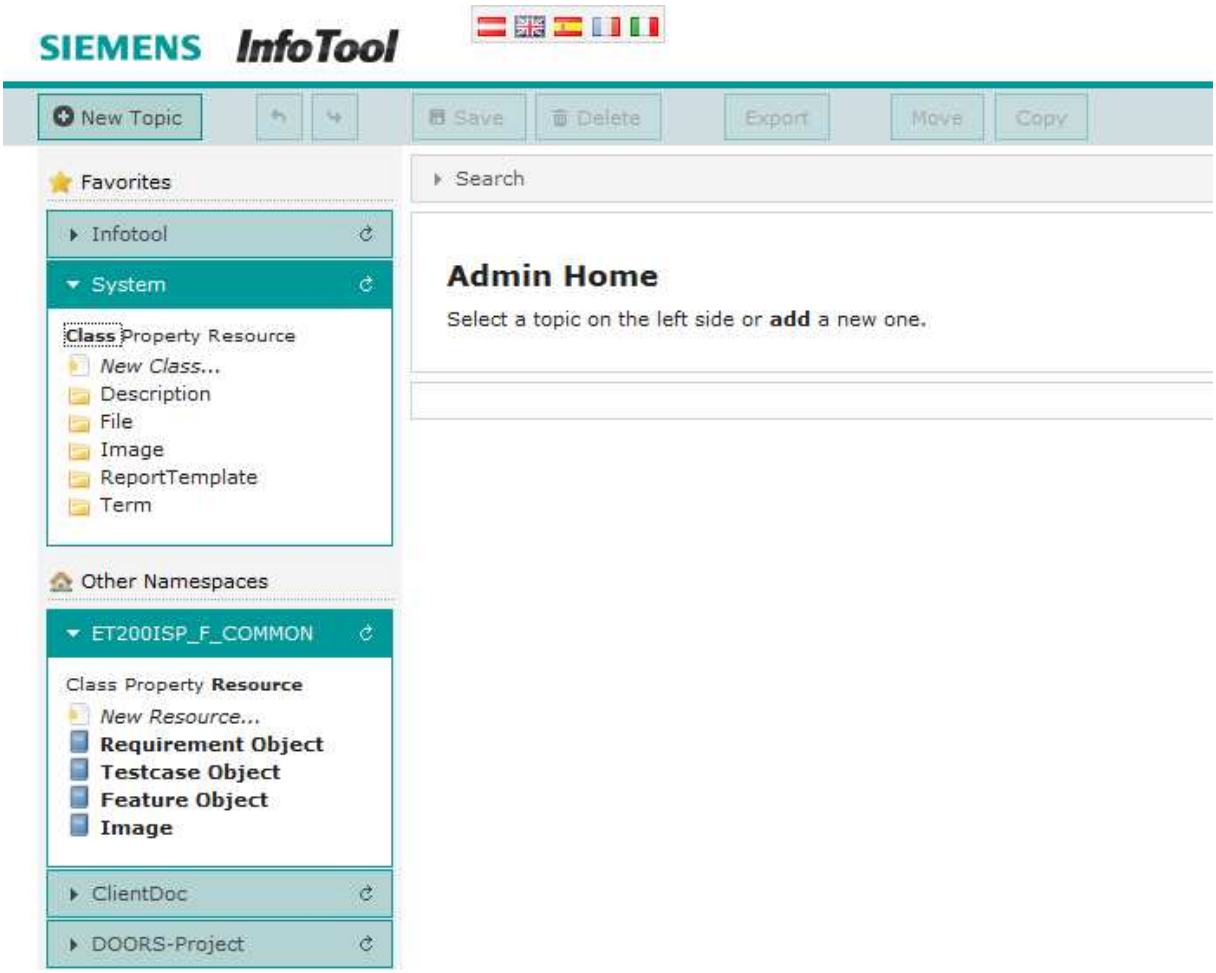


Abbildung 30: Darstellung von Namespaces im Infotool

Abbildung 31 zeigt eine mögliche Anwendung für die im Infotool gespeicherten Informationen, indem man eine Art Dokumentenansicht implementiert. Für diese Art der Darstellung wurde der Begriff *DocView* verwendet. Diese Darstellungsform zeigt, wie man die hierarchische Anordnung einzelner Informationseinheiten zur Dokumentenerstellung nutzen kann.

Jeder Abschnitt stellt ein eigenes Topic dar, der Name des Topics wird als Überschrift verwendet. Ein beliebiges Property mit Datentyp XHTML kann mit entsprechenden Daten gefüllt werden bzw. direkt in der *DocView* editiert werden. Man kann die Topics beliebig verschieben oder neu anordnen.

Dies ist nur eine mögliche Darstellungsform, die exemplarisch zur Veranschaulichung gewählt wurde. Entsprechende Clients könnten auch andere oder ähnliche Darstellungsformen wählen. Diese Darstellungsform zeigt nur eine mögliche Anwendung, mit Hilfe derer man wie gewohnt Information in Dokumentform erstellen könnte. Der Vorteil liegt jedoch darin, dass diese Information anschließend sehr strukturiert vorliegt.

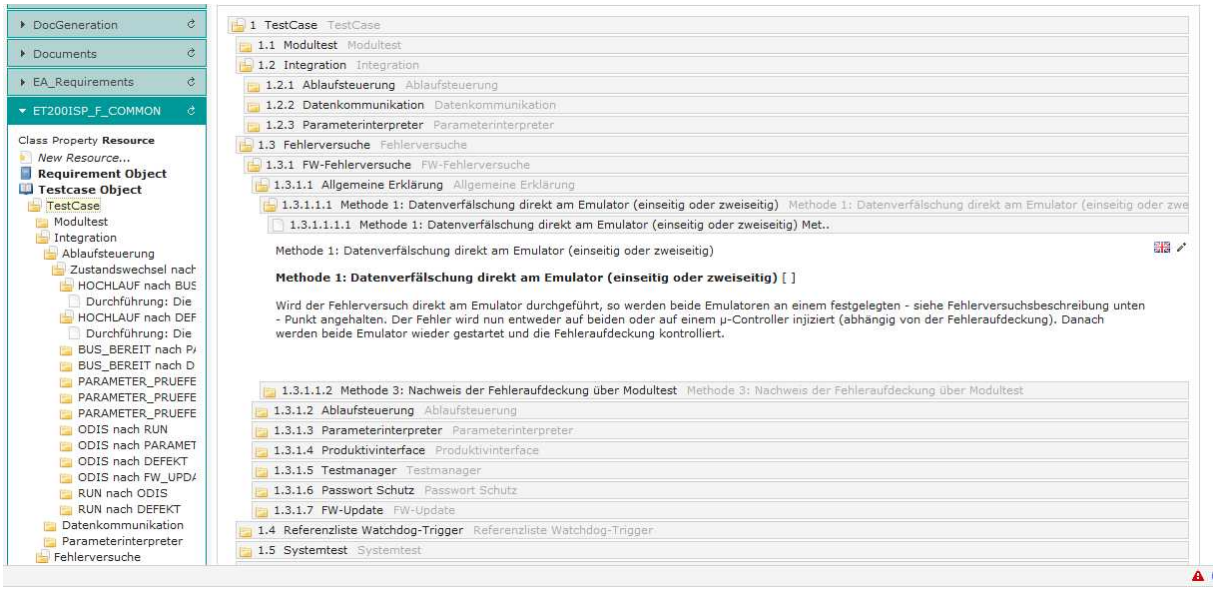


Abbildung 31: Darstellung der DocView im Infotool

5.6.4 Import / Export / Upload

Um externe Information in das Infotool integrieren zu können, wurden geeignete Schnittstellen (siehe Anhang A) definiert. Der Client stellt simple Upload Formulare zur Verfügung. Wie Dateien beim Upload im System integriert werden, wurde im vorhergehenden Kapitel zum Dateirepository beschrieben.

Import und Export unterstützen zwei unterschiedliche Formate. Einerseits per XML Datei, welche dem definierten XML-Schema entspricht und eine Liste von Topics beinhaltet. Andererseits per .zit Datei (ZIP Archiv, wobei das „*it*“ für Infotool steht), welche dieselbe XML Datei beinhaltet, diesmal aber gemeinsam mit zugehörigen Dateien, die wie im Dateirepository entsprechend ihren Hash-Werten abgelegt sind. Dies ist nötig, um auch die im Repository gespeicherten Dateien importieren oder exportieren zu können.

5.7 Generierung von Reports

Es gibt Szenarien, bei denen es nicht ausreicht die im Infotool abgelegte Information über die Online-Schnittstellen einzusehen oder darin zu navigieren. Beispiele hierfür wären:

- Es ist keine technische Infrastruktur vorhanden um eine Online-Verbindung zu einem Infotool-Server herzustellen
- Information muss als abgeschlossene Einheit, zum Beispiel als Word- oder PDF-Datei an Dritte zur Begutachtung verteilt werden
- Information wird explizit zum Zweck eingegeben, um daraus Dokumente wie zum Beispiel Bedienungsanleitungen, Requirements-Spezifikationen oder auch Masterarbeiten zu erstellen

Für alle angeführten Fälle ist es nützlich, eine Möglichkeit zur einfachen Generierung von Report-Dokumenten anzubieten. Anmerkung: Folgend wird „Report“, „Dokument“ und „Dokumentation“ synonym verwendet.

5.7.1 Formate von Reports

Da in den letzten Jahren Dokumentenformate auf Basis von XML vermehrt gebräuchlich wurden, oder sogar als internationale Standards etabliert wurden wie das OpenOffice XML Format [OpenOffice XML Format, 2011], das Microsoft Office Open XML Format [Microsoft Office Open XML Format, 2011] oder XHTML [W3.org XHTML, 2011], soll das Hauptaugenmerk für die Report-Generierung auf XML basierten Formaten liegen. Dafür spricht auch die Tatsache, dass die primäre Schnittstelle für Abfragen von Informationen aus dem Infotool das XML Format als Träger (siehe Anhang, Schnittstellen Referenz) verwendet.

Neben den Dateiformaten für diverse Textverarbeitungsprogramme (Microsoft Word, OpenOffice) ist vor allem im professionellen Umfeld das Portable Document Format (PDF – Verweis) von hoher Wichtigkeit. Dieses eignet sich wegen der weiten Verbreitung, der weitgehenden Herstellerunabhängigkeit und des langlebigen Dateiformats sehr gut für den Dokumentenaustausch zwischen Unternehmensgrenzen hinweg, aber auch für die Archivierung von sensitiven Inhalten. Daher ergab sich die Möglichkeit zur Generierung von Dokumenten in PDF-Format als Muss-Anforderung.

5.7.2 Werkzeuge und Standards

In der Entwicklungsabteilung des betreuenden Unternehmens sind in den vergangen Jahren bereits umfassende Erfahrungen zur Erstellung von Dokumentation aus XML Quellen gemacht worden. Diese Erfahrungen wurden bei der Konzeption und Implementierung des Infotool berücksichtigt. Insbesondere wurden die im Folgenden beschriebenen Werkzeuge und Standards eingesetzt.

5.7.2.1 XSLT

XSLT (siehe Kapitel 2.4.3) wird zur Umwandlung zwischen XML Darstellungen der verarbeiteten Information verwendet. Die Sprache selbst ist in XML notiert, was eine sehr einfache Generierung aller möglichen XML-basierten Ausgabeformate mit sich bringt. Es kommt die Version 2.0 der Sprache zum Einsatz, da hier sehr viele Einschränkungen der Vorgängerversion (Version 1.0, diese ist vor allem in Web-Browsern noch fast ausnahmslos verbreitet) beseitigt sind, und auch der Funktionsumfang (z.B. Unterstützung von Daten-Typen aus einem XML-Schema) stark erweitert wurde.

Als Implementierung des XSLT Prozessors wurde das kommerzielle Produkt „Saxon Enterprise Edition“ der Firma Saxonica [Saxonica.com, 2011] gewählt, da es sich hier um eine leicht am Infotool-Server einzubindende JAVA-Bibliothek handelt. Zudem scheint es sich derzeit um die einzige weitgehend Standard-konforme Implementierung von XSLT 2.0 am Markt zu handeln.

Meist wird XSLT zur Generierung von XML-Dokumenten aus in XML-Format vorliegenden Quellen eingesetzt. Die Version 2.0 der Sprache ermöglicht zwar die Einbeziehung von

unformatierten Text-Dokumenten und sogar die (eingeschränkte) Bearbeitung von geeignet kodierten Binärdaten, aber auch das in dieser Arbeit entwickelte Tool verwendet für das Generieren von Reports praktisch nur das Ausgabeformat XML. Sollten binäre Formate wie PDF erforderlich sein, muss deren Generierung in einem zusätzlichen Arbeitsschritt aus einem geeigneten XML-Format erfolgen. Mechanismen welche dies ermöglichen und erforderliche Werkzeuge werden weiter unten vorgestellt.

5.7.2.2 plain Text (text/plain)

Das Generieren von einfachem Klartext wird dem XSLT Prozessor mittels folgender Anweisung mitgeteilt:

```
<xsl:output method="text" encoding="UTF-8" indent="no"/>
```

Wegen der eingeschränkten Möglichkeit komplexere Texte geeignet zu strukturieren, zum Beispiel wäre eine Tabellen-Darstellung nur recht mühsam zu erhalten, hat dieses Ausgabeformat für technische Dokumentation eher geringe Bedeutung. Andererseits ist das Text-Format für die Generierung von Log-Dateien oder für manche ältere Dokumentenformate wie beispielsweise Rich Text Format (RTF) das Mittel der Wahl.

5.7.2.3 XHTML (application/xhtml+xml)

Mittels der eXtensible HyperText Markup Language [W3.org XHTML, 2011] abgelegte Information bildet den überwiegenden Inhalt der von Benutzern gesammelter Information. Die Semantik von XHTML bietet auch die von Textverarbeitungsprogrammen oder auch anderen Dokument-Formate [DocBook.org, 2011] bekannten Strukturierungs-Möglichkeiten wie zum Beispiel Abschnitte, Absätze, Aufzählungslisten, Tabellen, usw., ohne aber den Benutzer durch eine Vielzahl an Formatierungsmöglichkeiten abzulenken. Hier erleichtert die strikte Trennung von XHTML zwischen Inhalt (XHTML-Tags) und Formatierung (CSS-Styles) die spätere Aggregation der Inhalte in einen generierten Report mit einheitlichem Erscheinungsbild.

Eine weitere wesentliche Eigenschaft von XHTML, die es von anderen HTML Formaten unterscheidet, ist die Wahl von XML als Dateiformat und somit die Wohlgeformtheit der XHTML-Dokumente. Dies ist eine Grundvoraussetzung um überhaupt einer Bearbeitung mittels XSLT zugänglich zu sein. Ein Dokument in beispielsweise HTML4 [W3.org HTML4, 2011] würde bereits am XML-Parser einen Fehler generieren und die Transformation von vornherein verhindern.

Das Generieren von XHTML wird in XSLT mittels folgender Anweisung angezeigt:

```
<xsl:output method="xhtml" encoding="UTF-8" indent="no"/>
```

Wegen der Notierung von XHTML als XML ist aber auch folgende Anweisung für unsere Zwecke gleichbedeutend:

```
<xsl:output method="xml" encoding="UTF-8" indent="no"/>
```

Durch den Einsatz von XML-Namespaces ist in XHTML ein Weg geschaffen worden auch Nicht-HTML Inhalte aus anderen XML-Sprachen in einem Dokument unterzubringen. Bekannteste Beispiele sind hier sicher die Scalable Vector Graphic [W3.org SVG, 2011] oder mathematischer Formelsatz mittels MathML [W3.org MathML2, 2011]. Beides kann somit entweder aus einer internen Ablage im Infotool in einen Report kopiert, oder sogar über XSLT aus geeigneten Rohdaten generiert werden. Die Möglichkeiten zur Dokumenten-Generierung stehen somit in der Ausdrucksstärke den gängigen Textverarbeitungsprogrammen um nichts nach.

5.7.2.4 CSS

Mit der Einführung von XHTML wurde die direkte Formatierung (z.B. Farbe, Font, Zeichengröße, Rahmen, usw.) von Elementen eines HTML-Dokuments über Attribute verbannt. Formatierung auf folgende Art (style-Attribut) sollte vermieden werden:

```
<p style="float:right; font-size:80%; color: grey;">erforderlich</p>
```

Cascading Style Sheets [W3.org CSS, 2011] bieten nun die Möglichkeit die Formatierung von HTML-Elementen, diesmal einheitlich und an zentraler Stelle, vorzunehmen.

Die äquivalente Formatierung zum obigen Beispiel wäre, getrennt vom HTML Markup und nach Vergabe eines class-Attributes mit möglichst semantischer Bedeutung, in CSS formuliert:

```
table.parameter p.parameter-attribute {
        float:right; font-size:80%; color: grey;
}
```

Üblicherweise befinden sich CSS Anweisungen in einer separaten Datei und werden im Header von XHTML folgendermaßen eingebunden:

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Model Documentation</title>
    <link type="text/css" href="styles.css" rel="stylesheet" />
  </head>
```

Alternativ ist auch die Einbettung aller Formatierungs-Anweisungen direkt im Header einer XHTML-Datei möglich. Der CSS Code befindet sich dann innerhalb eines <*style*> Tags:

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Model Documentation</title>
    <style type="text/css" media="all">

          /* CSS Formatierungs-Anweisungen ...  */

    </style>
```

Dies hat den Vorteil, dass das gesamte Dokument in sich abgeschlossen und in nur einer einzigen Datei enthalten ist – was die Weitergabe an Kollegen etwas erleichtert. Mitunter sind CSS Stylesheets aber nicht die einzigen Dateien deren Inhalt nicht direkt im XHTML Text untergebracht werden. Obwohl technisch möglich, sind Bilder meist nicht in derselben Datei wie der HTML Code zu finden, sondern auf sie wird durch einen Link verwiesen (z.B.).

Gerade in einem Firmen-Intranet zeigen sich hier die Schwierigkeiten auf die man stößt wenn man versucht (X)HTML als Dokumentenaustausch-Format einzusetzen. Der Inhalt und das Erscheinungsbild eines HTML-Dokuments setzen sich meist aus mehreren einzelnen Teilen (Text, Bilder, Stylesheets) zusammen, welche als Ressourcen oft getrennt abgelegt sind und erst im Browser zusammengefügt werden. Das ist im Internet auch problemlos möglich, da in der Regel alle Inhalte für jeden und von jedem Ort aus gleichermaßen zugänglich sind.

Innerhalb einer Firma ist es für einzelne Mitarbeiter oder Arbeitsgruppen, schon allein aus Gründen der Informationssicherheit, nicht so einfach möglich einen HTTP-Server zu betreiben. Auch wenn zum Beispiel ein Infotool-Server für Mitglieder eines Entwicklungs-Teams installiert wird, muss der Zugriff für alle anderen Mitarbeiter der Firma unterbunden oder zumindest stark eingeschränkt werden. Will man aber doch Dokumentation an diese Kollegen (zum Beispiel vom Produkt-Marketing oder Systemtest) übergeben, bietet sich meist nur die Möglichkeit diese mittels E-Mail an den Adressaten zu schicken. Hier ist es besser eine einzige Dokumenten-Datei im Anhang vorzufinden als ein Sammelsurium von Einzeldateien (auch der Umgang mit Zip-Archiven ist wenig praktikabel).

Eine Möglichkeit nach der Dokumenten-Generierung nur eine einzige Datei in den Händen zu halten bietet die Konvertierung in das PDF-Format, und soll als nächstes beschrieben werden.

5.7.2.5 PDF

Lange Zeit war die Generierung von PDF aus XML/XHTML Quellen durch Einsatz von Tools möglich, die den Standard Extensible Stylesheet Language Formatting Objects (XSL-FO bzw. XSL, Anmerkung: Nicht zu verwechseln mit XSLT) [W3.org XSL, 2011] implementieren. Um PDF-Format zu erhalten, mussten zwei Schritte erledigt werden: Zuerst musste mittels XSLT Transformation aus den Quellen eine exakte, in XML notierte, Beschreibung von Inhalt und Layout des Dokuments erzeugt werden.

```
<fo:block>
  ..
  <fo:inline font-size=”80%” color="grey">erforderlich</fo:inline>
  ..
</fo:block>
```

Danach erledigt ein spezielles XSL-FO Formatter Tool die weitere Umsetzung in das binäre Ausgabeformat PDF. Als Open-Source Vertreter ist das Tool Apache FOP [Apache FOP, 2011] bekannt.

Diese Methode ist wegen der sehr detaillierten Einflussmöglichkeiten auf die Formatierung des Ergebnis-Dokuments für die Generierung hochwertiger Dokumente für

Publikationszwecke von Bedeutung. Für die schnelle und einfache Ausgabe von zum Beispiel Entwicklerdokumenten, wo der technische Inhalt mehr zählt als ein ausgeklügeltes Layout, ist der Aufwand doch recht hoch.

Vor allem wenn schon eine durch den Einsatz von CSS brauchbare Browser-Darstellung vorliegt und sich die gedruckte Fassung nicht allzu sehr davon unterscheiden soll, ist ein doppelter Aufwand der Erstellung von Format-Anweisungen für CSS und auch XSL-FO nicht sinnvoll. Das wurde auch vom W3C Konsortium erkannt, und der CSS Standard wurde im Laufe der Zeit so erweitert, dass damit auch andere Ausgabemedien als ein Computer-Bildschirm berücksichtigt sind.

Der für die spätere PDF-Erzeugung relevante Standard findet sich unter dem Namen CSS Paged Media [W3.org CSS Paged Media, 2011]. Hier wird CSS um Formatierungsanweisungen für zum Beispiel Seitengrößen, Seitennummerierung, Kopf- und Fusszeilen erweitert welche für gedruckte Dokumentation nötig sind.

Es exsistiert ein flexibler Mechanismus um aus demselben XML-Dokument wahlweise Formatierungen für jedes unterstützte Ausgabegerät wirksam werden zu lassen: Das „media-Attribut“ bei der Einbindung eines externen Stylesheets oder im *<style>* Element, somit ändern sich der Header im XHTML-Code zu:

```
<style type="text/css" media="all">
   /* gemeinsame CSS Formatierungs-Anweisungen für alle Ausgabemedien  */
</style>
<style type="text/css" media="print">
   /* CSS Formatierungs-Anweisungen für  seitenorientierte Medien */

      @page {
        size: A4;
        margin: 30mm 18mm 25mm 20mm;
        …

      }
</style>
```

Diese Methode wird auch beim Infotool für die Generierung von PDF-Dokumenten angewandt. Der überwiegende Teil der Formatierungs-Anweisungen befindet sich im Element *<style .. media=“all“>* das alle Ausgabemedien anspricht. Formatierungen die zusätzlich für druckbare Medien gedacht sind, oder welche vorhandene allgemeine Formatierungen ersetzen, sind im Element *<style .. media=“print“>* zusammengefasst.

Nach wie vor nötig ist ein Tool das die CSS paged media Formatierung in das gewünschte Dateiformat (PDF, oder auch andere Formate wären denkbar) umsetzt. In diesem Projekt wird dazu PrinceXML [PrinceXML, 2011] der Firma YesLogic eingesetzt.

5.7.2.6 WordML

Das Dokumentenformat Wordprocessing Markup Language [Microsoft WordML, 2011] wurde von der Firma Microsoft anlässlich der Einführung der Produktlinie Office 2003 als XML basiertes Dateiformat für das Textverarbeitungsprogramm Microsoft Word eingeführt.

WordML wurde zwar später vom standardisierten Office Open XML abgelöst, es gibt aber Gründe dieses Format an dieser Stelle zu behandeln: Zum einen ist die Office Version 2003 in den Intranets großer Firmen (wie auch der Siemens AG) noch sehr stark verbreitet und dient dort als Quasi-Standard zum Erstellen und Austausch von Dokumenten. Andererseits legt das Format sämtliche Information im Dokument (inklusive Bilder) in einer einzigen XML-Datei ab. Dies ermöglicht die automatisierte Generierung eines Dokuments als Ergebnis einer XSLT Transformation ohne irgendwelche Erweiterungen nutzen oder ohne ein zusätzliches Post-Processing durchführen zu müssen.

Da diese Masterarbeit mit Hilfe von Microsoft Office (in der Version 2007) erstellt wird, und im Rahmen der Arbeit ein Tool entwickelt wurde das zur automatisierten Generierung von Dokumenten fähig ist, liegt es nahe diese Fähigkeit nun auch anhand des folgenden Beispiels unter Beweis zu stellen.

5.8 Beispiel: Schnittstellen Dokumentation zum Infotool

Wie im vorhergehenden Kapitel erwähnt, wurde ein Mechanismus implementiert, mit Hilfe dessen man aus den im Infotool gespeicherten Informationen Reports generieren kann. Um dies zu veranschaulichen, wurde die gesamte Schnittstellen Dokumentation des Infotool im Infotool selbst erstellt und anschließend ein Dokument im WordML Format generiert. Dieses generierte Dokument wurde 1:1 in diese Arbeit importiert und ist im Anhang A verfügbar.

Dieses Kapitel soll die Vorgehensweise bzw. die Umsetzung veranschaulichen.

Wie schon in Kapitel 5.1 erwähnt handelt es sich beim Infotool um eine Client/Server Anwendung. Die Kommunikation zwischen dem Browser (Client) und der Web-Applikation (Server) erfolgt mittels Nachrichten genau definierten Inhalts. Diese Definitionen werden in der Schnittstellen-Dokumentation zusammengefasst, welche auch mit Hilfe des Infotools modelliert wurde.

Dazu wurden die Klassen *WebOperation*, *Parameter* und *XMLSchema_Images* im Infotool angelegt und mit Daten gefüllt. Es folgt eine Beschreibung der angelegten Informationen.

Die Klasse *WebOperation* stellt eine einzelne Web Operation, bzw. Server Schnittstelle dar. Sie besitzt aus in Tabelle 10 angeführten Properties.

Property	**Datentyp (Kardinalität)**	**Beschreibung**
method	Enumeration (0-1)	Auswahl der http Methode der Schnittstelle GET, PUT oder POST
operation	string (0-1)	Die Operation gibt wie der Name schon sagt die jeweilige Operation an. Beispielsweise op=list, op=save, op=new,

		etc.
parameters	Parameter (*)	Liste aller weiteren Parameter die für diese Schnittstelle angegeben werden können. Diese Liste hat als Datentyp die Klasse *Parameter* (siehe nachfolgende Tabelle)
pathname	string (0-1)	Angabe eines Pathnames für die Schnittstelle. Wird an die Server Root URL angehängt. Ein Beispiel wäre die Import Schnittstelle, die unter …/infotool/import aufgerufen werden kann. *import* stellt den *pathname* dar.
responseXMLSchema	File (0-1)	Dieses Property erlaubt es ein XML Schema anzuhängen
responseXMLSchema_ Images	XMLSchemaImage (*)	Dieses Property erlaubt es XML Schema Images anzuhängen, welche einfache Screenshots von XML Schemata darstellen.

Tabelle 10: Properties der Klasse WebOperation

Die Klasse *Parameter* stellt einen konkreten Parameter einer *WebOperation* dar. Tabelle 11 beschreibt die Properties der Klasse *Parameter*.

Property	**Datentyp (Kardinalität)**	**Beschreibung**
optional	boolean (0-1)	Gibt an, ob der Parameter optional ist
parameter_name	string (0-1)	Wie der Name schon aussagt, gibt *parameter_name* den Namen des Parameters an
multiple	boolean (0-1)	Gibt an, ob der Parameter mehrfach verwendet werden kann
possible_values	string (*)	Eine Liste möglicher Werte die dieser Parameter annehmen kann

Tabelle 11: Properties der Klasse Parameter

Abbildung 32 zeigt eine beispielhafte Ressource *List Hierarchy* der Klasse *WebOperation*. Auf diese Art wurden sämtliche Schnittstellen des Infotools definiert.

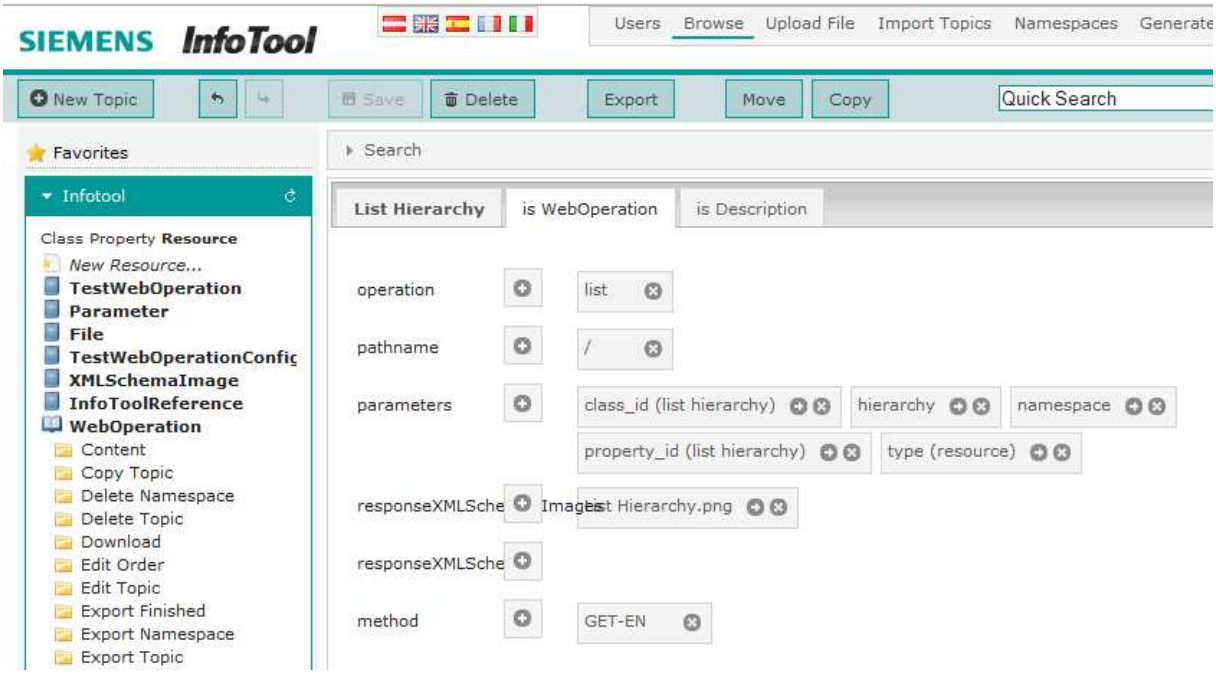


Abbildung 32: WebOperation List Hierarchy im Infotool

5.8.1 Generierung von Reports im Infotool – Überblick

Wie in Kapitel 5.7 beschrieben erfolgt die Generierung von Reports mittels XSLT Transformationen. Die Funktion *Generate Doc* des Client stößt am Server eine Transformation an, die ab diesem Zeitpunkt im Hintergrund abläuft.

Als Eingabe-Dokument für die Transformation dient die XML-Repräsentation des *ReportTemplate* (siehe nächstes Kapitel). Dies ist ausreichend, da sämtliche weitere Information vom XSLT-Prozessor über die *document()* Funktion von XSLT oder die Funktion *doc()* von XPath 2.0 Funktionen [W3.org XPath, 2011] vom Server geholt werden kann.

Im Prinzip kann die vollständige Transformation und somit die Dokumentengenerierung auch gänzlich außerhalb der Infotool Server-Infrastruktur ablaufen. Das Entwickeln von XSLT-Stylesheets kann zum Beispiel im XML-Editor oXygen [Oxygen XML Editor, 2010] erfolgen, auf Knopfdruck kann die Transformation angestoßen werden und die Daten werden über das HTTP-Protokoll vom Server geholt.

Für den Prozess der Generierung hat sich ein mehrstufiger Ansatz der „Pipeline Verarbeitung“ bewährt, der in den nächsten Abschnitten beschrieben wird. Es wird in jeder Stufe ein Aspekt des Generierungsprozesses implementiert und das Ergebnis im XML-Format als Input in die nächste Stufe eingespeist. Dies ist in XSLT ab der Version 2.0 auch sehr einfach implementierbar da man XML-Datenstrukturen als Variable speichern und (anders als die Version 1.0) darauf weitere Transformationen ausführen kann. Dieser Ansatz ist vergleichbar mit dem Vorgehen von Publishing-Tools wie Apache Cocoon [Apache Cocoon, 2011] oder der Grundidee hinter der *XML Pipeline Language* XProc [W3.org XProc, 2011]

Die Stufen der Verarbeitung sind:

1. Ausgangspunkt: XML-Repräsentation des *ReportTemplate* (siehe nächstes Kapitel)
2. Filtern und Vereinfachen der Information am Server
3. Aggregation der Information aus dem Infotool
4. Transformation in eine der Problem-Domäne angepasste Struktur
5. Transformation in das Ziel-Format
6. Internationalisierung

Dabei finden Stufen 3 bis 6 innerhalb einer gemeinsamen XSLT-Transformation statt.

5.8.2 Definieren des Inhalts eines Reports

Der Umfang der Information welcher im Report enthalten sein soll, wird vorab mittels folgender Parameter im *ReportTemplate* eingestellt

- *namespace*: Der Namespace aus dem die Eingabedaten zu nehmen sind
- *class*: Die Klasse der Topics welche bearbeitet werden sollen

Die Interpretation dieser Parameter obliegt dem verwendeten XSLT Stylesheet. Hierin ist kodiert, ob die Generierung zum Beispiel hierarchisch oder tabellarisch in Listenform erfolgen soll. Weiters können beliebige zusätzliche Parameter berücksichtigt werden welche durch die flexible Definition von *ReportTemplates* (d.h. davon abgeleiteter Klassen) dem XSLT-Prozessor mitgeteilt werden können. Abbildung 33 zeigt ein beispielhaftes *ReportTemplate*, welches für die Generierung eines Reports im PDF-Format verwendet wird.

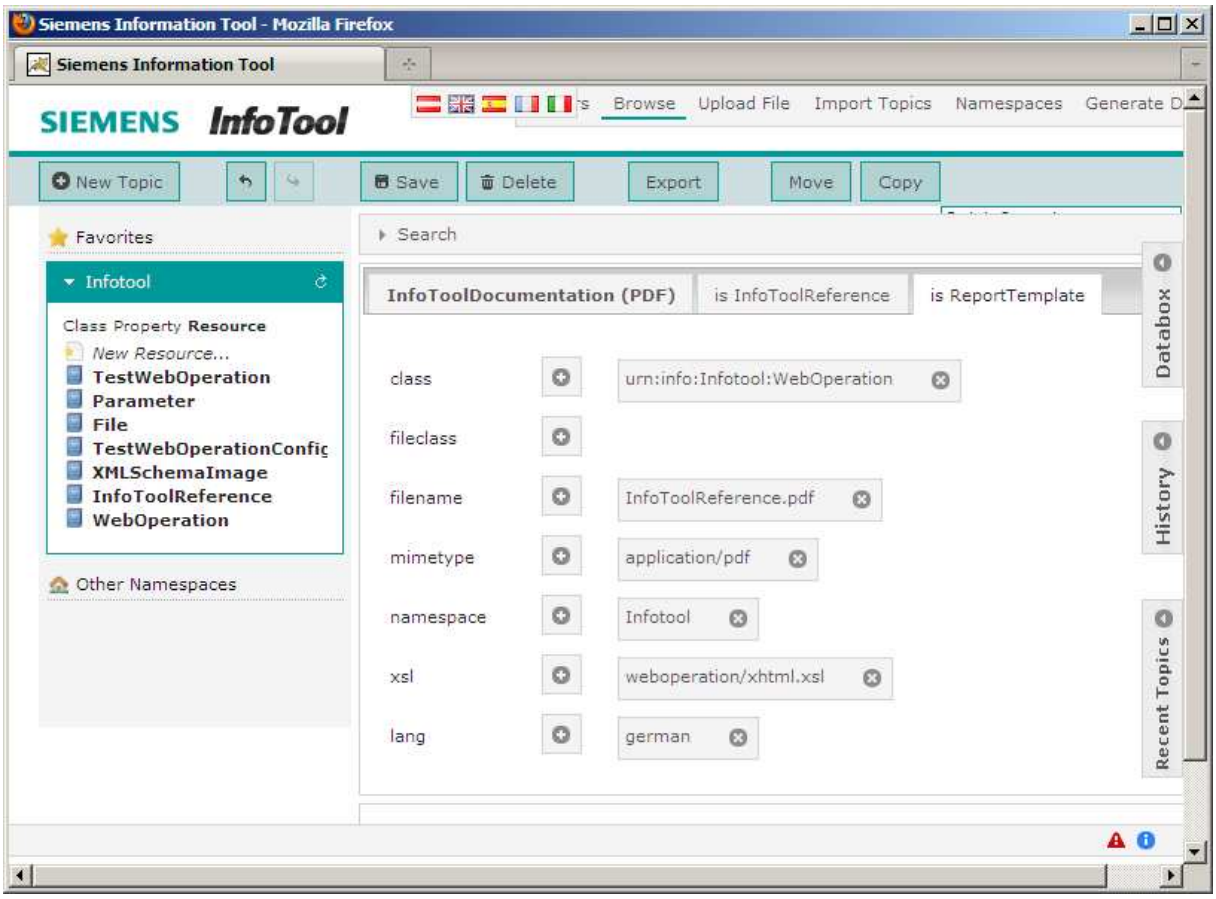


Abbildung 33: Beispielhaftes ReportTemplate

5.8.3 Filtern und Vereinfachen der Information am Server

Sofern nicht gerade Wörterbücher generiert werden, ist es nicht notwendig die Inhalte des Infotools in allen abgelegten Sprachvarianten an den Client zu übergeben. Deshalb ist bereits am Server eine einfache Art der Filterung vorgesehen, der relevante Parameter im *ReportTemplate* ist

- *lang*: Spracheinstellung für die Dokumenten-Generierung (z.B.: „de“, „en“)

Einen Vergleich zeigen die beiden nachfolgenden Web Operationen, abgebildet in Abbildung 34 und Abbildung 35. Während die *Get* Operation alle mehrsprachig vorhandenen Texte liefert, sind bei der *View* Operation mehrsprachig vorhandene Texte (z.B. siehe *name*, *short* und *item* Elemente) nur noch in der gewünschten Version vorhanden. Zusätzlich werden noch unnötige beschreibende Elemente (z.B. die Namen von Klassen oder Properties, siehe *class* oder *prop* Elemente) ausgefiltert. Falls kein Text in der gewünschten Sprache existiert, wird der erstbeste Text übernommen – dies ist der Grund warum das *lang* Attribut doch noch nötig ist. Damit können diese Texte in der „falschen“ Sprache speziell gekennzeichnet werden, oder bei der Generierung von WordML-Dokumenten kann so die Sprache für die Rechtschreibprüfung trotzdem korrekt gesetzt werden.

```
<topic id="urn:info:Infotool:modifier_search" type="resource" created="2010-11-11T10:36:11+01:00"
  <topicinfos>
    <name>
      <item lang="en">modifier (search)</item>
    </name>
    <short>
      <item lang="de">schränkt den Suchraum auf Topics ein welche zuletzt von Benutzer "modifier"
      <item lang="en">restrict the search space to topics that were last modified by user "modifi
    </short>
    <tags />
  </topicinfos>
  <properties>
    <class ref="urn:info:Infotool:Parameter" superclass="false">
      <name>
        <item lang="en">Parameter</item>
      </name>
      <prop datatype="boolean" count="0-1" hierarchy="0" sortable="0" language="0" type="data" re
        <name>
          <item lang="en">multiple</item>
        </name>
        <item>1</item>
      </prop>
      <prop datatype="boolean" count="0-1" hierarchy="0" sortable="0" language="0" default="false
        <name>
          <item lang="en">optional</item>
        </name>
        <item>1</item>
      </prop>
```

Abbildung 34: Operation GET: (/infotool/?op=get&id=urn:info:Infotool:modifier_search) (Auszug)

```
<topic id="urn:info:Infotool:modifier_search" type="resource" created="2010-11-11T10:36:11+01:00"
  <topicinfos>
    <name lang="en">modifier (search)</name>
    <short lang="en">restrict the search space to topics that were last modified by user "modifie
  </topicinfos>
  <properties>
    <class ref="urn:info:Infotool:Parameter" superclass="false">
      <prop datatype="boolean" count="0-1" hierarchy="0" sortable="0" language="0" ref="urn:info:
        <item>1</item>
      </prop>
      <prop datatype="boolean" count="0-1" hierarchy="0" sortable="0" language="0" default="false"
        <item>1</item>
      </prop>
      <prop ref="urn:info:Infotool:parameter_name" datatype="string" count="0-1" hierarchy="0" so
        <item lang="data">modifier</item>
      </prop>
      <prop ref="urn:info:Infotool:possible_values" datatype="string" count="*" hierarchy="0" sor
    </class>
  </properties>
  <children/>
</topic>
```

Abbildung 35: Operation „View Topic“ berücksichtigt das sprachspezifische Filtern mit optionalem Parameter „lang“: (/infotool/?op=view&id=urn:info:Infotool:modifier_search&lang=en) (Auszug)

5.8.4 Aggregation der Information aus dem Infotool

Dieser Schritt kann einmal gemeinsam für alle möglichen Generier-Fälle definiert werden. Deshalb sind die zugehörigen XSLT-Templates in einer Datei *report_common.xsl* bereits vorinstalliert. Eine Verwendung der Variablen und Templates darin ist durch einfache *import* Anweisung in anderen Stylesheets möglich.

Es werden hier, wie Listing 1 zeigt, einfach sämtliche Topics gemäß gegebenen Namespace und Klassenzugehörigkeit samt den mit ihnen per Relationen verbundenen Topics vom Server geholt und in einer Variablen *$topic-elements* abgelegt:

```
<!-- fetch a list of resources in a certain namespace
   - use operation "infotool/?op=list&type=resource&namespace=Infotool&hierarchy=true"
  -->

<xsl:variable name="top-resource-list"
  select="document(it:makeResourceListURL($namespace))//class[@ref eq $topic]/resources/resource"/>

<!--
  -   fetch all (possibly linked) topic documents from the server
  -   and collect in a global variable $topic.
-->

<xsl:variable name="topic-elements" as="element(topic)*">
  <xsl:apply-templates select="$top-resource-list" mode="topics">
    <xsl:sort select="name/item[1]"/>
  </xsl:apply-templates>
</xsl:variable>
```

Listing 1: Zugriff auf Topics per XSLT

Es wurde zunächst nur eine einfache Variante implementiert, die voraussetzt dass der Graph der Topics und ihrer Relationen kreisfrei ist. Wenn das nicht der Fall ist, können Variablen und Templates beim Importieren dieses Stylesheets beliebig ersetzt (überladen) werden.

Die nächste Stufe der Generierung erfolgt durch Transformation der Variable *$topic-elements*

5.8.5 Transformation in eine der Problem-Domäne angepasste Struktur

Trotz der flexiblen Definition von Klassen und Properties im Infotool weisen alle Topic-Instanzen dasselbe XML-Schema auf. Abbildung 36 zeigt noch einmal eine beispielhafte XML Repräsentation eines Topics.

```
<topic id="urn:info:Infotool:Copy_Topic" type="resource" created="2010-11-10T15:09:20+01:00" modified="201
   <topicinfos>
      <name lang="en">Copy Topic</name>
      <short lang="de">Kopiert den Inhalt eines Topics in ein neu angelegtes Topic.</short>
   </topicinfos>
   <properties>
      <class ref="urn:info:Infotool:WebOperation" superclass="false">
         <prop ref="urn:info:Infotool:operation" datatype="string" count="0-1" hierarchy="0" sortable="0"
            <item lang="data">copy</item>
         </prop>
         <prop ref="urn:info:Infotool:pathname" datatype="string" count="0-1" hierarchy="0" sortable="0" l
            <item lang="data">/</item>
         </prop>
         <prop ref="urn:info:Infotool:parameters" datatype="resource" count="*" hierarchy="0" sortable="1"
```

Abbildung 36: Beispielhafte XML Repräsentation eines Topic

Für die weitere Verarbeitung ist es meist nützlich wenn die XML-Elemente Namen besitzen welche der Problem-Domäne des generierten Inhalts entsprechen. Zum Beispiel sollte ein Topic einer *WebOperation* einfach mit folgendem Start-Tag beginnen:

```
<WebOperation id="urn:info:Infotool:Copy_Topic">
```

Eine weitere Transformation (Listing 2) führt dieses durch und speichert das Ergebnis in Variable *$domain-elements*:

```
<!--
   -  build domain specific data structure from topic hierarchy
   -
  -->

<xsl:variable name="domain-elements" as="element(report)">
  <report lang="{$language}">
    <xsl:apply-templates select="$topic-elements" mode="domain"/>
  </report>
</xsl:variable>
```

Listing 2: Transformation eines Topics in XSLT

Diese Transformation wird auch noch zur Gänze mit den Templates des gemeinsamen Stylesheets *report_common.xsl* durchgeführt, sofern diese nicht von importierenden Stylesheets überladen wurden. Die Namen der XML-Elemente der Problem-Domäne werden per Default einfach aus dem Namen der Topics generiert – den Fall von doppelt vergebenen Namen müsste man mit zusätzliche definierten Templates abfangen.

Die Darstellung eines Topics vereinfacht sich wie in Abbildung 37 gezeigt (Auszug):

```
<WebOperation id="urn:info:Infotool:Copy_Topic">
  <name lang="en">Copy Topic</name>
  <short lang="de">
    <para>Kopiert den Inhalt eines Topics in ein neu angelegtes Topic.</para>
  </short>
  <operation>
    <item lang="data">copy</item>
  </operation>
  <pathname>
    <item lang="data">/</item>
  </pathname>
  <parameters>
    <Parameter id="urn:info:Infotool:id">
      <name lang="en">id</name>
      <short lang="de">
        <para>die ID einer Topic.</para>
      </short>
```

Abbildung 37: Vereinfachte Darstellung eines Topic nach einer XSLT Transformation

Die nächste Stufe der Generierung erfolgt durch Transformation der Variable *$domain-elements*.

5.8.6 Transformation in das Ziel-Format

Das gewünschte Ziel-Format der Dokumentation muss im verwendeten *ReportTemplate* eingetragen werden, die relevanten Parameter sind:

- *filename*: wird später beim Download als lokaler Dateiname voreingestellt
- *mimetype*: ist beim Öffnen direkt im Browser relevant

- *xsl*: Pfad zu einem XSLT-Stylesheet, dass die Transformation durchführt

Ein beispielhaftes *ReportTemplate* ist noch mal unter Abbildung 33 zu sehen.

Soll aus dem Ergebnis der Generierung ein PDF Dokument generiert werden, so ist der *mimetype* auf „application/pdf“ zu setzen. Der Infotool-Server führt dann den zusätzlichen Schritt der PDF-Konvertierung durch.

Für die Ablage am Repository kann noch über den Parameter *fileclass* bestimmt werden unter welcher Klasse das generierte Dokument abgelegt werden soll; bleibt er leer, so wird später einfach *File* verwendet. Dieser Parameter ist für die Speicherung im Infotool selbst notwendig, da ja das generierte Dokument nicht nur zum Download zur Verfügung gestellt wird, sondern auch im Infotool selbst abgelegt wird.

Dies ist der erste Verarbeitungsschritt der nicht mit den vorgefertigten Templates in Stylesheet *report_common.xsl* ausgeführt werden kann - hier müssen je nach gewünschtem Format, Inhalt und Layout des generierten Dokuments eigene Templates bereitgestellt werden. Im Rahmen dieser Arbeit wurden zur Generierung der Schnittstellen-Referenz des Infotools folgende Stylesheets bereitgestellt:

- *word.xsl*: generiert die Schnittstellen-Referenz im WordML-Format
- *xhtml.xsl*: generiert die Schnittstellen-Referenz im XHTML-Format

Listing 3 zeigt ein Template für die Parameter-Tabelle in XHTML

```
<xsl:template match="Parameter">
  <tr>
    <td class="name">
      <p class="name">
        <xsl:value-of select="if (parameter_name/item) then parameter_name/item else name"/>
      </p>
      <xsl:apply-templates select="possible_values"/>
    </td>
    <td>
      <p class="attribute">
        <xsl:choose>
          <xsl:when test="it:isTrue(optional)"><i18n:text>optional</i18n:text></xsl:when>
          <xsl:otherwise><i18n:text>required</i18n:text></xsl:otherwise>
        </xsl:choose>
      </p>
      <xsl:if test="it:isTrue(multiple)">
        <p class="attribute"><i18n:text>multiple</i18n:text></p>
      </xsl:if>
      <xsl:apply-templates select="short"/>
    </td>
  </tr>
</xsl:template>
```

Listing 3: Template für Parameter Tabelle in XHTML

Listing 4 zeigt ein Template für die Parameter Tabelle in WordML (Auszug)

```
<xsl:template match="Parameter">
  <w:tr>
    <w:tc>
      <w:tcPr>
        <w:tcW w:w="3428" w:type="dxa"/>
        <w:shd w:val="clear" w:color="auto" w:fill="auto"/>
      </w:tcPr>
      <w:p>
        <w:r>
          <w:rPr>
            <w:b/>
            <w:noProof/>
          </w:rPr>
          <w:t><xsl:value-of select="if (parameter_name/item) then parameter_name/item else name"/>
        </w:r>
      </w:p>
      <xsl:apply-templates select="possible_values"/>
    </w:tc>
    <w:tc>
      <w:tcPr>
        <w:tcW w:w="5428" w:type="dxa"/>
        <w:shd w:val="clear" w:color="auto" w:fill="auto"/>
      </w:tcPr>
      <w:p>
        <w:pPr>
          <w:jc w:val="right"/>
        </w:pPr>
        <w:r>
          <w:rPr>
            <w:i/>
          </w:rPr>
          <xsl:choose>
            <xsl:when test="it:isTrue(optional)">
              <w:t><i18n:text>optional</i18n:text></w:t>
            </xsl:when>
            <xsl:otherwise>
              <w:t><i18n:text>required</i18n:text></w:t>
            </xsl:otherwise>
```

Listing 4: Template für die Parameter Tabelle in WordML (Auszug)

Beiden Stylesheets ist gemeinsam, dass das Ergebnis dieser Transformation wiederum in einer Variable abgelegt wird. Die nächste Stufe der Generierung erfolgt durch Transformation dieser Variable *$report-xhtml oder $report_word*.

5.8.7 Internationalisierung

Zwar liegen die Inhalte aus dem Infotool in der korrekten Sprachvariante vor, es kann aber immer noch Texte geben, die an die Sprache des Dokuments angepasst werden müssen, zum Beispiel die Beschriftung der Parameter-Tabelle oder Parameter-Eigenschaften wie *optional* oder *required*.

Diese Texte werden im generierten XML mit Elementen oder Attributen im Namespace *„i18“* ausgezeichnet. *„i18n“* ist ein allgemein üblicher Bezeichner für den Begriff *„internationalization“*

```
<table class="parameter">
  <caption><i18n:text>Parameter</i18n:text></caption>
  <xsl:apply-templates/>
</table>
```

Zusätzlich muss eine mehrsprachige Tabelle bereitgestellt werden, die sämtlichen verwendeten Text in allen Sprachversionen enthält, Listing 5 zeigt eine Tabelle für die Generierung der Infotool Schnittstellen Dokumentation:

```
<!--
  - do localisation
-->

<xsl:variable name="TRANSLATIONS" as="element(translation)*">
  <translation lang="en">
    <text id="Infotool Interface Reference">Infotool Interface Reference</text>
    <text id="Output XML Schema">Output XML Schema</text>
    <text id="operation">Operation</text>
    <text id="parameter">Parameter</text>
    <text id="no parameters">The operation needs no parameters.</text>
    <text id="optional">optional</text>
    <text id="required">required</text>
    <text id="multiple">multi-valued</text>
    <text id="possible value">possible value</text>
    <text id="possible values">possible values</text>
  </translation>
  <translation lang="de">
    <text id="Infotool Interface Reference">Infotool Schnittstellen Referenz</text>
    <text id="Output XML Schema">XML Schema des Ergebnisses</text>
    <text id="operation">Operation</text>
    <text id="parameter">Parameter</text>
    <text id="no parameters">Die Operation benötigt keine Parameter.</text>
    <text id="optional">optional</text>
    <text id="required">erforderlich</text>
    <text id="multiple">kann mehrmals auftreten</text>
    <text id="possible value">erlaubter Wert</text>
    <text id="possible values">erlaubte Werte</text>
  </translation>
</xsl:variable>
```

Listing 5: Tabelle für Internationalisierung der Infotool Schnittstellen Dokumentation in XSLT

Vorgefertigte Templates, wiederum im gemeinsamen Stylesheet *report_common.xsl*, ersetzen abschließend sämtliche Texte durch die korrekte Sprachversion (siehe nachfolgendes Listing 6):

```
<!-- translate text for <element i18n:key="..key.."/> -->
<xsl:template match="*[@i18n:key]" mode="i18n">
  <xsl:copy>
    <xsl:apply-templates select="@*" mode="i18n"/>
    <xsl:sequence select="i18n:localize(@i18n:key)"/>
  </xsl:copy>
</xsl:template>

<!-- translate text within <i18n:text>..key..</i18n:text> -->
<xsl:template match="i18n:text" mode="i18n">
  <xsl:sequence select="i18n:localize(.)"/>
</xsl:template>
```

Listing 6: Ersetzen sämtlicher Texte durch die korrekte Sprachversion in XSLT

Dies ist die letzte Stufe der Dokumenten-Generierung, das Ergebnis wird jetzt als Datei abgelegt.

5.8.8 Beispiele

Abbildung 38 zeigt die generierte Schnittstellen Referenz im WordML Format. Unter Anhang A ist die gesamte Schnittstellen Dokumentation abrufbar. Abbildung 39 zeigt dies in XHTML Format und abschließend zeigt Abbildung 40 dieselbe Schnittstellen Referenz im PDF Format.

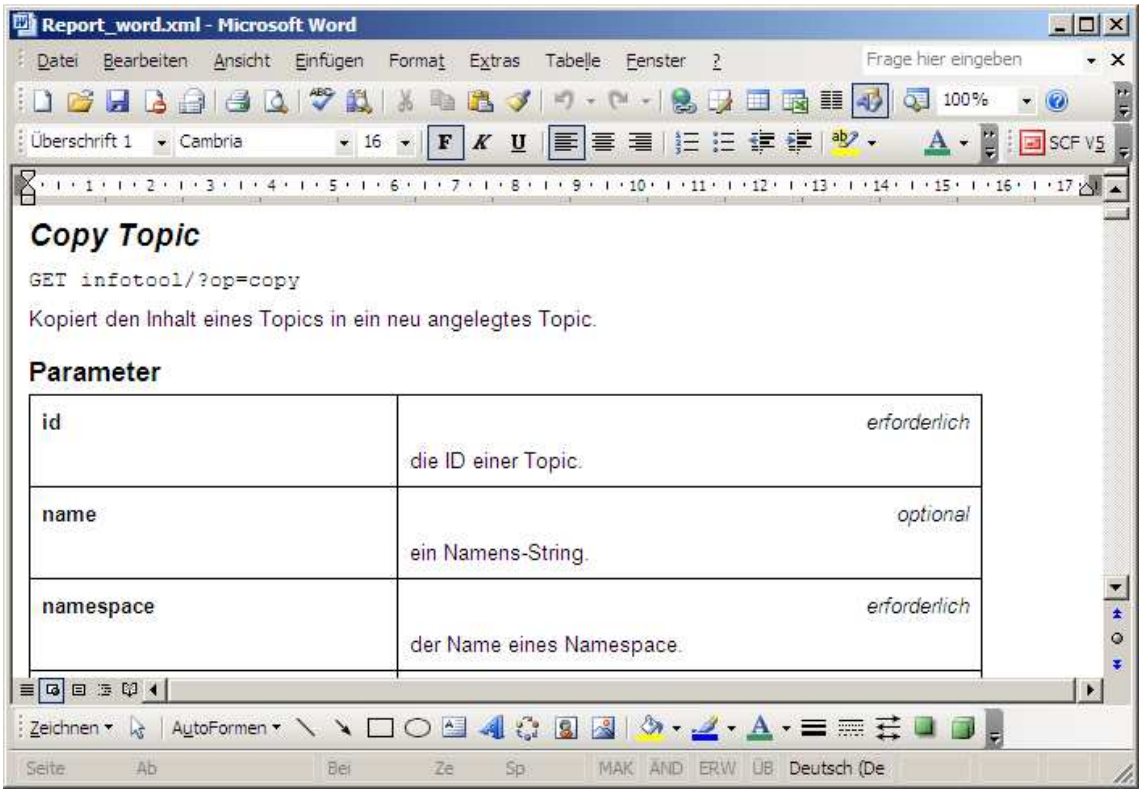


Abbildung 38: Schnittstellen Referenz im WordML Format

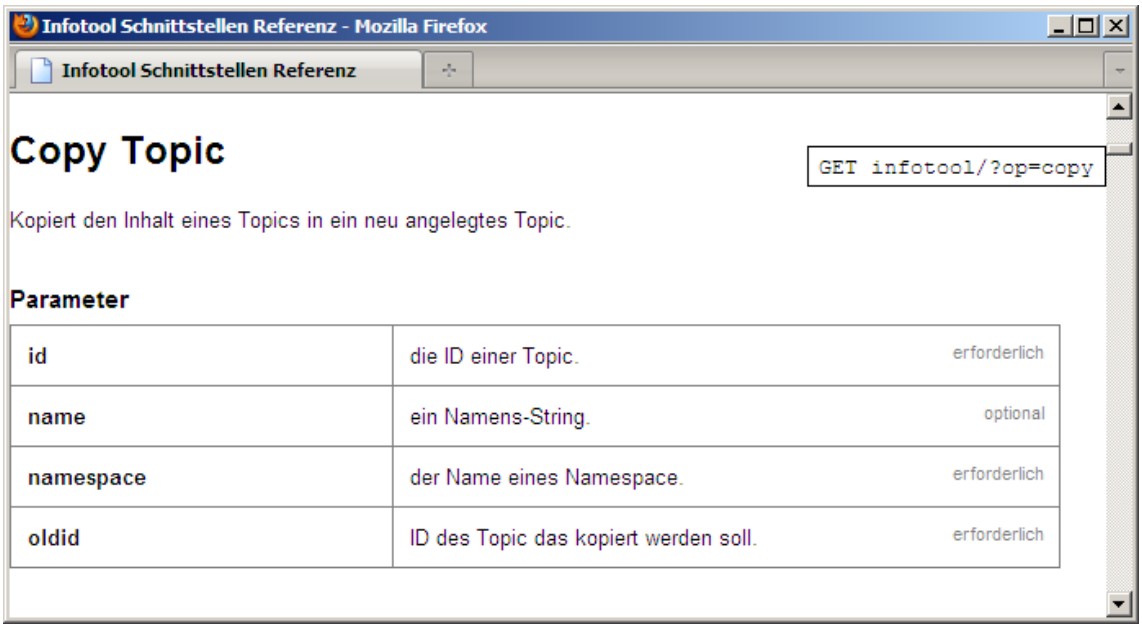


Abbildung 39: Schnittstellen Referenz im XHTML Format

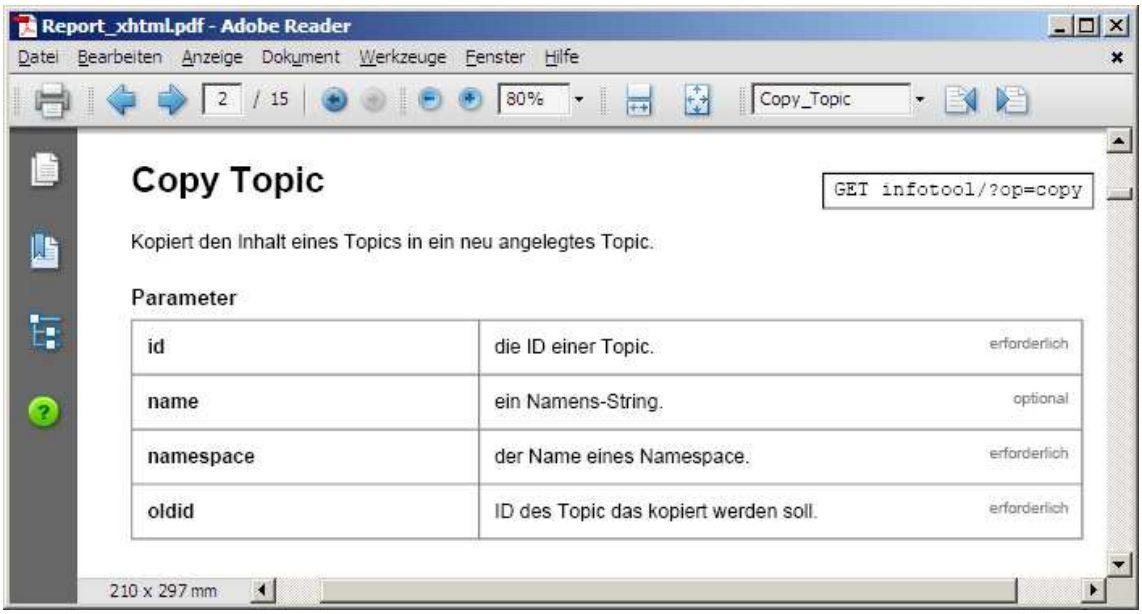


Abbildung 40: Schnittstellen Referenz im PDF Format

6 Zusammenfassung

Die in dieser Arbeit festgelegten Anforderungen an ein Wissensmanagement-Tool konnten allesamt erfüllt werden. Als Basis diente die einführende Literaturrecherche, sowie Analyse der Ausgangssituation. Es wurde gezeigt, dass eine eingehende Analyse der Problembereiche eines Projektumfelds unterstützend für die Identifizierung von Anforderungen und spätere Entwicklung eines geeigneten Tools wirkt. Gerade was das Managen von Wissen in Entwicklungsprojekten betrifft, ist es immens wichtig, die unterschiedlichen Informationsquellen im Projekt zu identifizieren, um diese Informationen anschließend in einheitliche Strukturen zusammenfassen und transformieren zu können.

Auf Basis der in Kapitel 2 und 3 gewonnenen Erkenntnisse, konnten ausführliche Anforderungen spezifiziert werden. Einen wichtigen Teil stellten die Anwendungsfälle dar, die allgemein, aber noch nicht zu technisch detailliert Aufschluss über die benötigten Funktionen eines Wissensmanagement-Tools geben.

Die wohl größte Herausforderung besteht darin, ein geeignetes Datenmodell zu spezifizieren. Die Kernidee war, Daten in einer semantischen Struktur so abzulegen, dass diese in beliebige andere Formate transformiert werden können. Diese Anforderung wurde erfüllt, es besteht hier noch einiges Potential für Erweiterungen, die auch kurz im nachfolgenden Ausblick behandelt werden.

Bei der Generierung von Reports bzw. Dokumenten zeigte sich, dass mit Hilfe von XSLT Transformationen die Daten, die in einer geeignete Struktur im XML Format aus dem Infotool vorliegen, eine flexible Möglichkeit besteht, Dokumente in üblichen Formaten wie Microsoft Word oder PDF automatisiert zu erstellen.

Im folgenden Unterkapitel wird die Abdeckung der Teilbereiche eines Wissensmanagementsystems nach [Lehner, 2009] noch mal analysiert. Es zeigte sich, dass sich jeder Teilbereich in einer gewissen Form im entwickelten Infotool wiederfindet.

Abschließend folgt im letzten Kapitel noch ein kurzer Ausblick, welche Erweiterungen bzw. Weiterentwicklungen zu empfehlen wären.

6.1 Abdeckung der Teilbereiche eines Wissensmanagementsystems

Abschließend werden die in Kapitel 2.2 genannten grundlegenden Teilbereiche eines Wissensmanagementsystems nach [Lehner, 2009] noch mal angeführt und die im Infotool abgedeckten Bereiche, bzw. die Umsetzung im Infotool beschrieben.

6.1.1 Kommunikation

Funktionen der synchronen und asynchronen Kommunikation, zum Beispiel E-Mail, Workflowmanagement oder Newsgroups. [Lehner, 2009]

Dieser Teilbereich wurde im Infotool nicht speziell abgedeckt, da betreffend Kommunikation andere Tools im Projekt bzw. gesamten Unternehmen zum Einsatz kommen. Allerdings bietet

das Infotool Möglichkeiten an, eine Kommunikationsplattform aufzubauen. Man könnte beispielsweise eigene „Nachrichtenobjekte“ definieren, die dann andere Personen einsehen können, oder auch nur für bestimmte Personenkreise einsehbar sind.

6.1.2 Inhaltsmanagement

Funktionen zum Umgang mit Inhalten, zum Beispiel: Bildern, Dokumenten und Lernobjekten. [Lehner, 2009]

Dieser Teilbereich wurde umfassend umgesetzt. Wissen im Projekt, Dokumente, Bilder etc. können einfach eingefügt, integriert oder verwaltet werden. Das Infotool bietet weiters unterstützende Funktionen wie Sortierungen oder hierarchische Einordnungen. Inhalte können einfach importiert oder exportiert werden.

6.1.3 Entscheidungsunterstützung

Funktionen zur Auswertung und Zusammenfassung von Informationen. [Lehner, 2009]

Durch die vom Infotool unterstützte hierarchische Anordnung von Informationseinheiten, lassen sich Informationen gut thematisch zusammenfassen. Durch die Implementierung spezieller Reports mit Hilfe von XSLT Transformationen lassen sich entsprechende Auswertungen der im Infotool gespeicherten Informationen automatisch generieren.

6.1.4 Suche

Funktionen zum Finden von Inhalten und Experten in unterschiedlichen Quellen, zum Beispiel: Datenbanken, Dateiservern, Dokumentmanagementsystemen. [Lehner, 2009]

Es wurde eine umfassende Suchfunktion implementiert. Mit dieser lassen sich Informationseinheiten einfach auffinden. Auch verknüpfte Informationen lassen sich einfach suchen. Beispielsweise welche Mitarbeiter in welchen Projekten arbeiten oder welche Dokumente wo referenziert werden. Es lassen sich entsprechende Filter anwenden, indem man die Suche auf bestimmte Personen einschränkt, oder auch zeitlich einschränkt.

6.1.5 Visualisierung

Funktionen zur Präsentation von Informations- und Wissenselementen sowie der Struktur dieser. [Lehner, 2009]

Ein ansprechender Client wurde entwickelt. Dieser präsentiert die gespeicherten Informationen. Hier besteht noch viel Erweiterungspotential, da die Schnittstellen so konzipiert wurden, dass viele weitere, teils sehr spezielle Clients entwickelt werden können.

6.2 Ausblick

Wie in dieser Arbeit dargestellt, deckt das Infotool bereits viele Anforderungen ab und kann für spezielle Aufgabenstellungen eingesetzt werden. Es existieren aber einige Erweiterungsmöglichkeiten, die sich grob in zwei Bereiche gliedern lassen. Einerseits für das Infotool selbst, andererseits für die gespeicherten Daten.

Für das Infotool wäre eine Betrachtung folgender Themen zu empfehlen:

- Performance Steigerung durch beispielsweise Cache Mechanismen, Optimierung der Datenbankschnittstellen etc.
- Offline-Variante für lokales Arbeiten. Zum Beispiel wäre der Einsatz von SQLite interessant. Die Idee ist, dass ein Entwickler lokal eine Infotool Anwendung laufen hat und später die Daten mit dem projektinternen Infotool Server synchronisiert.
- Synchronisation und Merge von Daten Beständen. Unterstützt man wie vorhergehend genannt eine Offline-Variante, muss man sich Gedanken über Synchronisation und Merge von Daten Beständen machen. Es wäre zu untersuchen in welcher Form man diese implementiert.
- Spezialisierte Clients. Der Infotool Server ist durch seine standardisierten Schnittstellen so ausgelegt, dass er unterschiedlichste und spezialisierte Clients unterstützen kann. Es wäre unter anderem möglich, einen „Textverarbeitungsclient“ zu implementieren, der die Erstellung und Bearbeitung von hierarchischen Dokumenten ermöglicht.

Für die im Infotool gespeicherten Daten, wäre es empfehlenswert, folgende Themenbereiche weiter zu untersuchen:

- Automatisierte Generierung von RDF mittels Stylesheet (siehe nachfolgendes Kapitel)
- SPARQL Queries für Reports oder Konsistenzprüfung (siehe nachfolgendes Kapitel)
- Adaption der unterstützten Datentypen in Richtung Datentypen aus XML Schema
- Abdeckung der ersten Stufe(n) des Semantic Web Stacks, siehe Abbildung 41 [W3.org Semantic Web, 2011]

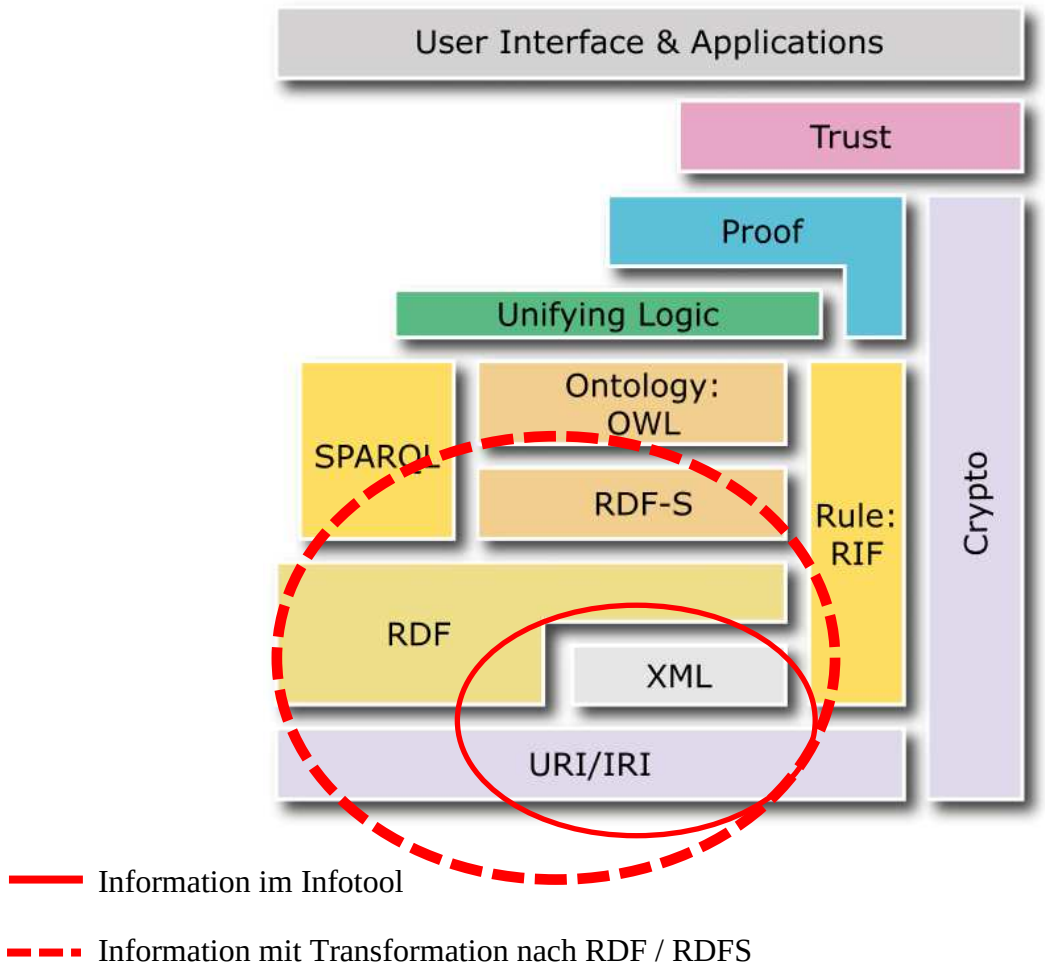


Abbildung 41: Semantic Web Stack [W3.org Semantic Web, 2011]

6.3 Verwendung von Infotool Daten in einem Semantic Web Framework

Das Datenmodell zur Ablage von Information im Infotool wurde nach dem Vorbild von RDF konzipiert. Eine Motivation für diese Entscheidung war die sich damit ergebende Möglichkeit das vom Infotool gesammelte Wissen einfach nach RDF umzuwandeln. Danach sollte der Verarbeitung der Information mit allen Werkzeugen und Tools modernen Wissensmanagements nichts mehr im Wege stehen.

6.3.1 Transformation der Information im Infotool nach RDF

Da in einer Wissensdatenbank sowohl die einzelnen Objekte als auch alle Beziehungen zwischen ihnen von Bedeutung sind, soll als Ausgangspunkt für die Transformation nach RDF ein möglichst großer Bereich der Information im Infotool erfasst werden. Hier bietet sich der exportierte Inhalt eines Namespace an, welcher mittels Operation *Export* einfach als XML Datei generiert werden kann.

Die Transformation der XML Datei nach RDF erfolgt wieder mittels XSLT Stylesheet und soll im Folgenden anhand einiger Templates skizziert werden.

Als Container für den Inhalt einer RDF/XML Datei dient das Element <*rdf:RDF*> und wird für das Element <*topiclist*> der Export Datei generiert:

```
<!--
   -  generate a RDF container element for the topiclist root
   -->
<xsl:template match="topiclist">
  <rdf:RDF>
    <xsl:apply-templates/>
  </rdf:RDF>
</xsl:template>
```

Stellvertretend für die drei möglichen Typen von Topics (*class*, *property* oder *resource*) sei die Transformation eines Topics vom Typ *class* gezeigt. Man kann in RDF/XML den Typ (also die Klassenzugehörigkeit) einer Ressource bereits im Elementnamen (hier <*rdfs:Class*>) zum Ausdruck bringen. Die vorhandenen Subklassenbeziehung lassen sich mittels Property <*rdfs:subClassOf*> angeben:

```
<!--
  -  a class
-->
<xsl:template match="topic[@type eq 'class']">
  <rdfs:Class>
    <xsl:apply-templates select="@*|*"/>
  </rdfs:Class>
</xsl:template>

<!-- super classes of the class -->
<xsl:template match="superclass/topicref">
  <rdfs:subClassOf rdf:resource="{@ref}"/>
</xsl:template>
```

Allen Typen von Topics gemeinsam ist die Generierung der eindeutigen URI. Diese wird einfach aus dem Topic Attribute *id* übernommen und im RDF/XML Attribute *rdf:about* abgelegt:

```
<!-- the identifier of a topic  -->
<xsl:template match="@id">
  <xsl:attribute name="rdf:about"><xsl:sequence
select="."/></xsl:attribute>
</xsl:template>
```

Einfache Metadaten aus dem Dublin Core werden aus den Attributen des Topic Elements abgeleitet:

```
<!-- the names of a topic, convert them to title attributes -->
<xsl:template match="topicinfos/name/item">
  <dc:title><xsl:call-template name="itemLanguageType"/></dc:title>
</xsl:template>
```

```
  <!-- the short descriptions of a topic, convert them to description
attributes -->
  <xsl:template match="topicinfos/short/item">
    <dc:description><xsl:call-template
name="itemLanguageType"/></dc:description>
  </xsl:template>

  <!-- the tags of a topic, convert them to subject attributes -->
  <xsl:template match="topicinfos/tag/item">
    <dc:subject><xsl:call-template name="itemLanguageType"/></dc:subject>
  </xsl:template>
```

Zusätzlich werden noch Metadaten aus dem Standard DCMI Metadata Terms [Dublin Core Metadata Terms, 2011] hinzugezogen. Beispielsweise haben die Topic Attribute *created* und *modified* ihre Entsprechung in den gleichnamigen Metadata Terms:

```
  <!--
     -  created: Date of creation of the resource.
     -  modified: Date on which the resource was changed.
     -->
  <xsl:template match="@created|@modified">
    <xsl:attribute name="dcterms:{local-name()}"><xsl:value-of
select="."/></xsl:attribute>
  </xsl:template>
```

Etwas mehr Überlegung muss in das Transformieren der eigentlichen Daten von Topics investiert werden. In RDF/XML muss das Property welches ein Datum beschreibt als Namespace-qualifiziertes Element vorliegen, beispielsweise muss der Wert *GET* für Property *urn:info:Infotool:method* als Element

```
<method xmlns="urn:info:Infotool:">GET</method>
```

geschrieben werden. Dies erledigt folgendes Template:

```
  <!-- datatype properties -->
  <xsl:template match="class/prop[@type eq 'data']">
    <xsl:variable name="qname" select="it:urn2qname(@ref)"/>
    <xsl:variable name="withLanguage"
select="xs:boolean(@language)"/>
    <xsl:for-each select="item">
      <xsl:element name="{$qname[2]}" namespace="{$qname[1]}">
        <xsl:choose>
          <xsl:when test="$withLanguage">
            <xsl:call-template name="itemLanguageType"/>
          </xsl:when>
          <xsl:otherwise><xsl:value-of select="."/></xsl:otherwise>
        </xsl:choose>
      </xsl:element>
```

```
    </xsl:for-each>
  </xsl:template>
```

Zusätzlich erfahren hier sprachspezifische Daten eine Sonderbehandlung, es muss hier die tatsächlich verwendete Sprache im Attribute *xml:lang* eingetragen werden.

Wenn man die Transformation auf die Export Datei eines Namespace im Infotool anwendet erhält man eine vollständige Darstellung der Information des Namespace im Format RDF/XML. Zum Beispiel ist die RDF Darstellung des Topics der WebOperation *Copy_Topic* aus der Schnittstellen Referenz in Abbildung 42 gezeigt.

```
<rdf:Description
     rdf:about="urn:info:Infotool:Copy_Topic"
     dcterms:created="2010-11-10T15:09:20+01:00"
     dcterms:modified="2011-01-20T15:08:09+01:00"
     dc:creator="christoph.scherr@siemens.com"
     dc:contributor="admin"
     dc:isPartOf="Infotool">
   <rdf:type rdf:resource="urn:info:Infotool:WebOperation"/>
   <dc:title xml:lang="en">Copy Topic</dc:title>
   <dc:description xml:lang="de">Kopiert den Inhalt eines Topics in ein neu angelegtes Topic.</dc:description>
   <dc:description xml:lang="en">Copies the contents of an existing topic to a new topic.</dc:description>
   <operation xmlns="urn:info:Infotool:">copy</operation>
   <pathname xmlns="urn:info:Infotool:">/</pathname>
   <parameters xmlns="urn:info:Infotool:" rdf:resource="urn:info:Infotool:id"/>
   <parameters xmlns="urn:info:Infotool:" rdf:resource="urn:info:Infotool:name"/>
   <parameters xmlns="urn:info:Infotool:" rdf:resource="urn:info:Infotool:namespace"/>
   <parameters xmlns="urn:info:Infotool:" rdf:resource="urn:info:Infotool:oldid"/>
   <method xmlns="urn:info:Infotool:">GET</method>
</rdf:Description>
```

Abbildung 42: RDF Darstellung eines Topics

6.3.2 Tool-unterstützte Abfrage der RDF Daten

Die Auswertung des vom Infotool gesammelten Wissens ist zwar Gegenstand von zukünftigen Arbeiten. Es soll aber bereits hier anhand eines einfachen Beispiels geprüft werden, ob die Information für die Weiterverarbeitung mit Standard Tools des modernen Wissensmanagements geeignet ist.

Als Tool wurde das Open Source JAVA Framework Sesame der Firma ADUNA [OpenRDF.org, 2011] gewählt. Die Funktionalität von Sesame umfasst das Speichern, die Abfrage und das Schlussfolgern über Daten in den Formaten RDF und RDFS. Zusätzlich existiert eine Browser Schnittstelle namens *Sesame Workbench* für den Import und die Abfrage von Daten, die von einer Web-Applikation im Tomcat Servlet Container implementiert ist, siehe Abbildung 43.

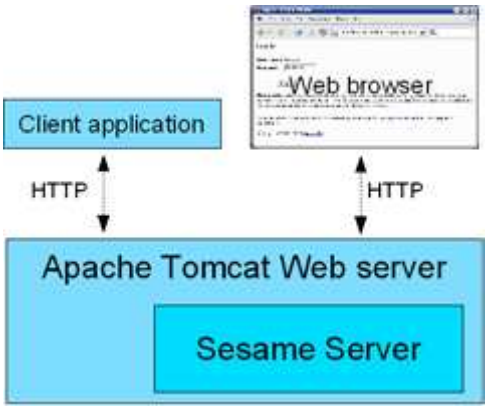


Abbildung 43: Sesame Server [OpenRDF.org Sesame Server, 2011]

Es soll nun eine einfache Abfrage über der Wissensbasis der Infotool Schnittstellen Referenz (Namespace Infotool) erfolgen, dazu sind folgende Schritte nötig:

1. Export des Namespace System aus dem Infotool und Transformation der exportierten XML Datei in die RDF/XML Datei *System.rdf*
2. Export des Namespace Infotool aus dem Infotool und Transformation der exportierten XML Datei in die RDF/XML Datei *Infotool.rdf*
3. Importieren von *System.rdf* und *Infotool.rdf* in die Sesame Workbench mittels File-Upload
4. Abfrage der Daten in der Sesame Workbench mittels der SPARQL Query Language for RDF [W3.org SPARQL, 2011]

Als Beispiel für eine Abfrage (Query) soll eine Liste von WebOperations erstellt werden, die alle den Parameter *namespace* verwenden. Das Query ist sehr einfach formuliert:

```
PREFIX info:<urn:info:Infotool:>

# query all WebOperations that have a parameter "namespace"

SELECT ?wo WHERE {
  ?wo info:parameters ?par .
  ?par info:parameter_name "namespace" .
}
```

Es wird nach Tripeln im RDF gesucht die folgendem Muster entsprechen:

1. <*WebOperation*> <info:parameters> <*Parameter*>
2. <*Parameter*> <info:parameter_name> “namespace”

Die *kursiv* dargestellten Bezeichner sind veränderlich, müssen für ein gültiges Ergebnis der Suche aber an allen Stellen an denen sie vorkommen identisch sein.

Beispiele für entsprechende Daten in der Datei *infotool.rdf* wären:

```
<rdf:Description rdf:about="urn:info:Infotool:List_Topics" …
  <rdf:type rdf:resource="urn:info:Infotool:WebOperation"/>
  <parameters xmlns="urn:info:Infotool:"
rdf:resource="urn:info:Infotool:namespace"/>
```

und

```
<rdf:Description rdf:about="urn:info:Infotool:namespace" …
  <rdf:type rdf:resource="urn:info:Infotool:Parameter"/>
  <parameter_name
xmlns="urn:info:Infotool:">namespace</parameter_name>
```

Am Sesame Server werden wie Abbildung 44 zeigt als Ergebnis der Abfrage die gefundenen 13 *WebOperations* gelistet:

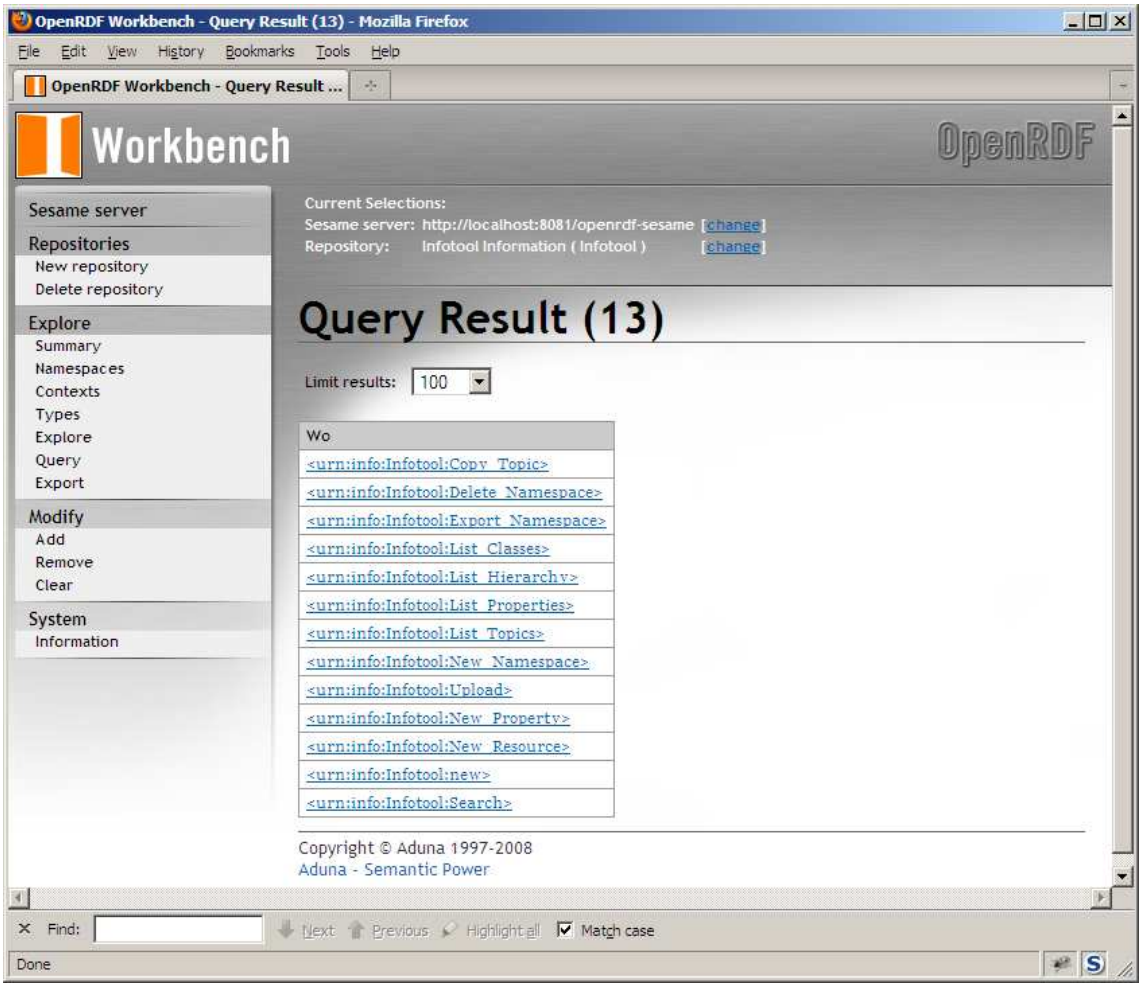


Abbildung 44: Ergebnis einer SPARQL Abfrage

Ein genauerer Blick auf den Quelltext der Ergebnis-Seite zeigt, dass die Abfrageergebnisse im Format SPARQL Query Results XML Format [W3.org SPARQL XML Results, 2011] vom Server geliefert und am Client mittels XSLT stylesheet in die HTML Darstellung umgewandelt werden. Bei nicht-interaktiver Abfrage durch Clients könnte das XML Format weiterverarbeitet werden. Abbildung 45 zeigt ein SPARQL Result im XML Format.

```
<?xml-stylesheet type='text/xsl' href='/openrdf-workbench/transformations/tuple.xsl'?>
<sparql xmlns='http://www.w3.org/2005/sparql-results#'
        xmlns:q='http://www.openrdf.org/schema/qname#'>
  <head>
    <variable name='wo'/>
    <link href='info'/>
  </head>
  <results ordered='false' distinct='false'>
    <result>
      <binding name='wo'>
        <uri>urn:info:Infotool:Copy_Topic</uri>
      </binding>
    </result>
    <result>
      <binding name='wo'>
        <uri>urn:info:Infotool:Delete_Namespace</uri>
      </binding>
    </result>
    <result>
      <binding name='wo'>
        <uri>urn:info:Infotool:Export_Namespace</uri>
      </binding>
    </result>
    <result>
      <binding name='wo'>
        <uri>urn:info:Infotool:List_Classes</uri>
      </binding>
    </result>
```

Abbildung 45: SPARQL Result (XML Format)

6.3.3 Mehr als RDF und RDFS: automatische Schlussfolgerungen (Inferenz)

Sesame unterstützt nur die Abfrage von RDF Daten und die Inferenz über Klassenhierarchien von RDFS. Für viele weiterführende Zwecke ist das nicht ausreichend, da diese Information auch einfach über XPath Queries auf dem bestehenden Export Datenformat des Infotools erhalten werden können.

Eine mögliche Erweiterung wäre die Anreicherung der Daten durch OWL Sprachkonstrukte

owl:inverseOf: das Property `<arbeitet für>` ist invers zum Property `<ist Arbeitgeber von>`

owl:inverseFunctionalProperty: die Inverse des Property `<email_adresse>` hat für jede Ressource nur genau einen Wert, weil eine E-Mail Adresse üblicherweise genau einer Person zugeordnet ist.

owl:SymmetricProperty: Beim Property `<verheiratet>` gilt die Beziehung immer, auch wenn Subjekt und Objekt vertauscht werden

owl:TransitiveProperty: aus

```
<A> <ist vorgesetzter von> <B>
```

und

```
<B> <ist vorgesetzter von> <C>
```

lässt sich schlussfolgern

```
<A> <ist vorgesetzter von> <C>
```

7 Abbildungsverzeichnis

8 Tabellenverzeichnis

9 Listings

10 Anhang A: Infotool Schnittstellen Referenz

Wie in Kapitel 5.8 zur Dokumentgenerierung erwähnt – wurde die komplette Schnittstellen Referenz dieses Projekts ins Infotool integriert. Mit Hilfe der zur Verfügung gestellten XSLT Transformation, wurde ein Schnittstellen Referenz Dokument erstellt, welches hier automatisch in diese Arbeit eingebunden wurde. Nachfolgend findet sich die automatisch generierte Schnittstellen Referenz.

10.1 Formate

Anzumerken ist, dass die hier vorgestellten Schnittstellen zu großen Teilen XML Format liefern. Es wurde für Clients die auf JavaScript basieren auch JSON Format unterstützt. Für den Großteil der hier verwendeten Schnittstellen ist es möglich, einfach den zusätzlichen Parameter *format=json* anzuhängen, um beispielsweise ein Topic in einer JSON Repräsentation zu erhalten. Die Angabe von *format=xml* ist nicht explizit nötig, da die Schnittstellen per Default XML liefern.

10.2 Content

`GET infotool/content`

Liefert Dateien aus dem Server Repository an den Client aus.
Alle Dateien sind über ihren Hash-Wert identifizierbar, somit sind sie einfach über die URL "/content/[Hash-Wert]" adressierbar.

Die Operation benötigt keine Parameter.

10.3 Copy Topic

`GET infotool/?op=copy`

Kopiert den Inhalt eines Topics in ein neu angelegtes Topic.

Parameter

id	*erforderlich* die ID eines Topics.
name	*optional* ein Namens-String.
namespace	*erforderlich* der Name eines Namespace.
oldid	*erforderlich* ID des Topics das kopiert werden soll.

10.4 Delete Namespace

`GET infotool/?op=delete`

Löscht einen Namespace mitsamt allen enthaltenen Topics.

Parameter

namespace	*erforderlich* der Name eines Namespace.
type erlaubter Wert: `namespace`	*erforderlich* muss auf den Wert "namespace" gesetzt sein.

10.5 Delete Topic

`GET infotool/?op=delete`

Löscht ein einzelnes Topic.

Parameter

id	*erforderlich* die ID eines Topics.

10.6 Download

`GET infotool/download`

Download eines Topics vom Server. Das Topic muss ein File repräsentieren.

Parameter

id	*erforderlich* die ID eines Topics.

10.7 Edit Order

`GET infotool/?op=edit`

Mit dieser Operation kann die Reihenfolge von Properties einer Class oder der Range-Werte einer Resource verändert werden.
Es ist eine JSON Datenstruktur als Eingabe erforderlich.

Parameter

type erlaubte Werte: `class property resource`	*erforderlich* der Typ eines Topics.

10.8 Edit Topic

`GET infotool/?op=edit`

Mit dieser Operation können die Werte einzelner Properties einer Resource verändert werden.
Es ist eine JSON Datenstruktur als Eingabe erforderlich.

Parameter

id	*erforderlich* die ID eines Topics.
property_id	*erforderlich* ID eines Property das editiert werden soll.

10.9 Export Finished

`GET infotool/export`

Prüft ob eine Export Operation beendet wurde.

Parameter

finished	*erforderlich* setze "finished="{Dateiname}" um abzufragen ob diese Export Operation beendet wurde.

10.10 Export Namespace

`GET infotool/export`

Exportiert den gesamten Inhalt eines Namespaces.

Mit dem optionalen "type" Parameter können selektiv nur Class-, Property oder Resource Topics exportiert werden.

Die exportierte Information kann später mittels Operation "Import" wieder in das Infotool importiert werden.

Die unterstützten Datenformate dieser Operation sind:
XML - nur Topics, ohne File Ressourcen
ZIT - gezipptes Archiv mit Topics und den verlinkten File Ressourcen

Parameter

filename	*erforderlich* Dateiname beim Export
namespace	*erforderlich* der Name eines Namespace.
type erlaubte Werte: `class property resource`	*erforderlich* der Typ eines Topics.
with_files erlaubter Wert: `true`	*erforderlich* falls "true" werden die exportierten Topics gemeinsam mit referenzierten Dateien in einem Archiv (mit Endung ".zit") abgelegt.

10.11 Export Topic

`GET infotool/export`

Exportiert ein einzelnes Topic im XML Format.

Parameter

filename	*erforderlich* Dateiname beim Export
id	*erforderlich* die ID eines Topics.
with_files erlaubter Wert: `true`	*erforderlich* falls "true" werden die exportierten Topics gemeinsam mit referenzierten Dateien in einem Archiv (mit Endung ".zit") abgelegt.

10.12 Generate Report

`GET infotool/export`

Generiert ein Dokument oder einen Report.

Die Operation benötigt keine Parameter.

10.13 Get Topic

`GET infotool/?op=get`

Empfängt eine XML oder JSON Repräsentation eines Topic vom Infotool Server.

Wenn ein oder mehr Werte für den optionalen Parameter "property_id" gegeben sind, werden nur die Werte für die entsprechenden Properties geliefert.

Parameter

id	*erforderlich* die ID eines Topics.
property_id	*optional, kann mehrmals auftreten* die ID einer Property Topic.

10.14 Get User Rights

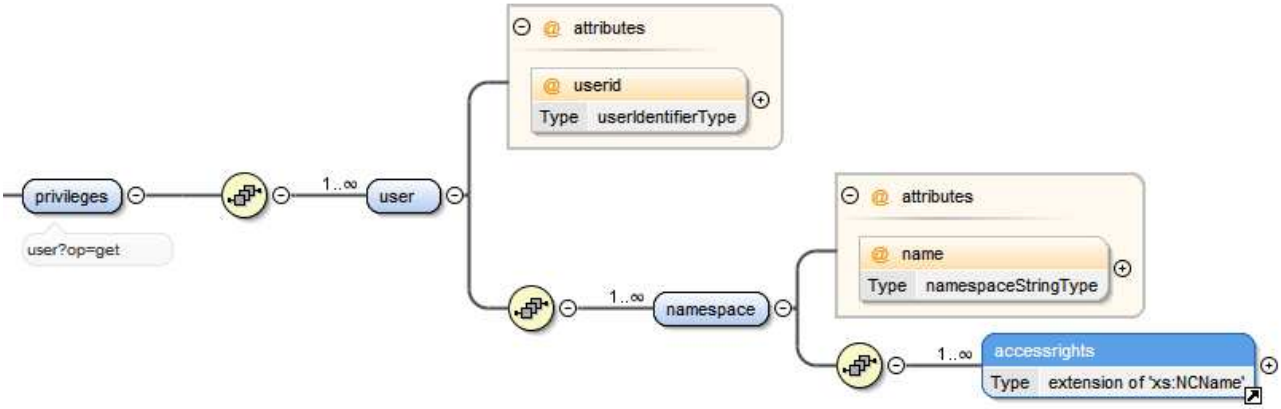
`GET infotool/user?op=get`

Liefert eine Liste mit Zugriffsrechten aller Benutzer für alle Namespaces.

Die Operation benötigt keine Parameter.

XML Schema des Ergebnisses

Die Information ist in ein <infotool> Element eingebettet.



10.15 Import

`GET infotool/import`

Importiert ein Infotool Topic Archiv in einen Namespace.

Die unterstützten Datenformate dieser Operation sind dieselben wie beim Export (siehe Operation "Export Namespace"):
XML - nur Topics, ohne File Ressourcen
ZIT - gezipptes Archiv mit Topics und den verlinkten File Ressourcen

Die Operation benötigt keine Parameter.

10.16 LastCreated / LastModified

`GET infotool/?op=list`

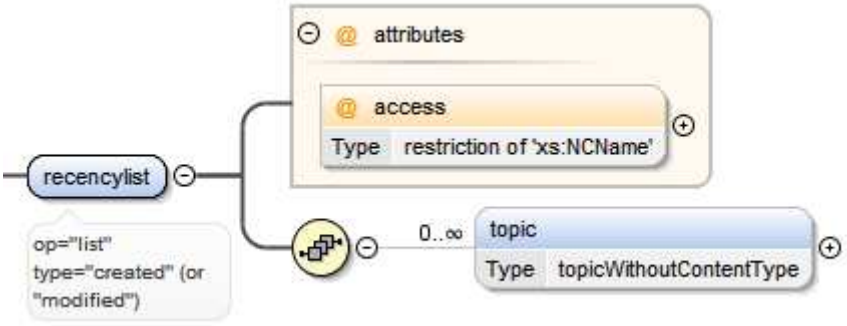
Listet die zuletzt angelegten oder geänderten Topics des aktuell eingeloggten Benutzers.

Parameter

count	*optional* bestimmt wie viele Einträge in der Ergebnisliste maximal sein sollen.
type erlaubte Werte: `created` `modified`	*erforderlich* wählt den Zeitpunkt des Anlegens ("created") oder der letzten Änderung ("modified") für die Ergebnislist aus.

XML Schema des Ergebnisses

Die Information ist in ein <infotool> Element eingebettet.



10.17 List Child Classes

`GET infotool/?op=list`

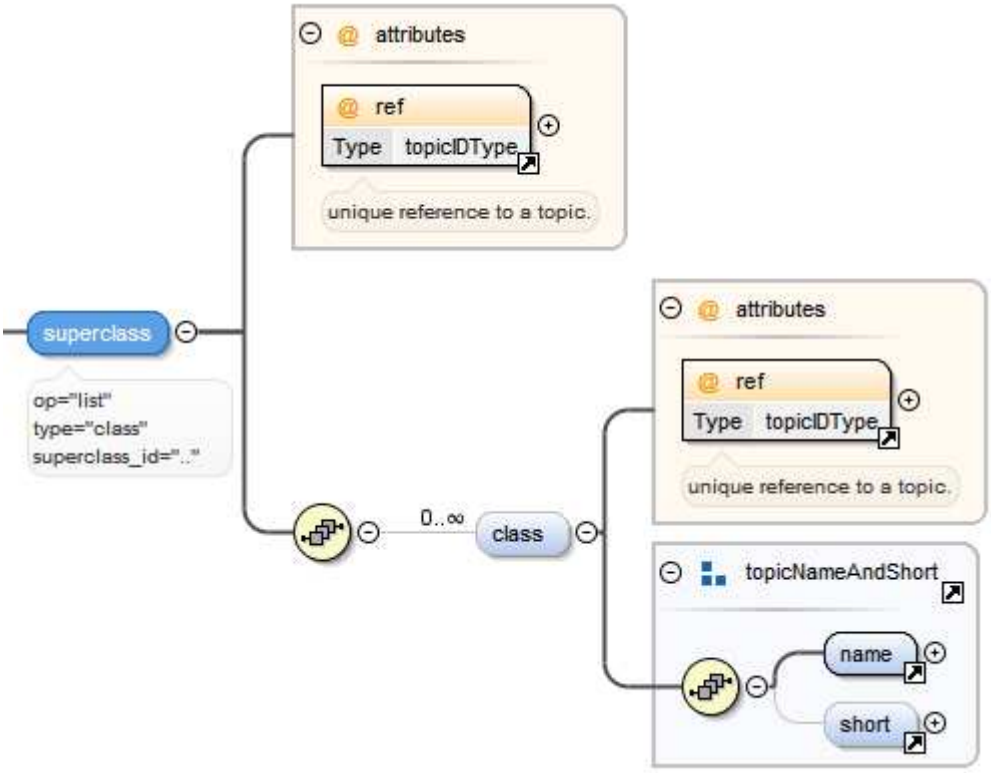
Listet alle von der gegebenen Class abgeleitete Classes.

Parameter

superclass_id	*erforderlich* die ID einer Class Topic.
type erlaubter Wert: `class`	*erforderlich* muss auf den Wert "class" gesetzt sein.

XML Schema des Ergebnisses

Die Information ist in ein <infotool> Element eingebettet.



10.18 List Classes

`GET infotool/?op=list`

Listet alle Class Topics in einem Namespace.

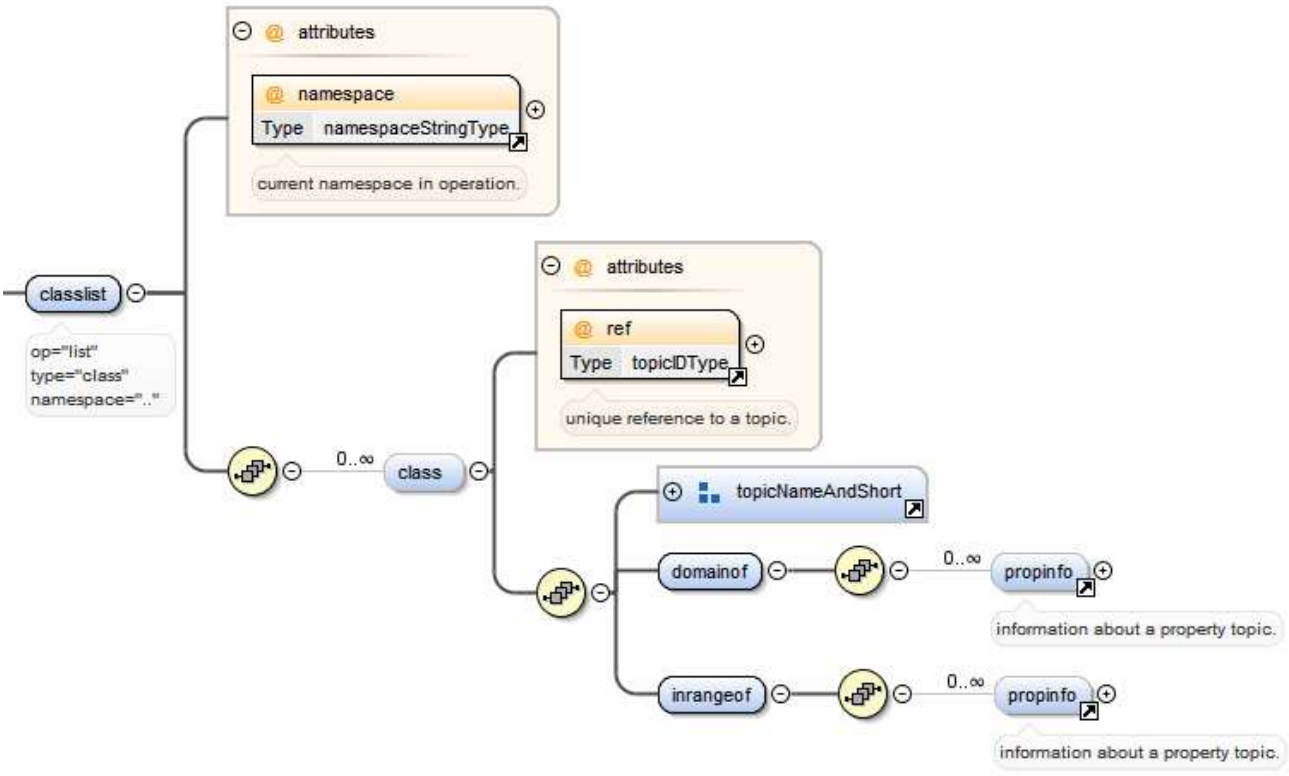
Parameter "namespace=all" listet die Class Topics in allen Namespaces.

Parameter

namespace	*erforderlich* der Name eines Namespace.
type erlaubter Wert: `class`	*erforderlich* muss auf den Wert "class" gesetzt sein.

XML Schema des Ergebnisses

Die Information ist in ein <infotool> Element eingebettet.



10.19 List Hierarchy

`GET infotool/?op=list`

Listet alle Resource Topics in einem Namespace.

Bei Parameter "hierarchy=true" werden nur die Ressourcen in der Wurzel einer Hierarchy berücksichtigt.

Mit den optionalen Parameter "class_id" und "property_id" kann die gewünschte Hierarchy ausgewählt werden, sofern mehrere vorhanden sind.

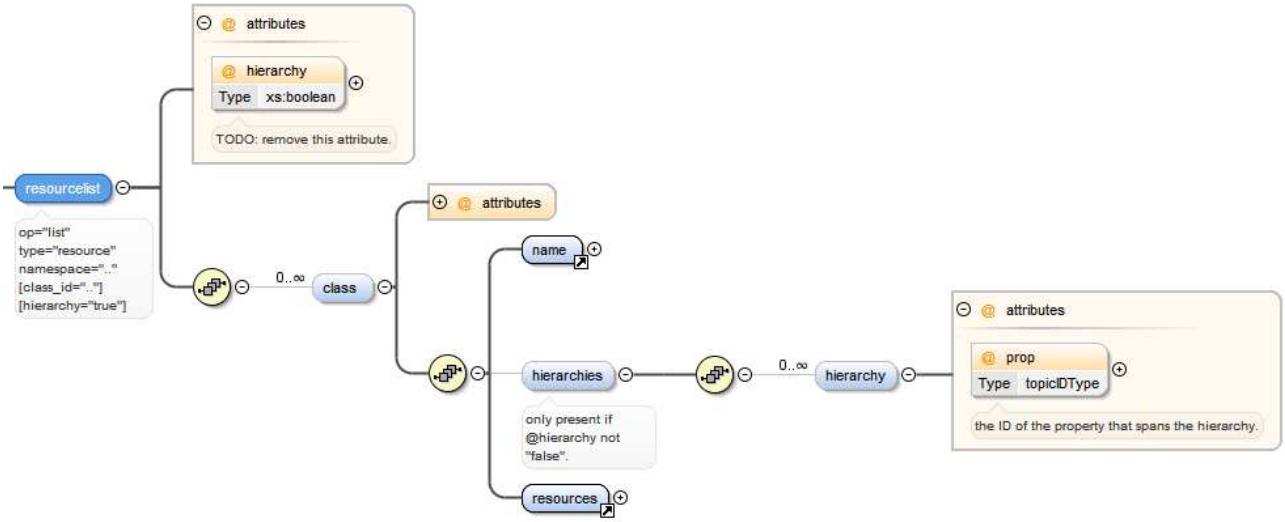
Parameter

class_id	*optional* die ID einer Class Topic.
hierarchy erlaubter Wert: `true`	*erforderlich* verwendet "hierarchy=true" um eine Liste von Child Resource Topics zu erhalten. verwendet "hierarchy={resource_id} um eine Liste von Child Resources von Resource {resource_id} zu erhalten.

namespace	*erforderlich* der Name eines Namespace.
property_id	*optional* wenn angegeben, muss es sich um die ID eines Hierarchy Property handeln.
type erlaubter Wert: `resource`	*erforderlich* muss auf den Wert "resource" gesetzt werden.

XML Schema des Ergebnisses

Die Information ist in ein <infotool> Element eingebettet.



10.20 List Hierarchy Children

`GET infotool/?op=list`

Listet die Child Resource Topics an der durch Parameter "hierarchy" gegebenen Stelle einer Resource Hierarchy.

Der Parameter "property_id" muss angegeben werden, falls mehrere Hierarchien existieren.

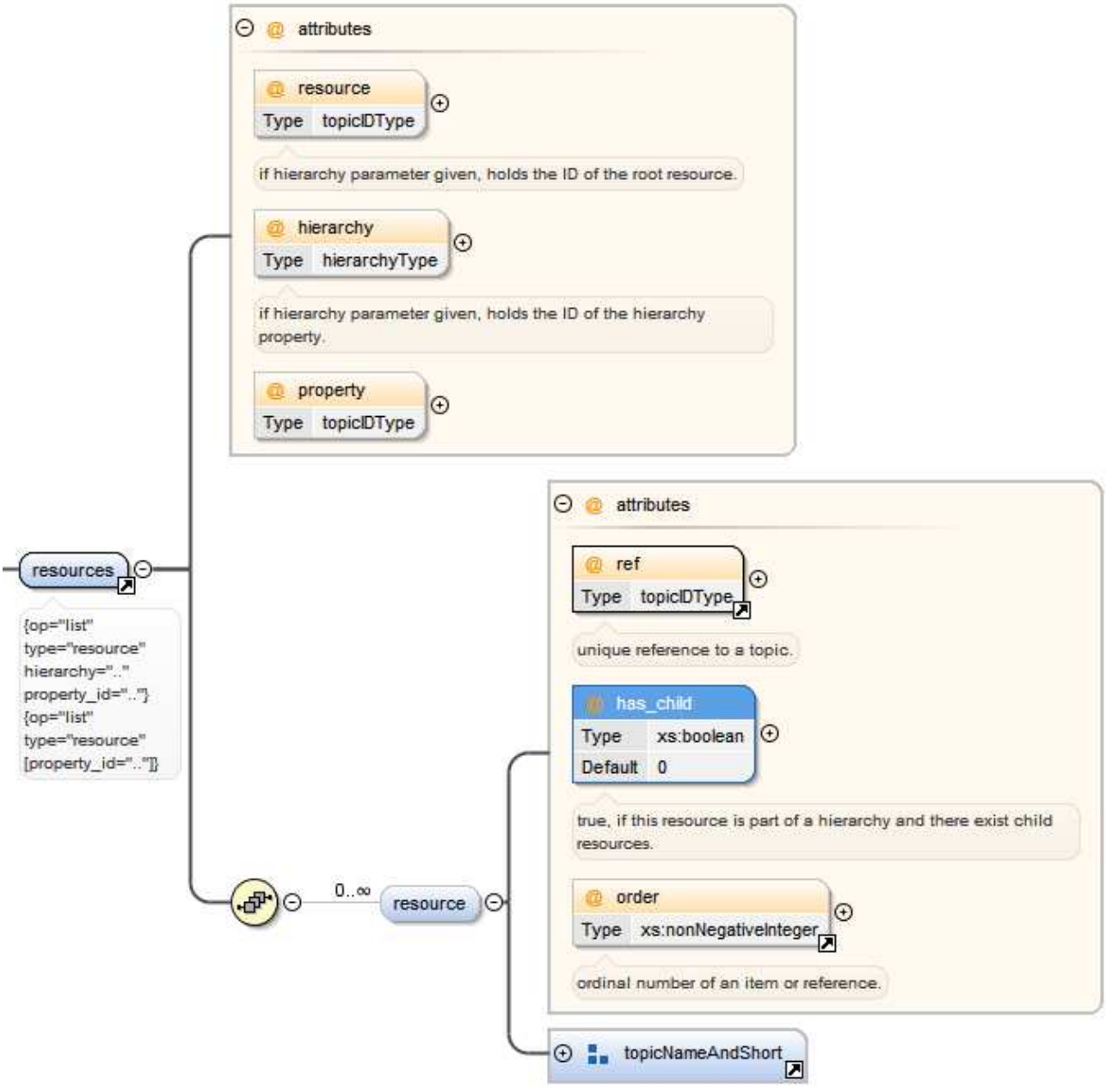
Parameter

hierarchy erlaubter Wert: `true`	*erforderlich* verwendet "hierarchy=true" um eine Liste von Child Resource Topics zu erhalten. verwendet "hierarchy={resource_id} um eine Liste von Child Resources von Resource {resource_id} zu erhalten.

property_id	*optional* wenn angegeben, muss es sich um die ID eines Hierarchy Property handeln.
type erlaubter Wert: `resource`	*erforderlich* muss auf den Wert "resource" gesetzt werden.

XML Schema des Ergebnisses

Die Information ist in ein <infotool> Element eingebettet.



10.21 List Namespaces

`GET infotool/?op=list`

Listet alle Namespaces im Infotool.

Parameter

type erlaubter Wert: `namespace`	*erforderlich* muss auf den Wert "namespace" gesetzt sein.

XML Schema des Ergebnisses

Die Information ist in ein <infotool> Element eingebettet.

10.22 List Properties

`GET infotool/?op=list`

Listet alle Property Topics in einem Namespace.

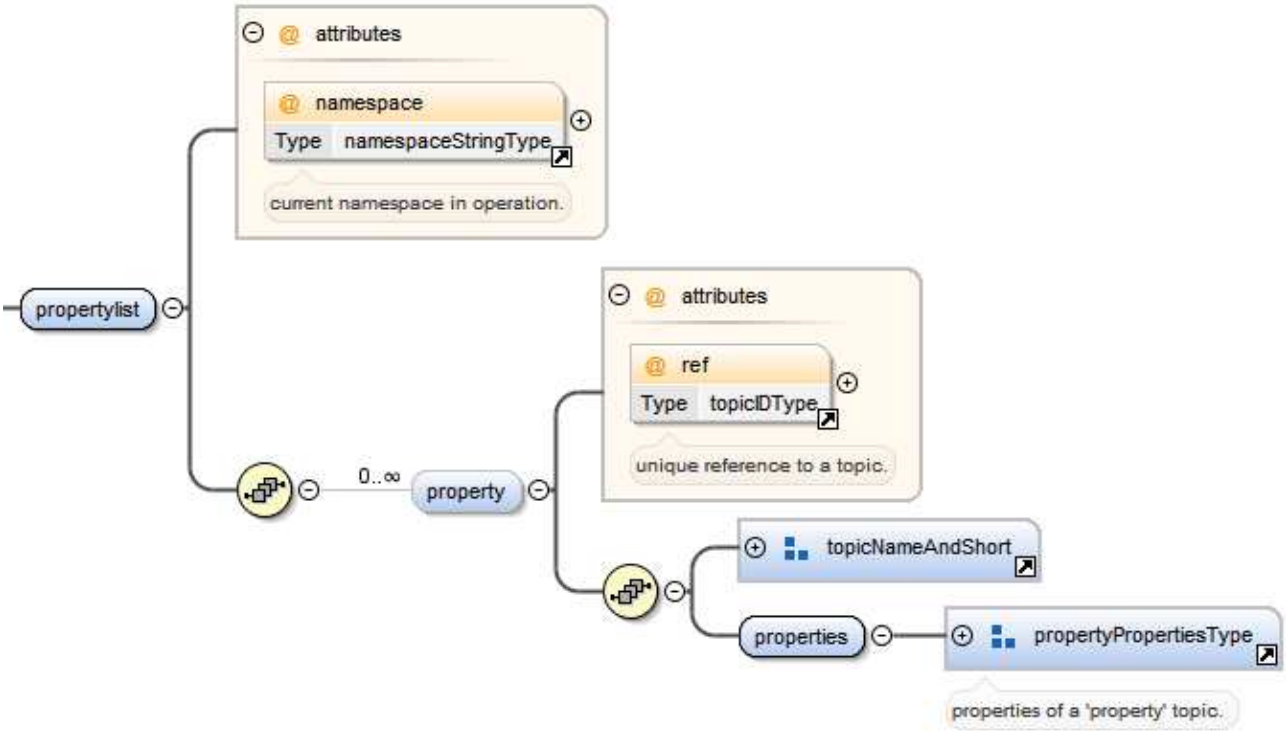
Mit Parameter "namespace=all" werden die Property Topics in allen Namespaces gelistet.

Parameter

namespace	*erforderlich* der Name eines Namespace.
type erlaubter Wert: `property`	*erforderlich* muss auf den Wert "property" gesetzt sein.

XML Schema des Ergebnisses

Die Information ist in ein <infotool> Element eingebettet.



10.23 List Resource Relations

`GET infotool/?op=list`

Listet Relationen zwischen Ressourcen.

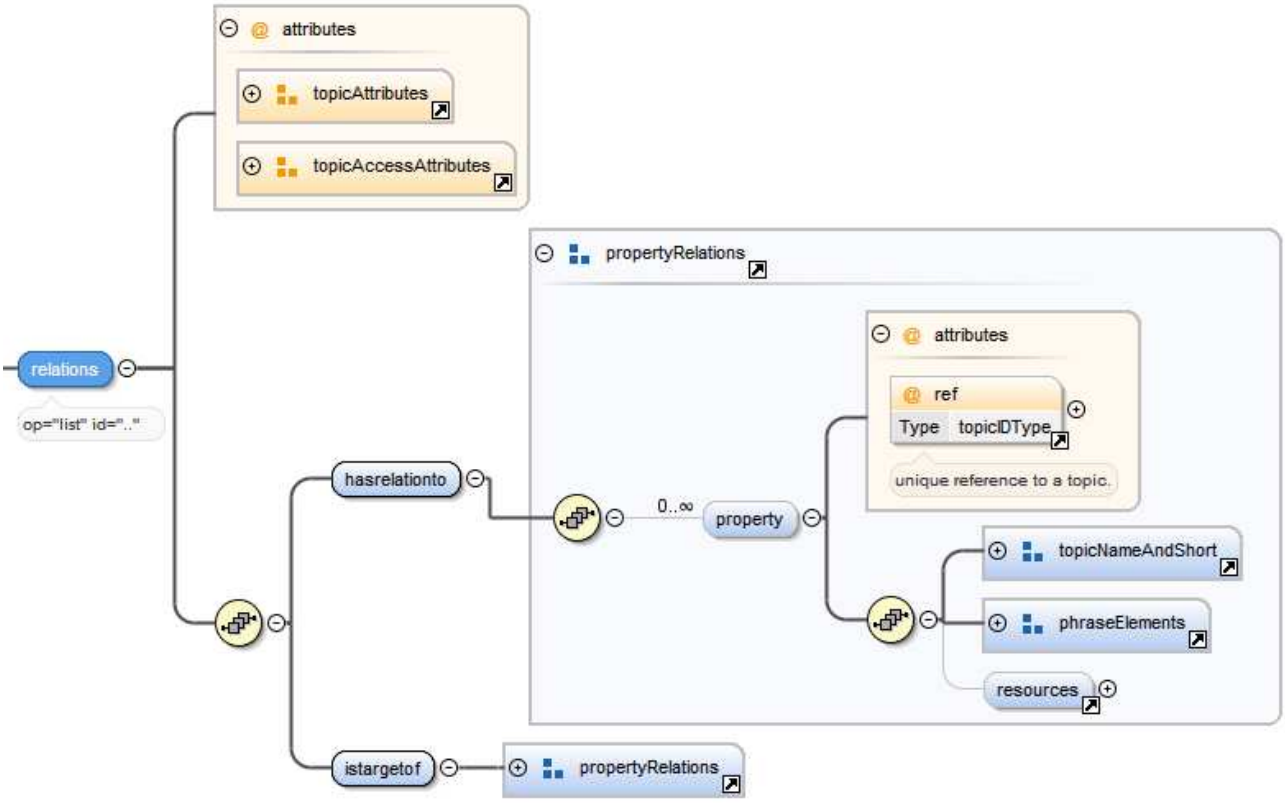
Beispiele:
Resource Project P1 "wird geleitet von" Manager M
Resource Manager M "leitet" Project P1

Parameter

id	*erforderlich* die ID eines Topics.

XML Schema des Ergebnisses

Die Information ist in ein <infotool> Element eingebettet.



10.24 List Resources By Class

`GET infotool/?op=list`

Listet alle Resource Topics von einer bestimmten Class (oder abgeleiteten Class).

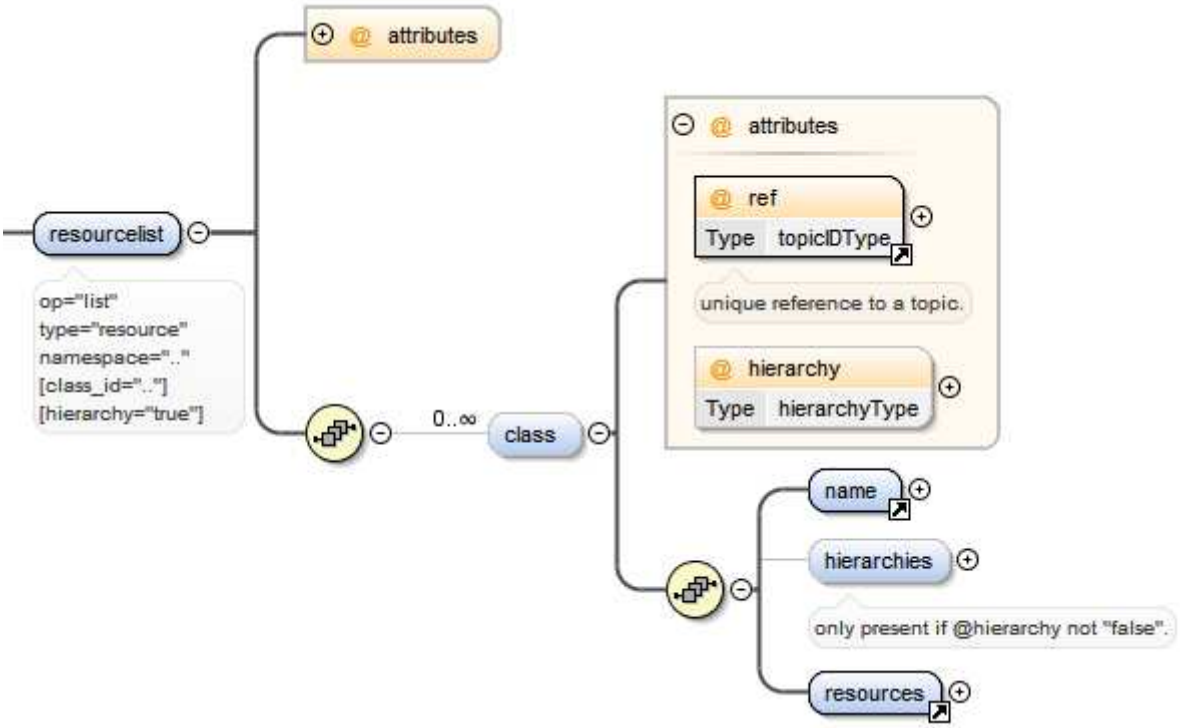
Parameter

class_id	*erforderlich* die ID eines Class Topic.
type erlaubter Wert: `resource`	*erforderlich* muss auf den Wert "resource" gesetzt werden.

XML Schema des Ergebnisses

Die Information ist in ein <infotool> Element eingebettet.

Das XML Schema ist identisch zum Schema der Operation "List Hierarchy", aber die optionalen Hierarchy-Informationen fehlen.



10.25 List Resources By Range Property

`GET infotool/?op=list`

Listet alle Resource Topics die den Range eines Property als Class besitzen.

Beispiel: Wenn das Property "parameters" den Range "Parameter" besitzt, so liefert die Operation eine Liste von Resource Topics mit Class "Parameter".

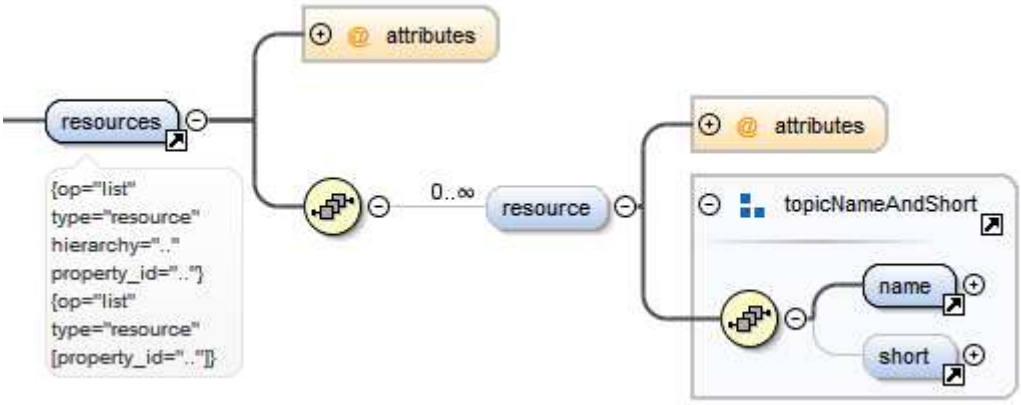
Parameter

ref	*erforderlich* die ID eines Property Topics.
type erlaubter Wert: `resource`	*erforderlich* muss auf den Wert "resource" gesetzt werden.

XML Schema des Ergebnisses

Die Information ist in ein <infotool> Element eingebettet.

Das XML Schema ist identisch zum Schema der Operation "List Hierarchy Children", aber die optionalen Hierarchy-Informationen fehlen.



10.26 List Topics

`GET infotool/?op=list`

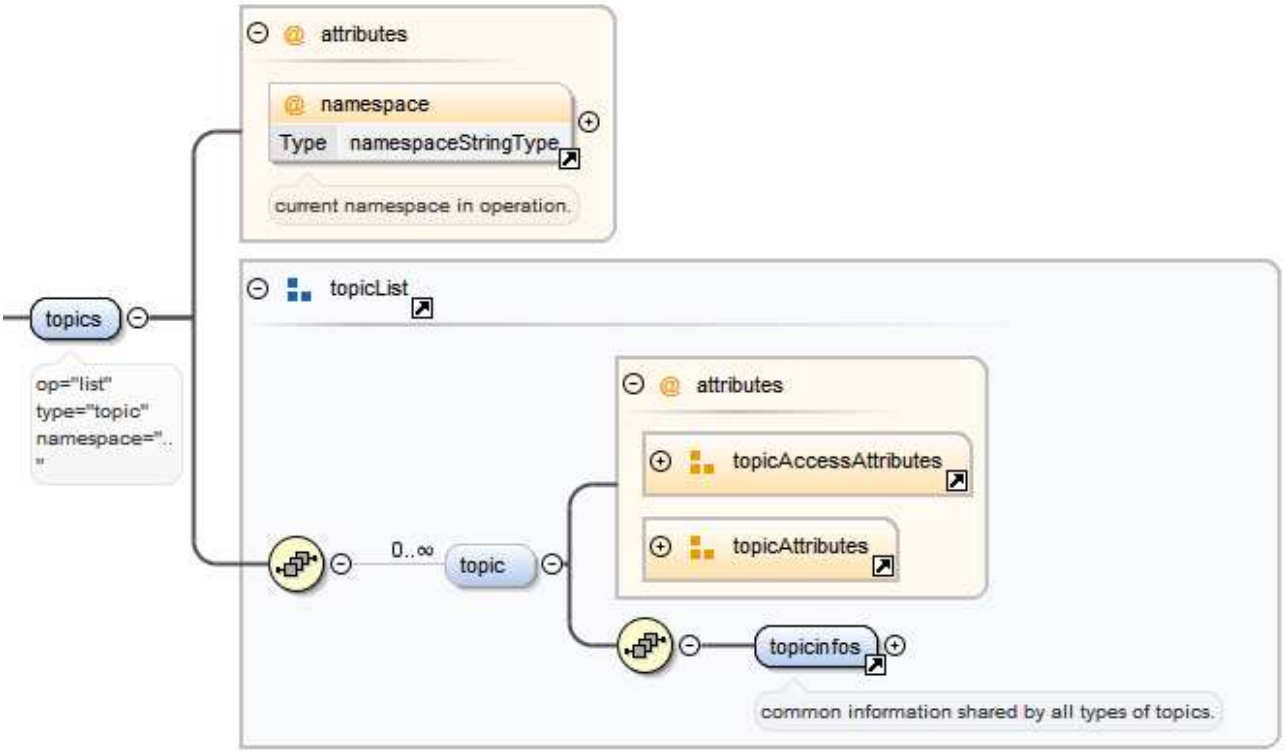
Listet Topics (mit allgemeiner Information wie "id", "name", "short",..) in einem Namespace.

Parameter

namespace	*erforderlich* der Name eines Namespace.
type erlaubter Wert: `topic`	*erforderlich* muss auf den Wert "topic" gesetzt werden.

XML Schema des Ergebnisses

Die Information ist in ein <infotool> Element eingebettet.



10.27 List Users

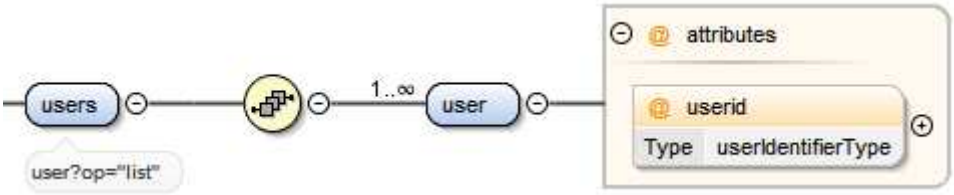
`GET infotool/user?op=list`

Listet alle bekannten Benutzer-Identifikationen.

Die Operation benötigt keine Parameter.

XML Schema des Ergebnisses

Die Information ist in ein <infotool> Element eingebettet.



10.28 Lock Topic

`GET infotool/?op=lock`

Implementiert das Locking (Schreibsperre) von Topics.

es sind folgende Operationen möglich:
"islocked" : liefert "true" falls ein Topic einen Lock aufweist, sonst "false"
"lock" : setzt einen Lock auf dem Topic, sodass er von keinem Benutzer geändert werden kann
"unlock" : nimmt den Lock auf dem Topic wieder zurück

Parameter

id	*erforderlich* die ID eines Topic.
mode erlaubte Werte: `islocked lock unlock`	*erforderlich* Art der Lock Operation.

10.29 Move Topic

`GET infotool/?op=namespace`

Verschiebt ein Topic von einem Namespace in einen anderen (und ändert auch die ID).

Parameter

mode erlaubter Wert: `move`	*erforderlich* wenn "mode=move", dann wird das Topic verschoben.
newnamespace	*erforderlich* Namespace in den ein Topic verschoben werden soll.
oldnamespace	*erforderlich* der Namespace aus dem das Topic verschoben werden soll.

10.30 New Namespace

`GET infotool/?op=new`

Legt einen neuen Namespace an.

Parameter

namespace	*erforderlich* der Name eines Namespace.
type erlaubter Wert: `namespace`	*erforderlich* muss auf den Wert "namespace" gesetzt sein.

10.31 New Property

`GET infotool/?op=new`

Legt ein neues Property Topic an.

Ähnlich wie Operation "New Topic" - aber die optionalen Parameter "class" und "datatype" müssen versorgt werden.

Parameter

class	*optional* die ID einer Class Topic.
datatype	*optional* Datentyp eines neu angelegten Property Topics.
id	*optional* die ID welche einem neuen Topic zugeordnet werden soll. Wenn nicht angegeben, wird die ID automatisch generiert.
lang	*optional* ein Sprach-Code nach ISO 639-1 (z.B. "en", "de").
name	*optional* ein Namens-String.
namespace	*optional* der Namespace in dem das neue Topic angelegt wird.
type erlaubter Wert: `property`	*erforderlich* muss auf den Wert "property" gesetzt sein.

10.32 New Resource

`GET infotool/?op=new`

Legt ein neues Resource Topic an.

Ähnlich wie Operation "New Topic" - aber der optionalen Parameter "class"muss versorgt werden.

Parameter

class	*optional* die ID einer Class Topic.
id	*optional* die ID welche einem neuen Topic zugeordnet werden soll.

	Wenn nicht angegeben, wird die ID automatisch generiert.
lang	*optional* ein Sprach-Code nach ISO 639-1 (z.B. "en", "de").
name	*optional* ein Namens-String.
namespace	*optional* der Namespace in dem das neue Topic angelegt wird.
type erlaubter Wert: `resource`	*erforderlich* muss auf den Wert "resource" gesetzt werden.

10.33 New Topic

`GET infotool/?op=new`

Legt ein neues Topic am Infotool Server an.

Parameter

id	*optional* die ID welche einem neuen Topic zugeordnet werden soll. Wenn nicht angegeben, wird die ID automatisch generiert.
lang	*optional* ein Sprach-Code nach ISO 639-1 (z.B. "en", "de").
name	*optional* ein Namens-String.
namespace	*optional* der Namespace in dem das neue Topic angelegt wird.
type erlaubte Werte: `class` `property` `resource`	*erforderlich* der Typ eines Topics.

10.34 Save Topic

`GET infotool/?op=save`

Speichert ein Topic am Infotool Server.

Die Operation benötigt XML oder JSON Daten.

Die Operation benötigt keine Parameter.

10.35 Save User

`GET infotool/user?op=save`

Speichert Zugriffsrechte von Benutzern am Infotool Server.

Benötigt eine JSON Datenstruktur mit Benutzer-Identifikationen und Rechten.

Die Operation benötigt keine Parameter.

10.36 Search

`GET infotool/search`

Durchsucht Topic Information nach Suchbegriff(en).

Liefert eine Liste von Topics - nur der Parameter "searchstring" ist nötig, alle anderen Parameter sind optional.

Parameter

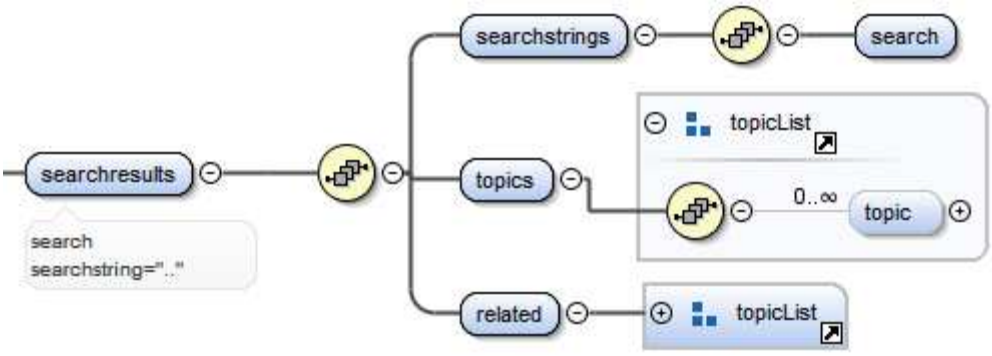
class_id	*optional, kann mehrmals auftreten* die ID einer Class Topic.
count	*optional* bestimmt wie viele Einträge in der Ergebnisliste maximal sein sollen.
createdfrom	*optional* wird gemeinsam mit Parameter "createto" verwendet. Beide schränken den Suchraum auf Topics ein welche innerhalb der gegebenen Zeitspanne angelegt wurden (z.B. zwischen 11-12-2010 und 13-12-2010)
createdto	*optional* siehe Parameter "createfrom".
creator	*optional, kann mehrmals auftreten* restrict the search space to topics that were created by

	user "creator".
modifiedfrom	*optional* wird gemeinsam mit Parameter "modifiedto" verwendet. Beide schränken den Suchraum auf Topics ein welche innerhalb der gegebenen Zeitspanne verändert wurden (z.B. zwischen 11-12-2010 und 13-12-2010).
modifiedto	*optional* siehe Parameter "modifiedfrom".
modifier	*optional, kann mehrmals auftreten* schränkt den Suchraum auf Topics ein welche zuletzt von Benutzer "modifier" geändert wurden.
namespace	*optional, kann mehrmals auftreten* Namespace der durchsucht werden soll.
property_id	*optional, kann mehrmals auftreten* wenn angegeben, findet die Suche nur innerhalb von Werten dieses Property statt.
searchname erlaubter Wert: `true`	*optional* wenn angegeben umfasst die Suche auch Namen von Topics.
searchshort erlaubter Wert: `true`	*optional* wenn angegeben umfasst die Suche auch den Short Text von Topics.
searchstring	*erforderlich* ein Suchbegriff.
searchtag erlaubter Wert: `true`	*optional* wenn angegeben umfasst die Suche auch Tags von Topics.
start	*optional* Index in der Liste der Suchergebnisse an der die Ergebnisse an den Client geliefert werden.

topic_type erlaubte Werte: `class` `namespace` `property` `resource`	*optional, kann mehrmals auftreten* ermöglicht das Filtern der Suchergebnisse nach den angegebenen Topic-Typ(en).

XML Schema des Ergebnisses

Die Information ist in ein <infotool> Element eingebettet.



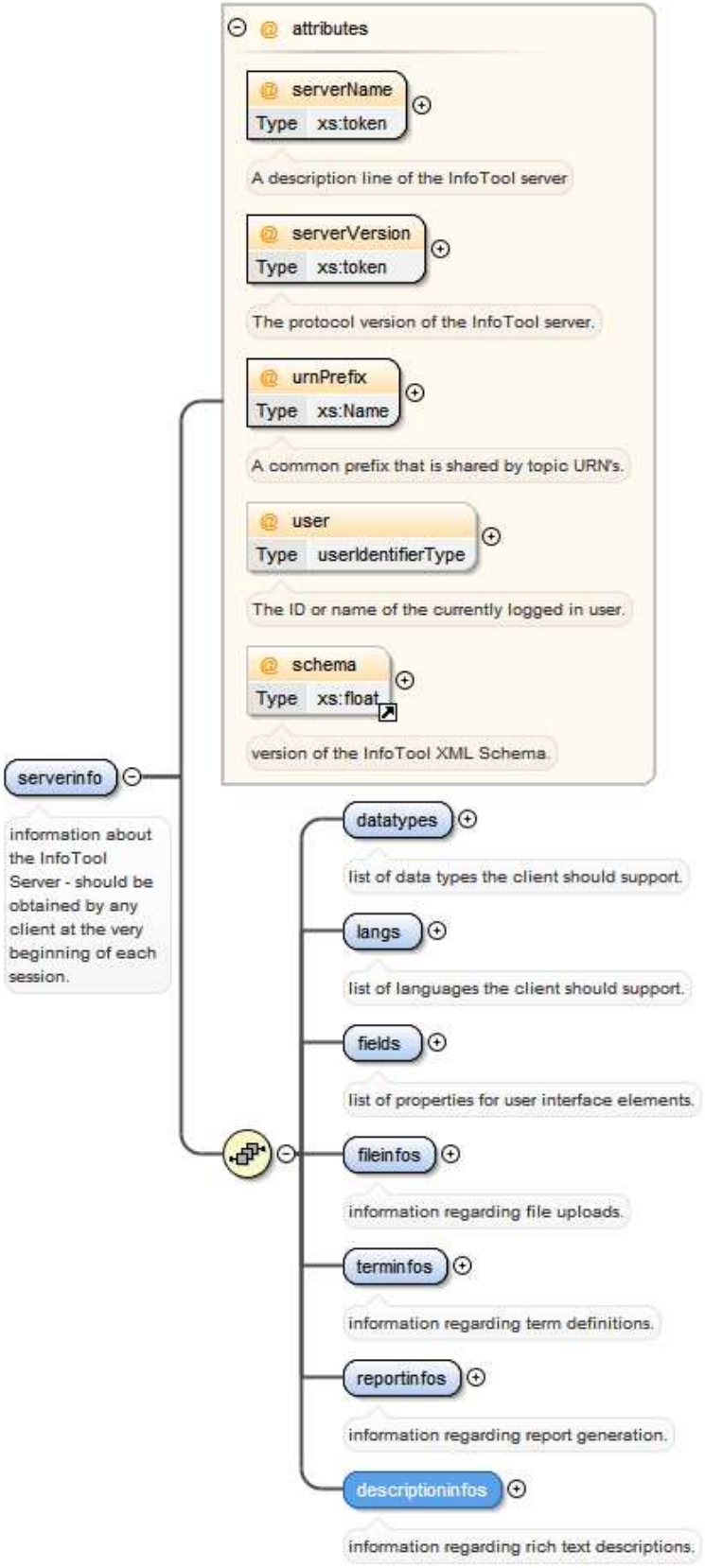
10.37 Server Information

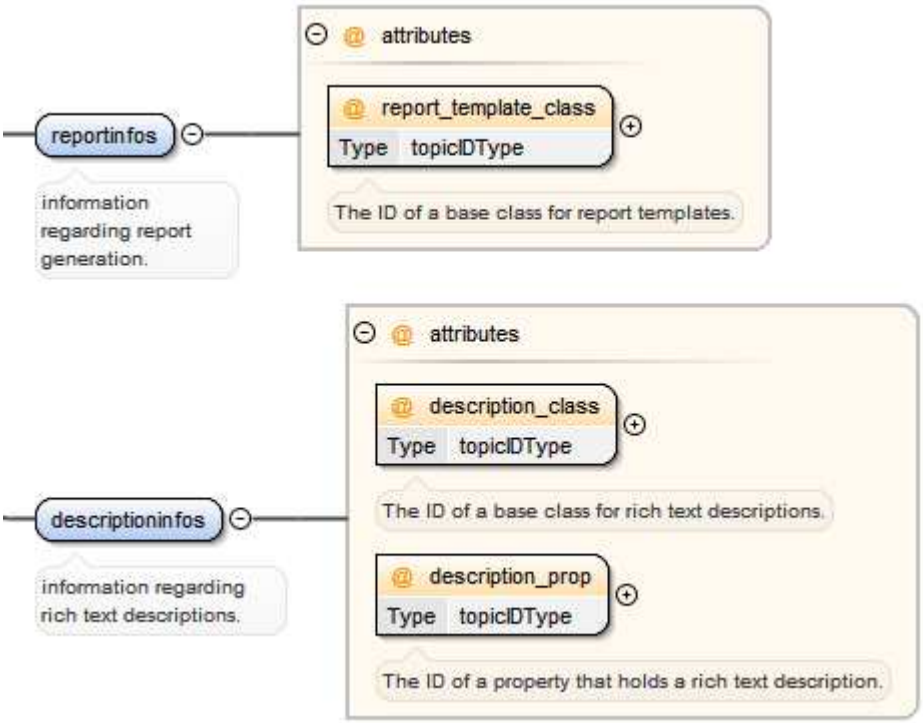
`GET infotool/?op=serverinfo`

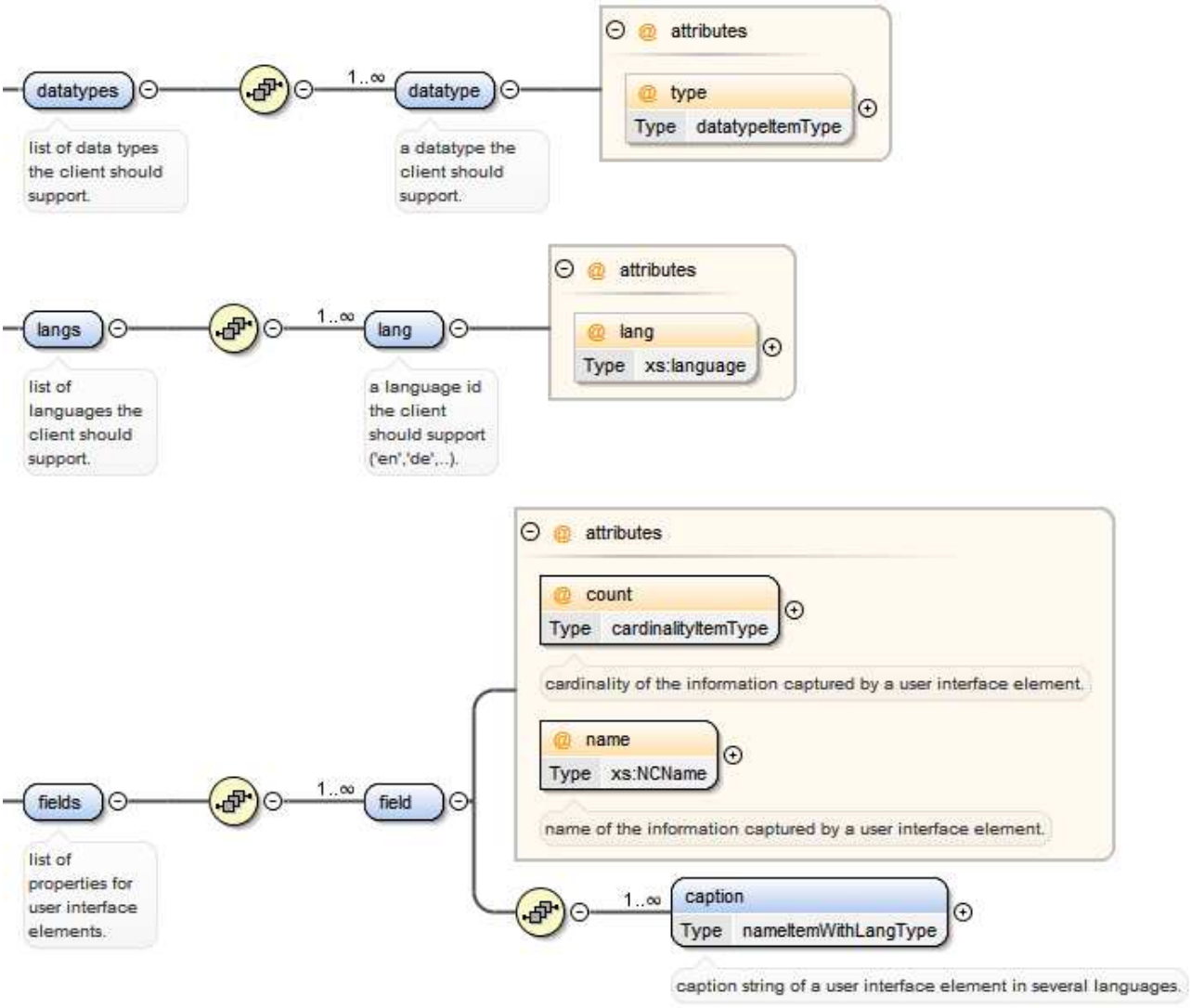
Holt Verwaltungsinformation zum Infotool vom Server.

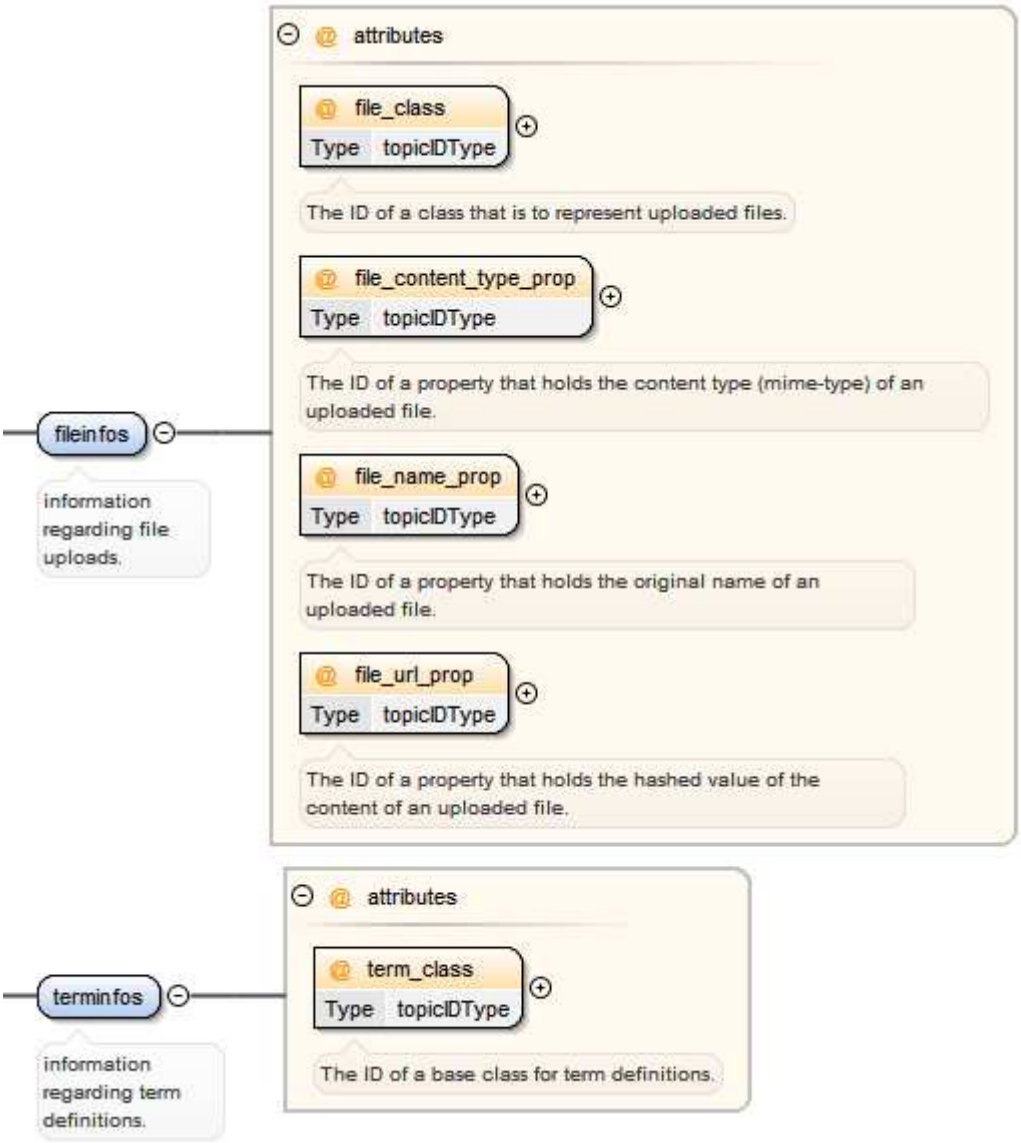
Die Operation benötigt keine Parameter.

XML Schema des Ergebnisses




attributes
report_template_class
Type topicIDType
The ID of a base class for report templates.
reportinfos
information regarding report generation.
attributes
description_class
Type topicIDType
The ID of a base class for rich text descriptions.
description_prop
Type topicIDType
The ID of a property that holds a rich text description.
descriptioninfos
information regarding rich text descriptions.


datatypes
list of data types the client should support.
1..∞
datatype
a datatype the client should support.
attributes
type
Type datatypeItemType
langs
list of languages the client should support.
1..∞
lang
a language id the client should support ('en','de',...).
attributes
lang
Type xs:language
fields
list of properties for user interface elements.
1..∞
field
attributes
count
Type cardinalityItemType
cardinality of the information captured by a user interface element.
name
Type xs:NCName
name of the information captured by a user interface element.
1..∞
caption
Type nameItemWithLangType
caption string of a user interface element in several languages.



10.38 Upload

`GET infotool/upload`

Lädt eine Datei in das Repository des Infotool Servers.

Die Datei kann entweder durch einen lokalen Dateipfad oder über eine URL lokalisiert werden.

Parameter

class_id	*optional* die ID einer Class Topic.
namespace	*erforderlich* der Name eines Namespace.

url	*optional* falls angegeben, erfolgt der Datei Upload aus dieser URL.

10.39 View Topic

`GET infotool/?op=view`

Liefert eine vereinfachte XML Repräsentation eines Topics.

Es sind nur noch sprach-spezifische Informationen entsprechend Parameter "lang" enthalten.

Wenn der Parameter "hierarchy" versorgt ist, sind auch Referenzen auf Child Topics enthalten.

Parameter

hierarchy erlaubter Wert: `true`	*erforderlich* verwendet "hierarchy=true" um eine Liste von Child Resource Topics zu erhalten. verwendet "hierarchy={resource_id} um eine Liste von Child Resources von Resource {resource_id} zu erhalten.
id	*erforderlich* die ID eines Topic.
lang	*optional* ein Sprach-Code nach ISO 639-1 (z.B. "en", "de").

11 Literaturverzeichnis

Allemang, D., & Hendler, J. (2008). *Semantic Web for the Working Ontologist.* Morgan Kaufmann.

Anahory, S., & Murray, D. (1997). *Data Warehouse.: Planung, Implementierung und Administration.* Addison Wesley Verlag.

Apache Cocoon. (2011). *Apache Cocoon*. Abgerufen am 20. 01 2011 von Apache Cocoon: http://cocoon.apache.org

Apache FOP. (2011). *Apache FOP*. Abgerufen am 20. 01 2011 von Apache FOP: http://xmlgraphics.apache.org/fop/

Apache Jackrabbit. (2011). *Apache Jackrabbit*. Abgerufen am 10. 01 2011 von Apache Jackrabbit: http://jackrabbit.apache.org/

Becher, M. (2009). *XML.* W3L Gmbh.

Brücher, H. (2004). *Leitfaden Wissensmanagement.* Zürich: vdf, Hochschulverl. an der ETH.

Burke, E. M. (2001). *Java and XSLT.* O'Reilly.

Cagle, K., Corning, M., & Diamond, J. (2001). *Professional XSL.* Wrox Press.

Chen, X., Snyman, M., & Sewdass, N. (2007). *Interrelationship between document management, information management and knowledge management.* Johannesburg: Department of Information and Knowledge Management, University of Johannesburg.

DocBook.org. (2011). *DocBook.org*. Abgerufen am 18. 01 2011 von DocBook.org: http://www.docbook.org/

Dublin Core Metadata Terms. (2011). *Dublin Core Metadata Terms*. Abgerufen am 20. 01 2011 von Dublin Core Metadata Terms: http://dublincore.org/documents/2008/01/14/dcmi-terms

DuBois, P. (2003). *MySQL Kochbuch.* O'Reilly Germany.

Enterprise Architect. (2011). *sparxsystems.com*. Abgerufen am 14. 01 2011 von sparxsystems.com: http://sparxsystems.com

GNU License. (20. 01 2011). Abgerufen am 20. 01 2011 von gnu.org: http://www.gnu.org/licenses/old-licenses/gpl-2.0.html

Hitzler, P., Krötzsch, M., Rudolph, S., & York, S. (2007). *Semantic Web: Grundlagen.* Springer.

Hüttenegger, G. (2006). *Open Source Knowledge Management.* Springer Verlag Berlin Heidelberg.

JQuery.com. (2011). *JQuery.com*. Abgerufen am 21. 01 2011 von JQuery.com: http://jquery.com/

Json.org. (2011). *json.org*. Abgerufen am 20. 01 2011 von json.org: http://www.json.org

Kleinhans, A. (1989). *Wissensverarbeitung im Management.* Peter Lang.

Kofler, M. (2005). *MySQL 5.* Addison-Wesley.

Lehner, F. (2009). *Wissensmanagement: Grundlagen, Methoden und technische Unterstützung.* Hanser Fachbuchverlag.

Maier, R., Hädrich, T., & Peinl, R. (2005). *Enterprise Knowledge Infrastructures.* Berlin, Heidelberg: Springer Verlag.

Microsoft Office Open XML Format. (2011). *Microsoft Office Open XML Format*. Abgerufen am 21. 01 2011 von Microsoft Office Open XML Format: http://msdn.microsoft.com/en-us/library/aa338205%28v=office.12%29.aspx

Microsoft WordML. (2011). *Microsoft WordML*. Abgerufen am 20. 01 2011 von Microsoft WordML: http://msdn.microsoft.com/en-us/magazine/cc164064.aspx

Oasis DITA. (2011). *Oasis DITA*. Abgerufen am 21. 01 2011 von Oasis DITA: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=dita

OpenOffice XML Format. (2011). *OpenOffice XML Format*. Abgerufen am 21. 01 2011 von OpenOffice XML Format: http://xml.openoffice.org/xml_specification.pdf

OpenRDF.org. (2011). *OpenRDF.org*. Abgerufen am 21. 01 2011 von OpenRDF.org: http://www.openrdf.org/

OpenRDF.org Sesame Server. (2011). *OpenRDF.org Sesame Server*. Abgerufen am 21. 01 2011 von OpenRDF.org Sesame Server: http://www.openrdf.org/doc/sesame/users/figures/sesame-server.png

Oxygen XML Editor. (20. 12 2010). *Oxygen XML Editor*. Abgerufen am 20. 12 2010 von Oxygen XML Editor: http://www.oxygenxml.com

Powers, S. (2003). *Practical RDF.* O'Reilly Media.

PrinceXML. (2011). *PrinceXML*. Abgerufen am 20. 01 2011 von PrinceXML: http://www.princexml.com

Reese, G., Randy, J. Y., & King, T. (2002). *MySQL: Einsatz und Programmierung.* O'Reilly Germany.

Riempp, G. (2004). *Integrierte Wissensmanagement-Systeme: Architektur und praktische Anwendung.* Springer.

Saxonica.com. (2011). *www.saxonica.com*. Abgerufen am 15. 01 2011 von www.saxonica.com: http://www.saxonica.com

Vogel, O., Arnold, I., Chughtai, A., Ihler, E., Kehrer, T., U., M., et al. (2008). *Software-Architektur: Grundlagen - Konzepte - Praxis.* Spektrum Akademischer Verlag.

W3.org CSS Paged Media. (2011). *W3.org CSS Paged Media*. Abgerufen am 20. 01 2011 von W3.org CSS Paged Media: http://dev.w3.org/csswg/css3-page/

W3.org CSS. (2011). *W3.org CSS*. Abgerufen am 20. 01 2011 von W3.org CSS: http://www.w3.org/TR/CSS/

W3.org HTML4. (2011). *W3.org HTML4*. Abgerufen am 19. 01 2011 von W3.org HTML4: http://www.w3.org/TR/html401/

W3.org MathML2. (2011). *W3.org MathML2*. Abgerufen am 20. 01 2011 von W3.org MathML2: http://www.w3.org/TR/MathML2/

W3.org RDF/XML. (2011). *W3.org RDF/XML*. Abgerufen am 15. 01 2011 von W3.org RDF/XML: http://www.w3.org/TR/REC-rdf-syntax/

W3.org Semantic Web. (2011). *www.w3.org*. Abgerufen am 18. 01 2011 von www.w3.org: http://www.w3.org/2007/Talks/0130-sb-W3CTechSemWeb/#%2824%29

W3.org SPARQL. (2011). *W3.org SPARQL*. Abgerufen am 21. 01 2011 von W3.org SPARQL: http://www.w3.org/TR/rdf-sparql-query

W3.org SPARQL XML Results. (2011). *W3.org SPARQL XML Results*. Abgerufen am 21. 01 2011 von W3.org SPARQL XML Results: http://www.w3.org/TR/rdf-sparql-XMLres

W3.org SVG. (2011). *W3.org SVG*. Abgerufen am 20. 01 2011 von W3.org SVG: http://www.w3.org/TR/SVG/

W3.org XHTML. (2011). *W3.org XHTML*. Abgerufen am 20. 01 2011 von W3.org XHTML: http://www.w3.org/TR/xhtml1/

W3.org XPath. (2011). *W3.org XPath*. Abgerufen am 20. 01 2011 von W3.org XPath: http://www.w3.org/TR/xpath-functions/#func-doc

W3.org XProc. (2011). *W3.org XProc*. Abgerufen am 20. 01 2011 von W3.org XProc: http://www.w3.org/TR/xproc/

W3.org XSL. (2011). *W3.org XSL*. Abgerufen am 20. 01 2011 von W3.org XSL: http://www.w3.org/TR/xsl/

Will, M. (2002). *Aufbau und Nutzung einer digitalen Bibliothek in einer universitären Ausbildungsumgebung.* Waxmann.

Willke, H. (2001). *Systemisches Wissensmanagement.* Lucius & Lucius.

XStandard. (2010). *Xstandard.com*. Abgerufen am 21. 12 2010 von Xstandard.com: http://www.xstandard.com