# Visualisierung von Leistungsdaten eines Softwareproduktes auf Basis Equinox OSGi und Eclipse BIRT

**Masterarbeit**

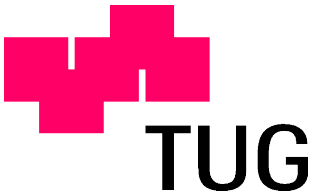

eingereicht von

**Johannes Bickel**

am

**Institut für Maschinenbau- und Betriebsinformatik**

Technische Universität Graz


November 2009


Gutachter: O.Univ.-Prof. Dipl.-Ing. Dr.techn. Siegfried Vössner

Betreuer: Dipl.-Ing. Wolfgang Vorraber

# 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, am .....  .....

(Unterschrift)

# DANKSAGUNG

Ich möchte in besonderer Weise meinen Eltern Helga und Othmar danken, die mir das Informatik-Studium ermöglicht haben und mir die nötigen Gene mitgegeben haben es auch durchzustehen.

Weiters möchte ich meine Freundin Elisabeth danken, deren Nervern in der Zeit, in der ich diese Masterarbeit verfasst habe öfters auf eine harte Probe gestellt wurden.

Ein besonderer Dank gilt meinem Betreuer DI Wolfgang Vorraber, der meine Arbeit immer wieder der Korrektur gelesen hat und durch seine Korrekturen und Kritik dieses Ergebnis erst möglich gemacht hat. Ich danke ihm für seine nicht endend wollende Ausdauer und die gute Zusammenarbeit.

Die Möglichkeit eine Masterarbeit bei der Firma XiTrust Secure Technologies zu schreiben wurde mir von DI Helmut Aschbacher eröffnet. Ich Danke ihm für diese Möglichkeit und die wertvolle inhaltliche Vorarbeit zu meiner Masterarbeit im Bereich Smart Services.

Seitens der Firma XiTrust gilt mein Dank dem Geschäftsführer DI Georg Lindsberger, der mir die Chance und das Vertrauen schenkte, eine Masterarbeit für ihn zu schreiben. Zusätzlich möchte ich DI Gerhard Fliess danken, der mir immer ein ausgezeichnter Ansprechpartner und Mentor war und mir mit seinem großen Wissen die Arbeit erleichterte.

Abschließend gilt mein Dank der Firma all car tuning und im besonderen dem Geschäftsführer Mario Deimbacher, der mir durch eine Reduzierung der Arbeitszeit die benötigte Zeit zur Erstellung meiner Masterarbeit gab.

# KURZFASSUNG

Die Vermarktung von Services stellt Dienstleistungsanbieter wie die Firma XiTrust Secure Technologies GmbH immer wieder vor eine große Herausforderung. Aus diesem Grund entwickelte DI Helmut Aschbacher im Rahmen seiner Masterarbeit an der TU Graz ein Service Konzept angepasst an die Firma XiTrust, welches das angesprochene Problem in Zukunft lösen soll. Sein Service Konzept verlangt die Entwicklung eines Tools, das in der Lage ist, die Leistung des XiTrust Business Servers (XBS) besser zu visualisieren als es bislang möglich ist.

In dieser Masterarbeit gilt es, ein solches Monitoring Tool für den E3 Server, die neueste Generation des XBS, zu entwickeln. Die Basis des XBS bildet die OSGi Implementierung von Eclipse. Ein Framework, das höchste Erweiterbarkeit und Wiederverwendbarkeit von Software Paketen (Plugins) zum Ziel hat. Dieses System stellt besondere Herausforderungen an die Entwicklung des Monitoring-Systems für den XBS.

Die Basis für das Konzept des Monitoring-Systems stellen drei Analysen dar. Diese umfassen Protokolle und Standards aus dem Monitoring Bereich, Software-Architekturen und bestehende Monitoring Tools. Für die Visualisierung der akkumulierten Daten kommt mit den Business Intelligence Reporting Tools (BIRT) eine weitere Eclipse Technologie zum Einsatz.

Die durch das hier entwickelte Monitoring Tool ermöglichte Visualisierung der Leistung des XBS soll den direkten Nutzen von XBS-Services für Kunden und Entwickler deutlich machen. In weiterer Folge soll die Zahlungsmoral der Kunden gesteigert, ebenso sollen zusätzliche Verkaufsargumente für den Hersteller geschaffen werden.

# ABSTRACT

The marketing of services always is a great challenge for service providers like XiTrust Secure Technologies GmbH. For this reason DI Helmut Aschbacher developed a new service conept as part of his masters thesis at the Technical University of Graz, which is adapted to XiTrust and should solve the problems that may rise in the future. His service concept requires the development of a tool that is capable of visualizing the performance of the XiTrust Business Server (XBS) better than before.

In this master thesis a monitoring tool for the E3 server, the latest generation of the XBS, is being developed. The basis of the XBS is Equinox, the OSGi implementation of Eclipse, a framework of highest expandability plus reusability of software packages. This system will represent a special challenge for the development of the monitoring system for the XBS.

A knowledge basis for the concept was build up through three researches. These include protocols and standards from the monitoring area, software architectures and existing monitoring tools. The visualization of the accumulated data will be done by Business Intelligence Reporting Tools (BIRT), which is another Eclipse technology.

In this master thesis the developed monitoring tool demonstrates a direct benefit from XBS services for customers and developers by enabling visualization of the performance of the XBS. Subsequently, the customers payment behavior will increase and additional manufacturer sales arguments will be created.

# Inhaltsverzeichnis

# 1 EINLEITUNG

Die Vorarbeit zu dieser Masterarbeit liefert DI Helmut Aschbacher mit seiner Master Thesis "Entwicklung eines Service Engineering Prozessmodells für Software Solution Provider im KMU Sektor unter Nutzung eines webbasierten Informationssystems". Er beschreibt darin das Problem der immer schwieriger werdenden Vermarktung von Services im Software Solution Bereich. Er verdeutlicht weiters, dass es immer wichtiger wird, dem Kunden die Leistungen seines Software-Produkts zu verdeutlichen.

Beruhend auf einer Publikation der Technologie Stiftung Hessen mit dem Titel "Dienstleistung - von der Renditefalle zum Wettbewerbsvorteil" beschreibt (Aschbacher 2009) drei wesentliche Renditefallen beim Einsatz von Services eines Dienstleistungsanbieters (vgl. TechnologieStiftung Hessen GmbH 2002 S. 24-28 online):

1. **Renditefalle 1: Preisvergleich ohne Leistungsvergleich**
   Der Kunde ist nur bereit für einen Service zu zahlen, wenn ihm auch der unmittelbare Nutzen des Services bewusst ist. Das heisst dem Kunden muss der Grund für das In-Anspruch-nehmen eines Services klar sein.

2. **Renditefalle 2: Tue Gutes und verschweig's**
   Für den Kunden ist der Wert eines Services vor der Erbringung oft nicht abschätzbar. Dies gilt besonders für Dienstleistungen mit hohem Wissensanteil. Dem Kunden muss der Nutzen, der sich durch die Erbringung eines Services ergeben hat auch dem entsprechend klar visualisiert werden.

3. **Renditefalle 3: Wir machen alles für den Kunden**
   Man überfordert den Kunden durch ein zu großes Serviceangebot.

Weitereführende Literatur zu diesem Thema bieten (Aschbacher 2009) oder (TechnologieStiftung Hessen GmbH 2002).

Besonders die ersten beiden Punkte der Aufzählung der Renditefallen sind Motivation für diese Masterarbeit. Durch die bessere Visualisierung der Leistung eines Softwareproduktes kann dem Kunden der Nutzen beispielsweise von Updates oder Upgrades einfacher deutlich gemacht werden. Nicht nur dem Kunden gegenüber entsteht ein direkter Nutzen aus der Lei-

stungsvisualisierung. Der Dienstleistungsanbieter kommt in den Besitz eines leistungsfähigen Instrumentariums für das Marketing seiner Services des jeweiligen Software-Produkts.

Im konkreten Fall der Firma XiTrust Secure Technologies GmbH, für die DI Helmut Aschbacher im Zuge seiner Masterarbeit ein Service Konzept entworfen hat, soll die Leistung des XiTrust Business Server (XBS) visualisiert werden. Ziel dieser Arbeit ist es, auf Basis der Analyse von Software-Architekturen, Standards und Protokollen aus dem Monitoring Kontext sowie bestehenden Monitoring-Systemen, ein Monitoring-System für den XBS zu entwickeln.

Die Abbildung 1.1 zeigt den Aufbau dieser Masterarbeit:

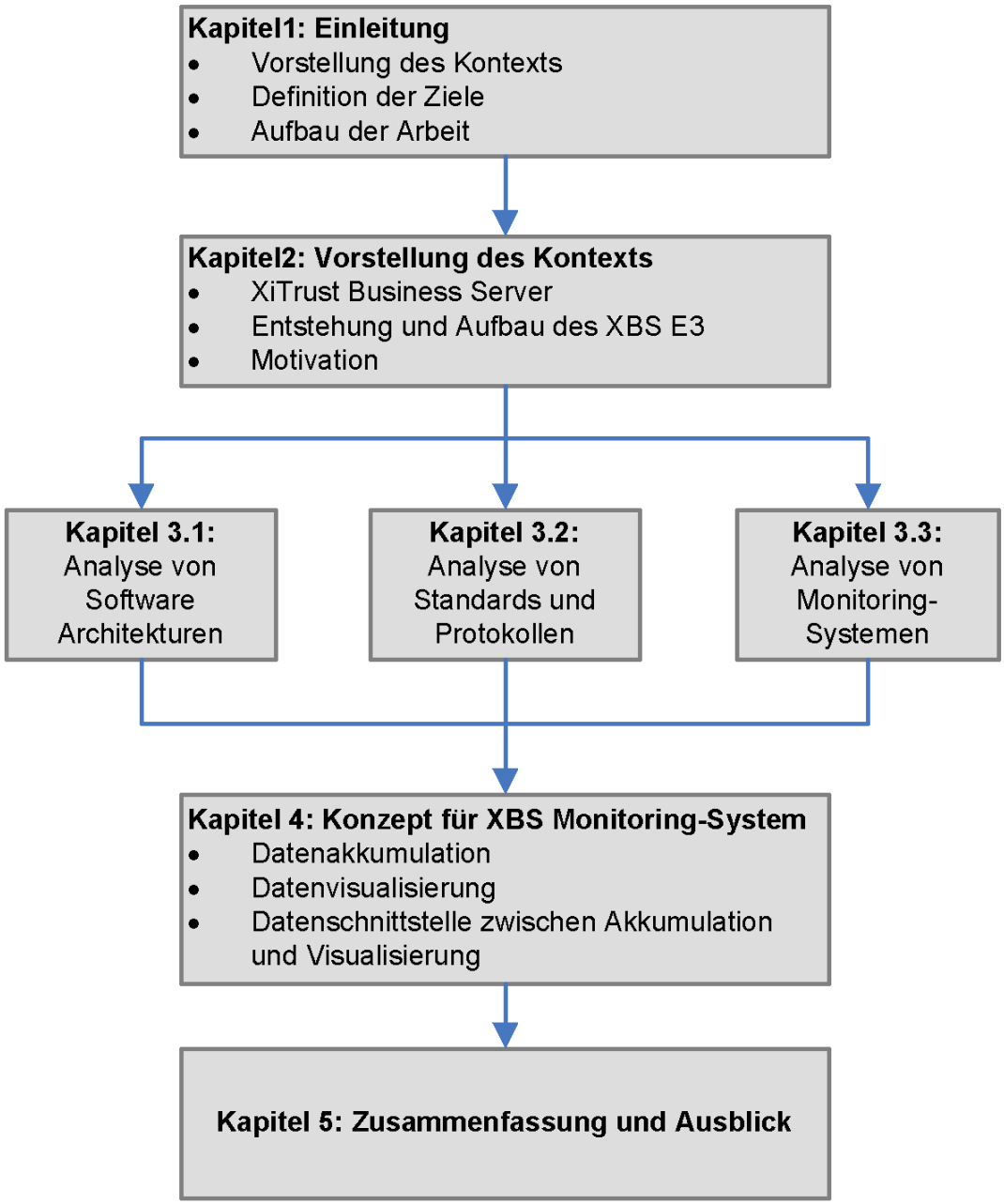



Abbildung 1.1: Aufbau der Masterarbeit

Auf die Einleitung in Kapitel 1 folgt die Vorstellung des Kontextes der Masterarbeit in Kapitel 2.

Es wird zuerst die Firma XiTrust Secure Technologies GmbH vorgestellt. Im Anschluss daran wird der XiTrust Business Server (XBS), das Produkt dessen Leistung es zu visualisieren gilt, präsentiert. Die Darstellung der geschichtlichen Entwicklung des XBS bis zur heutigen Version E3 soll die Architektur und die verwendeten Technologien unterlegen.

Mit OSGi Equinox wird die Basis des Client-Server Systems vorgestellt. Mit dem Aspekt der hohen strukturellen Komplexität, die ein Resultat der gestiegenen Modularität ist, wird auch eine Gefahr des Systems aufgezeigt.

Im Kapitel 3 werden die Grundlagen für das Design eines Monitoring-Systems für den XBS erarbeitet. Diese setzen sich aus drei essentiellen Analysen zusammen:

1. Analyse von Software-Architekturen
2. Analyse von Standards und Protokollen im Monitoring Kontext
3. Analyse von bestehenden Monitoring-Systemen

Die Analysen sollen Antworten für typische Fragen beim Entwurf eines Monitoring-Systems liefern. Beispielsweise wie Daten des Systems akkumuliert oder persistiert werden können unter Berücksichtigung der Architektur des XBS.

Aufbauend auf dem Output dieser Analysen wird im Kapitel 4 das Konzept für ein eigenes Monitoring-System des XBS, bestehend aus Datenakkumulation, Visualisierung basierend auf Eclipse BIRT und Datenschnittstelle vorgestellt.

Das letzte Kapitel fasst den Inhalt der Masterarbeit zusammen und diskutiert offene Punkte und weiterführende Aspekte.

# 2 AUSGANGSSITUATION

Mit dem XiTrust Business Server (XBS) entwickelte die Firma XiTrust Secure Technologies GmbH ein Werkzeug rund um Signatur, Verschlüsselung und Dokumentenkonvertierung (vgl. XiTrust Secure Technologies GmbH 2009 online). Der XBS erledigt, zentral in der IT-Struktur eines Unternehmens positioniert, unternehmenswichtige Aufgaben etwa in der elektronischen Kommunikation, Rechnungslegung oder Archivierung. Diese Tätigkeiten finden aus der Sicht der Mitarbeiter und des Managements transparent statt. Das große Problem dieser Arbeitsweise ist die fehlende Sichtbarkeit. Unweigerlich stellen sich bei Management Entscheidungen der Kunden die Frage: Was leistet der XBS überhaupt? Nur Systemadministratoren oder Entwicklern ist es bislang möglich geeignete Statistiken durch Ausarbeitung diverser Log-Files zu erstellen. Diese sind für betriebliche Entscheidungen unabdingbar. In diesem Kapitel wird der XBS und seine Entwicklung vorgestellt, ebenso werden die Probleme der fehlenden Transparenz analysiert.

## 2.1 XiTrust Business Server

Der XiTrust Business Server (XBS) ist ein modulares System zur Bewältigung verschiedener Aufgaben in einem Unternehmen. Zu den Kernaufgaben des XBS zählen beispielsweise Verschlüsselung von E-Mails oder Erstellung digitaler Signaturen. Mittels zahlreicher Schnittstellen kann der XBS in die Ziel IT-Landschaft des gewünschten Unternehmens integriert werden. Es existieren Schnittstellen etwa zu E-Mail Server, Enterprise Ressource Planing Systeme (ERP) oder Archiv-Server (vgl. XiTrust Secure Technologies GmbH 2009 online). Die Abbildung 2.1 zeigt diesen zentralen Einsatz des XBS.

Ziel der Entwickler war es ein System mit hoher Modularität zu entwerfen. Das Resultat ist ein schlanker Kern und eine Sammlung von Plugins, die frei zu verschiedenen Produkten kombiniert werden können. Solche Produkte sind zum Beispiel der XiTrust Timestamp Server[1] oder der XiTrust Interaction Server[2]. Zusätzlich zu diesen Produkten, entstehen durch die hohe Wiederverwendbarkeit der Plugins, schnell und einfach individuelle Lösungen für Kunden.

[1] http://www.xitrust.com/zeitstempel-timestamp-server.php
[2] http://www.xitrust.com/signatur-interaction-server.php


Abbildung 2.1: XiTrust Business Server
(XiTrust Secure Technologies GmbH 2009 online)

Der XBS ist die Erweiterung einer Idee des Institute for Applied Information Processing and Communications (IAIK)[3], die im Jahr 2001 entstand. Damals wurde der Plan gefasst, einen Proxy zur elektronischen Signatur von E-Mails zu entwickeln. Grund dafür war die fehlende Signatur-Funktionalität gängiger E-Mail Clients zu dieser Zeit. Die Firma XiTrust Secure Technologies GmbH[4] erwarb die notwendigen Rechte und entwickelte das System weiter.

Der Erfolg des Produkts brachte immer neue Kundenanforderungen, die die Firma XiTrust zur Entwicklung des XBS "E2" veranlasste. Der E2 brachte eine ganze Reihe zusätzlicher Funktionen mit. Darunter fielen Zertifikatsverwaltung, Signieren von Dokumenten, Verschlüsselung von Daten und viele mehr. Basis des monolithischen Systems war Java in der Version 5. Im Laufe der Zeit wuchs das System und damit die Probleme der monolithischen Architektur des E2. Die gravierendsten Probleme waren:

- Erweiterungen waren immer schwieriger umzusetzen. Beispielsweise war die geplante Einführung eines Rechtesystems nicht zu realisieren, da weitreichende Änderungen an zu vielen verschiedenen Stellen des Systems notwendig gewesen wären.
- Es entstanden Konflikte zwischen verwendeten Bibliotheken

[3] http://www.iaik.tugraz.at
[4] Damals noch unter dem Namen XiCrypt

- Die Abhängigkeiten des Systems wurden unüberschaubar
- Fehlende interne Systemgrenzen ermöglichen eine uneingeschränkten Zugriff auf alle Komponenten

Zu diesem Zeitpunkt umfasste der E2 in seiner Grundfunktionalität 275 000 Zeilen Code, 2200 Klassen und 250 packages.

Diese Probleme bewogen das Entwicklerteam der Firma XiTrust zur Neuentwicklung des Systems auf Basis moderner Konzepte, die den erforderlichen Anforderungen von hoher Modularität und Erweiterbarkeit des Systems gerecht werden konnten. Es entstand in den Jahren 2008 und 2009 die neueste Generation des XBS, der "E3" Server.

## 2.2 Der XBS E3

Der E3 Server ist die aktuelle Version des XBS. In den nachstehenden Unterkapiteln werden der Aufbau und der Kern mit seiner Basisfunktionaltiät vorgestellt. Im Anschluss werden vorhandene Probleme und mögliche Gefahren des Systems analysiert.

### 2.2.1 Aufbau

Wie bereits im vorigen Kapitel erwähnt, stieß man in Sachen Erweiterbarkeit an die Grenzen der monolithischen Architektur des E2 Servers. Dadurch wurde eine Neuentwicklung notwendig. Eine Analyse der Anforderungen im Vorfeld brachte zwei essentielle Punkte zu Tage:

1. **Kombination zu verschiedenen Produkten**
   Der E3 Server muss durch hohe Modularisierung die Möglichkeit bieten, Code-Teile miteinander frei zu verschiedenen Produkten zu kombinieren. Das fordert gleichzeitig ein hohes Maß an Wiederverwendbarkeit der einzelnen Code-Teile.
2. **Erweiterbarkeit und individuelle Lösungen**
   Das System soll leicht erweitert werden können. Zusätzliche Funktionalität oder individuelle Kundenlösungen müssen einfach und möglichst ohne weitreichende Adaptionen des bestehenden Codes umgesetzt werden können. Dies fordert einen stabilen Kern mit der entsprechenden Grundfunktionalität.

Diese zwei Anforderungen führten das Entwicklerteam zur OSGi Service Platform. Diese bietet ein entsprechendes Modularisierungskonzept in Java, das dynamische Integration von Softwarekomoponenten (Bundles) und Diensten (Services) ermöglicht (vgl. Wütherich *et al.* 2008 S. 12). Verantwortlich für die Spezifikation ist die OSGi Alliance, die im Jahr 1999 von den Organisationen Ericsson, IBM, Oracle und Sun gegründet wurde. Bis zum Jahr 2004 trug sie den

Namen Open Service Gateway Initiative (vgl. Wütherich *et al.* 2008 S. 12). Die Abbildung 2.2 zeigt den Aufbau des OSGi Frameworks, das die Basis der OSGi Service Platform bildet.

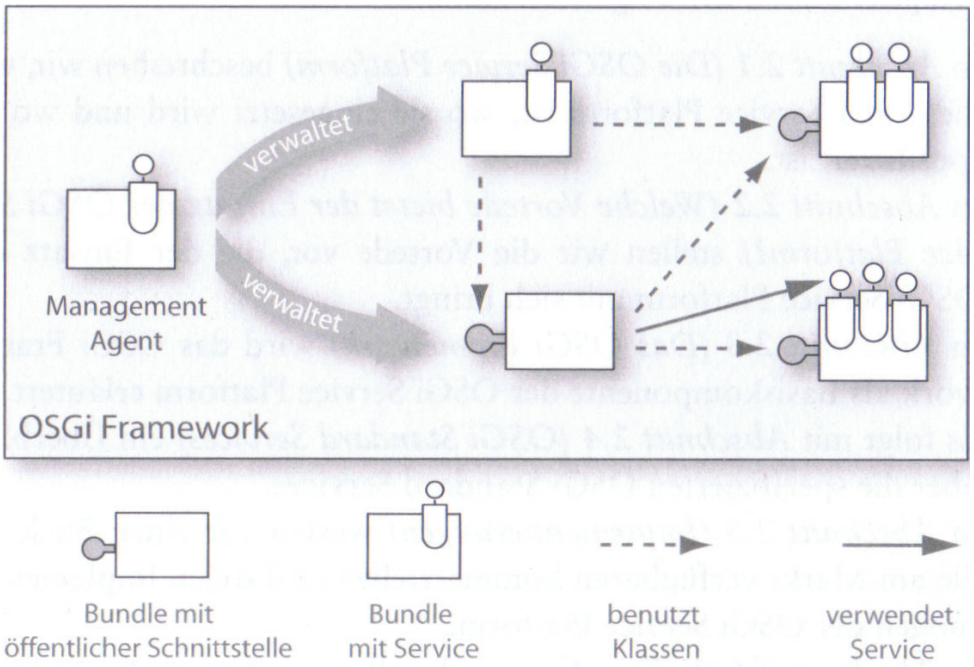



Abbildung 2.2: OSGi Framework
(Wütherich *et al.* 2008 S. 12)

Ein Bundle besteht aus Klassen und Ressourcen, die zur Laufzeit installiert und deinstalliert werden können. Um Klassen für andere Bundles sichtbar zu machen ist explizites Exportieren der gewünschten Klassen notwendig. Will ein anders Bundle diese verwenden, muss es sie explizit importieren. Im- und Export werden in der jeweiligen Manifest-Datei des Bundles definiert (vgl. Wütherich *et al.* 2008 S. 12f).

Ein Bundle hat durch Services die Möglichkeit Funktionalität global zur Verfügung zu stellen. Services sind nach deren Registrierung über die zentrale Service Registry verfügbar. Ein Service selbst ist ein Java Object, das mittels Interface-Name registriert wird. Durch die Abstraktion des Interfaces müssen Bundles den Anbieter des Services nicht kennen, was zu einer weiteren Entkoppelung der Bundles führt. Ebenso wie Bundles, können auch Dienste zur Laufzeit hinzugefügt oder entfernt werden (vgl. Wütherich *et al.* 2008 S. 13).

Als Resultat der Recherche um OSGi wurde der Entschluss gefasst, den E3 auf Basis der OSGi-Implementierung von Eclipse Equinox aufzubauen. Die Wahl von Equinox basierte auf drei wesentlichen Vorteilen:

1. Eclipse Equinox bildet die Grundlage der Eclipse Integrated Development Environment (IDE) und aller Eclipse basierten Eclipse Rich Client (RCP) Anwendungen (vgl. Wütherich *et al.* 2008 S. 30f). Die RCP Plattform bietet eine Zusammenstellung von Komponenten, die das Entwickeln von interaktiven Applikationen enorm vereinfachen. Das Rad sollte in diesem Punkt nicht neu erfunden werden.

2. Eclipse Equinox bietet eine Extension Registry, wodurch Bundles offen für Erweiterungen anderer Bundles gemacht werden können. Dazu bieten Bundles deklarativ in der `plugin.xml` Datei Extension Points an, die wiederum von anderen Bundles verwendet werden können, um dessen Funktionaliät zu erweitern (vgl. Wütherich *et al.* 2008 S. 389). Weitere Details zur Verwendung des Extension Point Mechanismus kann bei (Wütherich *et al.* 2008) im Kapitel 22.2 nachgelesen werden.

3. Die Eclipse IDE bietet mit dem Plugin Development Environment (PDE) eine hervorragende Entwicklungsumgebung zur Implementierung von OSGi Bundles. Neben den aus Eclipse bekannten Entwickler-Features, bietet PDO Editoren für Manfest und Plugin Abhängigkeiten sowie Wizards für Export, Projekterstellung und Ausführung an (vgl. Wütherich *et al.* 2008 S. 31f).

Erfahrungen aus dem Java Umfeld führten schließlich, basierend auf weiteren Analysen, zur Entwicklung der E3 Server Architektur. Die Abbildung 2.3 zeigt die Architekturen des Servers und des Clients. Der Client bietet die Möglichkeit der Remote-Konfiguration des Servers.

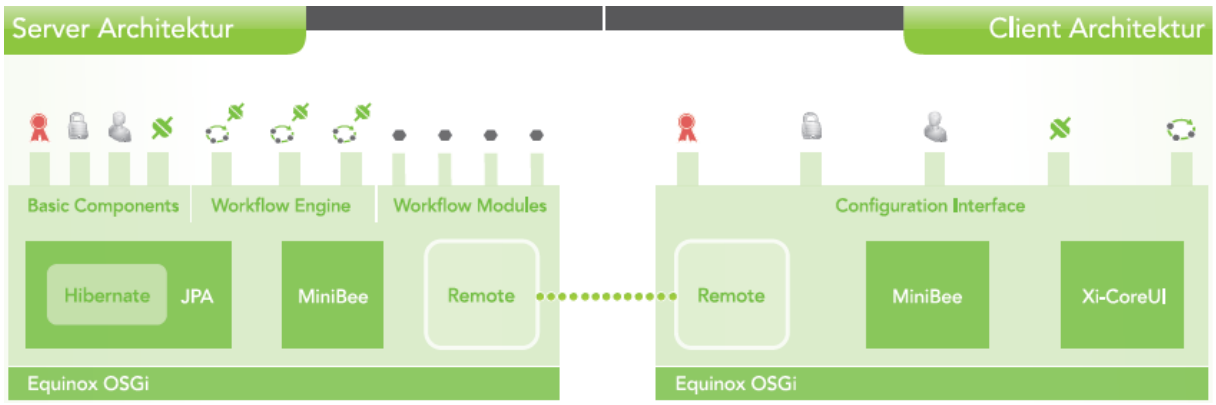



Abbildung 2.3: Architektur des XBS E3 Servers
(Marketingfolien der Firma XiTrust Secure Technologies GmbH)

Die verwendeten Technologien, die in der E3 Architektur Verwendung finden, sind:

- **Java**
  Obwohl nicht in der Abbildung 2.3 eingezeichnet, bildet Java die Grundlage aller Komponenten. Verwendet wird Version 6 der Java Plattform.

- **OSGi**
  Die Equinox Implementierung der OSGi R4 Spezifikation[5] bildet die Basis der Architektur. Mit Equinox konnte ein Framework gefunden werden, das die Anforderung der freien Kombinierbarkeit von Bundles zu verschiedenen Produkten ermöglicht.

- **Java Persistence API (JPA)**
  Die JPA ist eine Schnittstelle zur Abbildung von Java Objekten auf Datenbankeinträgen.

[5] http://www.osgi.org/Release4/HomePage

Hibernate[6] ist eine Implementierung dieser Schnittstelle und wird im XBS E3 in der Version 3.3 verwendet. Zum Basisumfang des XBS gehört die relationale Datenbank HyperSQL Database Engine[7] in der Version 1.8.

- **Enterprise Java Beans (EJB)**
  Mit EJB 3 wird Java ein Framework für die Entwicklung serverseitiger Komponenten zur Verfügung gestellt. Das Framework bietet Funktionalität wie etwa Multi-Threading, Connection Pooling oder State Management (vgl. EJB 3.0 Expert Group 2006 S. 29 online). Mit *MiniBee* wurde eine leichtgewichtige Implementierung der Spezifikation entwickelt. Diese wird im Unterkapitel 2.2.2 noch genauer vorgestellt.

- **Xi-CoreUI & Configuration Interface**
  Diese Komponente der XBS Architektur stellt den Kern und damit das Grundgerüst des Konfigurations-Client dar. Der Kern bietet eine Schnittstelle für Verwaltungsmodule. Beispiele solcher sind Benutzerverwaltung oder Diensteverwaltung. Das Xi-CoreUI Modul stellt die angedockten Verwaltungsmodule in einer Baumstruktur dar.

- **Basic Components**
  Zu den Basic Components des E3 Servers zählen Bundles, die in allen Produkten vorhanden sind. Das markanteste Beispiel einer Basic Component stellt die Benutzerverwaltung (MiniAas) dar. Viele Basic Components sind Teil des Kerns der im nächsten Unterkapitel vorgestellt wird.

- **Workflow engine & Workflow Modules**
  Die Workflow Engine stellt ein Prozessverarbeitunssystem dar, das aus Workflow Modulen aufgebaut ist. Das System modelliert einen endlichen Automaten, dessen Zustände Befehle ausführen. Diese Befehle werden von den Workflow Modulen zur Verfügung gestellt. Über den Konfigurations-Client kann die Zustandsmaschine konfiguriert werden.

### 2.2.2 Kernkomponenten

In diesem Unterkapitel werden die Kernkomponenten des XBS E3 vorgestellt. Die Abbildung 2.4 zeigt die Kernkomponenten und deren Abhängigkeiten.

XBS E3 Server Bundles können aus mehreren Plugins bestehen. Die gängige Variante ist Server-, Remote-, Client- und Test-Plugin. Das Remote-Plugin stellt gemeinsam verwendete Komponenten zur Verfügung, Server- und Remote-Plugins die jeweils nur einseitig verwendeten. Das Test-Plugin beinhaltet die Unit-Tests des jeweiligen Bundles. Die Komponenten (Bundles) des E3 Servers im Detail:

- **Service**

[6] http://www.hibernate.org
[7] http://www.hsqldb.org

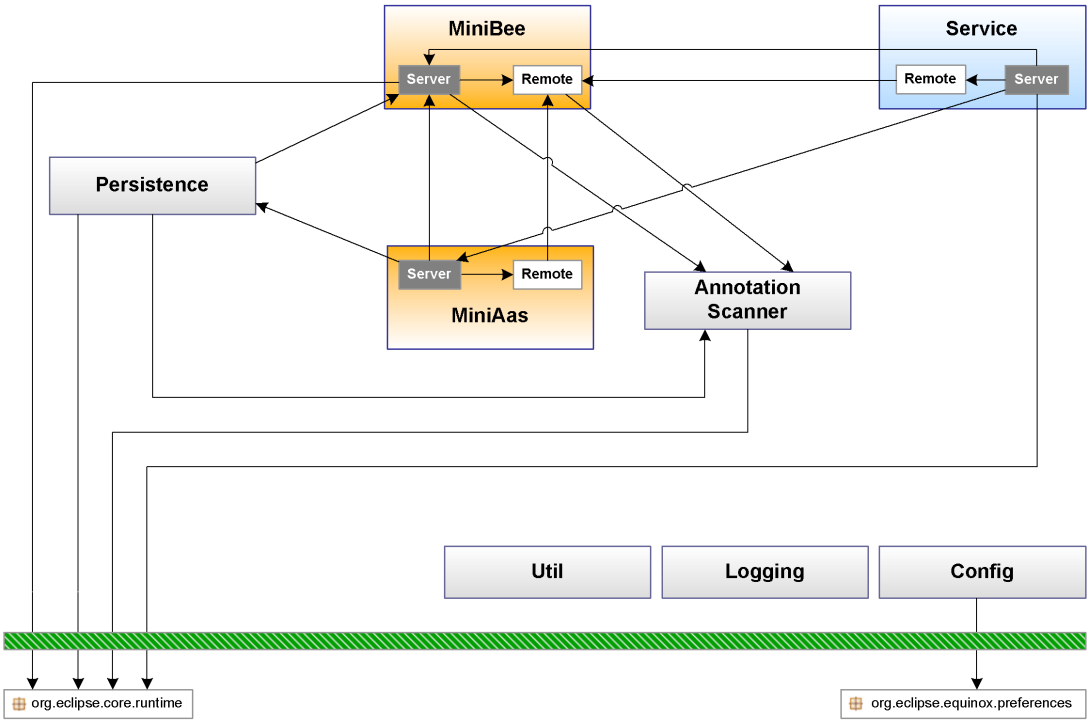



Abbildung 2.4: Kernkomponenten und Abhängigkeiten des XBS E3 Servers

Das Server Plugin des Service Bundle beherbergt die `Application` Klasse des E3 Servers. Diese stellt den Einstiegspunkt der Applikation bei der Ausführung dar. Der Name des Bundles beruht auf der Tatsache, dass der XBS als Dienst des Betriebssystems ausgeführt wird. Zusätzlich bietet das Service Plugin einen Extension Point für XBS Services. Diese sind Tasks die im "Hintergrund" des XBS meist zyklische Aufgaben erledigen.

- **MiniBee**
  Das MiniBee Bundle repräsentiert eine leichtgewichtige Implementierung von Teilen der EJB 3 Spezifikation. Im Vorfeld der E3 Server Entwicklung im Jahr 2007 wurde nach einer Möglichkeit gesucht, EJB 3 Container und Equinox gemeinsam zu verwenden. Zu diesem Zeitpunkt gab es noch keinen Application Server der dazu in der Lage war. MiniBee macht die gemeinsame Verwendung der beiden Technologien möglich. Bei der Entwicklung wurde die Verwendung eines Application Servers zu einem späteren Zeitpunkt vorgesehen und die entsprechenden Kompatibilitätsanforderungen eingehalten.

- **Annotation Scanner**
  Dieses Bundle besteht aus nur einem Plugin und bietet die Möglichkeit annotierte Klassen zu finden. Diese Funktionalität wird zur Registrierung der Beans vom MiniBee Bundle und der Entities vom Persistence Bundle benötigt.

- **MiniAas**
  Das MiniAas Bundle stellt die Implementierung eines Role-based Access Systems nach EJB 3 Spezifikation dar. Der Zugriff auf Beans kann durch eine Annotation, die die benötigte Rolle des Benutzers festgelegt, eingeschränkt werden. Das Bundle bietet zusätzlich die Möglichkeiten der Benutzerverwaltung, Autorisierung und Authentifizierung. Da vie-

le Bundles auf diese Funktionen zurückgreifen, stellt das MiniAas Bundle eine zentrale Komponente des E3 Servers dar.

- **Persistence**
  Dieses Bundle stellt einen Persistance Manager für MiniBee zur Verfügung. Er wurde bewusst von MiniBee herausgelöst um die beiden Bundles auch getrennt voneinander verwenden zu können. Für das Object-Relational Mapping wurde das Hibernate Framework in Version 3.3 gewählt.

- **Logging**
  Das Logging Bundle dient als Provider für die Simple Logging Facade for Java (SLF4J) API[8] in der Version 1.5.6. Das Bundle exportiert die Java Pakete der API und stellt sie anderen Bundles global zur Verfügung.

- **Util**
  Dieses Bundle stellt oft verwendete Funktionalität global zur Verfügung. Darunter fällt zum Beispiel das Einlesen von Files, das Ausführen von externen Prozessen oder das Vergleichen von Objekten auf Basis von Hash-Werten.

- **Config**
  Das Config Bundle stellt eine Schnittstelle für hierarchische Konfiguration bereit. Durch verschiedene Implementierungen können Schlüssel-Wert Paare der einzelnen Konfigurationen diverser Bundles in Dateien oder Tabellen einer Datenbank persistiert werden.

Der Aufbau des Konfigurations-Client sieht dem des Servers aus Abbildung 2.4 sehr ähnlich. Die Abbildung 2.5 zeigt eine Konfiguration, die ausschließlich zur Benutzerverwaltung gedacht ist. Diese wird vom MiniAas Bundle (UICore Plugin) zur Verfügung gestellt. Wird dieses Bundle weggelassen, bleibt die Minimalkonfiguration, die nur mehr in der Lage ist eine Verbindung zu einer XBS E3 Server Instanz herzustellen. Der Einstiegspunkt bildet das UICore Bundle, das eine RCP Applikation darstellt.

### 2.2.3 Analyse

In diesem Unterkapitel werden Konsequenzen und mögliche Gefahren der E3 Server Architektur analysiert. Eclipse Equinox auf Basis der OSGi Service Plattform Spezifikation bringt einige wesentliche Vorteile mit sich:

- **Modularisierung**
  OSGi strukturiert eine Java Anwendung in Module um Erweiterbarkeit und Wartbarkeit

[8] http://slf4j.org/

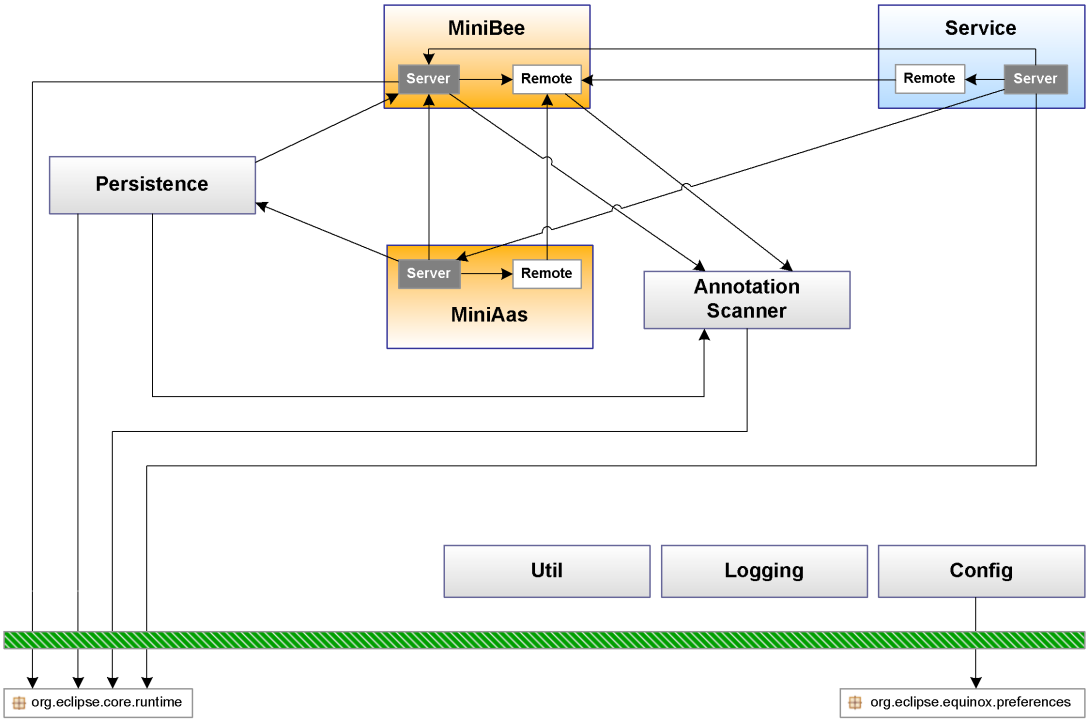



Abbildung 2.5: Aufbau des XBS E3 Clients

zu erhöhen. Die Module (Bundles) beinhalten interne Implementierungen und öffentliche Schnittstellen. OSGi bietet eine Implementierung eines Modulsystems, das den Austausch von Bundles zur Laufzeit ermöglicht (vgl. Wütherich *et al.* 2008 S.16f).

- **Abhängigkeitsmanagement**
  Die Abhängigkeiten zwischen den Bundles müssen explizit festgelegt werden. Die Freigabe erfolgt auf Paket-Ebene. Das OSGi Framework stellt sicher, dass nicht exportierte Pakete keinen Falls von anderen Bundles verwendet werden können. Zusätzlich können Bundles versioniert werden und es ist möglich, dass verschiedene Versionen des gleichen Bundles gleichzeitig in einer Applikation installiert sind (vgl. Wütherich *et al.* 2008 S. 17f).

- **Hot Deployment**
  Das OSGi Framework ermöglicht den Austausch von Bundles und Services zur Laufzeit. Dies stellt besonders für hochverfügbare Applikationen einen entscheidenden Vorteil dar (vgl. Wütherich *et al.* 2008 S. 18).

Bringt der hohe Grad der Modularisierung auf der einen Seite die erforderliche Möglichkeit der Kombination von Bundles zu Produkten, erhöht sie auf der anderen Seite die Komplexität des Systems enorm. Gerhard Fliess verdeutlichte diesen Umstand zuletzt auf der BASTA!2009 SPRING[9] in einem Vortrag über verteilte Architekturen.

[9] http://it-republik.de/konferenzen/bastaspring/

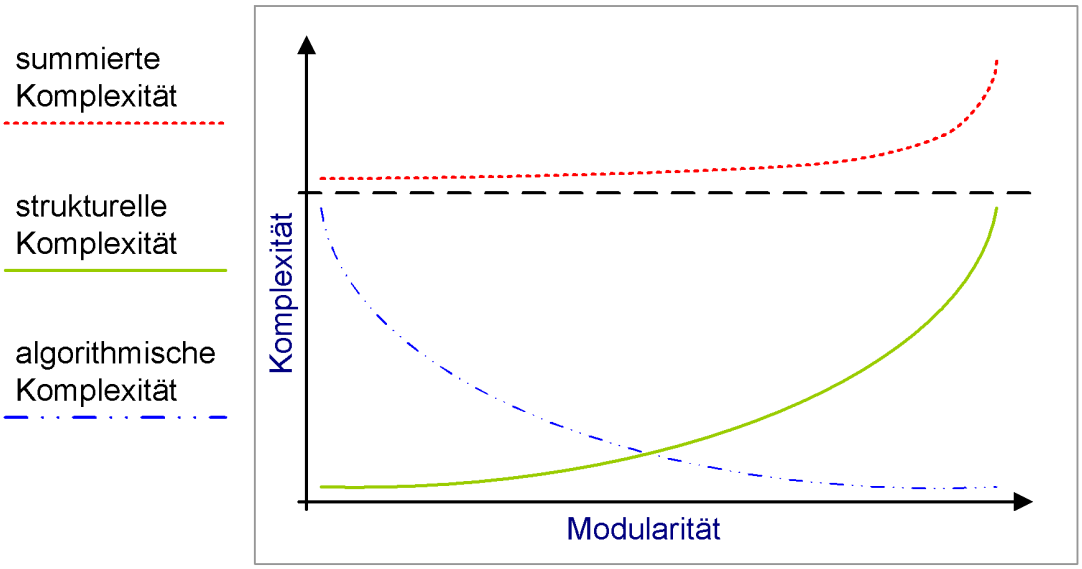



Abbildung 2.6: Modularität versus Komplexität
(in Anlehnung an Fliess 2009 S. 5)

Die Abbildung 2.6 zeigt, wie mit steigender Modularität die algorithmische Komplexität sinkt, in Gegenzug aber die strukturelle Komplexität überproportional ansteigt[10].

## 2.3 Fazit und Motivation dieser Arbeit

Eines der größten Probleme des XBS E3 Servers als Produkt liegt nicht in seiner Architektur sondern viel mehr in der Art und Weise, wie er zum Einsatz kommt. Wie bereits in Unterkapitel 2.1 erwähnt, kommt der XBS häufig zentral platziert in Unternehmen zum Einsatz und erledigt dort diverse Aufgaben im Hintergrund. Dieser Umstand ermöglicht auf der einen Seite eine einfache Integration in bestehende IT Umgebungen von Unternehmen, benötigt jedoch auf der anderen Seite wenig Interaktion mit den Anwendern, was zu einer geringen Wahrnehmung der Leistung des Produkts führt.

Ein plakatives Beispiel für den Einsatz des XBS E3, das die beschriebene Problematik verdeutlicht, ist der E3 im Einsatz als Signature-Server. Er übernimmt in diesem Szenario die digitale Signatur beispielsweise von E-Mails oder Portable Document Format (PDF) Dateien im Anhang einer E-Mail. Der XBS fungiert als Proxy, der die E-Mails und Anhänge vor dem Verlassen des Unternehmens signiert. Die Arbeit des E-Mail-Versenders ändert sich dadurch nicht. Er selbst bekommt nur schwer mit, dass er digital signierte E-Mails versendet.

Dieser Effekt ist sehr wohl erwünscht, es fehlt jedoch die Möglichkeit der statistischen Auswertung über die Leistung die der XBS E3 vollbringt, im oben beschriebenen Szenario etwa die

[10] Die zitierte Präsentation kann leider nicht von der BASTA Homepage abgerufen werden. Sie liegt dem Autor in digitaler Form vor und kann auf Anfrage von ihm bezogen werden.

Anzahl der E-Mails, die pro Tag und Mitarbeiter versendet werden. Die sich ergebende fehlende Sichtbarkeit des XBS E3 Servers bringt einige Nachteile mit sich:

- **Erschwertes Monitoring**
  Statistische Auswertungen über Leistungen des E3 Servers sind nur über Auswertung entsprechender Log Dateien möglich. Diese Arbeit ist mühsam und nur von Administratoren oder Entwicklern zu bewältigen.
- **Erschwerte Wartung**
  Da Log Dateien unter Umständen sehr groß sein können, gestaltet sich deren Auswertung zum Auffinden von Problemen oft als sehr mühsam. Die Wartbarkeit des Systems leidet unter diesem Umstand.
- **Fehlendes Feedback für Entwickler**
  Die Auswirkung von Updates in verschiedenen Bereichen wie etwa Performance Verbesserungen bleiben für Entwickler nur schwer nachvollziehbar.
- **Fehlendes Feedback für Anwender**
  Für den Kunden sind Auswirkungen von Updates noch viel schwerer ersichtlich.

Ein entsprechendes Monitoring-System für den XBS E3, das in der Lage wäre die Leistung des XBS zu visualisieren, würden diese Probleme beheben können. Im Rahmen seiner Masterarbeit entwickelte DI Helmut Aschbacher ein Servicekonzept für die Firma XiTrust. Dies ergab ebenso die Notwendigkeit der Entwicklung eines XBS Monitoring-Systems, aus dem sich für (Aschbacher 2009) drei wesentliche Vorteile ergeben:

1. Ein XBS Monitoring-System stellt dem Kunden ein internes Argumentationsinstrument durch statistische Auswertung der Leistung des XBS zur Verfügung. Gleichzeitig soll dadurch das Supportaufkommen seitens der Firma XiTrust verringert werden können (vgl. Aschbacher 2009 S. 113)
2. Die Visualisierung der Leistung des XBS bringt einen Mehrwert für Kunde und Entwickler durch Reports im PDF Format (vgl. Aschbacher 2009 S. 113)
3. Durch ein geeignetes Monitoring-Tool soll dem Kunden eine gewisse Sicherheit bezüglich der korrekten Arbeitsweise des XBS gegeben werden. Gleichzeitig ergeben sich für die Firma XiTrust die Möglichkeit Updates und Erweiterungen dem Kunden klarer kommunizieren zu können (vgl. Aschbacher 2009 S. 113)

Diese Vorteile untermauern zusätzlich die Analyse des E3 Servers und des daraus gezogenen Fazits: Ein Monitoring-System für den XBS E3 bringt für Kunde und Entwickler enorme Vorteile und eine Qualitätssteigerung des Produkts. Das Monitoring-System soll Daten des Systems sammeln können und auf Basis dieser Daten Statistiken erstellen können. Es ergeben sich sechs konkrete Anforderungen für das Projekt:

1. **Datenakkumulation**
   Es soll ein Konzept für die Datenakkumulation erstellt werden. Der Schwerpunkt liegt darin, zu erarbeiten, wie das Monitoring-System an die Daten der einzelnen Bundles gelangt. Hierfür sollen Architekturen gängiger Monitoring-Systeme analysiert werden, deren Vor- und Nachteile begutachtet und auf dieser Basis ein Monitoring-System entworfen werden. Zusätzlich soll eine Strategie für die Persistierung der gewonnenen Daten erarbeitet werden.

2. **Visualisierung der Daten**
   Die gewonnen Daten müssen in geeigneter Form wiedergegeben werden können. Zu diesem Zweck soll eine Visualisierungsengine die Darstellung von statistischen Auswertungen ermöglichen. Auswertungen sollen leicht erstellbar sein und in den Konfigurations-Client des XBS E3 Servers integriert werden können.

3. **Datenschnittstelle**
   Zwischen Visualisierungsengine und Datenakkumulation muss eine geeignete Schnittstelle entworfen werden. Sowohl XBS E3 als auch das Tool zur Erzeugung von statistischen Auswertungen müssen in der Lage sein Daten des Servers abzufragen.

4. **Implementierung**
   Im Anschluss an die Erstellung der in Punkt 1-3 genannten Konzepte sollen diese implementiert werden. Diese Implementierung soll die in Punkt 5 und 6 geforderten Reports erzeugen und darstellen können.

5. **Darstellung des Verlaufs eines Systemzustandes**
   Ein Ziel des Projekts ist die Darstellung eines Reports, der den aktuellen Systemzustand des E3 Servers darstellt.

6. **Darstellung des Verlaufs von internen Prozessdaten**
   Zum Schluss soll ein weiterer Report mit einer statistische Auswertung von Teilen der Leistung des XBS E3 Servers erstellt werden.

In einer weiteren Phase des Servicekonzepts sieht (Aschbacher 2009) die Entwicklung eines Benachrichtiungssystems vor, das den Anwender beispielsweise über auslaufende Zertifikate informiert (vgl. Aschbacher 2009 S. 113). Die entsprechenden Vorkehrungen für die Erweiterung des Monitoring-Systems um ein solches Benachrichtigungssystem sollen im Konzept berücksichtigt werden.

# 3 Logging und Monitoring

Im Kapitel 2 wurde die Notwendigkeit eines XBS Monitoring-Systems herausgearbeitet. Basis für die Konzeption dieses Systems sollen die nachfolgenden Analysen bilden. Zu Beginn werden Software-Architekturen untersucht und deren Vor- und Nachteile ermittelt. Im Anschluss werden in den Monitoring-Kontext fallende Standards und Protokolle analysiert. Den Abschluss bildet die Analyse etablierter Monitoring-Systeme. Deren Vor- und Nachteile sollen die Tauglichkeit der verwendeten Architekturen für das XBS Monitoring-System prüfen.

## 3.1 Architekuren

*In Softwareprojekten nimmt Architektur eine der zentralen Rollen ein. Sie fungiert als Brücke zwischen Analyse und Implementierung. (Starke 2008 S. 29).*

Diese Definition von Gernot Starke verdeutlicht treffend den Zweck dieses Unterkapitels. Die vorgestellten Architekturen bilden den Grundstein für den Entwurf des Monitoring-Systems für den XBS auf Basis der Analysen des bestehenden Systems, der Anforderungen, von verbreiteten Monitoring-Systemen sowie von Standards und Protokollen.

Im Unterkapitel 3.1.1 werden vorweg die Vielzahl der verschiedenen Architekturen kategorisiert und in weiteren Unterkapiteln Vertreter der Kategorien vorgestellt.

### 3.1.1 Kategorisierung

Die Auswahl der Architekturen basiert auf den verschiedenen Sichten auf Software-Architektur basierend auf (Zdun & Avgeriou 2005). In dieser Publikation von Paris Avgeriou und Uwe Zdun werden acht Sichten (Views) unterschieden:

- **Layered View**
  Beschäftigt sich mit der Strukturierung von komplexen Strukturen in einzelne miteinander kommunizierende unabhängige Teile. Die Layers Architektur wird dieser Sicht zugeordnet und wird in Unterkapitel 3.1.2 vorgestellt.

- **Data Flow View**
  Beschäftigt sich mit der Verarbeitung von Datenströmen. Als Beispiel führen (Zdun & Avgeriou 2005) Batch Sequential und die Pipes & Filter Architektur, die in Unterkapitel 3.1.3 analysiert wird.

- **Data-centered View**
  Diese Sicht befasst sich mit dem Problem des gleichzeitigen Zugriffs mehrerer Komponenten auf zentrale Daten. Bekanntester Vertreter ist die Blackboard Architektur. Weitere Vertreter sind Shared Repository und Active Repository.

- **Adaption View**
  Thema dieser View ist die Adaption des Systems bei veränderten Anforderungen. Die von (Buschmann *et al.* 1998) beschriebene Microkernel Architektur wird in Unterkapitel 3.1.7 vorgestellt.

- **Language Extension View**
  Beschäftigt sich mit dem Thema der Portierbarkeit von Systemen. Als Beispiele führen (Zdun & Avgeriou 2005) Interpreter, Virtual Machine und Rule-Based System auf. Eine Implementierung der Virtual Machine Architektur ist die Java Virtual Machine (JVM) (vgl. Bass *et al.* 1998 S. 98). Durch die gegebene hohe Portierbarkeit von Java Applikationen kann dieser Aspekt bei der Entwicklung eines Monitoring-Systems für den XBS nachrangig behandelt werden.

- **User Interaction View**
  Benutzerinteraktion ist das Thema dieser Sichtweise auf Software-Architekturen. Bekannteste Vertreter, ist die weit verbreitete MVC-Architektur (Unterkapitel 3.1.6). Mit Presentation-Abstraction-Control erwähnen (Zdun & Avgeriou 2005) einen weiteren Vertreter der in (Buschmann *et al.* 1998) detailliert beschrieben ist (vgl. Buschmann *et al.* 1998 S. 145-169).

- **Component Interaction View**
  Durch die komponentenbasierte OSGi-Architektur des XBS wird besonderes Augenmerk auf die Architekturen dieser Sicht gelegt. Sie befassen sich mit Kommunikation und Abhängigkeiten zwischen Komponenten. Es werden in separaten Unterkapiteln Client-Server sowie Event-based System (Implicit Invocation) vorgestellt. Weitere Vertreter nach (Zdun & Avgeriou 2005) sind Explicit Invocation, Peer-To-Peer und Publish-Subscribe.

- **Distribution View**
  Beschäftigt sich mit verteilten Komponenten in vernetzten Systemen. Neben der in Kapitel 3.1.5 vorgestellten Broker Architektur zählen Remote Procedure Calls und Message Queueing zu dieser Sicht.

In der einschlägigen Literatur findet man häufig ähnliche Kategorisierungen. (Starke 2008) unterteilt Architekturen nach Aspekten. Beispiele solcher Aspekte sind Persistenz, Integration, Verteilung, Kommunikation oder Sicherheit (vgl. Hruschka & Starke 2009 online). Ähnlich

den Views von (Zdun & Avgeriou 2005) beschreiben Aspekte die Motivationen, die hinter den verschiedenen Architekturen stecken. Weitere Kategorisierungen bieten etwa (Shaw & Garlan 1996 S. 19f) oder (Bass *et al.* 1998 S. 94f).

### 3.1.2 Layers

Die Layers oder auch Schichten Architektur hilft Anwendungen zu strukturieren, indem Teilaufgaben auf verschiedenen Abstraktionsebenen in Gruppen zusammengefasst werden. Diese Gruppen bilden die Schichten der Architektur (vgl. Buschmann *et al.* 1998 S. 32). In der einschlägigen Literatur findet man zur Layers Architektur meist das OSI-Schichtenmodell als Beispiel.

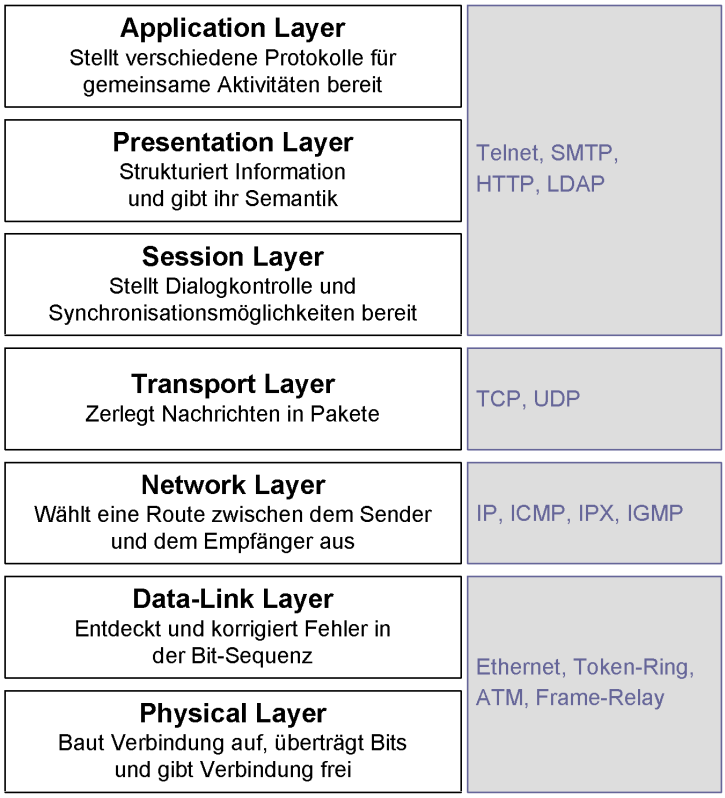



Abbildung 3.1: OSI-Schichtenmodell
(in Anlehnung an Tannenbaum 1997 S. 45)

Die Abbildung 3.1 zeigt das OSI-Schichtenmodell mit Beispielen von dazugehörigen Protokollen. Jede der sieben Schichten behandelt einen bestimmten Aspekt bei der Übertragung von Nachrichten über die Systemgrenzen hinweg (vgl. Tannenbaum 1997 S. 44-51). Dabei kann jede Schicht die Dienste der nächsttieferen Schichten verwenden (vgl. Buschmann *et al.* 1998 S. 32). Wichtig ist, dass die Abhängigkeiten zwischen den Schichten immer von oben nach unten verlaufen. Wird dieses Prinzip gebrochen, geht die Unabhängigkeit der Schichten verloren (vgl. Starke 2008 S. 146).

Ein bekanntes Beispiel, das gerne der Layers-Architektur zugeordnet wird, ist die "three tier

architecture" bestehend aus Präsentations-, Businesslogik- und Datenhaltungsschicht (siehe Abb. 3.2). Die Bezeichnungen Tiers und Layers werden oft fälschlich synonym verwendet. Tiers bezeichnen keine Abstraktionsebene, sondern die dem System zugrunde liegende Infrastruktur (vgl. Starke 2008 S. 148).

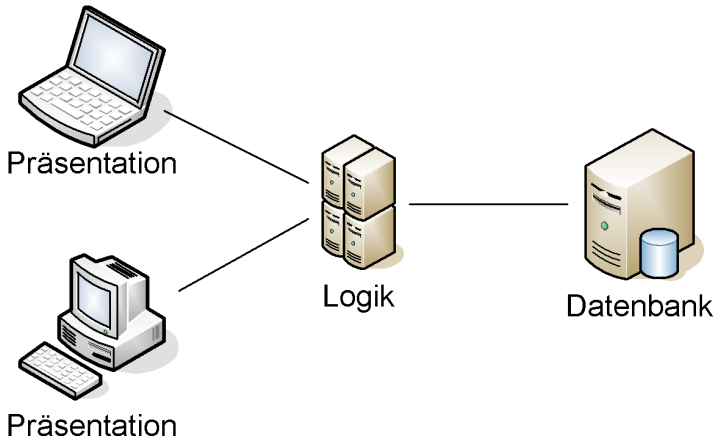



Abbildung 3.2: Three-Tier-Architecture
(in Anlehnung an Starke 2008 S. 148)

Weitere bekannte Einsatzmöglichkeiten der Layers-Architekur sind Virtuelle Maschinen, APIs oder Betriebssysteme.

**Vorteile:**

- Die einzelnen Schichten lassen sich durch ausreichende Abstraktion gut wiederverwenden (vgl. Buschmann *et al.* 1998 S. 50)
- Sind Schnittstellen zwischen den Schichten allgemein akzeptiert, können Implementierungen verschiedener Anbieter einfach ausgetauscht werden (vgl. Buschmann *et al.* 1998 S. 50)
- Standardisierte Schnittstellen zwischen den Schichten reduzieren die Auswirkungen von Code-Änderungen auf die Schicht der Änderung (vgl. Buschmann *et al.* 1998 S. 50f)
- Die Architektur bietet ein leicht verständliches Strukturkonzept (vgl. Starke 2008 S. 146)
- Abhängigkeiten zwischen Komponenten werden minimiert (vgl. Buschmann *et al.* 1998 S. 50)

**Nachteile:**

- Eine Veränderung des Verhaltens einer Schicht kann massive Konsequenzen für alle anderen Schichten herbeiführen (vgl. Buschmann *et al.* 1998 S. 51).
- Wenn eine Schicht stark von einer oder mehreren tieferen Schichten abhängig ist, müssen die Daten oft durch einige Schichten durchgereicht werden. Ebenso können diese

niedrigeren Schichten Arbeit vollbringen die von höheren Schichten nicht zwingend gebraucht wird. Diese beiden Umstände können die Effizienz des Systems mindern. Eine in der Literatur oft vorgestellte Lösung sind Layer-Bridges. Diese können die genannten Probleme teilweise lösen (vgl. Starke 2008 S. 146f). In (Buschmann *et al.* 1998) wird diese Lösung auch als die "Relaxed-Layered-System" Variante der Layers-Architektur bezeichnet (vgl. Buschmann *et al.* 1998 S. 47).

- Der Erfolg der Layers-Architektur ist stark von der nicht immer einfachen, Aufteilung der Aufgaben in den Schichten abhängig (vgl. Buschmann *et al.* 1998 S. 52f).

### 3.1.3 Pipes & Filter

Die Pipes & Filter Architektur kommt häufig bei der Verarbeitung von Datenströmen zum Einsatz. Bekannte Beispiele sind die Bearbeitung eines Bildes in mehreren Schritten oder das Kompilieren eines Programms (vgl. Starke 2008 S. 150).

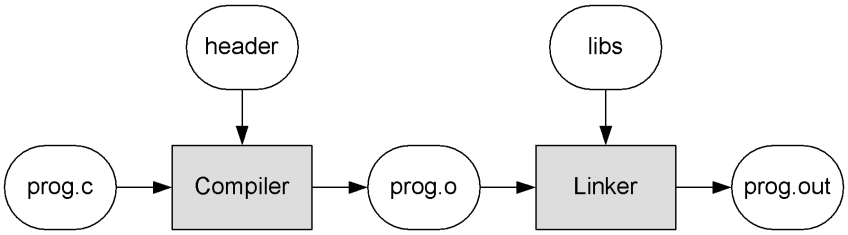



Abbildung 3.3: C-Compiler Steps
(basierend auf Schmaranz 2001 S. 10)

Die Abbildung 3.3 zeigt die Verarbeitungsschritte bei der Kompilierung eines C Programms (vgl. Schmaranz 2001 S. 10). Die abgerundeten Rechtecke symbolisieren den Inhalt der jeweiligen Pipe, die grau hinterlegten bilden die Filter des Systems.

- **Filter**
  Die datenverarbeitenden Teile des Systems werden als Filter bezeichnet. Diese verändern oder erweitern die eingehenden Daten. Die Aktivierung eines Filters kann durch Aktivierung der Vorgänger bzw. Nachfolgers in der Pipeline erfolgen (passiver Filter). Meistens ist der Filter jedoch in eine Schleife eingebunden, in der er ständig Daten von der eingehenden Pipe liest, bearbeitet und an die ausgehende Pipe weiter gibt (aktiver Filter). Jeder Filter hat eine eingehende und eine ausgehende Pipe (vgl. Buschmann *et al.* 1998 S. 56f).

- **Pipe**
  Pipes sind die daten-transportierenden Teile der Architektur und verbinden die Filter miteinander. Pipes entscheiden an, wen und welche Daten sie weitergeben (vgl. Starke 2008 S. 150).

Mit "Tee-and-Join-Pipeline-System" stellt (Buschmann *et al.* 1998) eine Variante der Architektur vor, bei der ein Filter mehrere eingehende und ausgehende Pipes besitzen kann. Dadurch bekommen Filter beispielsweise die Möglichkeit, Rückmeldungen an vorgelagerte Filter zu senden (vgl. Buschmann *et al.* 1998 S. 67f).

**Vorteile:**

- Durch verschiedene Kombinationsmöglichkeiten der Filter entsteht ein hoher Grad an Flexibilität und Wiederverwendbarkeit (vgl. Buschmann *et al.* 1998 S. 69)
- Bei der Verarbeitung des Datenstroms kann auf temporäre Dateien verzichtet werden. Der Einsatz ist aber dennoch möglich (vgl. Buschmann *et al.* 1998 S. 69)
- Durch paralleles Abarbeitung der Filter kann die Effizienz der Pipeline gesteigert werden (vgl. Buschmann *et al.* 1998 S. 70)

**Nachteile:**

- Die Fehlerbehandlung stellt sich, durch den Umstand, dass sich die Filter gegenseitig nicht kennen, als äußert komplex dar. Aus diesem Grund empfiehlt (Buschmann *et al.* 1998) die Layers-Architektur für den Einsatz, wenn verlässliche Operationen gefordert sind (vgl. Buschmann *et al.* 1998 S. 71).
- Der Zustand eines Filters ist unbekannt und muss als Teil der Daten einer Pipe mit übertragen werden (vgl. Starke 2008 S. 151)

(Shaw & Garlan 1996) beschreibt mit der Batch Sequential Architektur eine Alternative zu Pipes & Filter, bei der eigenständige Programme auf gemeinsamen Daten operieren. Dabei liest ein solches die Daten einer Datei und bearbeitet diese. Ist das Programm fertig beginnt das nächste mit der Arbeit. So etnsteht ein sequentieller (Batch) Vorgang (vgl. Shaw & Garlan 1996 S. 21).

### 3.1.4 Blackboard

Der Name Blackboard Architektur beschreibt eine Situation gemeinsamen Arbeitens verschiedener Experten an einem Problem an einer gemeinsamen Tafel. Die Experten des Systems sind voneinander unabhängige Programme mit Lösungen zu Teilen der Gesamtaufgabe. Der Zustand des Systems ist nicht von der Reihenfolge des Aufrufs der Experten abhängig, sondern vom Zustand der gemeinsamen Datenstruktur, der durch eine gemeinsame Kontrollkomponente ermittelt wird. Die Kombination der verschiedenen Experten und deren zeitliche Ablaufsteuerung wird meist durch einen Moderator geregelt (vgl. Buschmann *et al.* 1998 S. 75).

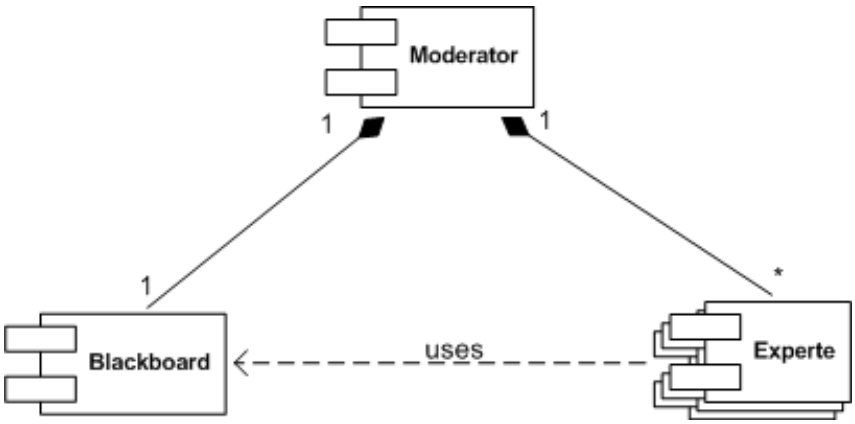



Abbildung 3.4: Blackboard Architektur
(in Anlehnung an Buschmann *et al.* 1998 S. 80)

Die Abbildung 3.4 zeigt die drei Komponenten der Blackboard Architektur. Das Blackboard beinhaltet die gemeinsame Datenstruktur sowie Teillösungen der Gesamtaufgabe. Die Experten werden vom Moderator in Abhängigkeit des Status des Gesamtfortschritts der Problemlösung aktiviert (vgl. Buschmann *et al.* 1998 S. 76). Im Prozess der Lösungsfindung entstehen Teillösungen, die von anderen Experten weiterverarbeitet, miteinander kombiniert oder verworfen werden können. Im Kontext der Blackboard Architektur wird das zu lösende Problem meist in Teilprobleme zerteilt, die sich auf unterschiedliche Fachgebiete verteilen und deren Lösung unterschiedliche Repräsentationsformen der Daten benötigt (vgl. Buschmann *et al.* 1998 S. 74). Meist besteht keine von vorherein festgelegte Strategie, wie die Experten gemeinsam zu einer Lösung des Problems gelangen (vgl. Buschmann *et al.* 1998 S. 73). Anwendungsgebiete sind Bild- und Spracherkennung , Bildverarbeitung und Systemüberwachung (vgl. Buschmann *et al.* 1998 S. 88ff).

Mit Repository beschreibt (Buschmann *et al.* 1998) eine, für den Kontext dieser Arbeit interessante, Variante der Blackboard Architektur. In dieser verallgemeinerten Variante wird die gemeinsame Datenstruktur als Repository bezeichnet. Nicht mehr der Zustand der Datenstruktur ist für die zeitliche Steuerung der Experten verantwortlich, sondern die Experten entscheiden selbst, wann sie die Daten des Repositories benutzen (vgl. Buschmann *et al.* 1998 S. 87f). Ein Monitoring-System, das die Zustände verschiedener Netzwerkkomponenten in eine Datenbank schreibt und deren Daten anschließend von einem Reporting Modul ausgewertet werden, bildet ein Beispiel dieser Variante.

**Vorteile:**

- Die Architektur ermöglicht das Experimentieren mit verschiedenen Lösungsstrategien (vgl. Buschmann *et al.* 1998 S. 93)
- Das System ist bedingt durch die Unabhängigkeit der Experten leicht zu erweitern (vgl. Buschmann *et al.* 1998 S. 93f)
- Die Experten selbst können gut wiederverwendet werden, sofern die Datenstruktur des

Blackboards gleich bleibt (vgl. Buschmann *et al.* 1998 S. 94)

- Da nur Hypothesen auf dem Blackboard bleiben, die von anderen Hypothesen weiterverarbeitet werden können entsteht eine Toleranz gegenüber verrauschten Daten (vgl. Buschmann *et al.* 1998 S. 94)

**Nachteile:**

- Das Testen des Systems ist durch die schwer reproduzierbaren Ergebnisse nur schwer möglich (vgl. Buschmann *et al.* 1998 S. 95)
- Die Systeme können nur einen geringen Anteil der gestellten Probleme lösen (vgl. Buschmann *et al.* 1998 S. 95)
- Die Definition der gemeinsamen Kontrolllogik ist schwierig (vgl. Buschmann *et al.* 1998 S. 95)
- Die Effizienz des Systems leidet durch den geringen Prozentsatz von Lösungen (vgl. Buschmann *et al.* 1998 S. 95)
- Keine Unterstützung paralleler Verarbeitung (vgl. Buschmann *et al.* 1998 S. 95)
- Meist hoher Entwicklungsaufwand (vgl. Buschmann *et al.* 1998 S. 95)

Der Einsatz der Blackboard Architektur und der damit verbundene hohe Entwicklungsaufwand lässt sich meist nur durch das Fehlen von deterministischen gesamtheitlichen Lösungsstrategien oder Probleme aus unausgereiften Anwendungsgebieten, in denen experimentieren hilfreich ist, rechtfertigen (vgl. Buschmann *et al.* 1998 S. 95f). Ein solches Szenario ist das HEARSAY-II[1] System, das von (Buschmann *et al.* 1998) im Zuge des Blackboard Architektur Kapitels genauer beleuchtet wird (vgl. Buschmann *et al.* 1998 S. 88f).

### 3.1.5 Broker

Die Broker Architektur kommt bei der Strukturierung verteilter Systeme zum Einsatz. Dabei interagieren entkoppelte und voneinander unabhängige Komponenten miteinander (vgl. Buschmann *et al.* 1998 S. 99). Die Einführung der Interprozesskommunikation in ein System voneinander unabhängiger Komponenten bringt Abhängigkeiten mit sich. Die Broker Architektur bildet eine Möglichkeit, miteinander kommunizierende Komponenten trotzdem voneinander unabhängig zu halten. Die Kommunikation findet nicht mehr direkt zwischen zwei Komponenten statt, sondern wird von einem Vermittler (Broker) gesteuert (vgl. Buschmann *et al.* 1998 S. 100). Dies führt dazu, dass keine Abhängigkeiten zwischen den Komponenten entstehen. Wird ein Broker zur Entkoppelung in ein Client-Server System eingeführt, übernimmt er die Aufgaben der

[1] Spracherkennungssystem der Advanced Research Projects Agency (ARPA)

Auffindung des Servers, Weiterleitung der Dienstaufforderung an den Server und Übermittlung des Ergebnisses an den Client. Der Broker kann das System flexibler machen, indem er die Möglichkeit vorsieht, Komponenten zur Laufzeit hinzuzufügen, zu verändern oder zu entfernen (vgl. Buschmann *et al.* 1998 S. 101f).

Beispiele für Implementierungen der Broker Architektur sind CORBA[2], J2EE[3] EJB[4] oder Java RMI[5] .

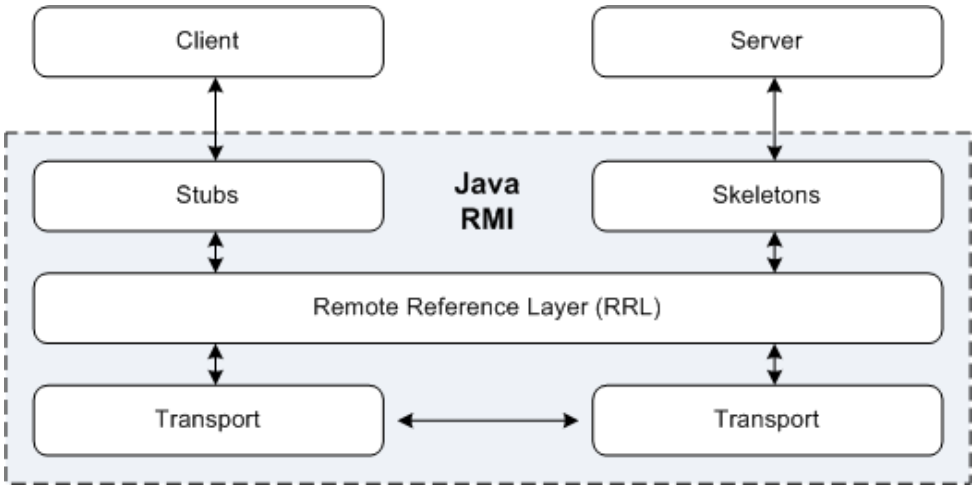



Abbildung 3.5: Java RMI Architektur
(in Anlehnung an Jia & Zhou 2005 S. 164)

Die Abbildung 3.5 zeigt die Java RMI Architektur[6]. Mit RMI führte Java eine eigene Möglichkeit des Remote Prozeduraufrufs ein. Dazu registriert der Server ein Remote Objekt mit eindeutigen Namen bei der RMI-Registry. Der Client sieht bei der RMI-Registry (RRL) unter dem eindeutigen Namen nach und bekommt eine Objektreferenz. Diese kann er verwenden um Methoden aufzurufen. Der Methodenaufruf wird vom RRL (Broker) an den Server vermittelt (Zuordnung wird von Stubs bzw. Skeleton vorgenommen) und das Ergebnis wieder an den Client zurückgesendet (vgl. Sun Microsystems 2006a online).

**Vorteile:**

- Die Clients müssen den Serverstandort nicht mehr kennen. Diese Aufgabe wird vom Vermittler übernommen (vgl. Buschmann *et al.* 1998 S. 119)
- Durch erreichte Unabhängigkeit der Komponenten ergibt sich eine hohe Erweiterbarkeit und Änderbarkeit (vgl. Buschmann *et al.* 1998 S. 119)
- Da nur mehr die Broker Komponente portiert werden muss steigt die Portierbarkeit des

[2]Common Object Request Broker Architecture: http://www.omg.org/gettingstarted/corbafaq.htm
[3]Java Platform, Enterprise Edition: http://java.sun.com/javaee/technologies/javaee5.jsp
[4]Enterprise Java Beans: http://java.sun.com/products/ejb/
[5]http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp
[6]Stammt aus Java Version 1.2. Ab Version 1.5 werden Stubs und Skeletons nicht mehr benötigt.

ganzen Systems (vgl. Buschmann *et al.* 1998 S. 119)

- Verwenden mehrere Broker Systeme das gleiche Protokoll zur Nachrichtenübertragung, können diese miteinander operieren (vgl. Buschmann *et al.* 1998 S. 120)
- Gute Wiederverwendbarkeit bereits implementierter Dienste. Diese beschränkt sich aber meist auf Komponenten des gleichen Systems (vgl. Buschmann *et al.* 1998 S. 120)

**Nachteile:**

- Systeme mit einer direkten Abhängigkeit zum Kommunikationsmechanismus oder fester Komponentenverteilung erreichen meist eine höhere Performance (vgl. Buschmann *et al.* 1998 S. 120)
- Die Fehlertoleranz des Systems sinkt durch gemeinsame Abhängigkeiten zum Vermittler (vgl. Buschmann *et al.* 1998 S. 120)
- Schwieriges Testen und Debuggen durch die hohe Anzahl beteiligter Komponenten (vgl. Buschmann *et al.* 1998 S. 120)

(Buschmann *et al.* 1998) bietet eine Reihe von Varianten an, in denen die Broker Architektur erweitert oder in veränderter Form zum Einsatz kommt (vgl. Buschmann *et al.* 1998 S. 117f).

### 3.1.6 Modell-View-Controller

Die Modell-View-Controller (MVC) Architektur wurde von Trygve M.H. Reenskaug in den Jahren 1978-1979 erfunden und in Smalltalk eingeführt (vgl. Reenskaug 1979 online). MVC unterteilt das interaktive System in die drei Komponenten Modell, View und Controller. Die Abbildung 3.6 zeigt diese drei Komponenten.

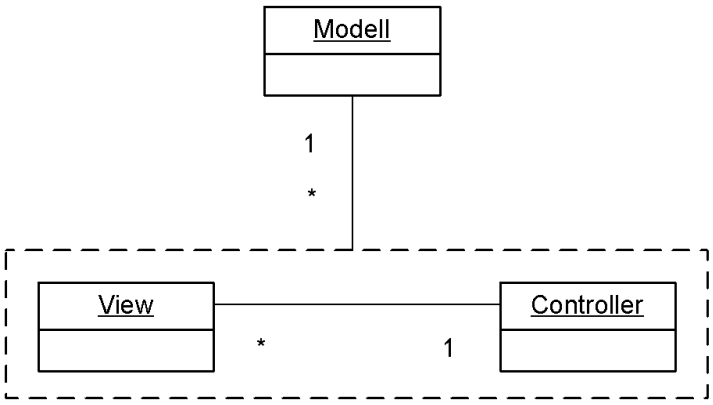



Abbildung 3.6: Modell-View-Controller
(in Anlehnung an Reenskaug 1979 online)

- **Modell**
  Das Modell repräsentiert Daten und bietet Operationen zu deren Bearbeitung und Zugriff (vgl. Reenskaug 1979 online).

- **View**
  Einem Modell werden ein oder mehrere Views zu dessen Darstellung angehängt. Dabei gibt es mehrere Views für verschiedene Darstellungen der Daten des Modells (vgl. Reenskaug 1979 online). (Buschmann *et al.* 1998) sieht für jedes View ein Aktualisierungsoperation vor, die vom Benachrichtigungssystem über Änderungen aktiviert wird. (vgl. Buschmann *et al.* 1998 S. 126f).

- **Controller**
  Der Controller stellt das Interface zwischen Benutzer und View dar. Es bietet Interaktionsmöglichkeiten beispielsweise in Form von Menüs oder Tastatureingabe.

Anwendungsgebiete von MVC sind das UI-Framework von Smalltalk, die Microsoft Foundation Class Library (MFC) oder das Anwendungsframework ET++ (vgl. Buschmann *et al.* 1998 S. 144). Moderne Webanwendungen sind interaktive Systeme, die immer mehr auf der MVC Architektur aufbauen. Beispielsweise bauen die meisten großen PHP Application Frameworks wie etwa Zend[7], Codeigniter[8] oder Zoop[9] auf MVC. Andere erfolgreiche MVC Implementierungen sind Ruby on Rails[10] oder Struts[11].

**Vorteile:**

- Strikte Trennung zwischen Modell und Bedienoberfläche (vgl. Buschmann *et al.* 1998 S. 142)
- Gute Ausstauschbarkeit von Views und Controllern durch die strikte Trennung der Komponenten (vgl. Buschmann *et al.* 1998 S. 142)
- Hohes Potenzial für Frameworks (vgl. Buschmann *et al.* 1998 S. 142)

**Nachteile:**

- Die strikte Trennung von Modell, View und Controller kann die Komplexität des Systems erhöhen (vgl. Buschmann *et al.* 1998 S. 142)
- Controller und Views sind eng gekoppelt. Die Wiederverwendung der einzelnen Komponenten kann dadurch leiden (vgl. Buschmann *et al.* 1998 S. 142f)

[7] http://framework.zend.com
[8] http://codeigniter.com
[9] http://zoopframework.com
[10] http://rubyonrails.org/
[11] http://struts.apache.org/

- Durch den direkten Aufruf der Operationen eines Modells durch View und Controller entsteht zwischen ihnen eine enge Kopplung (vgl. Buschmann *et al.* 1998 S. 143)
- Ineffizienter Datenzugriff auf das Modell, sollte ein View mehrere verschiedene Operationen des Modells benötigen um alle benötigten Daten zur Darstellung zu gewinnen (vgl. Buschmann *et al.* 1998 S. 143)
- Bei der Portierung eines MVC Systems müssen unter Umständen plattformabhängiger und plattformunabhängiger Code der Controller und Views getrennt werden (vgl. Buschmann *et al.* 1998 S. 143)
- Die Einführung von modernen Werkzeugklassen für Bedienoberflächen in eine MVC Architektur gestaltet sich meist kompliziert, da die Kontrollkomponente oft bereits in der Werkzeugklasse inkludiert ist (vgl. Buschmann *et al.* 1998 S. 143)

Weitere Informationen zu MVC können von der Homepage von Trygve M.H. Reenskaug (Reenskaug 1979) oder (Buschmann *et al.* 1998) entnommen werden.

### 3.1.7 Microkernel

Die Microkernel Architektur kommt bei Systemen mit variabler kundenspezifischer Funktionalität, aber mit gleichbleibender Kernfunktionalität zum Einsatz. Beispiele für solche Architekturen sind Anwendungsplattformen wie Betriebssysteme. Diese kommen oft über einen längeren Zeitraum von einigen Jahren zum Einsatz. Innerhalb dieser Jahre werden neue Technologien und Standards entwickelt oder verändert. Diese Systeme müssen entsprechend auf diese geänderten Anforderungen reagieren können (vgl. Buschmann *et al.* 1998 S. 172).

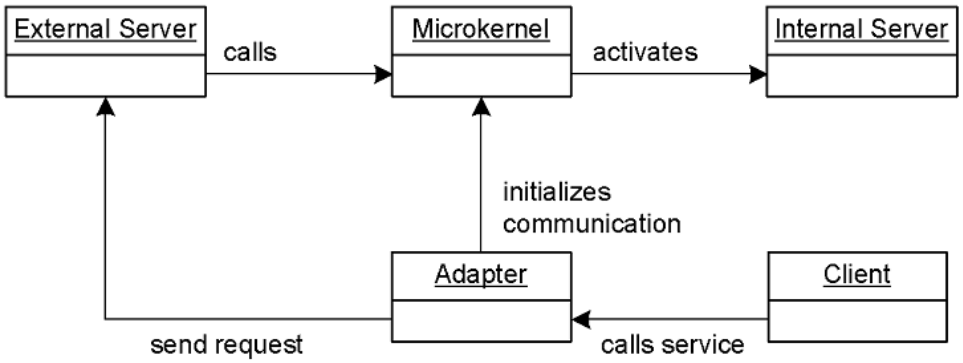



Abbildung 3.7: Microkernel Struktur von Hydra
(in Anlehnung an Buschmann *et al.* 1998 S.178)

Als Beispiel bringen (Buschmann *et al.* 1998) die Entwicklung einer eigenen Anwendungsplattform mit dem Namen "Hydra". Die Abbildung 3.7 zeigt den Aufbau des Systems. Der *Microkernel* kapselt die Kernfunktionalitäten des Systems. Darunter fallen grundlegende Dienste, Interprozesskommunikation, Verwaltung von Systemressourcen und Schnittstellen für den Zugriff auf die Kernfunktionen. Komplexe und zu große Kernfunktionalität wird in den *internen*

*Server* ausgelagert. *Externe Server* bilden konkrete Implementierung und verwendene Kernfunktionalität (vgl. Buschmann *et al.* 1998 S. 173). Beispiele solcher *externen Server* sind die OS/2-1.x, POSIX und Win32 des Windows NT Systems (vgl. Buschmann *et al.* 1998 S. 189). Die *Klienten* verwenden die Funktionalität des *externen Servers* und verwenden dazu die bereitgestellten Kommunikationsmechanismen des *Microkernels* (vgl. Buschmann *et al.* 1998 S. 173). Um eine lose Kopplung zwischen *Klient* und *externem Server* zu ermöglichen wird eine Schnittstelle (*Adaptor* oder *Emulator*) eingefügt (vgl. Buschmann *et al.* 1998 S. 177).

**Vorteile:**

- Externe Server müssen bei der Portierung in eine neue Hardware- oder Software-Umgebung nicht portiert werden. Bei der Portierung auf eine neue Hardware müssen nur die hardware-abhängigen Teile des Microkernels portiert werden (vgl. Buschmann *et al.* 1998 S. 189)
- Hoher Grad an Flexibilität und Erweiterbarkeit. Es reicht aus, einen neuen externen Server zu implementieren oder einen bestehenden zu erweitern (vgl. Buschmann *et al.* 1998 S. 189)
- Die strikte Trennung von Strategie (externer Server) und Mechanismus (Microkernel) erhöht die Wartbarkeit und Veränderbarkeit des Systems (vgl. Buschmann *et al.* 1998 S. 189f)

**Nachteile:**

- Die Architektur erreicht durch die erhöhte Flexibilität und Erweiterbarkeit gegenüber monolithischen Systemen eine geringere Performance (vgl. Buschmann *et al.* 1998 S. 190)
- Erhöhter Aufwand bei der Konzeption des Microkernels (vgl. Buschmann *et al.* 1998 S. 189)

Als Verianten zur Microkernel Architektur bieten (Buschmann *et al.* 1998) den "Microkernel System with indirect Client Server connections" oder das "Distributed Microkernel System" an (vgl. Buschmann *et al.* 1998 S. 188). Eine alternative Architektur für adaptive Systeme sind Reflection (vgl. Buschmann *et al.* 1998 S. 193-219) oder Interceptor (vgl. Schmidt *et al.* 2006 S. 109-140).

### 3.1.8 Event-based System

Typischerweise interagieren Komponenten eines Systems, beispielsweise in einer objektorientierten Applikation, durch explizite Methodenaufrufe (explicit invocation). Statt des direkten Methodenaufrufs registriert eine Komponente eines Event-basierten Systems ein oder mehrere Events (publish). An diesem Event interessierte Komponenten registrieren eine Funktion für

dieses Event (subscribe). Wird ein Event ausgelöst, werden alle für dieses Event registrierten Funktionen aufgerufen. Ein Event löst den Methodenaufruf in anderen Modulen indirekt (implicit invocation) aus (vgl. Shaw & Garlan 1996 S. 23f).

Das System mutiert durch diese Technik zu einem reaktiven System. Die Event-auslösende Komponente muss dabei die am Event interessierten Komponenten nicht kennen (vgl. Shaw & Garlan 1996 S. 24). Es entsteht ein System mit loser Kopplung zwischen den Komponenten (vgl. Bass *et al.* 1998 S. 102).

Als Beispiele für Event-basierte Architekturen nennen (Shaw & Garlan 1996) den Breakpoint Mechanismus vieler Entwickler-Editoren, Datenbank Systeme und die Trennung von Datenmanipulation und Präsentation in graphischen Oberflächen (vgl. Shaw & Garlan 1996 S. 24). Letztere beschreibt eine Situation für die Verwendung des Observer Patterns von (Gamma *et al.* 1995), das auf einer Event-basierten Architektur beruht. Das Observer (auch Publish-Subscribe) Pattern beschreibt ein beliebtes Muster zur Umsetzung der publish-subscribe Funktionalität (vgl. Gamma *et al.* 1995 S. 294).

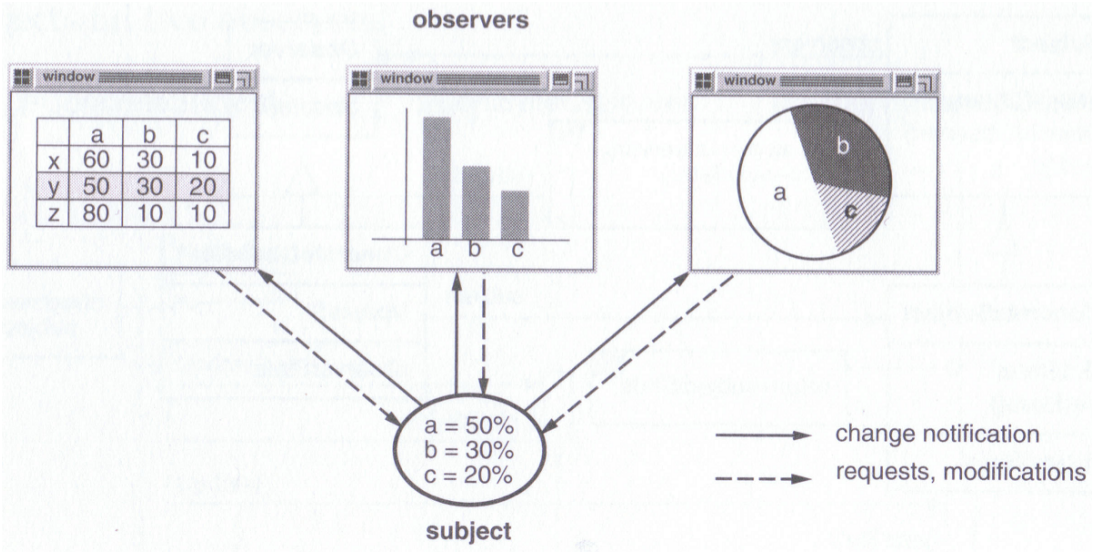



Abbildung 3.8: Observer Pattern im Einsatz (Gamma *et al.* 1995 S. 293)

Die Abbildung 3.8 zeigt eine typische Situation für den Einsatz des Observer Patterns und der Event-Architektur. Drei Benutzeroberflächen stellen Daten auf verschiedene Arten (Observer) dar und müssen bei einer Veränderung des Datensatzes (Subject) benachrichtigt werden (vgl. Gamma *et al.* 1995 S. 293f). Durch die Event-basierte Architektur können die Datendarstellenden von Daten-manipulierenden Komponenten entkoppelt werden. Das Subject muss die Observer nicht kennen (vgl. Shaw & Garlan 1996 S. 24).

**Vorteile:**

- hohe Wiederverwendbarkeit der Komponenten (vgl. Shaw & Garlan 1996 S. 24)
- entschärft Probleme bei Erweiterung des Systems (vgl. Shaw & Garlan 1996 S. 24)

- Interaktion von Komponenten bei gleichzeitiger Entkoppelung (vgl. Bass *et al.* 1998 S. 102)

**Nachteile:**

- Event-auslösende Komponenten verlieren die Kontrolle über das System. Weder können diese Komponenten bestimmen wie ein Observer auf das ausgelöste Event reagiert, noch können sie die Reihenfolge der Observer-Abarbeitung beeinflussen (vgl. Shaw & Garlan 1996 S. 24)
- Datenaustausch zwischen Observer und Listener stellt einen kritischen Punkt im Design einer Applikation dar. Manchmal können Daten als Teil des Events mitgegeben werden. Werden aber geteilte Daten verwendet kann dies zu Performance Problemen führen. (vgl. Shaw & Garlan 1996 S. 24)
- Die Folge der Funktionsaufrufe ist schwer nachzuvollziehen (vgl. Shaw & Garlan 1996 S. 24)

Weitere Details zu Event-basierten Systemen können bei (Shaw & Garlan 1996) oder (Bass *et al.* 1998) nachgelesen werden. Die genaue Spezifikation des Observer Patterns bietet (Gamma *et al.* 1995).

### 3.1.9 Client-Server

Der Client-Server Begriff beschreibt ursprünglich eine Software-Architektur, die das Zusammenarbeiten zweier Programme thematisiert. Dabei ist das eine Programm eine Applikation und das andere ein unterstützender Service, wobei beide auf dem gleichen Rechner ausgeführt werden. Erst später ging die Definition der Client-Server Architektur auf physikalisch getrennte Programme über (vgl. Berson 1996 S. 3).

Mit der Weiterentwicklung von Computer-Netzwerken und Arten der Verteilung von Ressourcen entwickelte sich auch die Client-Server Architektur weiter. Am Anfang dieser Entwicklung stand die Host-basierte Verarbeitung von Anfragen eines Clients an den Server (vgl. Berson 1996 S. 4). Die Abbildung 3.9 zeigt ein Beispiel einer solchen klassischen Host-basierten Rechner Umgebung.

In weiterer Folge gewannen LAN und verteilte Systeme immer mehr an Bedeutung. Die Entwicklung der Master-Slave Verarbeitung war der nächste Evolutionsschritt der Server-Client Architektur (Abbildung 3.10). Slaves übernehmen, im Gegensatz zur Host-basierten Verarbeitung, bereits Teilaufgaben bei der Verarbeitung von verteilten Applikationen (vgl. Berson 1996 S. 5).

Das nächste Level erreichte die Client-Server Architektur durch die gestiegene Anzahl geteilter Ressourcen, wie es beispielsweise in LANs der Fall ist. Beispiele typischer geteilter Ressour-

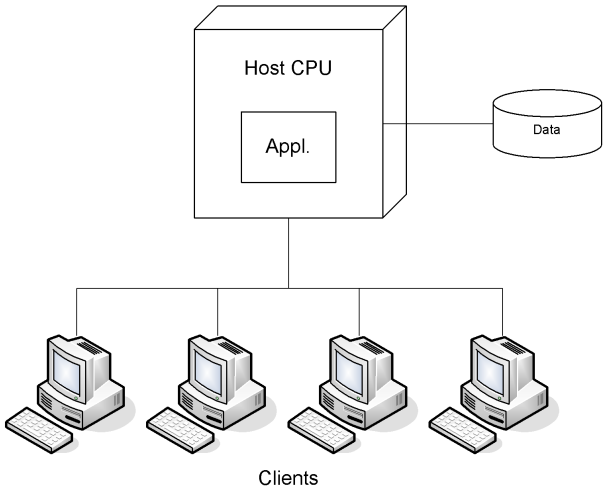



Abbildung 3.9: Host-basierte Rechner Umgebung
(in Anlehnung an Berson 1996 S. 5)

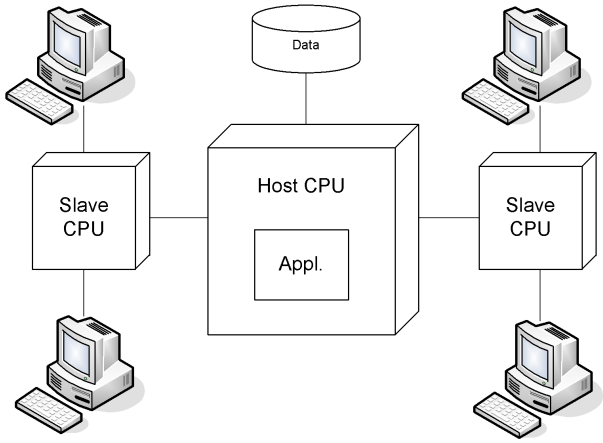



Abbildung 3.10: Master-Slave Rechner Umgebung
(in Anlehnung an Berson 1996 S. 6)

cen sind Dateien oder Drucker, die auch als Server bezeichnet werden, da sie Dienste für Clients anbieten. (Berson 1996) bezeichnet eine solche Umgebung als "Shared-device processing environment". Ein weiteres Merkmal dieser Umgebung ist das Verschieben der Ausführung der Applikation vom Server zu den Clients. Es werden nur mehr spezielle Applikationen (z.B. Drucker und Filesharing) verteilt ausgeführt (vgl. Berson 1996 S. 5-9).

Ein Beispiel einer solchen shared-device Umgebung stellt Novell's NetWare[12] dar (vgl. Berson 1996 S. 7). Mit dem Wachsen der LANs und der gestiegenen Leistung der Rechner konnten beispielsweise Fileserver immer mehr Daten verwalten und mehr Requests beteiligter Workstations verarbeiten. Es entstand das von (Berson 1996) als "Client-Server proccessing model" bezeichnete Modell. In diesem übernahm wieder der Server den Hauptteil der Applikations-Verarbeitung. Die Clients übernehmen die Initialisierung der Services, die Hauptlast trägt aber

[12] http://www.novell.com/products/openenterpriseserver/

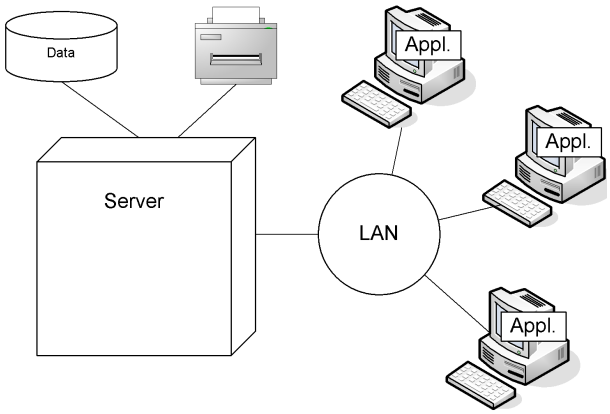



Abbildung 3.11: Shared-device Rechner Umgebung
(in Anlehnung an Berson 1996 S. 8)

der Server (vgl. Berson 1996 S. 7f). Beispiele solcher Client-Server Umgebungen stellen moderne Datenbanken dar.

Die letzte Evolutionsstufe stellt die Peer-to-Peer Architektur dar. Hierbei gibt es keine Unterscheidung mehr in Client und Server Rechner, jeder Teilnehmer im Netzwerk ist gleichberechtigt (vgl. Berson 1996 S. 9f).

**Vorteile:**

- Die Lasten können auf Clients verteilt werden (vgl. Berson 1996 S. 16)
- Erlaubt die Verarbeitung der Daten nahe der Quelle (vgl. Berson 1996 S. 16)
- Erleichtert die Verwendung von graphischen Benutzeroberflächen (vgl. Berson 1996 S. 16)
- Client und Server können auf verschiedenen Hard- und Software-Plattformen ausgeführt werden (vgl. Berson 1996 S. 16)

**Nachteile:**

- Server stellen den Flaschenhals der Architektur dar (vgl. Berson 1996 S. 16)
- Verteilte Applikationen sind komplexer als nicht verteilte (vgl. Berson 1996 S. 16f)

Weitere Details zu Client-Server Systemen können bei (Berson 1996) nachgelesen werden.

### 3.1.10 Fazit

Die Tabelle 3.1 zeigt einen Überblick über die in den Kapiteln 3.1.2 bis 3.1.9 vorgestellten Software-Architekturen.

| Architektur | Architekturelle Sicht | Vorteile | Nachteile |
|---|---|---|---|
| Layers | layers | Wiederverwendbarkeit<br>Austauschbarkeit<br>einfaches Strukturkonzept<br>minimierte Abhängigkeiten<br>Erweiterbarkeit | Gefahr bei Erweiterung der Funktionalität<br>Performance (Layer Bridges)<br>Erfolg stark von Einteilung abhängig |
| Pipes & Filter | dataflow | Wiederverwendbarkeit & Flexibilität<br>keine temporären Dateien<br>Parallelisierung | Fehlerbehandlung<br>unbekannte Zustände der Filter |
| Blackboard | data centered | erlaubt das Experimentieren<br>Erweiterbarkeit<br>Wiederverwendbarkeit<br>Toleranz gegenüber verrauschten Daten | Testbarkeit<br>geringe Anzahl lösbarer Probleme<br>komplexe Kontrolllogik<br>Effizienz<br>Parallelisierung<br>Entwicklungsaufwand |
| Broker | distribution | Client vom Server-Standort unabhängig<br>Erweiterbarkeit<br>Portierbarkeit<br>Wiederverwendbarkeit<br>Interoperabilität | Performance<br>Fehlertoleranz<br>Testbarkeit |
| MVC | user interaction | Austauschbarkeit<br>strikte Trennung<br>Framework Potential | Komplexität<br>enge Kopplung<br>Performance |
| Microkernel | adaption | Portierbarkeit<br>Erweiterbarkeit<br>strikte Trennung | Performance<br>Design des Kernels |
| Event-based System | component interaction | Wiederverwendbarkeit<br>Erweiterbarkeit<br>Interaktion und Entkoppelung | Kontrolle über das System<br>Datenaustausch<br>Nachvollziehbarkeit der Funktionsaufrufe |
| Client/Server | component interaction | Lastenverteilung<br>Quellen-nahe Datenverarbeitung<br>Erleichtert Verw. von Benutzeroberflächen<br>Plattformunabhängigkeit | Server ist Flaschenhals<br>Komplexität verteilter Applikationen |

Tabelle 3.1: Software Architekturen

Vergleicht man die Vor- und Nachteile der Architekturen, werden zwei Aspekte deutlich:

1. Systeme, die durch eine dieser Architekturen strukturiert sind, haben gegenüber monolithischen Systemen meist eine erhöhte Flexibilität und Erweiterbarkeit.
2. Als Konsequenz aus der gesteigerten Erweiterbarkeit ergibt sich jedoch meist aufgrund notwendiger Abstraktionen eine Performance-Einbuße gegenüber dem monolithischen System.

Die Verwendung und Auswahl einer Software-Architektur ist somit primär von den Anforderungen abhängig. Steht eine hohe Performance im Vordergrund, kann diese durch Verzicht auf hohe Erweiterbarkeit durch ein monolithisches System erreicht werden. (Bass *et al.* 1998) resümieren in diesem Punkt, dass die Auswahl der Software-Architektur stark von den Qualitäts-Merkmalen abhängig ist, die das System zu maximieren versucht (vgl. Bass *et al.* 1998 S. 121). Der XiTrust Business Server E2 ist ein Beispiel für die sehr eingeschränkte Erweiterbarkeit eines Systems, wenn dieses nicht durch eine entsprechende Software-Architektur strukturiert ist. Bei der Weiterentwicklung zum E3 Server wurde die Erweiterbarkeit in den Vordergrund gestellt, was zum Weggang vom monolithischem System geführt hat.

Moderne Software Systeme bauen immer häufiger auf mehreren Software-Architekturen auf und werden von (Bass *et al.* 1998) als heterogene Systeme bezeichnet. Die Autoren unterscheiden in bereichsabhängiger, hierarchieabhängiger und simultaner Heterogenität (vgl. Bass *et al.* 1998 S. 102). Als Musterbeispiel einer solchen heterogenen Architektur nennen (Dustdar *et al.* 2003) industrielle geschichtete Client/Server Systeme, die mittels CORBA Middleware kommunizieren (vgl. Dustdar *et al.* 2003 S. 82).

## 3.2 Protokolle und Standardisierung

Durch die Vielzahl verschiedener Hardware Hersteller stellt Standardisierung im Bereich der Netzwerküberwachung eine entscheidende Rolle. Ein solcher Versuch, die Art und Weise der Informationsgewinnung über den Zustand von Hardware zu standardisieren, ist IPMI. Geht man im OSI Schichtenmodell von unten nach oben, findet man weitere Protokolle, die bei der Netzwerküberwachung eine Rolle spielen (vgl. Tannenbaum 1997 S. 44-51). ICMP im Network Layer ist ein Protokoll zur Übermittlung von Status von IP Datagrammen. Das Ping Programm basiert auf ICMP und dient dem Netzwerk-Monitor die Erreichbarkeit einer Netzwerkkomponente zu überprüfen. Im Application Layer des OSI Schichtenmodells befinden sich SNMP und RMON. SNMP wurde eigens für das Netzwerk-Monitoring erschaffen und erfuhr mit RMON eine Erweiterung der Informationsbasis. Die letzten beiden Unterkapitel befassen sich mit Syslog und JMX. Syslog ist ein Standard zur Systemprotokollierung in das LAN. JMX wiederum ist ein Versuch von Java, das Monitoring von Java Applikationen zu vereinfachen und zu vereinheitlichen.

### 3.2.1 IPMI

IPMI steht für Intelligent Platform Management Interface und ist eine Sammlung von Schnittstellen für die Überwachung von Hardware. Auf Initiative einiger namhafter Hardware-Hersteller wurde 1998 die erste Version vorgestellt. 2001 wurde Version 1.5 vorgestellt und im Jahr 2004 mit 2.0 die bislang letzte Version (vgl. Schwenkler 2006 S. 169). Mittlerweile hat sich die Zahl der unterstützenden Hardware-Hersteller stark gesteigert [13] (vgl. Intel Corporation 2009 online).

Zentralle Rolle eines IPMI Systems bildet der Baseboard Management Controller (BMC). Dieser Mikrokontroller regelt die Systemschnittstelle zur Management-Software, bildet den Anschlusspunkt von Controllern zur Überwachung einzelner Hardware-Komponenten und bietet Schnittstellen für die Kommunikation (vgl. Schwenkler 2006 S. 170). Sowohl per serieller- als auch Ethernetverbindung kann der BMC angesprochen werden. Ein detailiertes Blockschaltbild einer solchen Architektur findet man in der IPMI Spezifikation (vgl. Intel Corporation *et al.* 2009 S. 39ff online). Die Features von IPMI laut (Intel Corporation *et al.* 2009) sind:

- Für den Betrieb des IPMI Systems ist die Standby Spannung ausreichend
- Möglichkeit des Versendens von SNMP Traps bei geeigneter Hardwarekonstellation
- Event Logging Datenbank
- Unterstützt eine Vielzahl von Sensoren in externer Hardware

[13] Vollständige Liste der Hardwarehersteller:
http://www.intel.com/design/servers/ipmi/adopterlist.htm

- Datenbank für Sensorwerte
- Betriebssystem unabhängig

Da mittlerweile ca. 180 Hersteller den Standard unterstützen und viele Server-Hauptplatinen bereits einen BMC Controller integriert haben, bietet der Standard eine praktikable Lösung zur Hardwareüberwachung (vgl. Intel Corporation 2009 online). Die Hardwarenähe des IPMI Standards bringt den Vorteil der Betriebssystemunabhängigkeit und somit mehr Möglichkeiten gegenüber softwarebasierten Management-Systemen bei der Hardwareüberwachung.

### 3.2.2 ICMP

Da Monitoring-Systeme meist nicht direkt auf das ICMP Protokoll zurückgreifen, spielt es im Kontext des System-Monitorings gegenüber SNMP eine untergeordnete Rolle. Client Applikationen, die auf ICMP basieren wie etwa PING (Packet Internet Groper), werden jedoch von Monitoring-Systemen häufig eingesetzt um die Erreichbarkeit von Netzwerkkomponenten zu überprüfen. Im Zusammenspiel mit unterstützenden Tools erfüllte PING in den 1980er Jahren die Anforderungen des Netzerk-Monitorings (vgl. Stallings 1999 S. 72f).

ICMP steht für Internet Control Message Protocol und ist Teil der Internetprotokollfamilie. Das IP Protokoll ist nicht entworfen, um Nachrichten (Datagrams) zuverlässig zuzustellen. Verwirft etwa ein Router ein solches Datagram, weil er überlastet ist, wird zum Host ein ICMP Paket gesendet. ICM dient vorwiegend dazu, Status- und Fehlermeldungen über Datagramme auszutauschen (vgl. Stallings 1999 S. 386). Es wurde im Jahr 1981 im RFC 792 spezifiziert (vgl. Postel 1981 online). Die Tabelle 3.3 zeigt die Form eines ICMP Pakets basierend auf dem IP Header.

| Bit 0-7 | Bit 8-15 | Bit 16-23 | Bit 24-31 |
|---|---|---|---|
| Typ | Code | Prüfsumme | |
| ID | | Sequenznummer | |
| Daten (optional) | | | |

Tabelle 3.2: ICMP Paketaufbau
(in Anlehnung an Postel 1981 online)

Die Tabelle 3.3 zeigt die elf verschiedenen Pakettypen eines ICMP Pakets.

Die wohl bekannteste Anwendung, die das ICMP Protokoll verwendet, ist das PING Programm. Es wird beispielsweise eingesetzt, um die End-to-End Erreichbarkeit innerhalb eines Netzwerkes zu überprüfen. PING sendet ein ICMP Paket vom Typ *Echo Request (Code 8)* an den Ziel-Host und wartet auf ein *Echo Reply (Code 0)* Paket. Neben der Erreichbarkeit wird gleichzeitig die Round Trip Zeit zum Ziel-Host gemessen (vlg. Stallings 1999 S. 72f).

Mit dem Aufkommen des IPv6 wurde gleichzeitig eine neue Version von ICMP entwickelt. Diese

| Type | Code | Name |
|---|---|---|
| **0** | **Echo Reply** | |
| **3** | **Destination Unreachable** | |
| | 0 | net unreachable |
| | 1 | host unreachable |
| | 2 | protocol unreachable |
| | 3 | port unreachable |
| | 4 | fragmentation needed and DF set |
| | 5 | source route failed |
| **4** | **Source Quench** | |
| **5** | **Redirect** | |
| | 0 | Redirect datagrams for the Network |
| | 1 | Redirect datagrams for the Host |
| | 2 | Redirect datagrams for the Type of Service and Network |
| | 3 | Redirect datagrams for the Type of Service and Host |
| **8** | **Echo Request** | |
| **11** | **Time Exceeded** | |
| | 0 | time to live exceeded in transit |
| | 1 | fragment reassembly time exceeded |
| **12** | **Parameter Problem** | |
| | 0 | pointer indicates the error |
| **13** | **Timestamp** | |
| **14** | **Timestamp Reply** | |
| **15** | **Information Request** | |
| **16** | **Information Reply** | |

Tabelle 3.3: ICMP Pakettypen
(in Anlehnung an Postel 1981 online)

ist unter der Bezeichnung ICMPv6 bekannt und ist im RFC 4443 spezifiziert (vgl. Conta *et al.* 2006 online).

### 3.2.3 SNMP

SNMP steht für *Simple Network Management Protocol*, wurde von der Internet Engineering Task Force (IETF)[14] im Jahr 1990 entwickelt und im RFC 1157 spezifiziert (vgl. Case *et al.* 1990 online). 1995 wurde mit SNMPv2 die zweite Version (vgl. McCloghrie *et al.* 1999 online) und 1998 mit SNMPv3 die neueste Version (RFC 3411-3418) von SNMP veröffentlicht. Das Protokoll hat sich als Standard für die Übertragung von Netzwerk-Management Informationen auf TCP/IP Basis durchgesetzt. William Stallings beschreibt in seinem Buch über SNMP (vgl. Stallings 1999) die einzelnen Versionen von SNMP sowie die zugrunde liegende Management Information Base (MIB).

Je nach Art der Netzwerkkomponente werden über das SNMP Protokoll verschiedene Daten

[14] http://www.ietf.org/

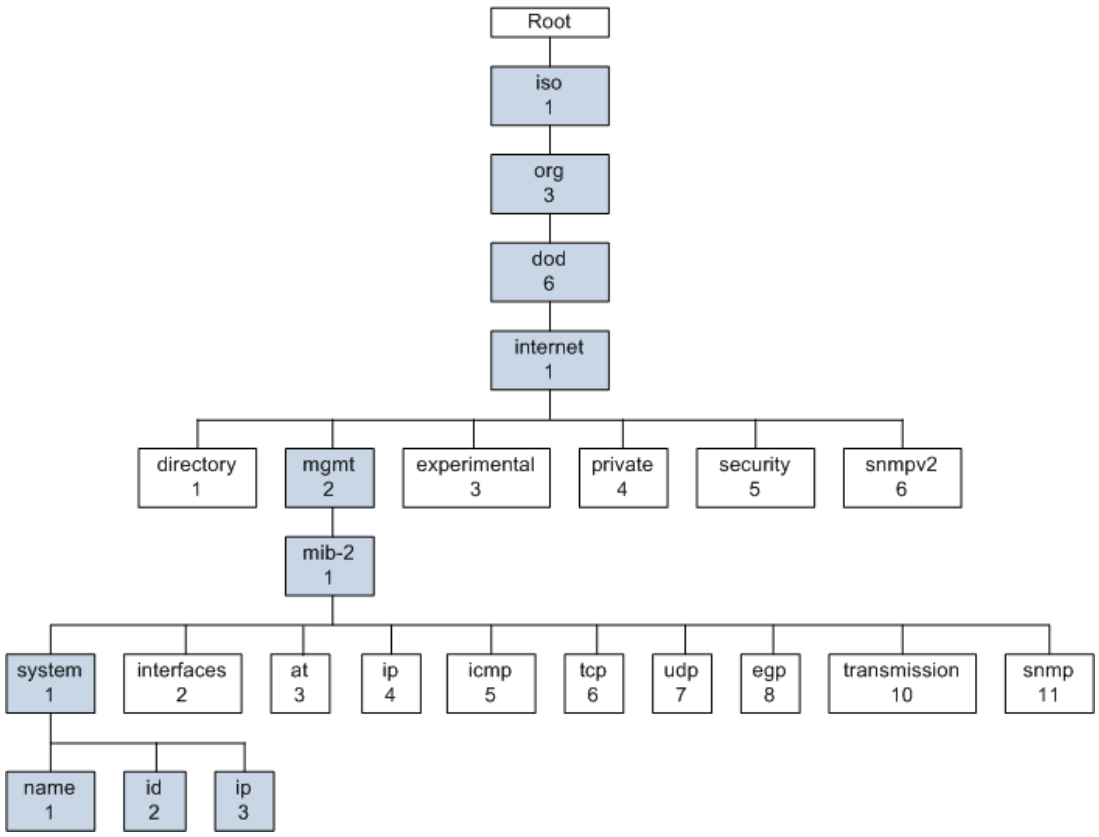



Abbildung 3.12: SNMP MIB-2 Tree
(in Anlehnung an Stallings 1999 S. 88)

ausgetauscht, sogenannte *Managed Objects*. Jedes dieser Objekte wird durch einen eindeutigen *Object Identifier* (OID) identifiziert. Mit deren Hilfe können die zur Verfügung stehenden Daten einer Netzwerkkomponente in einer Baumstruktur dargestellt werden (vgl. Stallings 1999 S. 26). Es gibt mehrere RFCs für MIBs, darunter MIB (vgl. Rose & McCloghrie 1991a online) und die Erweiterung MIB-2 (vgl. Rose & McCloghrie 1991b online) in der alle auf TCP/IP basierten Netzwerkkomponenten enthalten sind. Um die Kompatibilität von SNMP fähigen Netzwerkkomponenten zu gewährleisten, wird zur Beschreibung der Managed Objects die Sprache Structure of Management Information (SMI) verwendet, die eine Untermenge von Abstract Syntax Notation One (ASN.1) darstellt (vgl. Stallings 1999 S. 85-119). Die Abbildung 3.12 zeigt die Baumstruktur der MIB-2 und einen exemplarischen Teilbaum einer Router MIB.

Auf Basis der MIB bietet das SNMP Protokoll folgende drei Operationen (vgl. Stallings 1999 S. 163):

1. ***get***
   Eine Management Station holt einen skalaren Wert von einer anderen Station ab.

2. ***set***
   Eine Management Station setzt einen skalaren Wert einer anderen Station.

3. ***trap***
   Eine Management Station sendet unaufgefordert einen skalaren Wert zu einer anderen Station.

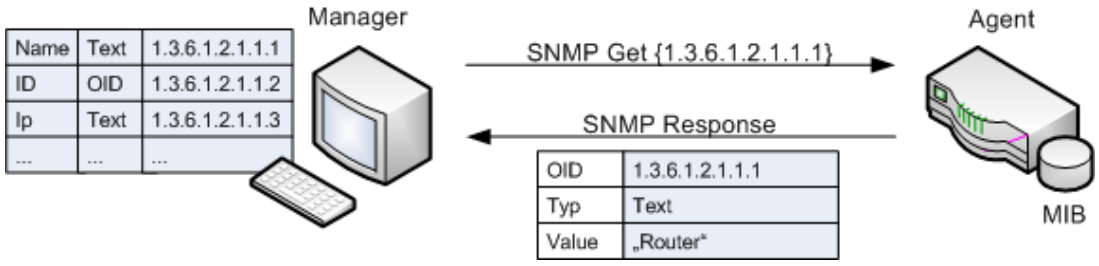



Abbildung 3.13: Beispiel einer SNMP Operation
(in Anlehnung an Stallings 1999 S. 382)

Es ist zu beachten, dass es keine Operationen gibt, die es ermöglichen, die MIB mit neuen Managed Objects zu erweitern oder solche zu löschen (vgl. Stallings 1999 S. 163). Die Abbildung 3.13 zeigt den beispielhaften Ablauf der *get* Funktion. Als MIB dient der Baum aus Abbildung 3.12. Zur Kommunikationen zwischen Management Stationen und Agents werden SNMP Messages eingesetzt. Sie bestehen aus der SNMP Versionsnummer, Community Name und einer von fünf protocol data units (PDU) (vgl. Stallings 1999 S. 174-190).

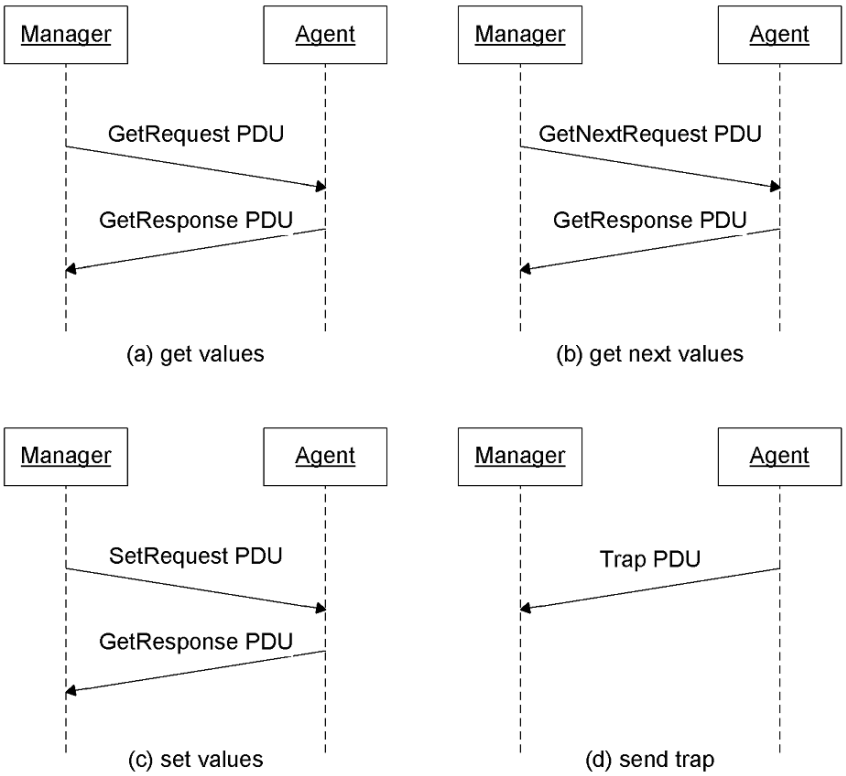



Abbildung 3.14: SNMP PDUs
(in Anlehnung an Stallings 1999 S. 178)

Die Abbildungen 3.14 zeigt die fünf verschiedenen PDUs im Einsatz bei den drei SNMP Operationen (vgl. Stallings 1999 S. 174-190). *GetRequest* und *GetNextRequest* kommen bei der *get*

Operation zum Einsatz, wobei *GetRequest* einen bestimmten Wert der MIB abfragt, *GetNextRequest* den aktuellen Wert einer sequenziellen Abfrage aller Daten. *SetRequest* ermöglicht eine *set* Operation zur Änderung von skalaren Werten der MIB. *Trap* wird vom Agenten verwendet, um unaufgefordert Werte an die Management Station zu schicken (vgl. Stallings 1999 S. 174-190). Eine Besonderheit der PDUs ist die Möglichkeit, mehrere Werte gleichzeitig, durch ein *variablebindings* Feld abzufragen. Den genauen Inhalt der verschiedenen PDUs kann bei Stallings (vgl. Stallings 1999 S. 180-190) oder im RFC 1157 (vgl. Case *et al.* 1990 online) nachgelesen werden.

Performance- und Sicherheitsprobleme führten schließlich zur Veröffentlichung von SNMPv2. Eines dieser Probleme war die Unbrauchbarkeit von SNMP im Management-Einsatz von großen IT-Netzwerken. Grund dafür ist der Polling Mechanismus, der für das Abfragen eines Wertes je ein Paket benötigt, was bei großen Netzwerken zu einem zu hohen Anteil von Steuersignalen führt. Weitere Probleme waren die Übertragung von Passwörtern als Klartext, sowie die fehlende Bestätigungsmöglichkeit von Traps auf Grund des UDP Protokolls (vgl. Stallings 1999 S. 202f). SNMPv2 brachte 1995 eine Lösung für die meisten dieser Probleme.

SNMPv3 verbesserte die Sicherheitsaspekte der Vorgänger noch weiter und führte wesentliche neue Erweiterungen im Security-Bereich ein (vgl. Stallings 1999 S. 483f):

- **User-Based Security Model**
  Das User-Based Security Model (USM) ermöglicht unter anderem Authentifizierung und Verschlüsselung von Nachrichten (vgl. Stallings 1999 S. 498-524). Die genaue Spezifikation des USM ist im RFC 3414 nachzulesen (vgl. Blumenthal & Wijnen 2002 online).
- **View-Based Access Control Model**
  Das View-Based Access Control Model (VACM) bietet die Möglichkeit der Zugriffskontrolle für MIB Objekte (vgl. Stallings 1999 S. 525-543). Die genaue Spezifikation des VACM ist im RFC 3415 nachzulesen (vgl. Wijnen *et al.* 2002 online).

Sowohl SNMPv2 als auch SNMPv3 liegen eine Reihe von RFCs zu Grunde. Die genauen Spezifikationen der beiden Protokollversionen 2 und 3, sowie eine Liste der RFCs können bei William Stallings nachgelesen werden (vgl. Stallings 1999 S. 331-544).

### 3.2.4 RMON

RMON ist eine Erweiterung des SNMP Protokolls und steht für Remote Network Monitoring. RMON selbst ist kein Protokoll im eigentlichen Sinne, sondern ein Standard, der die MIB-2 erweitert und so SNMP zum Beispiel Informationen über Netzwerkdatenverkehr oder Lastverhalten zur Verfügung stellt (vgl. Stallings 1999 S. 209f). Ziel von RMON ist es, diese Informationen verfügbar zu machen, ohne SNMP selbst zu verändern oder erweitern zu müssen (vgl. Stallings 1999 S. 210f). Mit der Entwicklung von RMON wurde von der IETF 1990 begonnen, 1991

die erste Version veröffentlicht und im RFC 1271 spezifiziert (vgl. Waldbusser 1991 online). Die Abbildung 3.15 zeigt die RMON MIB mit ihren 10 zusätzlichen Informationsgruppen.

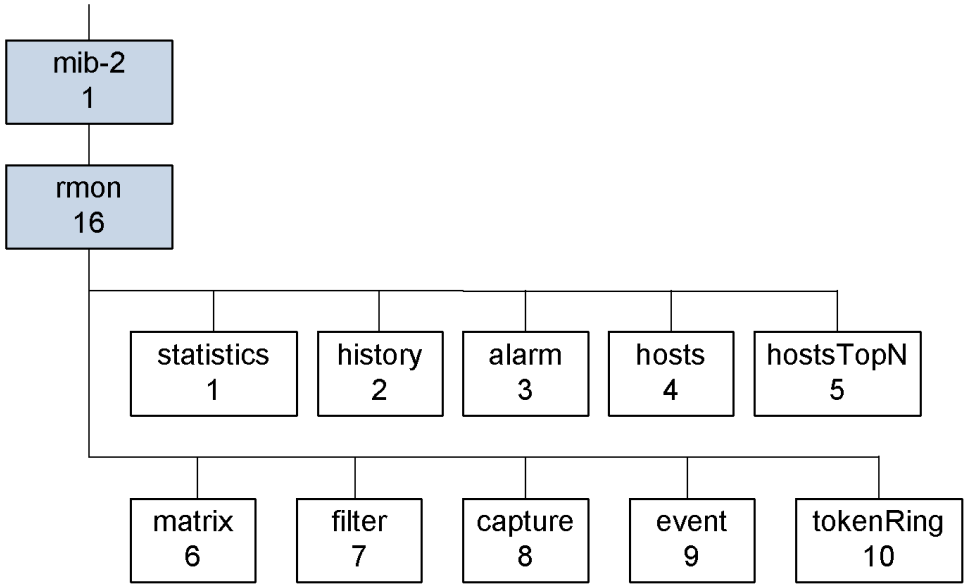



Abbildung 3.15: RMON1 MIB
(in Anlehnung an Stallings 1999 S. 221)

Die Daten der RMON MIB werden von sogenannten Analysatoren (Probes) gesammelt. Diese können eigenständige Hardware oder Teile von anderen Netzwerkkomponenten wie etwa Hubs oder Router darstellen (vgl. Stallings 1999 S. 209).

Die *statistic* Gruppe erfasst die Anzahl der gesendeten Pakete, deren Größe, Broad- sowie Multicast, Kollisionen und Netzwerkfehler. Diese Daten werden von der *history* Gruppe verwendet, um Analysen und Trends über den Netzwerktraffic zu erstellen. Die *alarm* Gruppe ermöglicht das Setzen von Schwellwerten oder Intervalle für überwachte Zähler oder ganzzahlige Werte. In der *host* und *hostTopN* Gruppe werden Daten über den Netzwerktraffic von Hosts in einem Subnet gesammelt. Um den Netzwerktraffic zwischen zwei speziellen Netzwerkadressen zu bekommen kann die Matrixform der Daten der *matrix* Gruppe herangezogen werden. Die *filter* Gruppe ermöglicht das Monitoring von speziellen Paketen, die durch einen Filter exakt bestimmt werden. Die Form der Daten, die den Management Stationen geschickt werden, wird durch die Informationen der *capture* Gruppe bestimmt. Events, die von der RMON Probe ausgelöst werden, sind in der *event* Gruppe gesammelt. Die *tokenRing* Gruppe ist die letzte Gruppe der RMON MIB und dient zum Zusammentragen von Statistikinformationen eines token ring subnets (vgl. Stallings 1999 S. 222-241). Weiterführende Details zu den Gruppen der RMON1 MIB bietet (Stallings 1999) in Kapitel 8.

1994 wurde mit der Arbeit begonnen, die RMON MIB zu erweitern, mit dem Ziel, auch Netzwerkinformationen über dem MAC-Level [15] sammeln zu können. Hintergrund dafür war zum

[15] Teil des Data Link Layers des OSI-Referenzmodells

einen, eine Möglichkeit zu erhalten, Netzwerktraffic auch über den LAN-Tellerrand hinweg zu bekommen und zum anderen, auch applikationsspezifischen Traffic, sammeln zu können (vgl. Stallings 1999 S. 277f). Die Abbildung 3.16 zeigt die Erweiterung durch die RMON2 MIB, spezifiziert in den RFCs 2021 (vgl. Waldbusser 1997 online) und 2074 (vgl. Bierman & Iddon 1997 online) im Jahr 1997.

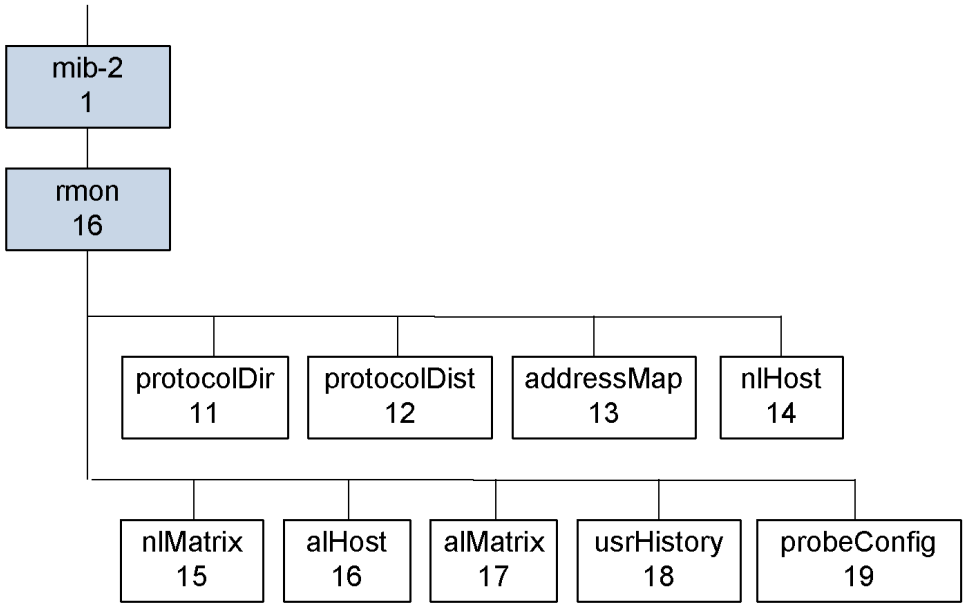



Abbildung 3.16: RMON2 MIB
(in Anlehnung an Stallings 1999 S. 279)

Die *protocol directory (protocolDir)* Gruppe ist eine Datenbank über alle Protokolle, die von dieser Probe überwacht werden. Die gesammelten Daten der Protokolle dieser Datenbank werden in der *protocol distribution (protocolDist)* Gruppe zusammengeführt. Die *addressMap* Gruppe ist für die Zuweisung von MAC- zu Netzwerkadressen zuständig. *network-layer host (nlHost)* sammelt eingehenden und ausgehenden Traffic von einzelnen Hosts, die Matrixform der *network-layer matrix (nlMatrix)* bietet die Möglichkeit den Netzwerktraffic zwischen zwei Hosts zu ermitteln. Die *application-layer host (alHost)* und *matrix (alMatrix)* folgen dem gleichen Prinzip auf dem Applikation Layer. Benutzerspezifische Daten werden in der *userHistory* Gruppe gesammelt, die Konfiguration der RMON Probe in der *probeConfig* Gruppe (vgl. Stallings 1999 S. 280). Weiterführende Details zu den Gruppen der RMON2 MIB bietet (Stallings 1999) in Kapitel 10.

Ausführliche Spezifikationen und Analysen zu RMON können zusätzlich in den RFCs 1513[16], 1757[17], 2021[18] und 2074[19] sowie bei (Stallings 1999) in den Kapiteln 8 bis 10 nachgelesen werden.

[16] Token Ring Extension to the Remote Network Monitoring MIB
[17] Remote Network Monitoring Management Information Base
[18] Remote Network Monitoring Management Information Base II
[19] Remote Network Monitoring MIB Protocol Identifiers

### 3.2.5 Syslog

Syslog ist ein Protokoll zur Weiterleitung von Log-Nachrichten in einem IP-Netzwerk. In den 1980er Jahren entwickelte Eric Allman das Syslog Protokoll als Teil der *sendmail* Implementierung (vgl. Vixie & Avolio 2002 S. 5). Später übernahmen viele Entwickler das System und so wurde es zum de-facto Standard für das Versenden von Log-Nachrichten.

Da über die Jahre verschiedene Versionen von Syslog entwickelt wurden, gründete die IETF im Jahr 2001 die syslog working group um syslog zu standardisieren. Diese veröffentlichte im August des selben Jahres das RFC 3164 (vgl. Lonvick 2001 online), das im März 2009 vom RFC 5424 (vgl. Gerhards 2009 online) abgelöst wurde. Da in der ersten Version des RFCs noch UDP als Transport Schicht für Syslog Nachrichten vorgesehen war, wurde ebenfalls noch im Jahr 2001 das RFC 3195 veröffentlicht, das eine Möglichkeit der zuverlässigen Übertragung der Log-Nachrichten definiert (vgl. New & Rose 2001 online). Das neue RFC 5424 definiert kein Transport Protokoll mehr, die Übertragung auf UDP-Basis wird eigens im separaten RFC 5426 spezifiziert (vgl. Okmianski 2009 online).

| HEADER | STRUCTURED-DATA |
|---|---|
| PRI | SD-ELEMENT |
| VERSION | SD-PARAM |
| TIMESTAMP | SD-ID |
| HOSTNAME | SD-NAME |
| APP-NAME | PARAM-NAME |
| PROCID | PARAM-VALUE |
| MSGID | MSG |

Tabelle 3.4: Syslog Nachrichtenformat
(in Anlehnung an Gerhards 2009 online)

Die Tabelle 3.4 zeigt den Aufbau des Syslog Nachrichtenformats laut RFC 5424. Die HEADER Daten bestimmen die Priorität, Version, Zeit und Herkunft der Log-Nachricht, das STRUCTURED-DATA Bereich enthält die eigentliche Log-Nachricht in einem strukturierten Format. Eine besondere Rolle spielt das PRI Feld. Es enthält eine Priorität die aus einer Einrichtung (Facility) und einer Härte (Severity) berechnet wird (vgl. Gerhards 2009 online).

Die Berechnung der Priorität erfolgt durch die Multiplikation des Faciliy Codes (siehe Tabelle 3.5) mit 8. Zum Resultat wird der Severity Code aus Tabelle 3.6 addiert. So hat zum Beispiel eine Nachricht, die einen Druckfehler beinhaltet die Priorität $6\ (lineprintersubsystem) * 8 + 3\ (Error) = 51$. Die exakten Definitionen zu den anderen Abschnitten des Syslog Nachrichtenformats können im RFC 5424 nachgelesen werden (vgl. Gerhards 2009 online).

Syslog bringt den entscheidenden Vorteil, die Log-Nachrichten von verschiedenen Systemen zentral sammeln zu können, in dem das Syslog-Port (meist 514) von einem Monitoring-System überwacht wird. Ein großes Problem von Syslog ist die Übertragung der Nachrichten als Klartext, dadurch der Inhalt der Log-Nachrichten ungeschützt ist. Fehlende Authentifizierungsmöglichkeiten sowie viele Implementierungen, die sich nicht an den Standard halten, bilden weitere

| Code | Facility | Code | Facility |
|---|---|---|---|
| 0 | kernel messages | 12 | NTP subsystem |
| 1 | user-level messages | 13 | log audit |
| 2 | mail system | 14 | log alert |
| 3 | system daemons | 15 | clock daemon |
| 4 | security/authorization messages | 16 | local use 0 |
| 5 | messages generated internally by syslogd | 17 | local use 1 |
| 6 | line printer subsystem | 18 | local use 2 |
| 7 | network news subsystem | 19 | local use 3 |
| 8 | UUCP subsystem | 20 | local use 4 |
| 9 | clock daemon | 21 | local use 5 |
| 10 | security/authorization messages | 22 | local use 6 |
| 11 | FTP daemon | 23 | local use 7 |

Tabelle 3.5: Syslog Facility Types
(in Anlehnung an Gerhards 2009 online)

| Code | Severity |
|---|---|
| 0 | Emergency: system is unusable |
| 1 | Alert: action must be taken immediately |
| 2 | Critical: critical conditions |
| 3 | Error: error conditions |
| 4 | Warning: warning conditions |
| 5 | Notice: normal but significant condition |
| 6 | Informational: informational messages |
| 7 | Debug: debug-level messages |

Tabelle 3.6: Syslog Severity Types
(in Anlehnung an Gerhards 2009 online)

Probleme des Syslog Protokolls.

### 3.2.6 JMX

JMX steht für Java Management Extensions und ist ein mögliches Mittel für die Überwachung einer Java Virtual Machine (JVM). Das Bestreben, Java Applikation besser überwachen zu können geht schon auf den Java Specification Request (JSR) 3[20] aus dem Jahre 1998 zurück. Er enthält die Beschreibung der JMX Komponenten bis zur Version 1.2 von Java. Mit Java 5 wurden Teile von JMX in die Standard API übernommen (vgl. Wunderlich 2006 S. 43).

Die JMX Spezifikation definiert drei Bereiche (Levels), die in Abbildung 3.17 der JMX Architekturübersicht dargestellt sind.

Einer überwachten Komponente (*Managed Component*) liegt immer eine Managed Bean (M-

[20] http://jcp.org/aboutJava/communityprocess/mrel/jsr003/index.html

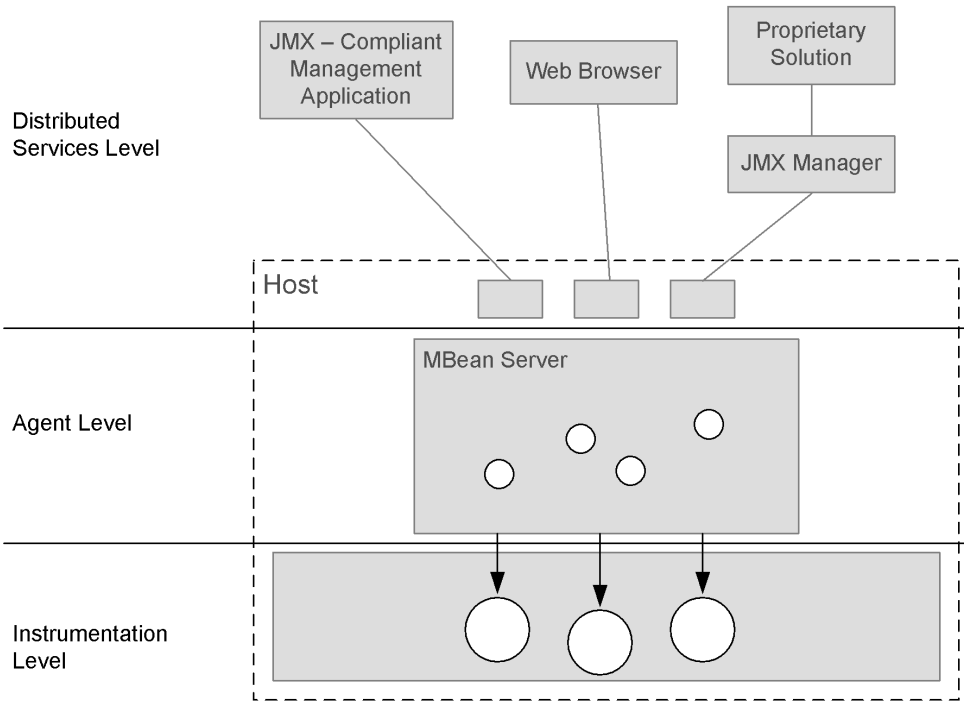



Abbildung 3.17: JMX Architektur
(in Anlehnung an Wunderlich 2006 S. 45)

Bean) zu Grunde. Diese sind im *Instrumentation Level* angesiedelt und sind für den Zugriff auf Informationen verschiedener Komponenten (Ressourcen) zuständig. Alle diese MBeans müssen sich beim MBean Server (*Agent Level*) registrieren. In der obersten Schicht, dem Distributed Services Level, können verschieden Adaptoren implementiert werden, die den Zugriff auf die MBeans ermöglichen (vgl. Wunderlich 2006 S. 44ff). Die Java 6 API spezifiziert fünf Arten von MBeans:

- **Standard MBeans**
  Diese Art von MBeans bilden den einfachsten Weg das JMX System zu nutzen. Die Arbeit des Entwicklers beschränkt sich auf das Erstellen eines Interfaces mit dem Suffix MBean und einer Implementierung des Interfaces mit dem gleichen Namen ohne Suffix. Das Listing 3.1 und 3.2 zeigen ein einfaches Beispiel einer MBean, das die Anzahl der Insassen eines Busses auslesen und setzen kann (vgl. Wunderlich 2006 S. 46-50).

Listing 3.1: BusMBean Interface

```
1 public interface BusMbean {
2         public int getNumberOfPassangers();
3         public void setNumberOfPassangers(int
             numberOfPassangers);
4 }
```

Listing 3.2: Bus Implementation

```
1  public class Bus implements BusMBean {
2          private int passangers_;
3
4          public int getNumberOfPassangers() {
5                  return passangers_;
6          }
7          public void setNumberOfPassangers(int
              numberOfPassangers) {
8                  passangers_ = numberOfPassangers;
9          }
10 }
```

Weitere Beispiele für Standard MBeans können beispielsweise bei (Wunderlich 2006)[21] in Kapitel 2 oder bei (Ullenboom 2007)[22] in Kapitel 23 nachgelesen werden.

- **Dynamic MBeans**
  Sind Attribute, die ein MBean an Management Applikationen publizieren will, von der Laufzeit abhängig, müssen Dynamic MBeans verwendet werden. Das Interface `DynamicMBean`[23] stellt Methoden zur Verfügung, die das Auslesen und Setzen von Werten aus einer Liste von Ressourcen ermöglicht. Die Liste enthält zu publizierende Attribute und wird zur Laufzeit erstellt (vgl. Wunderlich 2006 S. 50ff).

- **Model MBeans**
  Model MBeans sind eine Erweiterung der Dynamic MBeans, die zusätzlich die Möglichkeit des Speicherns und Ladens von Ressourcen etwa in einer Datenbank oder Datei bieteb (vgl. Sun Microsystems 2009 online).

- **Open MBeans**
  Diese Klasse von MBeans ermöglicht das Publizieren von neuen Objekten zur Laufzeit durch Beschreibung mittels Metadaten. Rückgabewerte der Operationen dürfen nur Datentypen der sogenannten "Basic Data Types" sein (vgl. Sun Microsystems 2009 online). Alle verfügbaren Datentypen können auf der Sun Homepage[24] nachgelesen werden.

- **MXBeans**
  MXBeans sind den Standard Beans sehr ähnlich. Sie verwenden wie Open MBeans ebenfalls nur einen eingeschränkten Satz von Datentypen, um die Unabhängigkeit des Clients vom Klassenmodell der Applikation zu erreichen. MXBeans bieten zusätzlich die Möglichkeit, Werte in Bündeln zusammenzufassen. Gleich wie Standard Beans werden sie durch ein Interface (`SomethingMXBean`) und eine implementierende Klasse erzeugt. Diese muss jedoch nicht den Namen `Something` haben (vgl. Ullenboom 2007 S. 1363ff).

[21] JMX in Java 5
[22] JMX in Java 6
[23] siehe JavaDoc
http://java.sun.com/javase/6/docs/api/javax/management/DynamicMBean.html
[24] http://docs.sun.com/app/docs/doc/816-7609/6mdjrf842?a=view

Alle MBeans bestehen aus Attributen, Operationen und Notifications. Attribute können von Management-Clients gelesen und geschrieben, Operationen von ihnen ausgeführt werden (vgl. Wunderlich 2006 S. 48). Mit den Notifications bietet JMX einen besonderen Benachrichtigungsmechanismus, mit dem Beans Management-Clients über Zustandsänderungen benachrichtigen können. Dazu muss ein MBean das Interface `NotificationEmitter` implementieren oder die Klasse `NotificationBroadcasterSupport` erweitern. Ein Event wird durch das Erstellen eines `Notification` Objekts und den `NotificationBroadcaster Support.sendNotification` an die `NotificationListener` geschickt. Ein solcher Listener ist die *jconsole.exe*, ein JMX Client, der im Java Development Kit inkludiert ist (vgl. Wunderlich 2006 S. 57ff).

Die JConsole bietet eine einfache Möglichkeit, die Informationen der in Java 6 bereits implementierten MXBeans, darzustellen. Diese sind:

| | |
|---|---|
| ClassLoadingMXBean | Anzahl der geladenen Klassen |
| CompilationMXBean | Compiler Daten |
| GarbageCollectorMXBean | Informationen zum Garbage Collector |
| MemoryManagerMXBean | Dient zum ermitteln der Memory Pools |
| MemoryMXBean | Zeigt die Auslastung des Heap Speichers |
| MemoryPoolMXBean | Zeigt die Auslastung der Memory Pools |
| OperatingSystemMXBean | Enthält Informationen zum verwendeten Betriebssystem |
| RuntimeMXBean | Enthält eine Reihe von Laufzeitinformationen |
| ThreadMXBean | Informationen zum JVM Thread-System |

Tabelle 3.7: Java MXBeans
(in Anlehnung an Ullenboom 2007 S. 1363)

Welche Attribute, Operationen und Notifications in den einzelnen MXBean Implementierungen enthalten sind kann in der Java 6 API unter der URL `http://java.sun.com/javase/6/docs/api/` nachgelesen werden.

### 3.2.7 Fazit

Alle vorgestellten Protokolle und Standards übernehmen in der System- oder Netzwerküberwachung unterschiedliche Aufgaben.

Die Tabelle 3.8 zeigt eine Übersicht über die vorgestellten Protokolle. Mit IPMI wurde 1995 versucht, das Monitoring von Hardwarekomponenten zu standardisieren. Schon früher im Jahre 1981 wurde ICMP vorgestellt, das Teil der Internet Protokoll Familie ist. Mit Hilfe dieses Protokolls werden Fehler- und Informationsnachrichten über IP Pakete zwischen den Netzwerkkomponenten ausgetauscht. Sowohl IMPI als auch ICMP spielen keine direkte Rolle im Kontext dieser Arbeit, bilden aber wertvolle Grundlage für Tools wie beispielsweise das, für das Netzwerk-Monitoring notwendige PING Programm.

Mit SNMP wurde 1990 ein Protokoll zur Überwachung von Netzwerkkomponenten veröffent-

| Standard | Typ | Aufgaben | Entwicklung |
|---|---|---|---|
| IPMI | Hardware Standard | Standardisierung der Überwachung von Hardwarekomponenten | 1995 |
| ICMP | Netzwerkprotokoll | Kontrollnachrichten des IP Protokolls | 1981 |
| SNMP | Netzwerkprotokoll | Überwachung von Netzwerkkomponenten | 1990 |
| RMON | Standard | Erweiterung der SNMP MIB | 1991 |
| Syslog | Log-Standard | Standard für die Übertragung von Log Nachrichten | 2001[a] |
| JMX | Software Standard | Versuch von Java das Monitoring von Software zu standardisieren | 1998 |

[a]Entwickelt bereits vorher im Zuge von Sendmail, 2001 Erscheinung der RFCs 3164 und 3195

Tabelle 3.8: Übersicht über Protokolle und Standards

licht. SNMP wurde im Laufe der Jahre weiterentwickelt und setzte sich immer mehr als das Protokoll im System-Monitoring Bereich durch. Nicht nur Hardwarehersteller bieten die Möglichkeit des Monitoring per SNMP, auch im Softwarebereich findet SNMP Anklang. Beispielsweise bietet die JVM SNMP-Monitoring basierend auf der JVM-Management MIB. Mit RMON erfuhr die SNMP-MIB Erweiterungen, um statistische Daten von Netzwerkkomponenten sammeln zu können.

Mit Syslog entstand bei der Entwicklung von sendmail von Eric Allman ein Nebenprodukt, das zu einem fixen Bestandteil der Monitoring Welt geworden ist. Es wurde zum de-facto Standard für die Übertragung von Log-Nachrichten in IP-Netzwerken.

JMX bildet einen softwaretechnischen Ansatz des Monitorings von Java-Applikationen. Bereits 1998 angedacht, bietet Java in der aktuellen Version 6 viele Features, um Applikationen zu überwachen.

Für die Entwicklung eines Monitoring-Systems für den XBS spielen besonders JMX und SNMP eine große Rolle. JMX ermöglicht das Auslesen von Systemzuständen wie CPU und Speicher Auslastung. Durch das Vorsehen einer SNMP Schnittstelle soll die Qualität und Skalierbarkeit des Systems hochgehalten werden, indem der XBS auch von externen Monitoring-Tools überwacht werden kann.

## 3.3 Beispiele von Monitoring-Systemen

Der Wunsch, ein System oder eine Applikation überwachen zu können, ist nicht neu. Aus diesem Grund findet man eine Vielzahl von kommerziellen Produkten und Lösungen aus dem Open Source Bereich, die in diesen Kontext fallen. Die nachfolgenden Unterkapitel stellen verschiedene Systeme genauer vor. Hauptaugenmerk bei der Auswahl lag dabei nicht in der Bekanntheit des jeweiligen Monitoring-Systems, sondern im Aufbau und verwendeter Architektur

des Systems. Anhand dieser Systeme werden Vor- und Nachteile der verwendeten Architekturen analysiert. Diese dienen als Basis für Architekturentscheidungen bei der Entwicklung des Monitoring-Systems des XiTrust Business Servers.

### 3.3.1 Nagios

Nagios steht für "Nagios Ain't Gonna Insist On Sainthood"[25] und ist ein Tool zur Überwachung einer kompletten IT-Infrastruktur (vgl. Nagios Enterprises 2009b online). Das System steht unter der GNU General Public License[26] und liegt momentan in der Version 3 auf (vgl. Nagios Enterprises 2009a S. 5 online).

- **Einsatzgebiet**
  Nagios kann zur Überwachung von Hardware und Software einer IT-Infrastruktur verwendet werden. Hardware Komponenten werden als Hosts bezeichnet, auf denen verschiedene Services (Software) zum Einsatz kommen (vgl. Nagios Enterprises 2009a S. 4 online).

- **Architektur**
  Nagios ist ein sehr großes und komplexes Gebilde, das durch seine Server-Agent Architektur bestimmt wird. Der Server ist ein zentraler Monitoring Rechner, auf dem ein Unix basiertes Betriebssystem zum Einsatz kommt. Wird ein PC von Nagios überwacht, muss dieser einen sogenannten Agent installiert haben, der als Proxy zwischen den Services auf dem Zielrechner und dem Nagios System fungiert (vgl. Nagios Enterprises 2009a S. 19f online). Die Abbildung 3.18 zeigt diese Architektur anhand der Überwachung von Services eines Windows PCs.

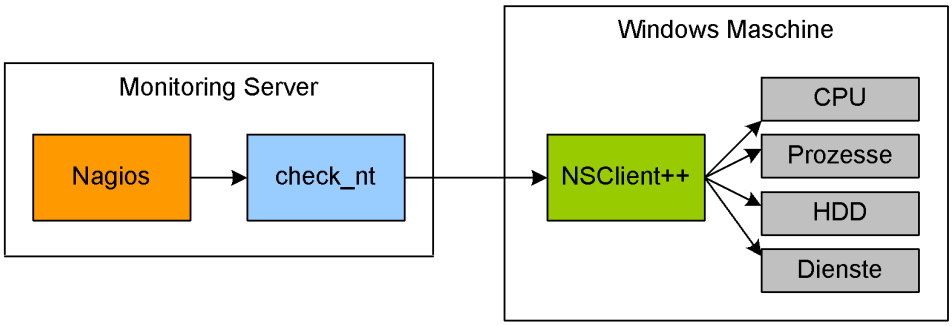



Abbildung 3.18: Windows Überwachung mit Nagios
(in Anlehnung an Nagios Enterprises 2009a S. 19 online)

Der Monitoring Server besteht aus dem Nagios System selbst und dem check_nt Plugin, das die Kommunikation zum Windows Agent (NSClient++) ermöglicht. Verschiede-

[25] bezieht sich auf den früheren Namen NetSaint, der wegen eines Trademark-Konflikts abgelegt werden musste (vgl. Nagios Enterprises 2009c online)
[26] http://www.gnu.org/copyleft/gpl.html

ne Services ermöglichen beispielsweise die Überwachung der CPU, des Speichers, der Festplatte, von Prozessen und Diensten. Alle Plugins sind ausführbare Programme oder Konsolen-Skripte, die von Nagios zum Abrufen eines Systemzustands aufgerufen werden. Der Output des Plugins wird von Nagios weiterverarbeitet (vgl. Nagios Enterprises 2009a S. 141 online). Es existiert eine Unzahl von Nagios Plugins. So findet man zum Beispiel auf der Nagios Exchange Homepage[27] ca. 1400 verschiedene Plugins.

Nagios unterteilt die Zustände von Services in die zwei Klassen Hard und Soft. Liefert die Überwachung eines Services oder Hosts einen kritischen Zustand, ist dieser vorerst ein Soft State. Ist das System auch nach dem wiederholten Check (Schwellwert in Konfiguration festgelegt) noch im kritischen Zustand, wechselt dieser zu einem Hard State. Bei einem Soft State wird die Event-Engine gestartet, bei einem Hard State wird zusätzlich die Notification-Engine aktiviert. Die Art der Benachrichtigung (E-Mail, SMS, o.ä.) und an welche Benutzer diese verschickt werden, wird durch die Nagios Konfiguration geregelt (vgl. Nagios Enterprises 2009a S. 182ff online).

Die Event-Engine ist ein komplexes Konstrukt, das Nagios die proaktive Lösung von Problemen ermöglicht. Ein möglicher Anwendungsfall ist der Neustart eines Services durch einen Event-Handler. Dieser kann ein ausführbares Programm oder Skript sein, das beim Eintritt eines Events ausgeführt wird. Herzstück der Event-Engine ist der Event-Broker, der auf einer Publisher-Subscriber Architektur basiert (vgl. Nagios Enterprises 2009a S. 210 online). Eine detaillierte Beschreibung der Nagios Event Broker Engine API findet man bei (Ingraham 2006).

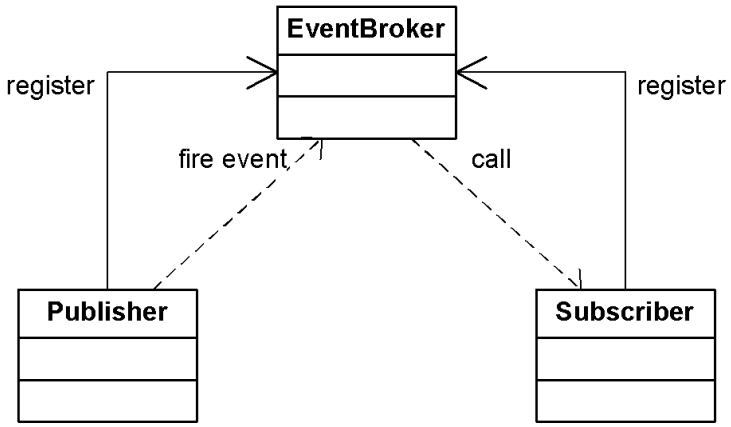



Abbildung 3.19: Nagios Event Broker Architektur
(in Anlehnung an Starke 2008 S. 171f)

Die Abbildung 3.19 zeigt den Aufbau des Event-Brokers. Die Event-Handler (Subscriber) registrieren sich beim Broker für ein Event. Publisher registrieren alle Events beim Broker, die sie erzeugen können. Löst ein Publisher ein Event aus, aktiviert der Event Broker alle Handler, die sich für dieses Event registriert haben. Diese Architektur ist in der Softwareentwicklung auch als Observer Pattern bekannt (vgl. Gamma *et al.* 1995 S. 293-303).

[27] http://exchange.nagios.org/directory/Plugins

- **Protokolle**
  Von den in Kapitel 3.2 vorgestellten Protokollen unterstützt Nagios SNMP und Syslog. Mit entsprechender Konfiguration loggt das Nagios System auf die Syslog Engine des Monitoring Servers. Mit dem entsprechenden Plugin horcht Nagios auf das korrespondierende Syslog Port des Zielrechners. Für die Unterstützung von SNMP gibt es eine Reihe von Plugins. Die Architekur von Nagios ermöglicht die Integration aller vorgestellten Protokolle, durch eine Implementierung eines geeigneten Plugins (vgl. Nagios Enterprises 2009a S. 176ff online).

- **Besonderheiten**
  Ähnlich der SNMP *Get* und *Trap* Funktionalität ermöglicht auch Nagios aktive und passive Wege der Systemüberwachung.

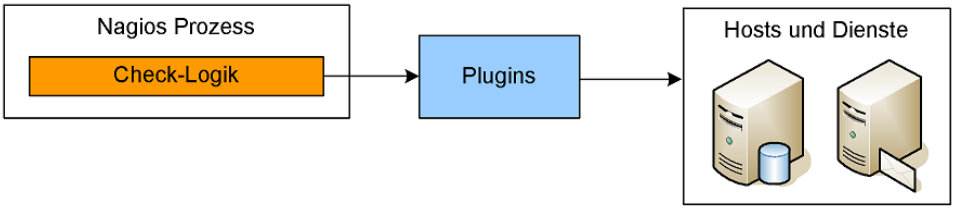



Abbildung 3.20: aktive Überwachung mit Nagios
(in Anlehnung an Nagios Enterprises 2009a S. 176 online)

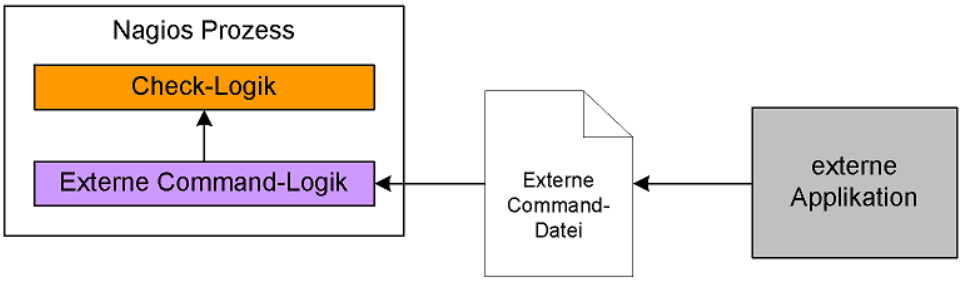



Abbildung 3.21: passive Überwachung mit Nagios
(in Anlehnung an Nagios Enterprises 2009a S. 179 online)

Die Abbildung 3.21 zeigt den passiven Weg. Dabei werden vom überwachten System die entsprechenden Werte in eine externe Command-Datei abgelegt. Diese werden in kurzen Perioden vom Nagios System abgeholt und zur Weiterverarbeitung in einer Queue deponiert. Beim aktiven Abfragen von Systemzuständen (siehe Abbildung 3.20) werden die resultierten Werte in die gleiche Queue abgelegt und später vom Nagios System weiterverarbeitet (vgl. Nagios Enterprises 2009a S. 176-181 online).

- **Vor- und Nachteile**
  Der Umstand, dass Nagios Plugins selbstständige Programme oder Konsolen-Skripts sind, bringt ein breites Feld von Einsatzmöglichkeiten. Das System ermöglicht beispielsweise eine programmiersprachenunabhängige Entwicklung von Plugins. Die Abhängigkeit von einem Unix basiertes Betriebssystem bringt jedoch gewisse Einschränkungen der Programmiersprachenunabhängigkeit.

Die Analyse des Nagios Systems basiert auf der Version 3.2.0 vom 12.08.2009. Weitere Details zu Nagios können der Hersteller Homepage[28] entnommen werden. Eine weitreichende Einführung mit praktischem Leitfaden in das Nagios System bietet beispielsweise (Barth 2005).

### 3.3.2 Collectd

Collectd ist ein Unix Daemon zum Sammeln von Statistiken über Netzwerkkomponenten und wurde im Sommer 2005 von Florian Forster ins Leben gerufen. Es war ursprünglich nur für die Darstellung von Statistiken über das lokale System gedacht, mittlerweile ist der Daemon zu einem Tool mit vielen Einsatzmöglichkeiten gewachsen. Collectd liegt bereits in der Version 4.7.1 auf und hat nun mehr als 30 Contributors (vgl. Harl 2008 S. 3ff online).

- **Einsatzgebiet**
  Collectd wird vorwiegend zum Monitoring von Hard- und Software eines Servers eingesetzt (vgl. Harl 2008 S. 5 online). Über ein Plugin können Hardwarekomponenten via SNMP in die Überwachung miteinbezogen werden (vgl. Harl 2008 S. 15 online). Zusätzlich können per Plugin mehrere Daemons mittels UDP Protokoll miteinander verbunden werden (vgl. Harl 2008 S. 19-25 online).

- **Architektur**
  Collectd basiert auf einer Microkernel Architektur und ist zur Gänze in C implementiert. Philosophie der Entwickler ist es, den Kern möglichst schlank zu halten und mit wenig Funktionalität auszustatten, aber gleichzeitig einen hohen Grad an Erweiterbarkeit durch Plugins zu erreichen[29]. Plugins sind im Vergleich zu Nagios keine eigenständigen Programme, sondern Programmteile die an eine Plugin API andocken (vgl. Harl 2008 S. 28 online).

  Die Abbildung 3.22 zeigt die Plugin Architektur von Collectd. Der Kern des Systems kann um read und write Plugins erweitert werden. Read Plugins sind dabei für die Gewinnung und Write Plugins für das Speichern der Daten zuständig. Die Abbildung 3.23 beschreibt den Zusammenhang und die zeitliche Abarbeitung der Plugins (vgl. Harl 2008 S. 28 online).

  Alle grau hinterlegten Komponenten sind Teil des Collectd Core und alle weiß hinterlegten sind Plugins, die nach Belieben hinzugefügt werden können. In der `main loop` wird das Plugin `plugin_read_all` periodisch aufgerufen. Dieses Plugin ruft der Reihe nach jedes registrierte Plugin auf um die jeweiligen Systemzustände in Erfahrung zu bringen (vgl. Harl 2008 S. 29f online).

  Neben der globalen Microkernel Architektur kommt für die Verarbeitung aller gesammelten Daten ein Filter Chain System zum Einsatz (vgl. Forster 2009b online).

[28] http://www.nagios.org
[29] vgl. Kapitel 3.1.7

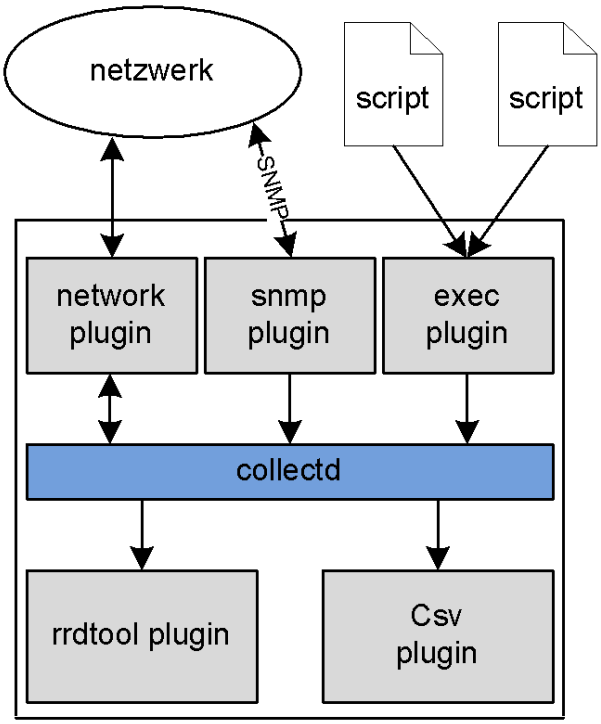



Abbildung 3.22: Die Plugin Architektur von Collectd
(in Anlehnung an Forster 2009c online).

Die Abbildung 3.24 zeigt den beispielhaften Aufbau einer Chain. Diese besteht aus mehreren Rules, die ihrerseits eine Menge von Targets und Matches enthalten. Ein Match ist ein Kriterium, das aussagt, ob der eingehende Wert von der entsprechenden Rule bearbeitet werden soll. Wenn dies der Fall ist, werden alle Targets der Rule abgearbeitet. Targets selbst sind Aktionen, die von einer Rule bei positivem Match ausgeführt werden. Beispiele wären: ein Write Plugin für das Schreiben in eine Datenbank aktivieren, oder ein Notify Plugin für die Benachrichtigung initiieren. Die für das Filter Chain System verwendete Architektur ist nicht neu und wurde größten Teils dem aus der Linux Welt bekannten iptables [30] Filter System nachgeahmt (vgl. Forster 2009b online).

- **Protokolle**
  Ähnlich wie bei Nagios können auch beim Collectd Monitoring-System verschiedene Protokolle über Plugins eingebunden werden. Für das SNMP Protokoll gibt es bereits ein Plugin und Syslog wird ohnehin vom Daemon benutzt, um den eigenen Zustand zu dokumentieren (vgl. Forster 2009c online).

- **Vor- und Nachteile**
  Wahrscheinlich die größten Nachteile des Systems liegen in der fehlenden Portierbarkeit und der Programmiersprachenabhängigkeit. Das Collectd Monitoring-System benötigt einen Unix basierten Host und Plugins die in C geschrieben sind. In der aktuellen Version stehen Perl und Java Interpreter Plugins zur Verfügung, diese stecken aber noch

[30] http://netfilter.org/projects/iptables/index.html

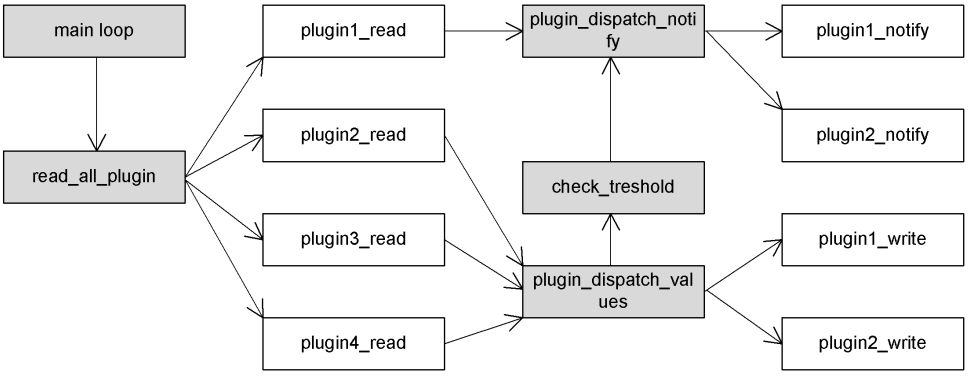



Abbildung 3.23: Plugin Verarbeitungszyklus von Collectd
(in Anlehnung an Harl 2008 S. 30 online)

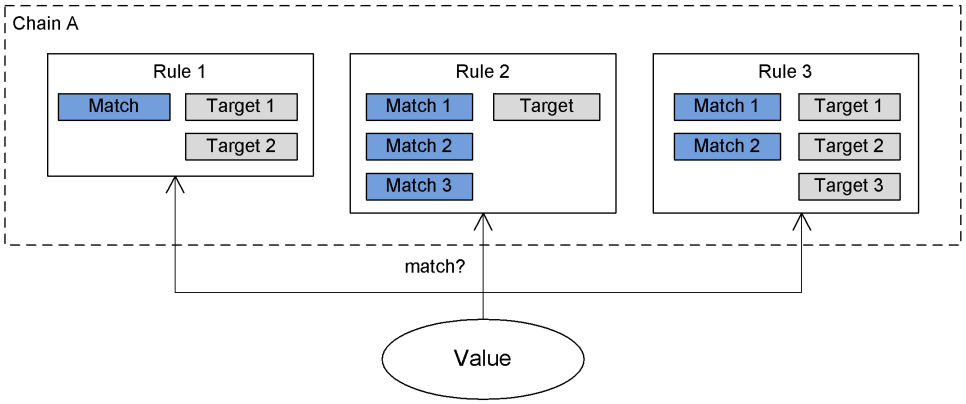



Abbildung 3.24: Collectd Filter Chain System
(in Anlehnung an Forster 2009b online)

am Beginn ihrer Entwicklung. Der Vorteil des Systems liegt in der Erweiterbarkeit, welche die Microkernel Architektur ermöglicht. Da der Kern selbst sehr wenig Funktionalität bietet, kann das Collectd Monitoring-System durch Plugins exakt an das jeweilige Zielsystem angepasst werden und es bleibt trotzdem immer sehr schlank.

Die Analyse des Collectd Systems basiert auf der Version 4.7.1 vom 02.06.2009. Zusätzliche Informationen liefert eine Präsentation von Sebastian Harl, die er im Zuge des LUG Camp Flensburg vom 03. Mai 2008 abhielt (vgl. Harl 2008 online). Er ist neben Florian Forster das zweite permanente Projektmitglied und stellt in der Präsentation Collectd und dessen Architektur vor.

### 3.3.3 Argus

Das Argus Monitoring-System wurde im Jahr 1996 erstmals veröffentlicht. Damals trug es noch den Namen system monitoring thing (smt) und war ein Konglomerat aus Perl und Shell Skripten, die grundlegende ping und SNMP Funktionalität boten (vgl. Weisberg 2009c online). Das

System wird von Jeff Weisberg entwickelt und steht unter der Open Source "Artistic License"[31]. Es wird fortlaufend weiterentwickelt und liegt nun in der Version 3.6 vom 27. Oktober 2008 auf (vgl. Weisberg 2009c online).

- **Einsatzgebiet**
  Ähnlich dem Collectd Monitoring-System ist Argus für die Überwachung von Servern, Services und Hardware Komponenten eines IT-Netzwerks einsetzbar (vgl. Weisberg 2009a online).

- **Architektur**
  Die Analyse des aktuellen Quelltextes ergab, dass Argus auf einer gänzlich monolithischen Architektur basiert und durch ein oder mehrere Konfigurationsdateien angepasst werden kann. Die Hauptkomponenten dieser Dateien bilden die Groups oder auch Hosts. Diese fassen zu überwachende Services der Zielsysteme zu Gruppen zusammen. Ein weiterer wichtiger Bestandteil, die Notification Settings, ist für die Konfiguration des Benachrichtigungssystems verantwortlich (vgl. Weisberg 2009b online).

  Listing 3.3: Argus Beispielkonfiguration

```
1 Group "Foo" {
2   hostname: foo.example.com
3         Service TCP/SMTP
4         Service TCP/HTTP
5         Service PING
6 }
```

  Das Listing 3.3 zeigt eine Minimalkonfiguration eines Argus Monitoring-Systems, das den Host `foo.example.com` überwacht. Speziell werden SMTP- und HTTP-Dienst sowie die Verfügbarkeit des Servers mittels PING ermittelt. Alle Tests sind als Perl Module realisiert und werden statisch in das monolithische System eingebunden.

- **Protokolle**
  Zur Basisfunktionalität von Argus gehört das SNMP Protokoll. Auf Basis von TCP und UDP kann Argus den Status von einer Reihe von Services überprüfen. Das System kann nur durch Erweiterung des aktuellen Source Codes mit weiteren Protokollen ausgestattet werden (vgl. Weisberg 2009b online).

- **Vor- und Nachteile**
  Das größte Problem des Systems liegt in der beschränkten Erweiterbarkeit durch seine monolithische Architektur. Alle Perl Module werden statisch und unabhängig von Konfiguration und Gebrauch in das Gesamtsystem eingebunden. Eine Erweiterung der Funktionalität ist somit ohne Veränderung des bestehenden Source Codes nicht möglich. Einen weiteren Nachteil bildet das eingeschränkte Notification System, das Benachrichtigun-

[31] http://www.opensource.org/licenses/artistic-license.php

gen nur per E-Mail erlaubt. Wie Nagios und Collectd ist Argus von einem Unix basierten Betriebssystem abhängig.

Die Analyse des Argus Monitoring-Systems basiert auf der Version 3.6 vom 27.10.2008. Weitere Details können der Homepage[32] oder dem aktuellen Source Code in Version 3.6 entnommen werden.

### 3.3.4 NeDi

NeDi steht für Network Discovery und ist ein leichtgewichtiges Management Framework, entworfen von Remo Rickli. Es begann 2006 mit ein paar Perl Skripten, um die MAC Adresstabellen von Routern auszulesen und die verbundenen Stationen anzuzeigen. Mittlerweile ist NeDi, unter der GNU General Public License entwickelt, zu einem Framework gewachsen, das die Analyse komplexer IT-Netzwerke ermöglicht (vgl. Rickli 2009b online).

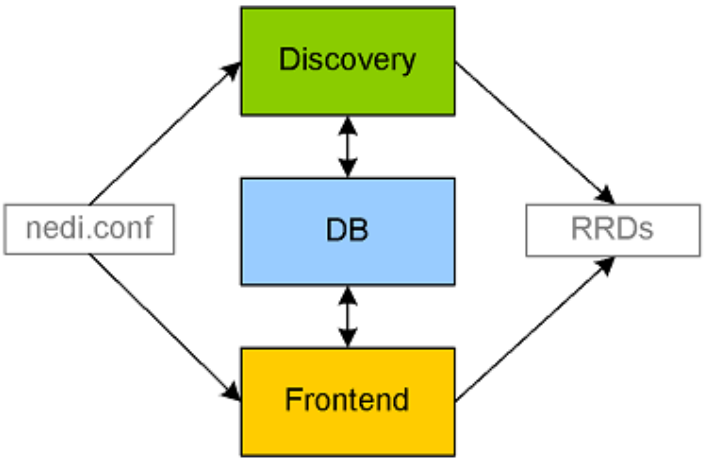



Abbildung 3.25: NeDi Architektur
(in Anlehnung an Rickli 2009a online)

- **Einsatzgebiet**
  Der Hersteller empfiehlt den Einsatz von NeDi für IT-Netwerke, die größten Teils aus Komponenten bestehen, welche das Cisco Discovery Protocol (CDP)[33] oder das Link Layer Discovery Protocol (LLDP)[34] unterstützen. Mit Hilfe dieser Protokolle wird die Topologie des Netzwerks ermittelt (vgl. Rickli 2009c online).

- **Architektur**
  Das Nedi Monitoring-System, in Abbildung 3.25 dargestellt, basiert auf einer Blackboard

[32] http://argus.tcp4me.com/
[33] http://www.cisco.com/en/US/docs/ios/12_1/configfun/configuration/guide/fcd301c.html
[34] http://standards.ieee.org/getieee802/download/802.1AB-2005.pdf

(Repository Variante) Architektur, die aus den Komponenten DB, Discovery und Frontend aufgebaut ist (vgl. Rickli 2009c online).

- **DB**
  Das DB Module ist eine SQL fähige Datenbank und stellt das Blackboard des Systems dar.
- **Discovery**
  Die Abbildung 3.26 zeigt die drei verschiedenen Perl Programme `moni.pl`, `nedi.pl` und `syslog.pl` des Discovery Moduls. `nedi.pl` ist das Hauptprogramm des Systems und dient zur Ermittlung aller verbundenen Netzwerkkomponenten des Zielnetzwerks. `moni.pl` wird verwendet um den Zustand der Netzwerkkomponenten zu ermittelt. Dabei wird nur überprüft, ob die Komponenten erreichbar ist. Wie der Name des `syslog.pl` bereits besagt, findet dieses Modul zum Ermitteln der Nachrichten an den jeweiligen Syslog-Schnittstellen Verwendung (vgl. Rickli 2009c online).

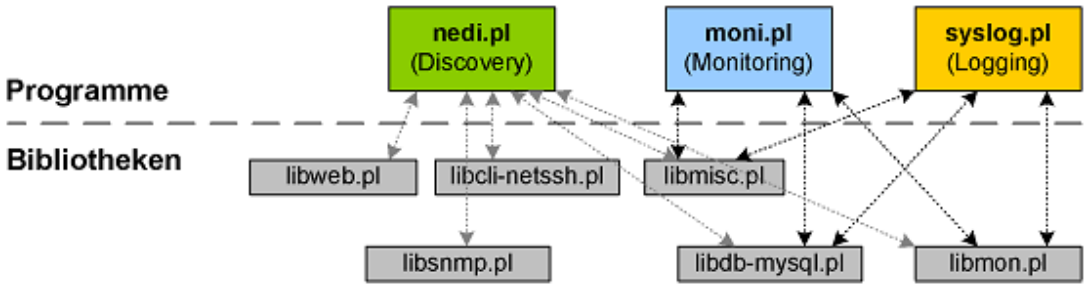



Abbildung 3.26: NeDi Programmaufbau
(in Anlehnung an Rickli 2009c online)

- **Frontend**
  Dieses Modul dient der Auswertung der gesammelten Daten. Hierbei können zum Beispiel Graphen des Netzwerks oder Statistiken über die Verfügbarkeit verschiedener Netzwerkkomponenten erstellt werden (vgl. Rickli 2009c online).

- **Besonderheiten**
  Vergleicht man NeDi mit den bereits vorgestellten Systemen aus den vorigen Unterkapiteln, fällt auf, dass NeDi keine ausführliche Konfiguration der zu überwachenden Komponenten benötigt. Die Komponenten des Systems werden automatisch ermittelt und überwacht. Diverse Einstellungen zu den Netzwerkkomponenten können nach dem ersten Discovery Vorgang, recht einfach durch das Frontend vorgenommen werden (vgl. Rickli 2009c online).

- **Protokolle**
  Wie bereits erwähnt, empfiehlt sich NeDi für den Einsatz bei CDP oder LLDP fähigen Netzwerkkomponenten. Diese zwei Protokolle bieten den größten Erfolg für das Discovery Modul beim Ermitteln der Netzwerktopologie. Als Alternative bietet NeDi die Mög-

lichkeit Organizationally Unique Identifiers (OUIs)[35] für zu überwachende Netzwerkkomponenten per Konfigurationsdatei anzugeben (vgl. Rickli 2009c online). SNMP wird zur Ermittlung von Details der Netzwerkkomponenten eingesetzt, für die jeweils SNMP Read Access benötigt wird (vgl. Rickli 2009d online).

- **Vor- und Nachteile**
  NeDi ist im Vergleich zu den bereits vorgestellten Systemen ein kleines, aber dafür leichtgewichtiges System. Darin liegt auch seine größte Stärke. Die von `syslog.pl` und `moni.pl` gesammelten Netzwerkdaten werden im Frontend übersichtlich dargestellt und ausgewertet. Dem gegenüber stehen jedoch einige Nachteile. Der wohl größte ist die fehlende Möglichkeit, Benachrichtigungen zu versenden. Ein weiteres Problem stellt die Erweiterbarkeit der Discovery Programme dar. Diese sind in Perl programmiert, verwenden eine Reihe von shared libraries und sind alle monolithisch aufgebaut.

Die Analyse des NeDi Management-System basiert auf der Version 1.0.4 vom 10.04.2009. Weitere Informationen können der Homepage unter der URL `www.nedi.ch` entnommen werden.

### 3.3.5 Fazit

Wie schon am Eingang zum Unterkapitel 3.3 erwähnt, lag der Fokus bei der Auswahl der vorgestellten Systeme in deren Architektur. Ihre Vor- und Nachteile derer wurden analysiert und dienen als Basis für Architekturentscheidungen beim Entwurf eines Monitoring-Systems für den XiTrust Business Server. Vergleichbar kleine Systeme wie NeDi und Argus wurden in die Analyse miteinbezogen, da diese gegenüber den "großen" Vertretern wie Nagios andere Architekturen verwenden.

Die Tabelle 3.9 bietet einen Überblick über die vorgestellten Systeme.

Nagios ist der wohl bekannteste Vertreter der Open Source Monitoring Tools. Seit der Entstehung hat sich eine große Community um das Produkt gebildet, die zur ständigen Weiterentwicklung des Systems beiträgt. Die Größe der Community kann im Open Source Umfeld als Gradmesser für die Verbreitung eines Produkts angesehen werden. Stöbert man in der Liste[36] der Firmen, die Nagios einsetzen, findet man große Namen wie Yahoo, Ubisoft, Texas Instruments, Toshiba oder DHL. Schenkt man den Informationen der Nagios Homepage Glauben, verwenden bereits mehr als 250 000 Benutzer das Nagios Monitoring-System. Ein weiterer Indikator für die Qualität eines Open-Source Produktes ist das Angebot von kommerziellen Supports. Nach kurzer Internetrecherche wird man schnell fündig und findet eine Reihe von Dienstleistungsanbietern, die Support um Nagios in ihrem Portfolio haben. Beispiele dafür sind

[35] `http://standards.ieee.org/regauth/oui/index.shtml`
[36] `http://users.nagios.org`

| System | Architektur | Protokolle | Vorteile | Nachteile |
|---|---|---|---|---|
| Nagios | Server-Agent<br>Event-Broker | SNMP, ICMP<br>SSH, LDAP | Programmiersprachenunabhängigkeit<br>vielfältige Einsatzmöglichkeiten<br>Open Source Lizenz<br>gute Erweiterbarkeit<br>große Community | benötigt Unix-basiertes Betriebssystem |
| Collectd | Microkernel | SNMP (Plugin) | gute Erweiterbarkeit<br>vielfältige Einsatzmöglichkeiten<br>Open Source Lizenz<br>schlanker Core | benötigt Unix-basiertes Betriebssystem<br>Plugins müssen in C geschrieben sein[a] |
| Argus | monolitisch | SNMP (im Core) | Open Source Lizenz | benötigt Unix-basiertes Betriebssystem<br>geringe Erweiterbarkeit<br>eingeschränktes Benachrichtigungssystem<br>Perl abhängig |
| NeDi | Blackboard | SNMP (im Core) | leichtgewichtiges System<br>Open Source Lizenz | keine Beanchrichtigungen<br>geringe Erweiterbarkeit<br>Perl abhängig |

Tabelle 3.9: Monitoring Tools

[a] Interpreter für Java und C in Entwicklung

GroundWork Open Source[37], op5[38] oder Würth Phoenix[39]. Die Popularität von Nagios bestätigt den guten Entwurf des Systems und die verwendeten Architekturen.

Auch CollectD hat einen hohen Grand an Popularität erreicht. So sind etwa auf der Homepage ca. 70 Plugins der Community veröffentlicht worden (vgl. Forster 2009a online). Ein weiterer Indikator für die Popularität ist der Umstand, dass andere Netwerk Monitoring-Systeme wie etwa OpenNMS Schnittstellen für CollectD bieten und den Daemon zum Sammeln von Systemdaten einsetzen (vgl. The OpenNMS Group 2009 online).

Der Fokus aller dieser Systeme liegt im Netzwerk-Monitoring. Die Funktionalität der Überwachung von Software-Applikationen beschränkt sich meist auf die Überprüfung des Zustands von Services oder Diensten. Daraus ergibt sich eine sehr eingeschränkte Einsetzbarkeit für den XiTrust Business Server (XBS). Die Analyse des Aufbaus der Systeme dient als Grundlage für den Entwurf eines geeigneten Monitoring-System für den XBS. Zusätzlich soll das Konzept des XBS Monitoring-Systems geeignete Schnittstellen zur Anbindung von Netzwerk Monitoring-Systemen per SNMP beinhalten.

[37] http://www.groundworkopensource.com
[38] www.op5.com/
[39] http://www.wuerth-phoenix.com

# 4 Das XBS Monitoring-System

Aus Kapitel 2.3 ging die Notwendigkeit eines Monitoring-Systems für den XiTrust Business Server hervor. Dieses soll Daten diverser Bundles sammeln (Datenakkumulation) und auswerten können (Visualisierungsengine). Eine entsprechende Datenschnittstelle zwischen Akkumulation und Visualisierung wird benötigt. In den nachfolgenden Unterkapiteln werden vorweg die Anforderungen und Erwartungen an dieses System analysiert. Anschließend werden in weiteren Unterkapiteln die Anforderungen von Datenakkumulation, Visualisierung und Schnittstelle analysiert sowie Konzepte für deren Umsetzung vorgestellt. Am Ende des Kapitels werden die Konzepte der drei Komponenten des XBS Monitoring-Systems zu einem Gesamtkonzept zusammengefügt und analysiert.

## 4.1 Anforderungen

Aus Kapitel 2.3 gingen sechs globale Anforderungen für das XBS Monitoring-System hervor, die zugleich die zeitliche Abfolge des Projekts darstellen. Diese Schritte sind:

1. Datenakkumulation
2. Visualisierung der Daten
3. Datenschnittstelle
4. Implementierung
5. Darstellung des Verlaufs eines Systemzustandes
6. Darstellung des Verlaufs von internen Prozessdaten

In diesem Kapitel werden die globalen Anforderungen an das Monitoring-System verfeinert. Spezielle Anforderungen für die Punkte 1-3 werden in nachfolgenden Unterkapiteln separat behandelt. Vorab werden alle Anwendungsfälle identifiziert und als Use Cases in UML 2 Notation dokumentiert. Diese soll das Sammeln der Anforderungen im Anschluss erleichtern (vgl. Leffingwell & Widrig 2003 S. 148).

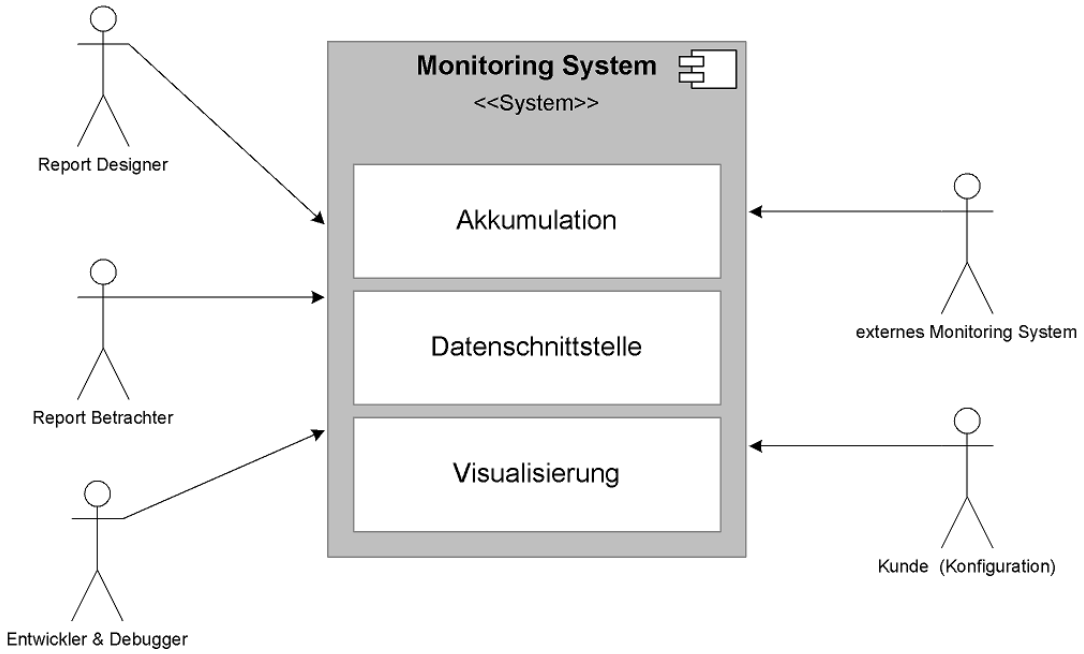



Abbildung 4.1: Use Cases des XBS Monitoring-Systems

Die Abbildung 4.1 zeigt das XBS Monitoring-System sowie die damit verbundenen Akteure in einem Systemkontextdiagramm (vgl. Oestereich 2006 S. 215). In der nachstehenden Auflistung werden die fünf sich ergebenden Use Cases beschrieben[1].

[1] Die Vorlage der Beschreibung stammt aus (Oestereich 2006) Seite 119

1. **Betrachtung von Reports**

| | |
|---|---|
| Kurzbeschreibung | Die vom Monitoring-System bereit gestellten Auswertungen von internen Daten werden im Client dargestellt |
| Akteure | Entwickler, Kunde, Debugger |
| Auslöser | Einer der Akteure möchte einen Report ansehen |
| Vorbedingungen | 1. Alle zur Darstellung der Daten benötigten Reporting Adaptoren müssen registriert sein<br>2. Es muss eine Verbindung zum XBS bestehen<br>3. Der Benutzer verfügt über die entsprechenden Rechte |
| Eingehende Informationen | Name und Pfad der Report Datei |
| Ergebnisse | Darstellung des Reports |
| Nachbedingungen | keine |
| Ablauf | 1. Der Anwender wählt einen gewünschten Report aus um ihn darzustellen<br>2. Die Visualisierungsengine ermittelt die benötigten Daten<br>3. Die Visualisierungsengine beschafft sich die Daten von der Datenschnittstelle<br>4. Die Visualisierungsengine stellt den Report in der gewünschten Form dar |

2. **Erstellung von Reports**

| | |
|---|---|
| Kurzbeschreibung | Erstellung von Reports durch XiTrust Mitarbeiter |
| Akteure | Entwickler |
| Auslöser | Es wird ein neuer Report benötigt |
| Vorbedingungen | 1. Alle zum Design des Reports benötigten Reporting Adaptoren müssen registriert sein<br>2. Es muss eine Verbindung zum XBS bestehen |
| Eingehende Informationen | |
| Ergebnisse | Eine neue Report Datei |
| Nachbedingungen | Ein neuer Report wurde erstellt |
| Ablauf | 1. Der Entwickler erstellt einen neuen Report in einer Design Umgebung<br>2. Der Entwickler erstellt eine Verbindung zum XBS<br>3. Der Entwickler wählt die entsprechenden Datensätze und Darstellungsform aus. |

3. **Debugging des XBS**

| | |
|---|---|
| Kurzbeschreibung | Die Entwickler nutzen die bereitgestellten Reports zur Analyse des Systems |
| Akteure | Entwickler |
| Auslöser | Analyse oder Debugging des Systems |

Der Rest der Use Case Beschreibung ist identisch mit dem Use Case - "Darstellung von Reports" aus Punkt 1.

4. **Konfiguration des Monitoring-Systems**

| | |
|---|---|
| Kurzbeschreibung | Das Monitoring-System wird konfiguriert |
| Akteure | Entwickler, Kunde, Debugger, Administrator |
| Auslöser | Einer der Akteure möchte das System konfigurieren |
| Vorbedingungen | 1. Es muss eine Verbindung zum XBS bestehen<br>2. Der Benutzer verfügt über die entsprechenden Rechte |
| Eingehende Informationen | |
| Ergebnisse | |
| Nachbedingungen | Das System übernimmt die Einstellungen des Clients |
| Ablauf | Der Anwender wählt die gewünschten Parameter |

5. **Monitoring durch ein externes System**

| | |
|---|---|
| Kurzbeschreibung | Ein externes Monitoring-Tool dockt an eine Schnittstelle des XBS Monitoring-Systems an |
| Akteure | externes Monitoring-System |
| Auslöser | Ein externes Monitoring-System möchte den XBS überwachen |
| Vorbedingungen<br>Eingehende Informationen | Die entsprechende Schnittstelle muss verfügbar sein |
| Ergebnisse | |
| Nachbedingungen | |
| Ablauf | 1. Das externe System dockt an eine Schnittstelle des XBS Monitoring-Systems an<br>2. Das externe System frägt Daten des Systems ab |

Bereits aus der Beschreibung des ersten Use Case kann eine Reihe von Anforderungen abgeleitet werden. Es gilt nun, zusätzlich mit den weiteren vier Use Cases, die Anforderungen zu definieren. Ziel ist es, diese zu identifizieren und möglichst alle Erwartungen von Kunde und Entwickler abzudecken (vgl. Bjorner 2006 S. 368). Die Verbindlichkeit der Anforderungen folgt der Einteilung nach (Oestereich 2001) in Pflichtanforderungen, optionale Anforderungen,

Absichten und Vorschläge (vgl. Oestereich 2001 S. 118). Die nachstehende Liste zählt alle globalen Anforderungen an das XBS Monitoring-System auf:

1. **Ziele**
   Die Ziele des Projekts wurden bereits im Eingang diese Kapitels erwähnt. Die nachfolgende Auflistung zählt diese noch einmal auf und weist ihnen die entsprechende Verbindlichkeit zu.

   - ZIEL 1: Datenakkumulation `(Pflichtanforderung)`
   - ZIEL 2: Visualisierung der Daten `(Pflichtanforderung)`
   - ZIEL 3: Datenschnittstelle `(Pflichtanforderung)`
   - ZIEL 4: Implementierung `(Pflichtanforderung)`
   - ZIEL 5: Darstellung des Verlaufs eines Systemzustandes `(Pflichtanforderung)`
   - ZIEL 6: Darstellung des Verlaufs von internen Prozessdaten `(Pflichtanforderung)`

2. **Funktionale Anforderungen**

   - FUNKANF 1: Das System muss Daten verschiedenen Ursprungs akquirieren können. Die funktionellen Anforderungen der Datenakkumulation werden in Kapitel 4.4.2 detailliert spezifiziert `(Pflichtanforderung)`
   - FUNKANF 2: Das System muss die akkumulierten Daten in verschiedenen Formaten darstellen können. Die funktionellen Anforderungen der Datenvisualisierung werden in Kapitel 4.5.1 detailliert spezifiziert `(Pflichtanforderung)`
   - FUNKANF 3: Das System muss eine abstrakte Datenschnittstelle anbieten, die eine Anlaufstelle für den Datenaustausch mit verschiedenen Applikationen darstellt. Die funktionellen Anforderungen dieser Schnittstelle werden in Kapitel 4.6.1 detailliert spezifiziert `(Pflichtanforderung)`

3. **Qualitätsanforderungen**

   - QUALANF 1: Das Monitoring-System darf die Kombinierbarkeit zu Produkten in keiner Weise beeinträchtigen `(Pflichtanforderung)`
   - QUALANF 2: Das Monitoring-System darf die hohe Erweiterbarkeit des Systems in keiner Weise einschränken `(Pflichtanforderung)`
   - QUALANF 3: Die Qualitätsanforderungen nach ISO/IEC 9126 müssen bestmöglich erfüllt werden. Die Hauptmerkmale sind Funktionalität, Benutzbarkeit, Effizienz, Änderbarkeit und Übertragbarkeit `(Pflichtanforderung)`

4. **Randbedingungen**

   - RANDBED 1: Alle in RFCs standardisierten Protokolle, die zum Einsatz kommen, müssen gegenüber diesen RFCs konform sein `(Pflichtanforderung)`

5. **Dokumentationsanforderungen**

   - DOKUANF 1: Use Cases müssen dokumentiert werden `(Pflichtanforderung)`
   - DOKUANF 2: Die Anforderungen müssen dokumentiert werden `(Pflichtanforderung)`
   - DOKUANF 3: Die Konzepte hinter Datenakkumulation, Visualisierung und Schnittstelle sowie zu Grunde liegende Design-Entscheidungen müssen entsprechend dokumentiert werden `(Pflichtanforderung)`
   - DOKUANF 4: Ein Ausblick über weiterführende Arbeit oder mögliche Erweiterungen sowie Vorschläge zur Umsetzung müssen dokumentiert werden `(Pflichtanforderung)`

6. **Abgeleitete Anforderungen aus den Analysen aus Kapitel 3**

   - ABGANF 1: Vorsehung einer SNMP Schnittstelle `(Pflichtanforderung)`
   - ABGANF 2: Vorsehung einer Syslog Schnittstelle `(Pflichtanforderung)`
   - ABGANF 3: Implementierung von aktiver und passiver Datengewinnung unter Verwendung der Nagios Event Broker Funktionalität `(Pflichtanforderung)`
   - ABGANF 4: Verwendung geeigneter Architekturen um klare Strukturierung und Erweiterbarkeit zu gewährleisten `(Pflichtanforderung)`

   Die Use Case Analyse war der Grundstein für die Identifizierung der globalen Anforderungen. Diese werden in weiteren Unterkapiteln über Datenakkumulation, Visualisierung und Schnittstelle weiter verfeinert. Die Konzepte für diese drei Teilsysteme des XBS Monitoring-Systems müssen diese Anforderungen entsprechend deren Verbindlichkeit berücksichtigen.

## 4.2 Erwartungen

Neben den Anforderungen an das XBS Monitoring-System, die zugleich Erwartungen widerspiegeln, gibt es eine Fülle von weiteren erwarteten Outputs, die bereits Helmut Aschbacher in seiner Masterarbeit ausgearbeitet hat. Eine Kernaussage seiner Arbeit ist: "Je höher die Wahrnehmung der Leistung eines Service, desto höher ist die Zahlungsbereitschaft des Kunden für dieses Service" (vgl. Aschbacher 2009 S. 14). Daraus folgernd, erwartet sich die Firma XiTrust eine erhöhte Zahlungsbereitschaft der Kunden durch eine geeignete Visualisierung der internen Vorgänge des XBS.

(Aschbacher 2009) ermittelt im weiteren Verlauf seiner Arbeit zusätzlichen Nutzen sowohl für Kunde als auch für die Firma XiTrust durch ein XBS Monitoring-System. Er bezeichnet dieses in seiner Arbeit als "XBS Status Monitor" (vgl. Aschbacher 2009 S. 113). Die Vorzüge wurden bereits in Kapitel 2.3 kurz angesprochen. Die Tabelle 4.1 thematisiert diese noch einmal.

| Kundennutzen | Herstellernutzen |
|---|---|
| Internes Argumentationsinstrument | Weniger Support |
| Leistungsvisualisierung | Datenauswertung |
| gibt Sicherheit | Update ist klarer kommunizierbar |

Tabelle 4.1: Nutzen des XBS Monitoring-Systems
(in Anlehnung an Aschbacher 2009 S. 113)

Greift man einige dieser Punkte heraus, bringen sie weitere Vorteile eines solchen Monitoring-Systems mit sich, wie die Datenauswertung auf der Entwicklerseite, die Möglichkeit von präventiven Eingriffen oder die Erkennung von Problemen vor dem Eintreten desselben. Dies bietet ein zusätzliches Verkaufsargument für Updates und Erweiterungen für die Firma XiTrust.

Die Auswertung und effektive Darstellung von Daten ermöglicht ein schnelleres Eingreifen bei Problemfällen, besonders das Auffinden der Ursache von Problemen kann dadurch beschleunigt werden.

Ein weiterer wertvoller Nebeneffekt der Datenauswertung ist die Erfassung des Kundenverhaltens. Dadurch können Erweiterungen und Verbesserungen des Systems gezielter vorgenommen werden.

## 4.3 Übersicht

In diesem Unterkapitel wird die Architektur des XBS Monitoring-Systems vorgestellt. Die Abbildung 4.2 zeigt die involvierten Komponenten und deren strukturelle Lage in den Schichten des Systems. Die Komponenten von unten nach oben im Detail:

- **XBS Bundles**
  In der untersten Schicht befinden sich die zu überwachenden Bundles, die aus einem oder mehreren Plugins aufgebaut sind.
- **Maggie & Core**
  Maggie ist der interne Projektname des XBS Monitoring-Systems und unter anderem für das Akkumulieren der Daten verantwortlich. Zusätzlich bietet das Maggie Bundle Platz für die Implementierung von SNMP oder Notification Funktionalität. Mit dem EventBroker wird ein Teil der Datenakkumulation in den Kern des XBS ausgelagert.
- **Open Data Access (ODA) XBS Implementierung**
  Eine Implementierung der ODA Interfaces für den XBS dient als Schnittstelle zwischen Akkumulation und Visualisierung.
- **Visualisierung**
  Die Visualisierung der Daten basiert auf Eclipse BIRT Reports. Zur Erstellung und zum Design der Reports kann die Eclipse IDE verwendet werden. Zur Darstellung von Reports

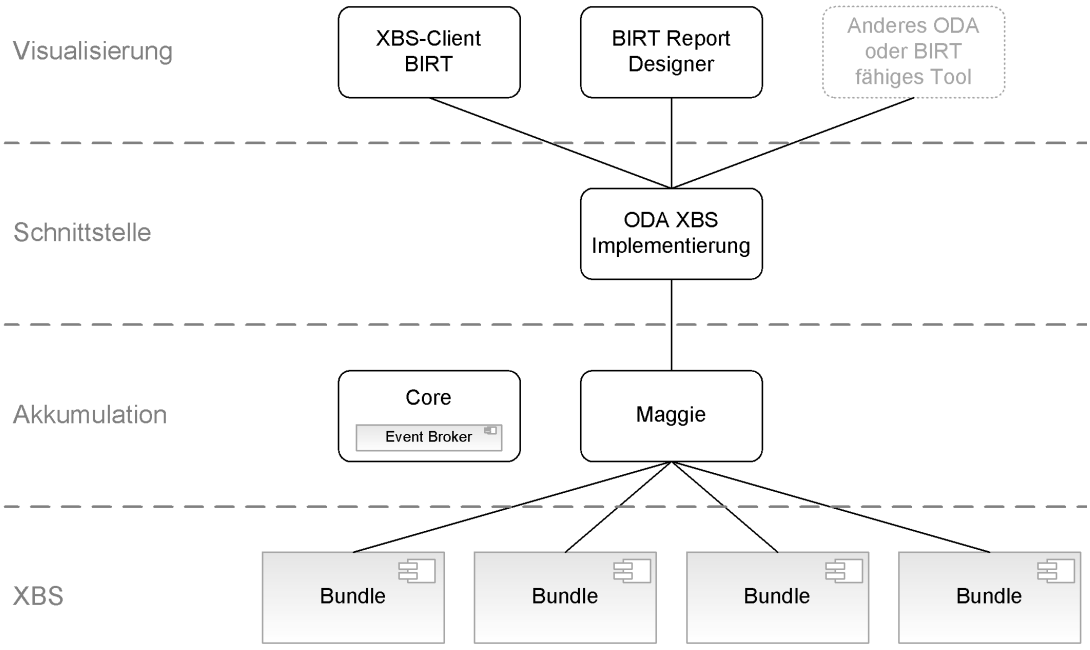



Abbildung 4.2: XBS Monitoring-System Architektur

kann jede Applikation mit einer BIRT Engine eingesetzt werden, die durch Verwendung der ODA XBS Schnittstelle Zugriff auf die Daten des XBS hat.

Die Konzepte von Akkumulation, Visualisierung und Schnittstelle werden in den nachfolgenden Unterkapiteln genauer vorgestellt.

## 4.4 Datenakkumulation

Die Datenakkumulation soll Daten des XBS akquirieren können. Für Benutzer und Administrator sind weitere Parameter aus dem "Umfeld" des XBS interessant. In diese Umfeld fallen Daten zum Betriebssystem oder zur Hardware des Systems. Das Unterkapitel 4.4.1 befasst sich vorweg mit der Analyse vorhandener Daten und deren Akquirierung. Basierend auf dieser Analyse und einer Use Case Analyse werden im Unterkapitel 4.4.2 die Anforderungen an die Datenakkumulation ermittelt. Diese wiederum bilden den Grundstein für das Konzept, das in Unterkapitel 4.4.3 vorgestellt wird.

### 4.4.1 Analyse der Daten

Für die Konzeption der Datenakkumulation ist eine vorhergehende Analyse der Daten notwendig. Sie soll Klarheit über zwei wesentliche Fragen bringen:

1. *Welche Daten sind interessant für Entwickler und Kunde?*

2. *Wie können diese Daten akquiriert werden?*

Die nachstehende Auflistung zählt interessante Daten sowie deren Quelle und Akquirierungsmöglichkeiten auf.

- **System**

| Typ | Parameter | Akquirierung | Interesse |
|---|---|---|---|
| Allgemein | Systemname, Domain | java.net | Entwickler & Kunde |
| | Zeit, Zeitzone | java.util | Entwickler & Kunde |
| CPU | Auslastung, Takt | | Entwickler & Kunde |
| | Typ, Anzahl | JMX | Entwickler |
| Speicher | Auslastung | JMX | Entwickler & Kunde |
| | Größe, Auslagerung | JMX | Entwickler |
| Festplatte | Auslastung | java.io | Entwickler & Kunde |
| | Anzahl, Größe, Partitionen | java.io | Entwickler |
| Netzwerk | Auslastung, Geschwindigkeit | | Entwickler & Kunde |
| | Geräte | java.net | Entwickler |

Tabelle 4.2: Systemdaten

Systemname und Domain können mit Hilfe der Klasse `java.net.InetAddress` ausgelesen werden (vgl. Sun Microsystems 2006e online). Für Zeit und Zeitzone kann die Klasse `java.util.Calendar` verwendet werden (vgl. Sun Microsystems 2006b online).

Bei den Daten der CPU verhält es sich nicht mehr so einfach. Zwar können Typ und Anzahl der Kerne der CPU durch das `OperatingSystemMXBean` ermittelt werden (vgl. Sun Microsystems 2006f online), die Auslastung dagegen kann nicht problemlos ermittelt werden. Grund dafür ist das Fehlen einer einheitlichen Schnittstelle für das Abfragen der Systemauslastung. Besonders Windows weigert sich noch, eine solche API anzubieten (vgl. Sun Developer Network 2005 online).

Per `OperatingSystemMXBean` lassen sich ebenfalls Auslastung, Größe des Hauptspeichers und Swap Bereichs sowie deren Auslastung ermitteln.

Partition, deren Größe und Auslastung können per `java.io.File` Klasse ermittelt werden (vgl. Sun Microsystems 2006d online). Die Kalkulation der Auslastung erfolgt durch Summierung der Dateigrößen. Dies kann unter Umständen sehr zeitaufwändig und rechenintensiv sein.

Die vorhandenen Netzwerkadapter können per `java.net.InetAddress` ermittelt werden (vgl. Sun Microsystems 2006e online). Für Auslastung und Geschwindigkeit bietet Java keine Lösung an.

Die Werte dieser Kategorie sind vorwiegend für den Entwickler und Systemadministratoren von Bedeutung. Sie bilden oft die Grundlage für Entscheidungen im Hardwarebereich oder liefern wertvolles Feedback bei der Analyse der Performance des XBS.

| Typ | Parameter | Akquirierung | Interesse |
|---|---|---|---|
| Allgemein | Name, Version, Architektur | JMX | Entwickler & Kunde |
| Dienste | Anzahl, Status, Auslastung | | Entwickler |
| Anwendungen | Anzahl, Status, Auslastung | | Entwickler |
| Prozesse | Anzahl, Status, Auslastung | | Entwickler |
| Threads | Anzahl, Status, Auslastung | | Entwickler |

Tabelle 4.3: Betriebssystemdaten

- **Betriebssystem**

  Betriebssystem Daten spielen wiederum haupsächlich für Systemadministratoren und Entwickler eine Bedeutung. Für Akquirierung dieser Daten stellt java keine Implementierung zur Verfügung. Lediglich Name, Version und Architektur des Betriebssystems können per `OperationSystemMXBean` akquiriert werden (vgl. Sun Microsystems 2006f online).

- **Java Virtual Machine**

| Typ | Parameter | Akquirierung | Interesse |
|---|---|---|---|
| Klassen | Anzahl, Loaded, Unloaded | JMX | Entwickler |
| Threads | Anzahl, Status, Auslastung | JMX | Entwickler |
| Speicher | Typen, Pools, Auslastung | JMX | Entwickler |

Tabelle 4.4: Java Virtual Machine Daten

  Für alle diese Werte bietet JMX die entsprechenden Beans. Daten zu Klassen können per `ClassLoadingMXBean` ermittelt werden (vgl. Sun Microsystems 2006c). Das `ThreadMXBean` bietet eine Hülle von verschiedenen Informationen über die Threads der JVM (vgl. Sun Microsystems 2006g online).

- **XiTrust Business Server**

| Typ | Parameter | Akquirierung | Interesse |
|---|---|---|---|
| Allgemein | Version, Plugins, Features | Monitoring | Entwickler & Kunde |
| Workflows | Ausführungen , Fehler | Monitoring | Entwickler & Kunde |
| Benutzer | Anzahl, Aktivität | Monitoring | Entwickler & Kunde |
| Zertifikate | Verwendungen, Fristen | Monitoring | Entwickler & Kunde |
| E-Mail | Anzahl, Spam, abgewiesen,<br>zugestellt | Monitoring<br>Monitoring | Entwickler & Kunde<br>Entwickler & Kunde |

Tabelle 4.5: Daten des XiTrust Business Server

  Die Tabelle 4.5 enthält einen Auszug von Daten des XBS. Welche Daten vom XBS Monitoring-System erfasst werden können, ist von der jeweiligen Bundle-Zusammenstellung abhängig.

Für alle Werte, die nicht direkt oder nur schwer durch Java ermittelt werden können, kann der Umweg über Bibliotheken oder Programme, basierend auf anderen Programmiersprachen mit

entsprechenden Interfaces, gegangen werden.

### 4.4.2 Anforderungen

In Kapitel 2.2.1 wurden die zwei Hauptanforderungen bei der Entwicklung des E3 Servers vorgestellt. Diese waren die freie Kombinierbarkeit zu Produkten und die hohe Erweiterbarkeit um kundenspezifische Lösungen anbieten zu können. Da die Datenakkumulation ein zentrales System des E3 darstellen wird und viele Bundles zu erweitern sein werden, muss bei der Konzeption besonders darauf geachtet werden, dass diese zwei Anforderungen weiterhin erfüllt bleiben.

Zusätzlich zu den zwei Hauptanforderungen ergeben sich eine Reihe von weiteren Anforderungen an das XBS Monitoring-System:

1. **Ziele**
   - ZIEL 1.1: Die Datenakkumulation soll in der Lage sein, Daten von verschiedenen Stellen (Bundles) im XBS zu sammeln `(Pflichtanforderung)`
2. **Funktionale Anforderungen**
   - FUNKANF 1.1: Die Daten sollen aktiv und passiv akkumuliert werden können. Mit aktiv ist das direkte Abfragen von Daten gemeint, vergleichbar mit der SNMP `get` Funktionalität. Dem entsprechend ist passiv mit der SNMP `trap` Operation gleichzusetzen. Das überwachte Bundle soll auch ohne Aufforderungen Daten an das Monitoring-System senden und es so über interne Ereignisse benachrichtigen können. `(Pflichtanforderung)`
   - FUNKANF 1.2: Unterschiedliche Daten verlangen nach verschiedenen Arten der Persistenz. Das System soll unterschiedliche Persistenz-Lösungen unterstützen `(Pflichtanforderung)`
   - FUNKANF 1.3: Die Datenakkumulation soll durch den XBS-Client konfigurierbar sein `(Pflichtanforderung)`
3. **Qualitätsanforderungen**
   - Die Qualitätsanforderungen gelten sinngemäß denen der globalen Anforderungen QUALANF 1, QUALANF 2 und QUALANF 3 aus Kapitel 4.1

### 4.4.3 Konzept

Um die Erweiterbarkeit des Systems hoch zu halten, wird eine Snap-In Lösung, ähnlich der Nagios oder CollectD Plugins gewählt. Die Analyse dieser Systeme ergab eine hohe Erweiterbarkeit, basierend auf Plugins (vgl. Unterkapitel 3.3.5).

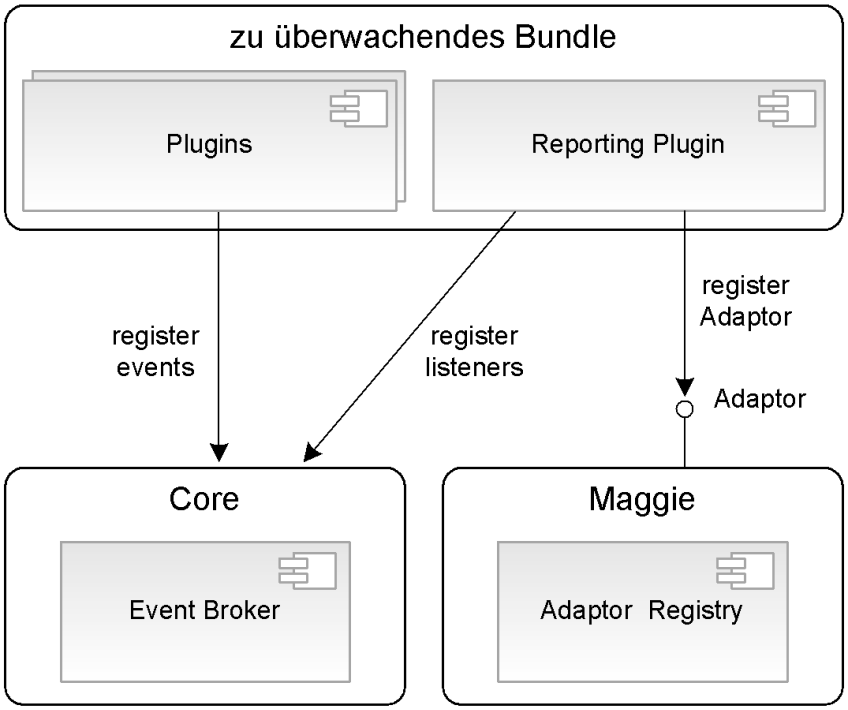


Abbildung 4.3: Maggie Datenakkumulation Architektur

Die Abbildung 4.3 zeigt ein Architekturdiagramm der Datenakkumulation. Die Komponenten des Diagramms im Detail sind:

- **Plugins**
  Wie bereits in Kapitel 2.2.2 erwähnt, können XBS E3 Bundles aus mehreren Plugins bestehen. Die interessantesten Daten bietet meist das Server Plugin.

- **Reporting Plugin**
  Das Reporting Plugin, auch Reporting Adaptor genannt, stellt das Snap-In des zu überwachenden Bundles dar. Es erweitert das Bundle um die Reporting Funktionalität und registriert seine Listener beim Event Broker und den Reporting Adaptor am entsprechenden Extension Point des Maggie Bundles. Die Listener des Reporting Plugins sind in der Lage sich für Events des überwachten Plugins zu registrieren. Dadurch kann es an Daten des überwachten Plugins auf dem passiven Weg gelangen. Direktes Abfragen von Werten ist ebenfalls möglich (aktiver Weg). Das Reporting Plugin ist für das Speichern relevanter Daten selbst verantwortlich. Durch diesen Umstand besteht die Möglichkeit der Verwendung verschiedener Persistenz Mechanismen für verschiedene Arten von Daten.

- **Maggie & Core**
  Das Maggie Bundle ist das Herzstück des Monitoring-Systems für den XBS. Es bietet einen Extension Point für Reporting-Plugins und die dazugehörige Adaptor Registry an. Die zweite wichtige Komponente der Akkumulation ist der Event Broker. Dieser wird in den XBS Kern ausgelagert. Beide Komponenten werden im Verlauf des Unterkapitels noch detailliert vorgestellt. Die Abbildung 4.3 zeigt nur die für die Datenakkumulation

relevanten Komponenten des XBS Monitoring-Systems. Das Gesamtkonzept wird in Unterkapitel 4.7 detailliert vorgestellt.

Eine Reihe von Überlegungen wurde im Vorfeld des Designs angestellt. Die Anforderung, die Anzahl hinzukommender Abhängigkeiten möglichst gering zu halten, um auch weiterhin die freie Kombinierbarkeit des Systems zu gewährleisten, machte die Konzeptionierung eines so zentralen Systems wie der Datenakkumulation nicht einfach.

Um die freie Kombinierbarkeit weiterhin zu ermöglichen, darf die zentrale Akkumulation keine Abhängigkeiten auf die zu überwachenden Bundles besitzen. Wäre das der Fall, müssten alle Bundles, zu denen das Monitoring-System eine Abhängigkeit hat, im Produkt mit ausgeliefert werden. Diese kann durch Reporting Plugins und dem Eclipse Extension Point Mechanismus gelöst werden. Zusätzlich muss es möglich, sein Produkte gänzlich ohne Monitoring-System auszuliefern. Aus diesem Grund darf der Kern und die zu überwachenden Bundles keine Abhängigkeit auf das Monitoring-System bekommen. Dies wird durch den Event Broker, der in den Kern ausgelagert wird, ermöglicht.

Ein Vorteil der gewählten Snap-In Lösung bildet die niedrige arithmetische Komplexität des Systems. Wie bereits in Unterkapitel 2.2.3 erwähnt, sinkt diese durch einen steigenden Grad an Modularisierung. Die Implementierung der Reporting Plugins, die für viele bestehende Bundles notwendig wird, gestaltet sich dadurch verhältnismäßig einfach.

Ein weiterer Vorteil der Snap-In Lösung bietet der sich aus der Analyse der Monitoring-Systeme ergebende Umstand, dass das Monitoring-System sehr genau an das jeweilige Produkt angepasst werden kann (vgl. Vorteile von CollectD in Kapitel 3.3.2).

Das Reporting Plugin hat eine direkte Abhängigkeit auf das überwachte Plugin und kann dadurch Daten des Plugins direkt (aktive Datengewinnung) akquirieren.

Event Broker und Adaptor Registry stellen die wichtigsten Komponenten der Datenakkumulation dar. Die Abbildung 4.4 zeigt den Aufbau des Event Brokers, der für die passive Gewinnung der Daten verantwortlich ist.

Zu überwachende Plugins registrieren von ihnen auslösbare Events unter Angabe der konkreten `Event` Klasse beim `EventBroker`. Dieser ist als Singleton[2] im Service Core Bundle instanziert. Reporting Plugins wiederum registrieren ihre Listener mittels konkreter `Listener` Instanz für jede `Event` Klasse beim `EventBroker`. Dieser verwaltet die `Events` und die dazugehörigen `Listener` in einer zwei-dimensionalen Liste. In einer weiteren Liste verwaltet er nur die aktiven `Listener`. Grund dafür ist die Anforderung[3], Listener zur Laufzeit aktivieren und deaktivieren zu können.

Löst ein überwachtes Plugin ein `Event` aus, ruft es die `notify` Funktion des `EventBroker`

[2] Singleton Design Pattern (Gamma *et al.* 1995 S. 127-134)
[3] abgeleitet aus Anforderung FUNKANF 1.3

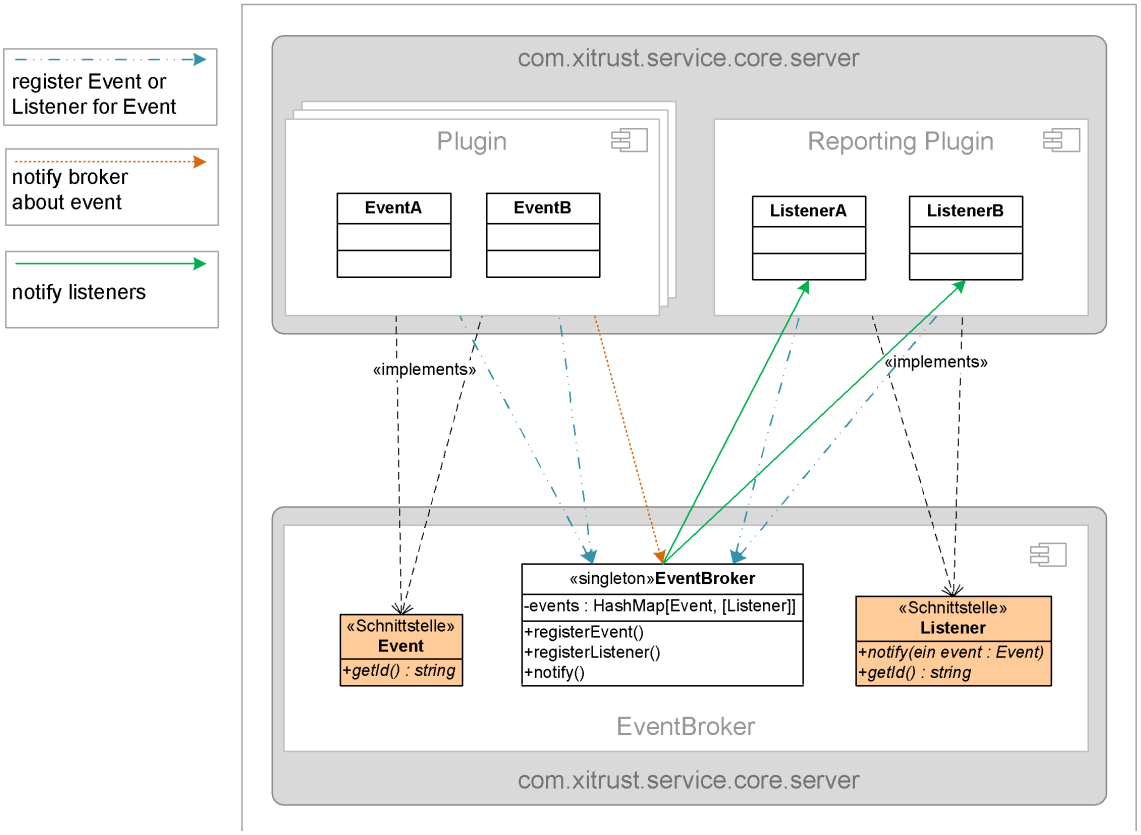



Abbildung 4.4: Maggie Event Broker

auf. Diese Funktion aktiviert nun alle `Listener` durch Aufruf deren `notify` Funktion und übergibt dieser das ausgelöste `Event`.

Die Grundidee des Event Brokers wurde vom Nagios Monitoring-System übernommen, aber im XBS Monitoring-System um eine zusätzliche Funktionalität erweitert.

Der Notify Mechanismus benachrichtigt nicht nur `Listener`, die sich für das konkrete `Event` registriert haben, sondern auch alle, die sich für eine Super Klasse oder Super Interface des ausgelösten `Event` registriert haben. Die Abbildung 4.5 zeigt eine beispielhafte `Event` Object-Hierarchie und die dazugehörige Liste des `EventBrokers`. Wird das `EventB2` ausgelöst, werden der `ListenerC`, aber auch `ListenerA` und `ListenerB` aktiviert, da diese sich für die Superklasse `EventB` registriert haben. Dieser Mechanismus erlaubt das gleichzeitige Registrieren eines `Listener` für alle Events eines Interfaces oder Superklasse ohne alle konkreten Implementierungen kennen zu müssen.

Grund für die Auslagerung in den Kern ist der bereits erwähnte Umstand, auch Produkte ohne Monitoring-System ausliefern zu können. Wäre er Teil von Maggie selbst, würden alle Bundles ihre Events dort registrieren, was eine unerwünschte Abhängigkeit mit sich bringt, die die freie Kombinierbarkeit einschränken würde.

Die zweite wichtige Komponente der Datenakkumulation ist die Adaptor Registry. Sie ist in Ab-

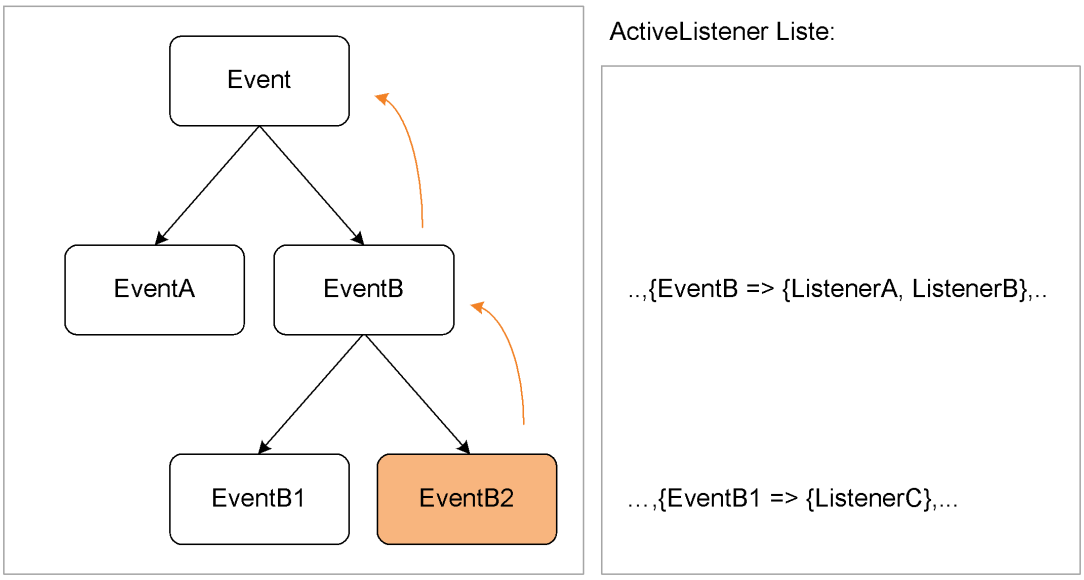



Abbildung 4.5: Maggie Event Broker Notify Mechanismus

bildung 4.6 dargestellt und dient zur Verwaltung der Reporting Plugins. Wird Monitoring Funktionalität für ein Bundle benötigt, muss dieses mit einem Reporting Plugin ausgestattet werden, das einen entsprechenden konkreten `ReportingAdaptor` zur Verfügung stellt. Dieser muss das `ReportingAdaptor` Interface implementieren und wird am entsprechenden Extension Point der Datenakkumulation mit einer eindeutigen Id registriert. Die `AdaptorRegistry` verwaltet alle registrierten Adaptoren und stellt so den zentralen Angriffspunkt für die Datenschnittstelle dar. Jeder konkrete `ReportingAdaptor` bietet eine Reihe von Datensätzen (`DataSet`) an. Jeder Datensatz wird durch eine eindeutige Id identifiziert. Die Daten können unter Angabe der Id des `Adaptors` und Id des `DataSets` bei der `AdaptorRegistry` abgeholt werden. Eine zusätzlich Angabe von Parameter ist ebenfalls möglich.

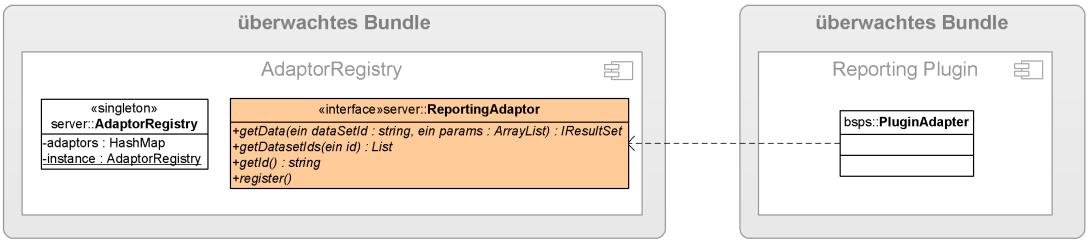



Abbildung 4.6: Maggie Reporting Adaptor Registry

Wie bereits in diesem Unterkapitel erwähnt, bleibt durch diesen Mechanismus das System frei von unnötigen Abhängigkeiten. Die Adaptor Registry muss, begünstigt durch den Eclipse Extension Point Mechanismus, die konkreten `ReportingAdaptor` Implementierungen nicht kennen. Die Adaptor Registry kann verwendet werden um verfügbare Reporting Plugins und Datensätze abzufragen.

Die Persistierung der Daten wird dem jeweiligen Reporting Plugin, basierend auf der Art der zu speichernden Daten selbst überlassen. Es gibt also keine globale Datenbank wie etwa in

einer Repository Architektur. Speichert ein Reporting-Plugin zyklisch die CPU Auslastung, ist diese nur für einen bestimmten Zeitraum interessant. Für die Persistierung bietet sich entweder der Speicher selbst oder eine Round Robin Datenbank Lösung an. Müssen Daten über einen längeren Zeitraum oder große Mengen von Daten gespeichert werden, bieten größere Datenbanksysteme wie etwa Microsoft SQL Server[4], Mysql[5] oder Oracle[6] entsprechende Lösungen an.

## 4.5 Visualisierungsengine

Die Visualisierungsengine übernimmt die Darstellung der akkumulierten Daten in Form von Reports. Diese stellen die Daten in geeigneter Form, beispielsweise als Diagramme oder Auflistungen dar.

### 4.5.1 Anforderungen

Die globalen Anforderungen an das XBS Monitoring-System implizierten keine weiteren allgemeinen Anforderungen für die Visualisierung. Die speziellen Anforderungen werden in der nachfolgenden Auflistung aufgezählt:

1. **Ziele**
   - ZIEL 2.1: Die Visualisierungsengine muss in der Lage sein Reports des XBS darstellen zu können `(Pflichtanforderung)`
2. **Funktionale Anforderungen**
   - FUNKANF 2.1: Die Darstellung der Reports soll in unterschiedlichen Formaten möglich sein. Die Darstellung in PDF und HTML Format sind die Mindestanforderungen. Jedes weitere Format würde die Qualität des Systems erhöhen und zusätzliche Verkaufsargumente bringen `(Pflichtanforderung)`
   - FUNKANF 2.2: Die Darstellung der Reports muss im XBS Client möglich sein `(Pflichtanforderung)`
   - FUNKANF 2.3: Für die Erstellung der Reports soll es einen graphischen Editor mit entsprechender Vorschaufunktionalität gegeben sein `(optionale Anforderung)`

[4] http://www.microsoft.com/germany/sql/2008/default.mspx
[5] http://www.mysql.de/products/
[6] http://www.oracle.com/lang/de/database/index.html

3. **Qualitätsanforderungen**

   - Die Qualitätsanforderungen gelten sinngemäß denen der globalen Anforderungen QUALANF 1, QUALANF 2 und QUALANF 3 aus Kapitel 4.1

4. **Randbedingungen**

   - RANDBED 2.1: Verwendung eines bestehenden Reporting Frameworks. Man will in diesem Punkt auf etablierte Lösungen zurückgreifen. Die Entwicklung eines komplett eigenen Reporting Frameworks würde den Rahmen des Projekts sprengen `(Pflichtanforderung)`
   - RANDBED 2.2: Das Dateiformat der Reports muss Plattform-unabhängig sein `(Pflichtanforderung)`
   - RANDBED 2.3: Die Darstellung der Reports muss Plattform-unabhängig gegeben sein `(Pflichtanforderung)`

### 4.5.2 Reporting Frameworks - BIRT versus Jasper Reports

Bevor die Konzeption der Visualisierungsengine beginnen kann ist es notwendig, verbreitete Reporting Frameworks zu untersuchen. Die beiden bekanntesten Vertreter der Java Welt sind Jasper Reports[7] und BIRT[8].

Die Tabelle 4.6 zeigt eine Gegenüberstellung der beiden Reporting Frameworks.

Nach einer kurzen Analyse der Gegenüberstellung haben folgende Punkte die Wahl zu Gunsten von BIRT entschieden:

1. **BIRT ist Teil des Eclipse Projektes**
   Aus diesem Umstand kann die Weiterentwicklung des BIRT Projektes als Open Source Projekt angenommen werden, wogegen Jasper Reports mit steigender Popularität und in absehbarer Zeit zu einem kommerziellen Produkt mutieren könnte. Man könnte einhaken, dass auch Eclipse in Zukunft ein kommerzielles Produkt werden könnte, aber für die Entwicklung von Eclipse ist die Eclipse Foundation verantwortlich, hinter der sich zahlreiche namhafte Firmen wie IBM, Siemens oder SAP verbergen[9]. Im Unterschied dazu wird Jasper Reports "nur" von der Jaspersoft Corporation[10] entwickelt.

2. **Report Design Tool**
   BIRT bietet als Teil des Eclipse Projekts die notwendigen Plugins für die Eclipse IDE

[7] http://jasperforge.org/projects/jasperreports
[8] http://eclipse.org/birt/phoenix/
[9] eine vollständige Liste der Eclipse Foundation Mitglieder findet man unter der URL: http://www.eclipse.org/membership/exploreMembership.php
[10] http://www.jaspersoft.com/

| | Jasper Reports | BIRT |
|---|---|---|
| **Entwicklung** | Jaspersoft Corporation[a] | Eclipse Foundation[b] |
| **Preis** | GNU Library Public License[c]<br>Nicht freie Professional Version[e] | Eclipse Public License[d] |
| **Output Formate** | XML, DOCX, HTML, XLS, PDF, RTF, weitere Formate möglich[f] | XML, HTML, XLS, PPT, DOC, PS weitere Emitte möglich[g] |
| **Designer** | IReport von JasperSoft | Eclipse IDE mit BIRT |
| **Datenanbindung** | JDBC | DTP ODA |
| **RCP Integration** | Eigene Implementierung notwendig | Beispiel vorhanden[h] |
| **Vorteile** | Standardtechnologien<br>Dokumentation | Eclipse RCP Integration<br>Report Designer<br>Dokumentation<br>Standardtechnologien |
| | | Datenanbindung |
| **Nachteile** | Report Designer | keine |

[a] http://www.jaspersoft.com
[b] http://www.eclipse.org/org/
[c] http://jasperforge.org/plugins/project/project_home.php?group_id=102
[d] http://www.eclipse.org/org/documents/epl-v10.php
[e] http://www.jaspersoft.com/mrktcamp/campaigns/Q3_09/jrpro36/
[f] (vgl. Jaspersoft Corporation 2009 online)
[g] (vgl. Weathersby *et al.* 2008 S. 435-462)
[h] http://wiki.eclipse.org/RCP_Example_(BIRT)_2.1

Tabelle 4.6: Gegenüberstellung von Jasper Reports und BIRT

für die Erstellung von Reports. Dieses Tool kann als Stand-alone Lösung von der BIRT Homepage geladen werden, oder per Eclipse Update Mechanismus in eine bestehende Entwicklungsumgebung integriert werden (vgl. The Eclipse Foundation 2009a online). Da die Firma XiTrust zur Entwicklung des XBS auch die Eclipse IDE verwendet, ist die zu erwartende Akzeptanz sehr hoch und der Aufwand zur Einführung als relativ gering zu erachten. Im Gegensatz dazu bietet Jasper Reports mit iReport[11] einen Report Designer der auf der NetBeans RCP[12] basiert.

3. **Einfache Integration**
   Wie im Punkt 2 beschrieben, kann BIRT einfach in die Eclipse IDE integriert werden (vgl. The Eclipse Foundation 2009b online). Diese ist gleich wie der XBS-Client eine Eclipse RCP Applikation. Daraus folgernd ist die Kompatibilität von BIRT und dem XBS-Client gegeben und einer schnellen Integration steht nichts im Wege. Ein Beispiel der Integration von BIRT in eine RCP Application kann von der BIRT Homepage heruntergeladen werden[13]. Auch Jasper Reports kann als Java Bibliothek recht einfach in den XBS-Client integriert werden, die Implementierung wäre aber mit mehr Aufwand verbunden.

[11] http://jasperforge.org//website/ireportwebsite/IRWebsite/ir_download.html?header=project&target=ireport
[12] http://netbeans.org/
[13] http://wiki.eclipse.org/RCP_Example_(BIRT)_2.1

4. **Dokumentation**
   Neben ausreichendem Material im Internet gibt es zwei Bücher zu BIRT. (Peh *et al.* 2008) thematisiert die Erstellung von professionellen Reports mit BIRT. Ist man auf der Suche nach Background-Informationen zur Integration und Erweiterung von BIRT bietet (Weathersby *et al.* 2008) ausreichend Material. Viele Hintergrundinformationen, Beispielreports, Code-Snippets und Dokumentationen bieten darüber hinaus das BIRT Wiki[14] oder die BIRT Exchange Homepage[15]. Auch für Jasper Reports gibt es eine Reihe von Büchern und Dokumentationen im Internet. Die durchgeführte Analyse der beiden Systeme hat aber einen höheren Anteil von frei zugänglichen Ressourcen für BIRT ergeben.

5. **Datenschnittstelle**
   Jasper Report benötigt als Datenschnittstelle einen JDBC Treiber. Im Gegensatz dazu verwendet BIRT als Datenschnittstelle einen ODA Treiber, kann aber durch eine bereits vorhandene ODA Treiber Implementierung mit JDBC Datenquellen umgehen. Durch die hohe Flexibilität der ODA Schnittstelle wird ein Vorteil bei der Datenanbindung erwartet. Weitere Details zu ODA werden in Kapitel 4.6 vorgestellt.

### 4.5.3 Konzept

Durch die Entscheidung das Eclipse BIRT Framework zu verwenden, beschränkt sich die Konzeption der Visualisierungsengine auf die Integration von BIRT in den XBS.

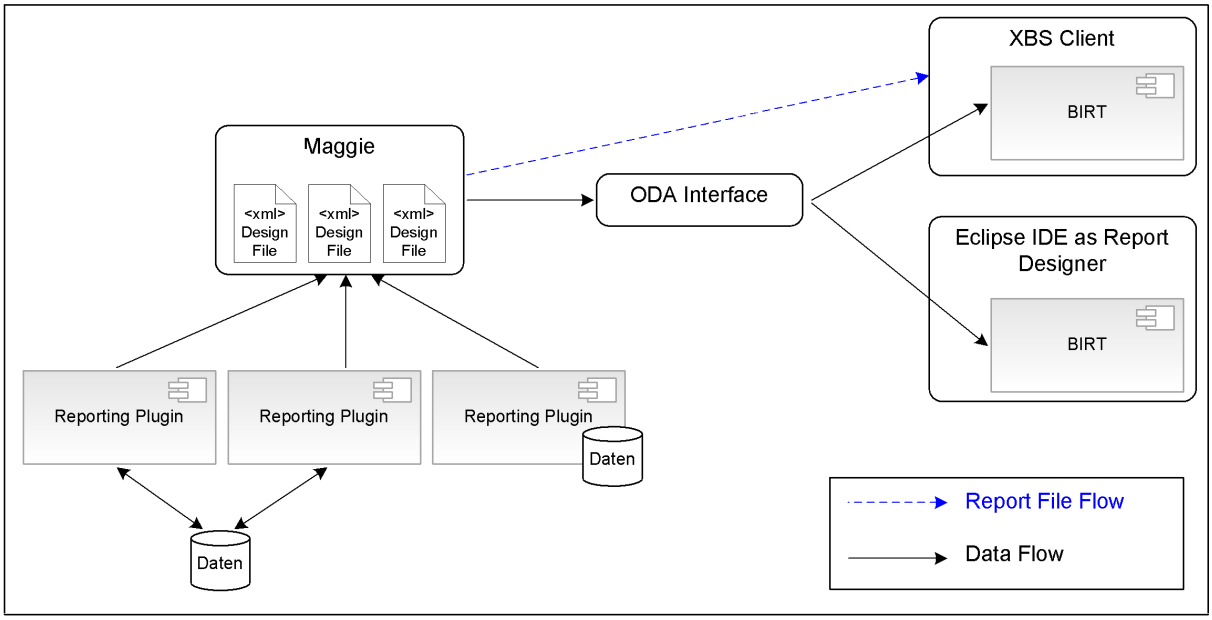



Abbildung 4.7: Maggie Visualisierungsengine Architektur

Die Abbildung 4.7 zeigt die Architektur der Visualisierungsengine für den XBS. Die Report

[14] http://wiki.eclipse.org/index.php/BIRT_Project
[15] http://www.birt-exchange.com/be/home/

Design Files werden zentral als Teil des Maggie Server Bundles in einem Verzeichnis des XBS abgelegt. Verbindet sich ein Konfigurations-Client zum Server, werden die Reports vom Server zum Client kopiert. Zur Beschleunigung des Vorgangs werden alle Reports zuerst verpackt und nur eine Datei zum Client gesendet. Dieser Vorgang bringt einen zusätzlichen Performance-Gewinn beim Betrachten eines Reports, da dieser bereits davor vom Server kopiert wurde und auf dem XBS-Client lokal verfügbar ist.

Die in den XBS-Client integrierte BIRT Engine ist für das Visualisieren der Reports verantwortlich. Die im Report definierten Datensätze und Quellen werden über die entsprechenden Schnittstellen von ODA akquiriert. Über die Adaptor Registry und den gewünschten Adaptor greift die Datenschnittstelle auf den jeweiligen Datensatz des Adaptors zu.

Die Eclipse IDE als BIRT Report Designer verwendet die gleiche Schnittstelle zur Abfrage der Daten vom XBS. Das Erstellen der Reports kann folglich mit Live-Datensätzen des XBS erfolgen. Nach Abschluss des Designs kann das Report Design File in das zentrale Reporting Verzeichnis der jeweiligen XBS Instanz kopiert werden und steht somit den XBS-Clients ohne Neustart des Servers zur Verfügung.

Mit den Details zum Datenfluss und der Datenschnittstelle befasst sich das nächste Unterkapitel.

Die Abbildung 4.8 zeigt die Konfiguration des XBS Monitoring-Systems durch den Konfigurations-Client. Alle beim Event Broker registrierten Listener können hier aus- und eingeschaltet werden.

Die Abbildung 4.9 zeigt die Darstellung eines Reports, der die aktuelle Systemauslastung im XBS-Client anzeigt.

## 4.6 Datenschnittstelle

In diesem Unterkapitel wird das Konzept der Datenschnittstelle vorgestellt. Diese stellt das Bindeglied zwischen Visualisierung und Akkumulation der Daten dar. Zu Beginn werden die Anforderungen definiert und im Anschluss daran das Konzept im Unterkapitel 4.6.2 vorgestellt.

### 4.6.1 Anforderungen

Die Entscheidung, das Eclipse BIRT Framework für die Visualisierung der Daten zu verwenden, bringt auch den Vorteil mich sich, dass nur ein ODA Treiber für den XBS implementiert werden muss. Auf die Spezifikation der Eclipse Data Tools ODA Spezifikation wird im nächsten Unterkapitel detailliert eingegangen.

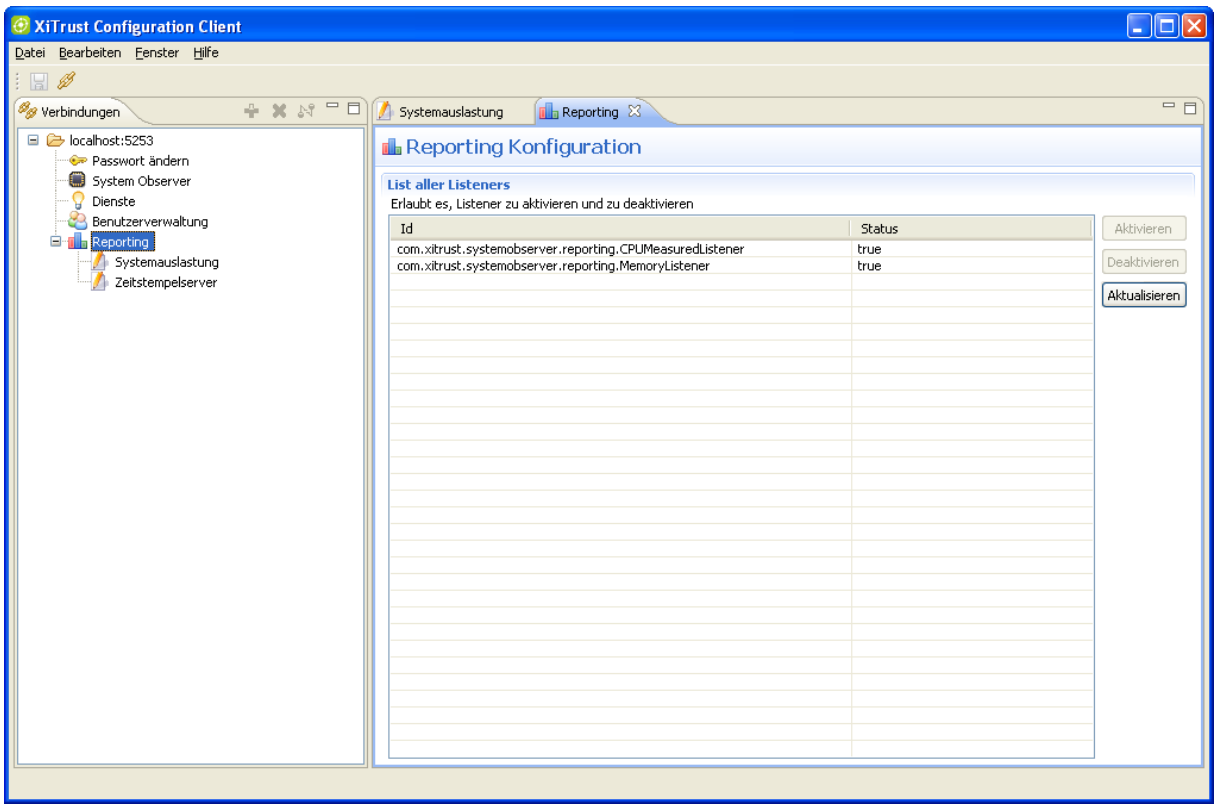



Abbildung 4.8: XBS Reporting Konfiguration

1. **Ziele**

   - ZIEL 3.1: Die Datenschnittstelle muss in der Lage sein die Daten des XBS Monitoring-Systems an die BIRT Visualisierungs Engine ausliefern zu können `(Pflichtanforderung)`

2. **Funktionale Anforderungen**

   - FUNKANF 3.1: Implementierung eines XBS ODA Drivers konform der Eclipse Data Tools ODA Spezifikation `(Pflichtanforderung)`

3. **Qualitätsanforderungen**

   - Die Qualitätsanforderungen gelten sinngemäß denen der globalen Anforderungen QUALANF 1, QUALANF 2 und QUALANF 3 aus Kapitel 4.1

4. **Randbedingungen**

   - RANDBED 3.1: Korrekte Implementierung der ODA Schnittstelle um kompatibel für ODA Consumer zu sein `(Pflichtanforderung)`

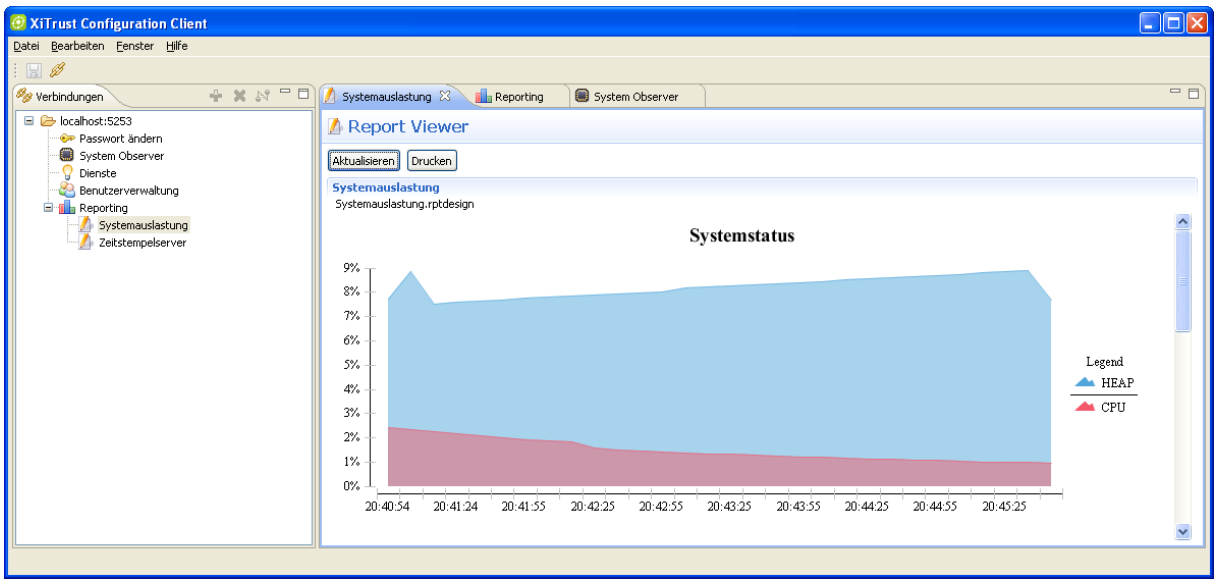



Abbildung 4.9: Reportdarstellung im XBS-Client

### 4.6.2 Konzept

Die Abbildung 4.10 zeigt das Konzept des Zugriffs auf Daten des XBS Monitoring-Systems. Jedes Reporting Plugin (Adaptor) kann eine Reihe von Datensätzen, identifiziert durch eine eindeutige ID, zur Verfügung stellen. Der Adaptor wird, ebenfalls durch eine eindeutige ID identifiziert, in der Adaptor Registry des Maggie Server Plugins registriert. Der Zugriff auf Monitoring Daten erfolgt durch eine Implementierung der Open Data Access (ODA) Spezifikation im Maggie Remote Plugin. Diese Implementierung wird von XBS-Client und BIRT Report Designer als Datenschnittstelle zur Gewinnung der Report Daten verwendet.

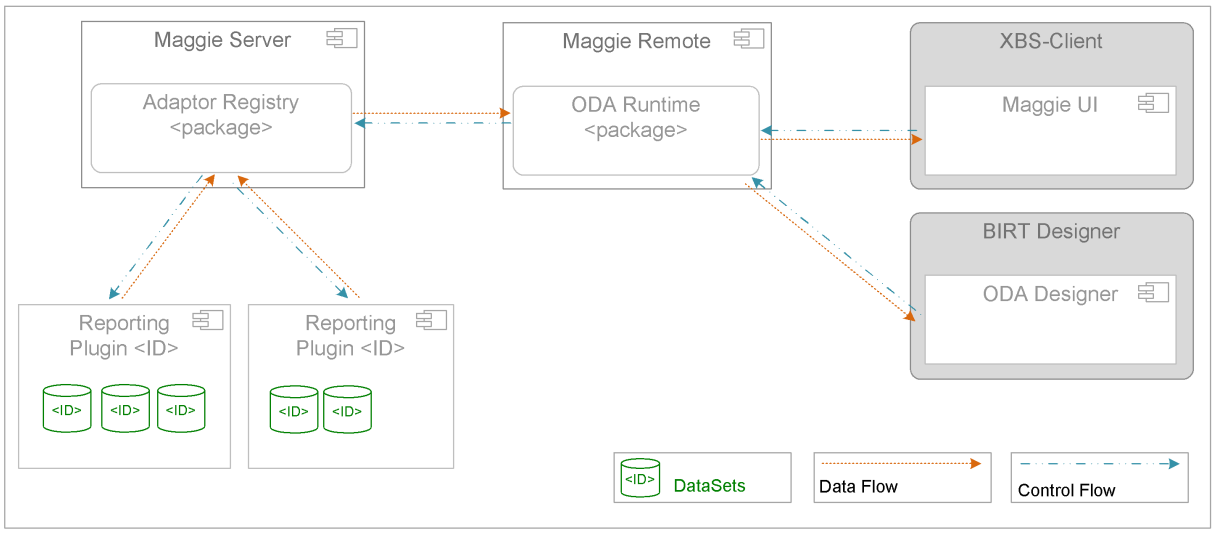



Abbildung 4.10: XBS Datenschnittstelle

ODA ist Teil des Eclipse Data Tools Platform (DTP) Projekts. Das BIRT Framework ist ein sogenannter ODA Consumer. Das heisst, BIRT verwendet einen ODA konformen Treiber, um

auf die Datenquellen, die in Reports spezifiziert sind, zuzugreifen (vgl. Weathersby *et al.* 2008 S. 477). Die ODA API spezifiziert folgenden Aufgaben (vgl. Weathersby *et al.* 2008 S. 477):

1. Herstellung der Verbindung zur Datenquelle
2. Erstellung und Ausführung von Datenabfragen
3. Verwaltung von Daten und Metadaten in Result Sets
4. Mapping der Repräsentation der Daten und der Datenquelle

Ziel der ODA API ist es Applikationen, die Daten der ODA Schnittstelle konsumieren (ODA Consumer), von der Art der Datenquelle unabhängig zu machen. Dadurch kann BIRT recht einfach erweitert werden, um auch mit speziellen Datenquellen agieren zu können (vgl. Weathersby *et al.* 2008 S. 477).

Eine konkrete ODA API Implementierung besteht aus den zwei Plugins Runtime und Designer. Das Runtime Plugin ist die Implementierung des ODA Treibers selbst, das Design Plugin erweitert die BIRT Designer Applikation. Diese Erweiterungen ermöglichen die Anbindung an die Datenquelle und Änderungen der Einstellungen der Parameter des Treibers durch konkrete Wizards des BIRT Designers (vgl. Weathersby *et al.* 2008 S. 479).

Das Runtime Plugin des XBS implementiert die folgenden Interfaces:

- `IDriver`
  IDriver ist der Einstiegspunkt des Runtime Plugins und stellt die Factory zur Erstellung eines IConnection Objekts dar.
- `IConnection`
  Dieses Interface stellt die Verbindung zur Datenquelle dar. Diese Verbindung kann geöffnet und geschlossen werden. Die Instanz einer `IConnection` Klasse dient zur Erzeugung eines `IQuery` Objekts.
- `IQuery`
  Das `IQuery` Interface definiert Methoden zur Präparierung des Query sowie Setzen der Parameter und Sortier-Attribute. Weitere Methoden dienen zur Ausführung und Schließen des Queries sowie Gewinnung der Metadaten aus dem Ergebnis.
- `IResultSet`
  Die Methoden dieses Interfaces ermöglichen den Zugriff auf die Daten des ResultSets. Die Daten liegen in tabellarischer Form vor und können durch Methoden der `IResultSet` Implementierung durchlaufen und abgefragt werden.
- `IResultSetMetaData`
  Meta-Daten eines ResultSets werden durch das `IResultSetMetaData` Interface repräsentiert. Sie beschreiben beispielsweise Index, Namen und Type jeder Spalte des ResultSets.

- `IDataSetMetaData`
  Gleich wie eine ODA Treiber Implementierung verschiedene `IConnection` Typen anbieten kann, so ist auch eine `IConnection` Implementierung in der Lage verschiedene Arten von DataSets anzubieten. DataSets werden durch `IDataSetMetaData` Klassen beschrieben und enthalten Informationen beispielsweise über Art der Input und Output Parameter oder Version des DataSet Types.

- `IParameterMetaData`
  Ein Query kann mit Parametern ausgestattet werden. Ähnlich der `IResultSetMeta Data` Klasse beschreibt ein `IParameterMetaData` Objekt etwa den Type, Genauigkeit oder Name des jeweiligen Parameters.

Das Diagramm aus der Abbildung 4.11 zeigt den sequenziellen Ablauf der Abfrage von Daten des XBS Monitoring-Systems über die ODA Schnittstelle.

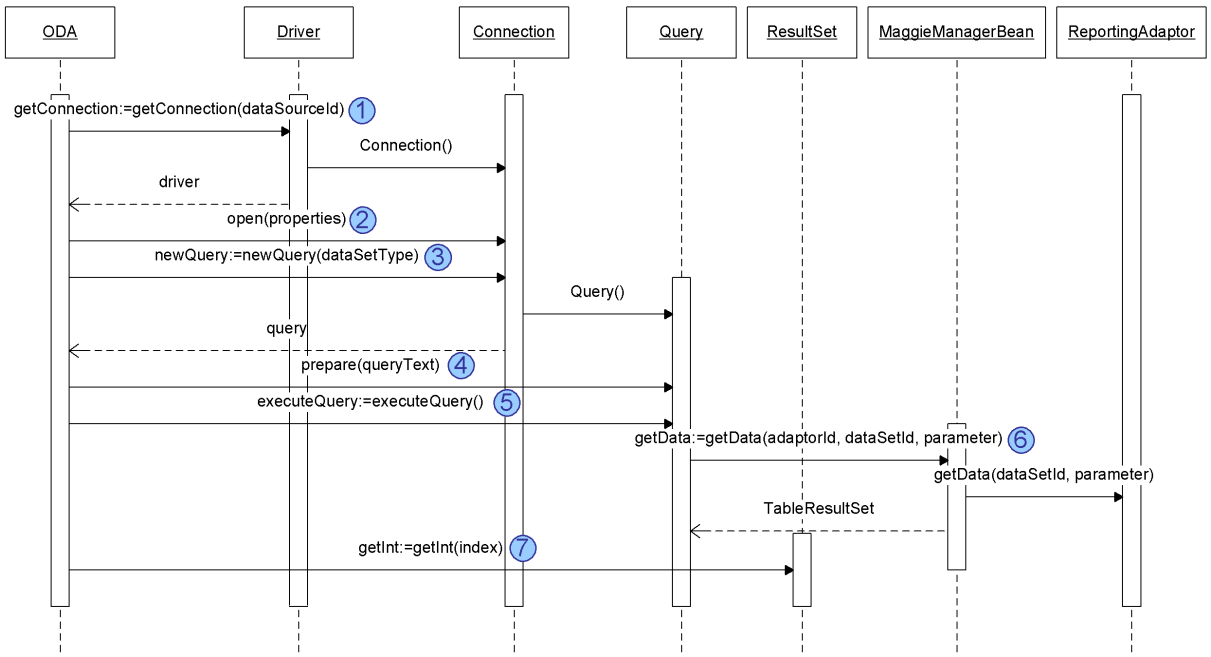



Abbildung 4.11: Sequenz eines ODA Datenzugriffs

Die 7 Schritte im Detail:

1. `Driver.getConnection`
   Der erste Schritt ist das Erstellen eines Connection Objekts. Der `Driver.getConnection` Methodenaufruf erstellt ein solches Objekt, das in der Lage ist, eine Verbindung mit dem XBS herzustellen.

2. `Connection.open`
   Mit diesem Methodenaufruf wird die Verbindung mit dem XBS per MiniBee initialisiert. Der `Properties` Parameter des Methodenaufrufs beinhaltet die Einstellungen für die

Verbindung bestehend aus Host, Port, Username und Passwort. Diese sind Teil des Reports und werden beim Aufruf eines Reports im XBS-Client durch die Daten der aktuellen Server Verbindung ersetzt.

3. `Connection.newQuery`
   Im nächsten Schritt wird ein neues `Query` Objekt basierend auf dem geforderten DataSet Type erstellt.

4. `Query.prepare`
   In diesem Schritt gilt es den Query String vorzubereiten. Der Query String besteht nach der ODA Spezifikation aus einem einzelnen `String`. Dieser besteht in der XBS ODA Implementierung aus den Teilen AdaptorId, DataSetId und Parameter getrennt durch Strichpunkte. Die `Query.prepare` Methode extrahiert die einzelnen Teile des Query Strings und bereitet den Query für die Ausführung vor.

5. `Query.executeQuery`
   Die Ausführung des Queries durch `Query.executeQuery` delegiert die Parameter an die `getData` Methode des `MaggieManagerBeanRemote` Beans mit den entsprechenden Funktionsparametern.

6. `MaggieManagerBeaneRemote.getData`
   Das `MaggieManagerBeaneRemote` ist die Schnittstelle der Adapor Registry des XBS Monitoring-Systems für Remote Procedure Calls. Diese werden durch das MiniBee Bundle abgewickelt. Die Methode sucht die Instanz des ensprechenden Adaptors aus der Registry und ruft dessen `getData` Methode mit DataSetId und Parametern auf.

7. `ResultSet.getInt`
   Das Ergebnis des Aufrufs ist ein tabellarisches ResultSet. Jedes Reporting Plugin kann eine Reihe solcher DataSets, die durch eine ID eindeutig identifiziert sind, zur Verfügung stellen. Der Aufruf beispielsweise von `ResultSet.getInt` mit gegebenem Spalten-Index gibt den Wert der aktuellen Reihe und Spalte als Integer Objekt zurück. Per Iterator kann das ResultSet durchlaufen werden.

Die genauen Spezifikationen der ODA API können der Eclipse Data Tools Platform Homepage[16] entnommen werden. Eine genaue Anleitung zur Implementierung einer ODA konformen Datenschnittstelle für BIRT liefert (Weathersby *et al.* 2008) in Kapitel 20.

## 4.7 Fazit

In diesem Kapitel werden die drei Teilkonzepte von Datenakkumulation, Visualisierungsengine und Datenschnittstelle zusammengefügt. Die Abbildung 4.12 zeigt das resultierende Konzept

[16] http://www.eclipse.org/datatools/

für das XBS Monitoring-System.

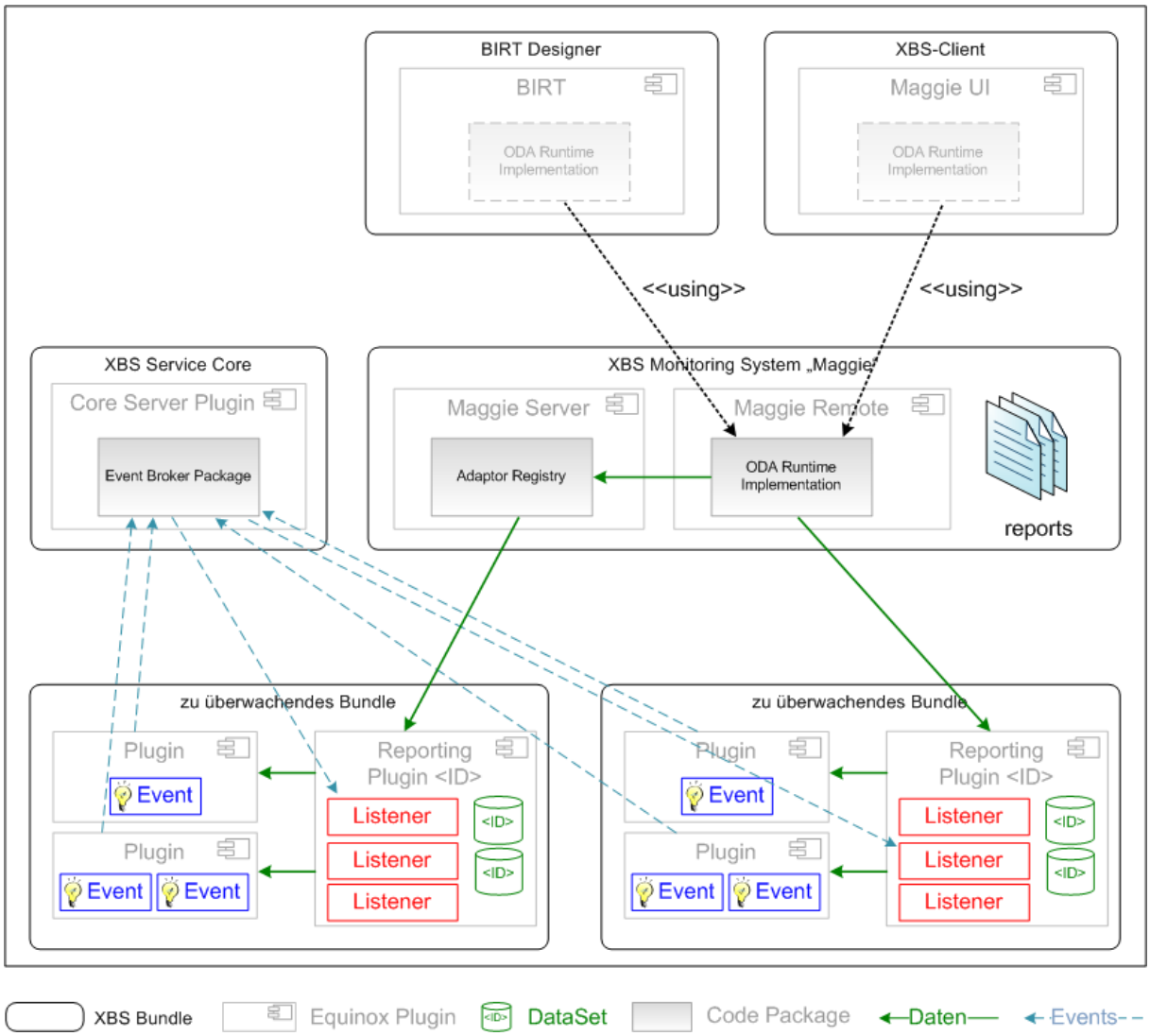



Abbildung 4.12: Architektur des XBS Monitoring-Systems "Maggie"

Die Komponenten der XBS Monitoring-System Architektur im Detail:

- **Zu überwachende Bundles**
  Diese Bundles können aus mehreren Plugins bestehen, die es zu überwachen gilt. Zu diesem Zweck muss für jedes solche Bundle ein sogenanntes Reporting Plugin (oder auch Reporting Adaptor) implementiert werden. Dieses Plugin stellt eine Reihe von DataSets zur Verfügung. Diese werden entweder durch direkten (aktiven) Zugriff auf Reporting Daten der Plugins oder durch Listener befüllt. Listener werden vom Event Broker aktiviert, wenn ein Event, für das sie sich registriert haben, auftritt. Events werden von den Plugins des zu überwachenden Bundles ausgelöst.

- **XBS Service Core**
  Das XBS Service Core beherbergt den Event Broker. Dieser ist für die "Zustellung" der Events an die jeweiligen Listener zuständig. Dabei werden beim Auftreten eines Events

alle Listener benachrichtigt, die sich für dieses Event oder eines in der Objekt-Vererbungshierarchie höher gelegenes Event registriert haben. Durch den Event Broker Mechanismus können Daten der zu überwachenden Bundles auf passivem Weg gewonnen werden.

- **Maggie**
  Das Maggie Bundle beherbergt zum einen die Reporting Adaptor Registry und die ODA Runtime Implementierung. Alle Reporting Plugins registrieren sich an der zentralen Adaptor Registry mittels einer eindeutigen Adaptor Id. Die Registry wird von der ODA Runtime Implementierung verwendet um an die Daten der Reporting Plugins zu gelangen. Diese Implementierung stellt eine ODA Datenquelle für ODA Consumer zur Verfügung. Die Reports im XML Format werden zentral in einem Verzeichnis des XBS abgelegt.

- **XBS-Client & BIRT Designer**
  Sowohl XBS-Client als auch BIRT Designer sind ODA Consumer und verwenden somit die ODA Runtime Implementierung des Maggie Remote Plugins zur Abfrage der Report Daten.

Blickt man auf die globalen Anforderungen zurück, die in Unterkapitel 4.1 definiert wurden, so kann das Erreichen der sechs Ziele des Projektes bestätigt werden. Auch die globalen funktionellen Anforderungen konnten erfüllt werden. Auf die Qualitätsanforderungen wurde großer Wert gelegt. Besonders die Erweiterbarkeit und Kombinierbarkeit zu Produkten des XBS wurde durch die gewählte Architektur in keiner Weise eingeschränkt.

Weitere wichtige Anforderungen aus Datenakkumulation, Visualisierung und Datenschnittstelle wurden erfüllt. So ist etwa die aktive und passive Datengewinnung durch Reporting Plugins und Event Broker Mechanismus möglich (FUNKANF 1.1). Da die Art der Persistierung den Reporting Plugins überlassen ist, kann auch die Anforderung FUNKANF 1.2 als erfüllt betrachtet werden. Die Konfiguration der Datenakkumulation ist durch das Aktivieren und Deaktivieren der Listener im XBS-Client möglich FUNKANF 1.3. FUNKANF 2.1 bis FUNKANF 2.3 werden durch BIRT erfüllt. Die Anforderung der Datenschnittstelle werden durch die Implementierung eines ODA konformen Treibers für den XBS erfüllt.

Zum Ende dieses Unterkapitels soll noch die Frage beantwortet werden, warum nicht gleich ein Monitoring-System wie Nagios verwendet wurde. Alle in Kapitel 3.3 vorgestellten Systeme benötigen ein Unix basiertes Betriebssystem und sind dadurch nur eingeschränkt portierbar. Ein weiterer Grund für die Entwicklung eines eigenen Systems ist der benötigte Aufwand für die Anbindung eines Monitoring-Systems. Der Aufwand für die Entwicklung der entsprechenden Plugins zur Anbindung ist gegenüber der Neuentwicklung in etwa der gleiche. Durch das Vorsehen entsprechender Schnittstellen, wie etwa SNMP, können Monitoring-Systeme einfach und unkompliziert an den XBS angedockt werden. Die Verwendung eines bestehenden Open Source Monitoring-Systems würde zusätzlich eine Abhängigkeit von einem externen System mit sich bringen, dessen Weiterentwicklung nur eingeschränkt beeinflusst und nicht garantiert werden kann. Im Unterkapitel 3.3.5 wurde mit der Verwendung der vorgestellten Monitoring-

Systeme hauptsächlich zur Netzwerküberwachung ein zusätzlicher Grund aufgeführt, warum ein eigenes System von Vorteil wäre, besonders da die Netzwerk Monitoring-Systeme sich nur beschränkt zur Überwachung von Software einsetzen lassen. Die Möglichkeiten beschränken sich meist auf das Überprüfen eines Dienstes oder Services nicht aber auf interne Daten.

# 5 Zusammenfassung und Ausblick

In diesem Kapitel lassen wir im ersten Unterkapitel noch einmal die Entstehung des XiTrust Business Server (XBS) Monitoring-Systems Revue passieren. Dazu betrachten wir die Gründe, die zur Entwicklung geführt haben. In weiterer Folge werden die Ergebnisse der durchgeführten Analysen aus Kapitel 3 und das Konzept für das Monitoring-System aus Kapitel 4 zusammengefasst.

## 5.1 Zusammenfassung

Einleitend wurden drei Renditefallen beim Angebot von Services von Dienstleistungsanbietern, basierend auf (TechnologieStiftung Hessen GmbH 2002) vorgestellt. Aus diesen geht die Wichtigkeit hervor, dem Kunden den unmittelbaren Nutzen von Services zu verdeutlichen, denn nur dann ist er auch bereit dafür Ressourcen zu veräußern. DI Helmut Aschbacher lieferte den Anlass zu vorliegender Masterarbeit. Er entwickelte in seiner Arbeit ein Servicekonzept für die Firma XiTrust Secure Technologies GmbH.

Einer der Vorschläge des Servicekonzepts von DI Helmut Aschbacher war die Umsetzung eines Tools, das die Leistung des XBS ausreichend visualisiert. Der XBS ist ein neu entwickeltes Software-Produkt, das vielschichtige Aufgaben zentral in Unternehmen übernimmt. Zu seinen Kernaufgaben zählen das Verschlüsseln von E-Mails oder das Erstellen digitaler Signaturen.

Der XBS durchwanderte im Laufe der Zeit verschiedene Evolutionsstufen. Er entwickelte sich von einer prototypischen Implementierung einer E-mail Signatur Proxy Idee des IAIK, zu einer immer weiter wachsenden monolithischen Software Applikation. Durch die Einschränkung, die eine monolithische Software-Architektur im Bereich Erweiterbarkeit mit sich brachte, musste der XBS später neu entwickelt werden. Auf Basis von OSGi Equinox entstand mit dem E3 Server die neueste Generation des XBS.

Bei der Entwicklung wurde besonders auf die Erweiterbarkeit und die Möglichkeit, Software-Pakete frei zu Produkten kombinieren zu können, Wert gelegt. Wie diese Anforderungen mit Hilfe von Eclipse Equinox umgesetzt werden konnten und welche Gefahren damit verbunden sind, wird in Kapitel 2 detailliert beleuchtet. Im weiteren Verlauf der Masterarbeit wurden sechs Ziele für das XBS Monitoring-System Projekt definiert:

1. Datenakkumulation
2. Visualisierung der Daten
3. Datenschnittstelle
4. Implementierung
5. Darstellung des Verlaufs eines Systemzustandes
6. Darstellung des Verlaufs von internen Prozessdaten

Bevor mit der Konzeptionierung des Monitoring-Systems begonnen wurde, galt es vorab eine Wissens-Basis im Monitoring Bereich zu schaffen. Zu diesem Zweck wurden Software-Architekturen, Protokolle und Standards sowie bestehende Monitoring-Systeme analysiert.

Der Analyse von Software-Architekturen ging eine Kategorisierung basierend auf (Zdun & Avgeriou 2005) nach acht verschiedenen Sichten (Views) auf Software-Architekturen voraus. Im Anschluss wurden für den Kontext dieser Arbeit relevante Architekturen der einzelenen Views vorgestellt. Darunter fallen die Architekturen Layers, Pipes and Filter, Blackboard, Broker, Modell-View-Controller, Microkernel, Event-based System und Client-Server (Kapitel 3.1.2 bis 3.1.9). Als Fazit aus der Analyse gingen zwei wesentliche Aspekte hervor:

1. Systeme, die durch eine dieser Architekturen strukturiert sind, haben gegenüber monolithischen Systemen meist eine erhöhte Flexibilität und Erweiterbarkeit.
2. Als Konsequenz aus der gesteigerten Erweiterbarkeit ergibt sich jedoch meist aufgrund notwendiger Abstraktionen eine Performance-Einbuße gegenüber dem monolithischen System.

Neben diesen beiden Punkten, die die Verwendung einer Software-Architektur von den Anforderungen abhängig macht, ergab die Analyse, dass moderne Software-Systeme immer häufiger heterogene Systeme sind, die auf mehreren Software-Architekturen aufbauen. Die Vor- und Nachteile der einzelnen Software-Architekturen wurden in der Tabelle 3.1 zusammengefasst.

Die nächste Analyse befasste sich mit Protokollen und Standards aus dem Monitoring-Bereich. Darunter fallen beispielsweise IPMI als Hardware Standard oder die Protokolle ICMP und SNMP. Besonders SNMP als reines Monitoring Protokoll spielte im weiteren Verlauf der Arbeit noch eine größere Rolle. Zusätzlich wurden die SNMP MIB Erweiterung durch RMON sowie Syslog und JMX vorgestellt. Für die Entwicklung eines Monitoring-Systems für den XBS spielen besonders JMX und SNMP eine große Rolle. JMX ermöglicht das Auslesen von Systemzuständen wie beispielsweise CPU und Speicherauslastung. Durch die Entwicklung einer SNMP Schnittstelle soll die Qualität und Skalierbarkeit des Systems hochgehalten werden.

Die letzte Analyse galt verbreiteten Monitoring-Systemen. Hier wurden mit besonderem Augenmerk auf die verwendeten Software-Architekturen, Nagios, CollectD, Argus und NeDi analysiert. Zusätzlich wurde untersucht, wie diese Systeme die Problemstellungen der Datenakkumulation und Visualisierung lösen. Ein weiterer Punkt war die Frage der Persistierung der gesammelten Daten. Einige Ideen der Systeme wurden zu Vorbildern bei der Konzeptionierung eines eigenen Monitoring-Systems für den XBS. Darunter fielen etwa der Event-Broker von Nagios oder der Plugin Mechanismus von Collectd. Die Vor- und Nachteile der einzelnen Monitoring-Systeme wurden in der Tabelle 3.9 zusammengefasst.

Wie bereits erwähnt wurde das XBS Monitoring-System, das den internen Projektnamen "Maggie" bekam, in drei große Teile unterteilt. Die einzelnen Konzepte im Detail waren:

- **Datenakkumulation**
  Die Konzeptionierung der Datenakkumulation war zugleich der komplizierteste Teil der Entwicklung des Monitoring-Systems. Auf der einen Seite sollte weiterhin eine große Erweiterbarkeit des Systems gewährleistet werden, auf der anderen Seite die freie Kombinierbarkeit zu Produkten auf keinen Fall eingeschränkt werden. Zusätzliche Anforderungen bestehen darin, die Daten unterschiedlich persistieren und aktiv und passiv gewinnen zu können. Ein ständiges Pollen der Werte sollte vermieden werden.

  Im Vorfeld wurden die zu erhebenden Daten analysiert und deren Akquirierungs-Möglichkeiten untersucht. Erst dann wurde ein auf Snap-Ins basierendes Konzept entworfen, das den Anforderungen gerecht werden konnte. Die Idee war, jedes zu überwachende Bundle mit einem sogenannten Reporting-Plugin zu versehen. Dieses ist für die Akkumulation und Persistierung der jeweiligen Daten zuständig.

  Zur passiven Datengewinnung wurde der Event Broker Mechanismus des Nagios Systems adaptiert. Die Idee dahinter ist ein Publisher-Subscriber Mechanismus basierend auf Events und Listener. Events werden von beliebigen Software-Teilen eines überwachten Bundles ausgelöst. Die Listener sind die datensammelnden Teile des Monitoring-Systems und werden vom Broker, im Falle des Eintreffens eines passenden Events (Listener hat sich für das entsprechende Event registriert), benachrichtigt. Die registrierten Listener können durch den Konfigurations-Client (XBS-Client) aktiviert und deaktiviert werden.

  Die zweite wichtige Komponente der Datenakkumulation neben dem Event-Broker ist die Adaptor Registry. An dieser zentralen Stelle müssen sich die Reporting Plugins registrieren. Zugleich dient sie als Anlaufstelle für das Abfragen der Daten durch die Schnittstelle zur Visualisierungsengine.

  Das Konzept der gesamten Datenakkumulation sowie Event-Broker und Adaptor-Registry im Detail werden im Unterkapitel 4.4 ausführlich dargestellt.

- **Visualisierung**
  Nach einem Vergleich zweier häufig eingesetzter Java-basierter Visualisierungs-Frameworks - Jasper Reports und Business Intelligence and Reporting Tools (BIRT) von Eclipse

- bei dem sich für BIRT mehr Vorteile ergaben - wurde die Integration des Frameworks in den XBS entworfen.

Die XML-basierten BIRT Report Design Files werden zentral in einem Verzeichnis XBS Servers abgelegt. Der XBS-Client holt diese nach dem Erstellen der Verbindung vom Server ab und verwendet die Datenschnittstelle zur Darstellung der eingebetteten Reporting Daten. Da sowohl der XBS (Client und Server) als auch BIRT auf dem Eclipse System basieren, war die Integration nicht besonders schwer, da lediglich die geforderten Plugins, die der Visualisierung von Reports dienen, in den leicht erweiterten XBS-Client integriert werden mussten.

Details zum Konzept der Visualisierungsengine des XBS Monitoring-Systems können im Unterkapitel 4.5 nachgelesen werden.

- **Datenschnittstelle**
  Durch die Entscheidung BIRT zu verwenden reduzierte sich die Entwicklung der Datenschnittstelle auf die Implementierung eines eigenen Open Data Access Treibers für den XBS. ODA ist Teil des Eclipse Data Tools Platform (DTP) Projekts mit dem Ziel Applikationen, die Daten der ODA Schnittstelle konsumieren (ODA Consumer), von der Art der Datenquelle unabhängig zu machen. Es wurde ein ODA Treiber entwickelt, der es ermöglicht, Daten unter Angabe von Reporting-Plugin Id und Datensatz Id vom Monitoring-System abzufragen. Das Konzept wird in Unterkapitel 4.6 detailliert vorgestellt.

Am Ende des Kapitels 4 werden im Unterkapitel 4.7 die drei Teilkonzepte zu einem Ganzen zusammengefügt. Weiters wird die Erfüllung der Anforderungen sowie der Aspekt diskutiert, welche Vorteile die Entwicklung eines eigenen Monitoring-Systems gegenüber der Anpassung eines bestehenden Systems wie beispielsweise Nagios hat.

## 5.2 Ausblick

Das Monitoring-System Projekt hat die Erweiterbarkeit des XBS auf eine harte Probe gestellt. Es musste ein System entwickelt werden, das in weiterer Folge mit sehr vielen bestehenden Bundles zu interagieren hat, die um Reporting Funktionalität erweitert werden. Das Konzept sieht vor, dass jedes zu überwachende Bundle mit einem Reporting Plugin ausgestattet wird. Die Erweiterung des Bundles beschränkt sich jedoch auf die Implementierung geeigneter Event Klassen und deren Auslösen an den gewünschten Stellen. Durch die Adaptor Registry und die Auslagerung des Event-Brokers in den Core konnte gewährleistet werden, dass keine zusätzlichen unerwünschten Abhängigkeiten entstehen, die die freie Kombinierbarkeit zu Produkten einschränken. Durch den Plugin Mechanismus von Eclipse Equinox konnte das XBS Monitoring-System als Bundle bestehend aus Remote, Server und UI Plugins in die bestehende Architektur integriert werden. Die Erweiterung des bestehenden Systems ware mit Ausnahme des Event-Brokers, der als Software-Paket in den Core integriert wurde, nicht nötig. Obwohl

der XBS um ein System erweitert wurde, das Daten vieler bestehender Bundles sammelt, war die Architektur des XBS sehr robust gegenüber diesen Erweiterungen.

Das entwickelte Monitoring-System kann durch die Auswahl der verwendeten Technologien als zukunftssicher angesehen werden. BIRT wird mit Eclipse weiterentwickelt und aus diesem Grund kann das Fortbestehen auch der Datatools Plattform und im speziellen von ODA angenommen werden.

Für das XBS-Monitoring-System sind einige Teile in erster Instanz noch nicht implementiert, mögliche zukünftige Erweiterungen im Konzept aber vorgesehen worden:

- **SNMP Modul**
  Die Abbildung 5.1 zeigt die mögliche Erweiterung des Monitoring-Systems um ein SNMP Modul.

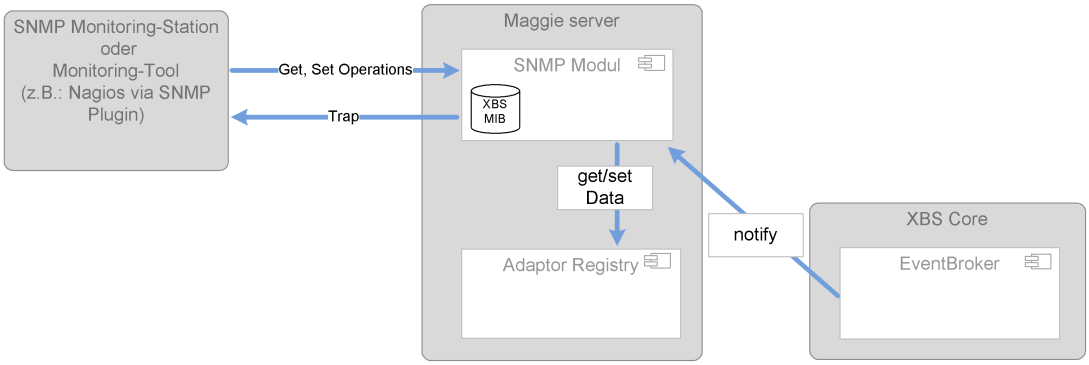



Abbildung 5.1: SNMP Modul für das XBS Monitoring-System

  Das Maggie Bundle müsste mit einem SNMP Modul versehen werden. Dieses enthält ein oder mehrere Listener, um auf Events zu reagieren. Dies könnte durch eine kleine Erweiterung des Event-Brokers (Abfrage aller registrierten Events) dynamisch implementiert werden. Wird ein Event empfangen, kann durch eine geeignete Implementierung der SNMP Spezifikation eine Trap ausgelöst werden. Die aktive Operation des SNMP Protokolls zur Gewinnung von Daten (get) kann über die Adaptor-Registry direkt vom SNMP Modul ausgeführt werden. Für die Set Operation ist eine kleine Erweiterung der Adaptor-Registry notwendig, die zusätzlich das Setzen von Werten ermöglicht.

  Ein SNMP Modul würde das Monitoring durch SNMP fähige Monitoring-Stations oder Monitoring-Tools erlauben. Da Nagios oder CollectD bereits SNMP Plugins zur Verfügung stellen, könnte der XBS mit Hilfe des SNMP Moduls ohne weiteren Aufwand durch diese Systeme überwacht werden.

- **Benachrichtigungsmodul**
  Die Abbildung 5.2 zeigt die mögliche Erweiterung des Monitoring-Systems um ein Benachrichtigungsmodul.

  Der Aufbau ist dem des SNMP Moduls sehr ähnlich. Wieder wird das Maggie Bundle mit einem Modul erweitert. Diese Modul empfängt gleich wie das SNMP Modul Events

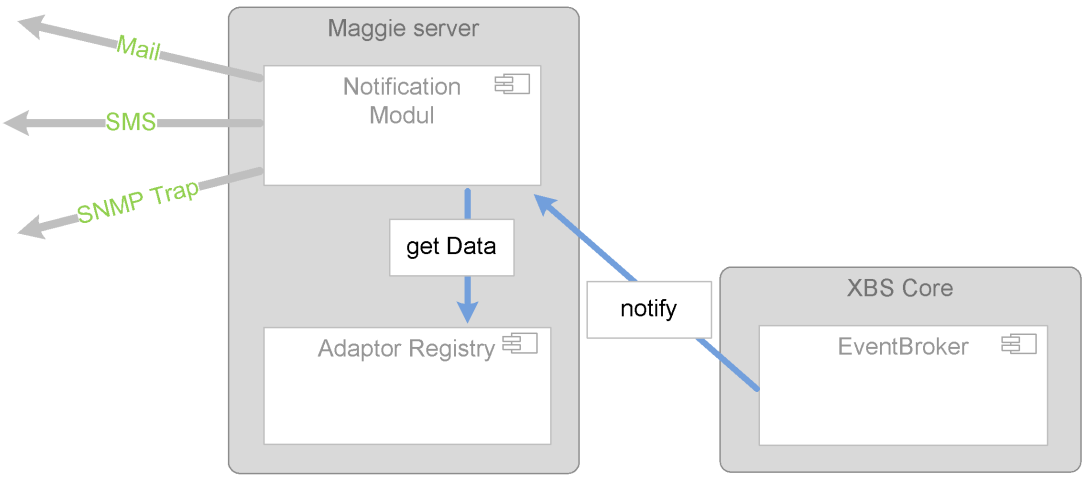



Abbildung 5.2: Benachrichtigungs Modul für das XBS Monitoring-System

und akquiriert Daten aktiv von der Adaptor-Registry. Durch verschiedene Erweiterungen können Benachrichtigungen beispielsweise per E-Mail oder SMS versendet werden. Ist das SNMP Modul ebenfalls vorhanden, können auch spezielle SNMP Traps ausgesendet werden. Das Module selbst müsste ein Regelverarbeitssystem beinhalten, das die Benachrichtigungen triggert. Geeignet für diese Aufgabe wäre die bereits implementierte Lucille Engine des XBS (Entscheidungsfindung in Form von Automaten) oder eine Rule Engine.

- **Optimierung des Event-Brokers**
  Der Event-Broker verwendet bei der Ermittlung der zu benachrichtigenden Listener einen sehr trivialen Algorithmus. Trifft ein Event ein, werden alle registrierten Events auf eine Abhängigkeit in der Objekt-Hierarchie überprüft. Es werden alle jene Listener, die sich nicht für das eintreffende Event selbst, aber für ein in der Hierarchie höher liegendes Objekt registriert haben, benachrichtigt. Um den Prozess zu beschleunigen, könnten die Objekte in einer baumartigen Datenstruktur organisiert sein, die das Auffinden "verwandter" Objekte beschleunigen würde. Die Verwaltung der Datenstruktur würde zwar aufwändiger, das Registrieren von Events und Listener beschränkt sich aber zum größten Teil auf das Starten des XBS.

- **Round Robin Database**
  Wie bereits behandelt, sind die Reporting-Adaptoren selbst für die Art der Datenspeicherung verantwortlich. Kurzlebige Daten werden im Speicher gehalten, Daten, die auch länger von Interesse sind, in einer Datenbank gespeichert. Für zyklische Daten wird von den vorgestellten Monitoring-Systemen meist eine Round Robin Datenbank eingesetzt. Diese ist besonders für das Speichern von Logging und Monitoring Daten ausgelegt. Eine Round Robin Datenbank speichert die Daten ähnlich einer Queue, deren Größe fixiert ist. Detaillierte Informationen zu Round Robin Datenbanken können der RRDTool Ho-

mepage[1], die bekannteste Implementierung einer Round Robin Datenbank, entnommen werden.

Alle diese Erweiterungen würden die Qualität des XBS Monitoring-Systems noch steigern. Zum aktuellen Zeitpunkt liegen noch keine Kundenwünsche für die beschriebenen Erweiterungen vor. Der Einsatz des entwickelten Systems beim Kunden wird die Notwendigkeit der Implementierung der einen oder anderen Erweiterung zeigen. Durch das Vorsehen dieser Erweiterungen im Konzept können die notwendigen zusätzlichen Module schnell und einfach implementiert und in das Monitoring-System integriert werden.

[1] http://oss.oetiker.ch/rrdtool/tut/rrdtutorial.en.html

# Abkürzungsverzeichnis

| | |
|---|---|
| AJAX | Asynchronous JavaScript and XML |
| API | Application Programming Interface |
| ASN | Abstract Syntax Notation |
| BIRT | Business Intelligence and Reporting Tools |
| BMC | Baseboard Management Controller |
| CDP | Cisco Discovery Protocol |
| CPU | Central Processing Unit |
| CSS | Cascading Style Sheets |
| DOC | Microsoft Word |
| DTP | Data Tools Platform |
| EJB | Java Enterprise Beans |
| ERP | Enterprise Ressource Planing Systeme |
| GNU | GNU is not Unix |
| HTML | Hypertext Markup Language |
| HTTP | Hypertext Transfer Protocol |
| IAIK | Institute for Applied Information Processing and Communications |
| ICMP | Internet Control Message Protocol |
| IDE | Integrated Development Environment |
| IETF | Internet Engineering Task Force |
| IP | Internet Protocol |
| IPMI | Intelligent Platform Management Interface |
| JDBC | Java Database Connectivity |
| JMX | Java Management Extensions |
| JPA | Java Persistence API |
| JSR | Java Specification Request |
| JVM | Java Virtual Machine |
| LAN | Local Area Network |
| LLDP | Link Layer Discovery Protocol |
| MAC | Media Access Control |
| MIB | Management Information Base |
| MVC | Modell-View-Controller |
| ODA | Open Data Access |

PC ..... Personal Computer
PDE ..... Plugin Development Environment
PDF ..... Portable Document Format
PDU ..... Protocol Data Unit
PPT ..... Microsoft PowerPoint
RCP ..... Rich Client Platform
RFC ..... Request For Comments
RMON ..... Remote Monitoring
SLF4J ..... Simple Logging Facade for Java
SMI ..... Structure of Management Information
SMS ..... Short Message Service
SNMP ..... Simple Network Management Protocol
SQL ..... Standard Query Language
TCP ..... Transmission Control Protocol
UDP ..... User Datagram Protocol
URL ..... Uniform Resource Locator
XBS ..... XiTrust Business Server
XLS ..... Microsoft Excel

# Stichwortverzeichnis

**B**



**C**



**D**

## E



## F

## G



## H



## I

## J



## K



## L



## M

**N**

## O



## P

## Q



## R



## S

**T**



**U**



**V**



**W**



**X**

## Z

# Listings

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Literaturverzeichnis


ASCHBACHER, H. 2009. *Entwicklung eines Service Engineering Prozessmodells für Software Solution Provider im KMU Sektor unter Nutzung eines webbasierten Informationssystems*. 1, 14, 15, 66, 67

BARTH, W. 2005. *Nagios: System- und Netzwerk-Monitoring*. Open Source Press. ISBN 3-937-51409-0. 52

BASS, L., CLEMENTS, P., & KAZMAN, R. 1998. *Software Architecture in Practice*. Addison Wesley Longman. ISBN 0-201-19930-0. 17, 18, 29, 30, 34

BERSON, A. 1996. *Client/Server Architecture*. 2. edn. McGraw-Hill Series on Computer Communications. McGraw-Hill. ISBN 0-07-005664-1. 30, 31, 32

BIERMAN, A., & IDDON, R. 1997 (Jänner). *RFC 2074 - Remote Network Monitoring MIB Protocol Identifiers*. http://www.ietf.org/rfc/rfc2074.txt [01.08.2009]. 42

BJORNER, D. 2006. *Software Engineering 3 - Domains, Requirements and Software Design*. Springer. ISBN 3-540-21151-9. 64

BLUMENTHAL, U., & WIJNEN, B. 2002 (Dezember). *RFC 3414 - User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)*. http://www.faqs.org/rfcs/rfc3414.txt [01.08.2009]. 40

BUSCHMANN, F., MEUNIER, R., ROHNERT, H., SOMMERLAD, P., & STAL, M. 1998. *Pattern-orientierte Software-Architektur - Ein Pattern System*. Addison Wesley. ISBN 3-8273-1282-5. 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28

CASE, J., FEDOR, M., SCHOFFSTALL, M., & DAVIN, J. 1990. *RFC 1157 - A Simple Network Management Protocol (SNMP)*. http://www.ietf.org/rfc/rfc1157.txt [01.08.2009]. 37, 40

CONTA, A., DEERING, S., & GUPTA, M. 2006 (März). *RFC 4443 - Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification*. http://www.ietf.org/rfc/rfc4443.txt [01.08.2009]. 37

DUSTDAR, S., GALL, H., & HAUSWIRTH, M. 2003. *Software-Architekturen für verteilte Systeme*. Springer. ISBN 3-540-43088-1. 34

EJB 3.0 EXPERT GROUP. 2006. *JSR 220 FR - Enterprise JavaBeans 3.0 Spezifikation*. http://java.sun.com/products/ejb/docs.html [16.09.2009]. 9

FLIESS, G. 2009. *Verteilte Architekturen - Grundlegende Überlegungen und Patterns*. Basta!2009 Spring, http://it-republik.de/konferenzen/bastaspring/speaker/. 13

FORSTER, F. 2009a. *Collectd Homepage*. http://collectd.org [01.08.2009]. 60

FORSTER, F. 2009b. *Collectd Homepage - Chains*. http://collectd.org/wiki/index.php/Chains [27.09.2009]. 52, 53, 54

FORSTER, F. 2009c. *Collectd Homepage - Features*. http://collectd.org/features.shtml [27.09.2009]. 53

GAMMA, E., HELM, R., JOHNSON, R., & VLISSIDES, J.M. 1995. *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison Wesley. ISBN 0-201-63361-2. 29, 30, 50, 73

GERHARDS, R. 2009 (März). *RFC 5424 - The Syslog Protocol*. http://tools.ietf.org/rfc/rfc5424.txt [01.08.2009]. 43, 44

HARL, S. 2008 (Mai). *collectd - the statistics collection & monitoring daemon*. http://osprey.tokkee.org/~tokkee/tmp/collectd.pdf [01.08.2009]. 52, 54

HRUSCHKA, P, & STARKE, G. 2009. *arc42 - Resourcen für Software-Architekten*. http://www.arc42.de/template/template/08-arc-aspects.html [28.09.2009]. 17

INGRAHAM, R.W. 2006. *The Nagios 2.X Event Broker Module API*. http://nagios.sourceforge.net/download/contrib/documentation/misc/NEB2xModuleAPI.pdf [01.08.2009]. 50

INTEL CORPORATION. 2009. *Intel IMPI Homepage*. http://www.intel.com/design/servers/ipmi/index.htm [01.08.2009]. 35, 36

INTEL CORPORATION, HEWLETT-PACKARD COMPANY, NEC CORPORATION, & DELL INC. 2009. *IPMI Spezifikation*. http://download.intel.com/design/servers/ipmi/IPMI2_0E4_Markup_061209.pdf [01.08.2009]. 35

JASPERSOFT CORPORATION. 2009. *Jasper Reports API - Exporter*. http://jasperreports.sourceforge.net/api/index.html [03.11.2009]. 78

JIA, W., & ZHOU, W. 2005. *Distributed network systems: from concepts to implementations*. Springer. ISBN 0-387-23839-5. 24

LEFFINGWELL, D., & WIDRIG, D. 2003. *Managing Software Requirements - A Use Case Approach*. 2. edn. Pearson Education. ISBN 0-321-12247-X. 61

LONVICK, C. 2001 (August). *RFC 3164 - The BSD syslog Protocol*. http://www.ietf.org/rfc/rfc3164.txt [01.08.2009]. 43

McCloghrie, K., Perkins, D., Schoenwaelder, J., Case, J., McCloghrie, K., Rose, M., & Waldbusser, S. 1999 (April). *RFC 2578 - Structure of Management Information Version 2 (SMIv2)*. http://tools.ietf.org/html/rfc2578.html [01.08.2009]. 37

Nagios Enterprises. 2009a. *Nagios 3 Dokumentation*. http://nagios.sourceforge.net/docs/nagios-3.pdf [01.08.2009]. 49, 50, 51

Nagios Enterprises. 2009b. *Nagios Homepage - FAQ*. http://support.nagios.com/knowledgebase/faqs/?section_id=30&expand=false&showdesc=true [27.09.2009]. 49

Nagios Enterprises. 2009c. *NetSaint Homepage*. http://netsaint.sourceforge.net [01.08.2009]. 49

New, D., & Rose, M. 2001 (November). *RFC 3195 - Reliable Delivery for syslog*. http://www.ietf.org/rfc/rfc3195.txt [01.08.2009]. 43

Oestereich, B. 2001. *Objektorientierte Softwareentwicklung - Analyse und Design mit der Unified Modeling Language*. 5. edn. Oldenbourg. ISBN 3-486-25573-8. 64, 65

Oestereich, B. 2006. *Analyse und Design mit UML 2.1 - Objektorientierte Softwareentwicklung*. 8. edn. Oldenbourg. ISBN 3-486-57926-6. 62

Okmianski, A. 2009 (März). *RFC 5426 - Transmission of Syslog Messages over UDP*. http://tools.ietf.org/rfc/rfc5426.txt [01.08.2009]. 43

Peh, D., Hague, N., & Tatchell, J. 2008. *BIRT - A Field Guide to Reporting*. 2. edn. the eclipse series. Addison-Wesley. ISBN 978-0-321-58027-6. 79

Postel, J. 1981 (September). *RFC 792 - INTERNET CONTROL MESSAGE PROTOCOL*. http://www.ietf.org/rfc/rfc792.txt [01.08.2009]. 36, 37

Reenskaug, T. 1979. *MVC XEROX PARC 1978-79*. http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html [12.09.2009]. 25, 26, 27

Rickli, R. 2009a. *NeDi Homepage*. http://www.nedi.ch [01.08.2009]. 56

Rickli, R. 2009b. *NeDi Homepage - About*. http://www.nedi.ch/about [26.09.2009]. 56

Rickli, R. 2009c. *NeDi Homepage - Development*. http://www.nedi.ch/dev:development [26.09.2009]. 56, 57, 58

Rickli, R. 2009d. *NeDi Homepage - Requirements*. http://www.nedi.ch/install:requirements [26.09.2009]. 58

Rose, M., & McCloghrie, K. 1991a (März). *RFC 1212 - Concise MIB Definitions*. http://www.ietf.org/rfc/rfc1212.txt [01.08.2009]. 38

Rose, M., & McCloghrie, K. 1991b (März). *RFC 1213 - Management Information Base for Network Management of TCP/IP-based internets: MIB-II*. http://www.ietf.org/rfc/rfc1213.txt [01.08.2009]. 38

SCHMARANZ, K. 2001. *Softwarenentwicklung in C*. Springer. ISBN 3-540-41958-6. 20

SCHMIDT, D., STAHL, M., ROHNERT, H., & BUSCHMANN, F. 2006. *Pattern-Oriented Software Architecture - Patterns for Concurrent and Networked Objects*. 2 edn. Wiley. ISBN 0-471-60695-2. 28

SCHWENKLER, T. 2006. *Sicheres Netzwerkmanagement*. Springer. ISBN 3-540-23612-2. 35

SHAW, M., & GARLAN, D. 1996. *Software architecture - perspectives on an emerging discipline*. Prentice Hall. ISBN 0-13-182957-2. 18, 21, 29, 30

STALLINGS, W. 1999. *SNMP, SNMPv2, SNMPv3 and RMON 1 and 2*. third edn. Addison Wesley. ISBN 0-201-48534-6. 36, 37, 38, 39, 40, 41, 42

STARKE, G. 2008. *Effektive Software-Architekturen - Ein praktischer Leitfaden*. 3. edn. Carl Hanser Verlag. ISBN 978-3-446-41215-6. 16, 17, 18, 19, 20, 21, 50

SUN DEVELOPER NETWORK. 2005. *request for enhancement 6336608*. http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=6336608 [02.10.2009]. 69

SUN MICROSYSTEMS. 2006a. *JDK 6 Documentation*. http://java.sun.com/javase/6/docs [12.09.2009]. 24

SUN MICROSYSTEMS. 2006b. *JDK 6 Documentation - Calendar Class*. http://java.sun.com/javase/6/docs/api/java/util/Calendar.html [02.10.2009]. 69

SUN MICROSYSTEMS. 2006c. *JDK 6 Documentation - ClassLoadingMXBean Class*. http://java.sun.com/javase/6/docs/api/java/lang/management/ClassLoadingMXBean.html [02.10.2009]. 70

SUN MICROSYSTEMS. 2006d. *JDK 6 Documentation - File Class*. http://java.sun.com/javase/6/docs/api/java/io/File.html [02.10.2009]. 69

SUN MICROSYSTEMS. 2006e. *JDK 6 Documentation - InetAddress Class*. http://java.sun.com/javase/6/docs/api/java/net/InetAddress.html [02.10.2009]. 69

SUN MICROSYSTEMS. 2006f. *JDK 6 Documentation - OperatingSystemMXBean Class*. http://java.sun.com/javase/6/docs/api/java/lang/management/OperatingSystemMXBean.html [02.10.2009]. 69, 70

SUN MICROSYSTEMS. 2006g. *JDK 6 Documentation - ThreadMXBean Class*. http://java.sun.com/javase/6/docs/api/java/lang/management/ThreadMXBean.html [02.10.2009]. 70

SUN MICROSYSTEMS. 2009. *Sun Microsystems Homepage*. http://java.sun.com/javase/6/docs/technotes/guides/jmx/index.html [01.08.2009]. 46

TANNENBAUM, A.S. 1997. *Computernetzwerke*. Übersetzung von a. shafir, 3. edn. Prentice Hall. ISBN 3-8272-9536-X. 18, 35

TECHNOLOGIESTIFTUNG HESSEN GMBH. 2002. *Dienstleistung - von der Renditefalle zum Wettbewerbsvorteil*. http://www.syneco.de/publikationen/0209Dienstleistungen.pdf. 1, 89

THE ECLIPSE FOUNDATION. 2009a. *Eclipse BIRT Homepage - Downloads*. http://download.eclipse.org/birt/downloads/ [21.10.2009]. 78

THE ECLIPSE FOUNDATION. 2009b. *Eclipse BIRT Homepage - Integration*. http://www.eclipse.org/birt/phoenix/deploy/ [21.10.2009]. 78

THE OPENNMS GROUP. 2009. *OpenNMS Homepage*. http://www.opennms.org/wiki/Data_Collect [01.08.2009]. 60

ULLENBOOM, C. 2007. *Java ist auch eine Insel - Programmieren mit der Java Platform, Standard Edition Version 6*. 6. aktualisierte und erweiterte edn. Galileo Press. ISBN 978-3-89842-838-5. 46, 47

VIXIE, P.A., & AVOLIO, F.M. 2002. *Sendmail - Theory and Practice*. 2. edn. Butterworth-Heineman. ISBN 1-55552-229-X. 43

WALDBUSSER, S. 1991 (November). *RFC 1271 - Remote Network Monitoring Management Information Base*. http://www.ietf.org/rfc/rfc1271.txt [01.08.2009]. 41

WALDBUSSER, S. 1997 (Jänner). *RFC 2021 - Remote Network Monitoring Management Information Base Version 2 using SMIv2*. http://www.ietf.org/rfc/rfc2021.txt [01.08.2009]. 42

WEATHERSBY, J., BONDUR, T., CHATALBASHEVA, I., & FRENCH, D. 2008. *Integrating and Extending BIRT*. 2. edn. the eclipse series. Addison-Wesley. ISBN 978-0-321-58030-6. 78, 79, 83, 85

WEISBERG, J. 2009a. *Argus Homepage*. http://argus.tcp4me.com [01.08.2009]. 55

WEISBERG, J. 2009b. *Argus Homepage - Documentation*. http://argus.tcp4me.com/docs.html [27.09.2009]. 55

WEISBERG, J. 2009c. *Argus Homepage - History*. http://argus.tcp4me.com/history.html [27.09.2009]. 54, 55

WIJNEN, B., PRESUHN, R., & MCCLOGHRIE, K. 2002 (Dezember). *RFC 3415 - View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP)*. http://www.faqs.org/rfcs/rfc3415.html [01.08.2009]. 40

WÜTHERICH, G., HARTMANN, N., KOLB, B., & LÜBKEN, M. 2008. *Die OSGi Service Platform - Eine Einführung mit Eclipse Equinox*. dpunkt.verlag. ISBN 978-,-89864-457-0. 6, 7, 8, 12

WUNDERLICH, L. 2006. *Managing Java - Operating, Monitoring, Performance-Tests*. Entwickler Press. ISBN 3-939084-13-1. 44, 45, 46, 47

XITRUST SECURE TECHNOLOGIES GMBH. 2009. *XiTrust Secure Technologies GmbH Homepage: XiTrust Business Server*. http://www.xitrust.com/signatur-verschluesselung-systemlandschaft.php [23.09.2009]. 4, 5

ZDUN, U., & AVGERIOU, P. 2005. *Architectural Patterns Revisited - A Pattern Language*. http://www.infosys.tuwien.ac.at/Staff/zdun/publications/ArchPatterns.pdf [11.09.2009]. 16, 17, 18, 90