Dokumentation von IT-Infrastrukturarchitekturen:

Unterstützung mittels konventioneller Datenbanken und der Konzeptorientierten Inhaltsverwaltung

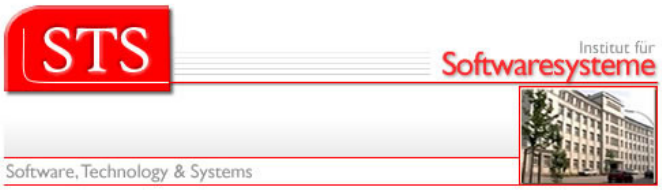
Diplomarbeit

zur Erlangung des akademischen Grades
Diplom Informatik-Ingenieur

vorgelegt von

Wjatscheslaw Lebsack

4. Oktober 2007



Gutachter:

Prof. Dr. Joachim W. Schmidt
Dr. Hans-Werner Sehring

Institut für Softwaresysteme (STS)
Technische Universität Hamburg-Harburg
`http://www.sts.tu-harburg.de/`
Harburger Schloßstr. 20
D-21073 Hamburg (Germany)

Gutachter:

Sven Müßig, MBA
Senior IT-Architect

IBM Deutschland GmbH
`http://www.ibm.de/`
Beim Strohhause 17
20097 Hamburg

Ehrenwörtliche Erklärung

Ich versichere, dass ich diese Diplomarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Ort, Datum: ..... Unterschrift: .....

Danksagung

An dieser Stelle möchte ich mich bei meinen Betreuern, Prof. Dr. Joachim W. Schmidt und Dr. Hans-Werner Sehring, dafür bedanken, dass sie es mir ermöglicht haben, mich mit diesem interessanten Thema auseinanderzusetzen.

Ganz herzlich bedanke ich mich bei meinem IBM-internen Betreuer Sven Müßig, der wohl den größten Aufwand mit mir hatte und mir mit seiner hervorragenden Betreuung und der investierten Zeit nicht nur bei der Fertigstellung der Arbeit mit konstruktiven Vorschlägen zur Seite stand, sondern mir auch im Rahmen der Arbeit durch unterschiedliche Interviews, Meetings, Telefonkonferenzen und Workshops einen Einblick in die Aufgabenbereiche eines IT-Architekten bei IBM gewährt hat. Den auf diesem Wege kennen gelernten Kollegen möchte ich ebenfalls an dieser Stelle für die Unterstützung danken.

Ich bedanke mich ganz herzlich bei den Kollegen des gesamten Student Development Team, die mir innerhalb von IBM zu diesem Thema verholfen und einen reibungslosen Ablauf während des kompletten Aufenthalts im Unternehmen gewährleistet haben.

Den gleichgesinnten Studenten sowohl innerhalb des Unternehmens als auch außerhalb danke ich für den einen oder anderen Tipp und eine angenehme Pausen- und Freizeitgestaltung.

Darüber hinaus möchte ich gerne meinen Eltern und meiner Freundin Nina Helrod für die Unterstützung und das entgegengebrachte Verständnis danken.

Inhaltsverzeichnis

Abbildungsverzeichnis

Quellcodeverzeichnis

Abkürzungen

AADL	Architecture Analysis & Design Language
AD	Architecture Description
ADL	Architecture Description Language
ADM	Architecture Development Method
ADS	Architecture Description Standard oder im Kontext der Konzept-orientierten Inhaltsverwaltung Assetdefinitionssprache
ANSI	American National Standards Institute
CCM	Conceptual Content Management
CCMS	Conceptual Content Management System
CDM	Custom Development Method
DU	Deployment Unit
GSM	Global Services Method
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Standards Organization
LPAR	Logical partition
MIME	Multipurpose Internet Mail Extensions
OEP	Object Engineering Process
OMG	Object Management Group
OPENUP	Open Unified Process
PI	Politische Ikonographie
RM-ODP	Reference Model For Open Distributed Processing
RUP	Rational Unified Process
SAE	Society of Automotive Engineers
SEI	Software Engineering Institute
SO	Strategic Outsourcing
TOGAF	The Open Group Architecture Framework
UML	Unified Modeling Language
WEL	Warburg Electronic Libriary
XMI	XML Metadata Interchange

1 Einleitung

Diese Arbeit ist in Zusammenarbeit mit dem Institut für Softwaresysteme der Technischen Universität Hamburg-Harburg und der IBM Deutschland GmbH entstanden. Es werden zwei wesentliche Ziele verfolgt. Zum einen die Dokumentation einer IT-Infrastrukturarchitektur in einer Datenbank und zum anderen der Einsatz der Konzeptorientierten Inhaltsverwaltung als alternative Möglichkeit zur Architekturmodellierung.

IBM ist ein Informationstechnologieunternehmen und besitzt eine sehr breit gefächerte Erfahrung. Es wurde vor fast 100 Jahren gegründet und hat seine Ursprünge sowohl in den USA als auch in Deutschland. In dieser Zeit hat sich das Unternehmen mehrmals neu definiert und hat es geschafft, durch Innovationen, die weit über Technologie hinausgehen, zu einer der stärksten Marken der Welt aufzusteigen.

Die Struktur des Unternehmens und dessen Geschäftsfelder haben sich im Laufe der Zeit stark verändert. Die am weitesten verbreitete Assoziation mit IBM ist die eines großen Herstellers von Hardware-Lösungen. Dieses entspricht nicht mehr den tatsächlichen und aktuellen Geschäftsmodellen von IBM. Der Konzern hat sich zu Beginn des 21. Jahrhunderts zu einem global integrierten Business- und Technologiepartner weiter entwickelt. Es werden mittlerweile neben modernsten Hardware- und Softwarelösungen industriespezifische Beratungs- und Implementierungsleistungen angeboten. Zur Unterstützung von Geschäftsprozessen werden maßgeschneiderte Lösungen sowohl für mittelständische Firmen als auch für die so genannte Global Player angeboten. Daneben gibt es auch einen großen Bereich, der IT-Lösungen für Unternehmen aller Größen betreut, das so genannte Strategic Outsourcing[1].

Im Rahmen eines solchen Outsourcing-Auftrags an die IBM von einem Finanzdienstleister unterstützt die IBM beim Management und Betrieb der Anwendungen und IT-Systeme dieses Finanzdienstleisters. Zu Beginn der Vertragslaufzeit sind zum Teil die Mitarbeiter und die bestehende IT-Landschaft des Auftraggebers übernommen worden. Durch den Einsatz zusätzlicher, eigener Fachkräfte und die Verwendung von den im eigenen Haus entwickelter Technologien entstand ein Projekt, in dessen Rahmen die gesamte IT-Landschaft des Kunden betreut, am Laufen gehalten, weiter ausgebaut und an die sich ständig ändernden Anforderungen angepasst wird. Dieses geschieht in einer engen Zusammenarbeit mit dem Auftraggeber und unter Verwendung neuer innovativer Lösungen, aber auch durch den fortführenden Einsatz bestehender, funktionierender und unentbehrlicher Technologie und Tools.

1.1 Motivation

Zusätzlich zur Beschreibung einzelner Softwaresysteme gewinnt die Dokumentation von System- und Anwendungslandschaften, bestehend aus einer Vielzahl von Soft- und Hardwarekomponenten, zunehmend an Bedeutung. Die IT-Infrastruktur von Unternehmen ist in den letzten Jahren sehr schnell und

[1] Strategic Outsourcing ist ein Begriff für die Auslagerung von Unternehmensbereichen in andere Unternehmen, die sich auf den entsprechenden Bereich spezialisiert haben und dadurch ein höheres Maß an Erfahrung besitzen.

organisch gewachsen. Darüber hinaus sind Anwendungssysteme keine autonom funktionierenden Einheiten, sondern stellen komplexe Systeme dar, die in sich und mit anderen Einheiten verbunden sind und mit diesen zusammen große IT-Landschaften ergeben. Eine der Folgen davon ist, dass Unternehmen heute große Schwierigkeiten haben, den Überblick über die sich im Einsatz befindende Hard- und Software zu behalten. Es existiert vielfach kein einheitliches und umfassendes Modell zum Architekturmanagement [Uni06].

Daraus ergeben sich weitere Probleme:

- die Interaktionen zwischen den Arbeitsabläufen und den Hard- und Softwarekomponenten sind nur schwer nachvollziehbar
- die Anschaffung und Integration neuer Geschäftsanwendungen lässt sich aufgrund mangelhaften Überblicks über die IT-Landschaft nur mühsam steuern
- eine zeitnahe und gezielte Reaktion auf auftretende Probleme gestaltet sich schwierig
- es werden nicht einheitliche Tools und Methoden zur Lösung gleicher Aufgaben eingesetzt
- ein Zusammenführen mehrerer IT-Landschaften und -Systeme, nach z.B. einer Firmenübernahme (engl. Merger & Acquisition), gestaltet sich schwierig

Im Zusammenhang mit der Modellierung einer IT-Architektur wird mittlerweile die Frage gestellt, ob die Konzeptorientierte Inhaltsverwaltung (engl. Conceptual Content Management, abgekürzt CCM) [Seh04] als alternative Möglichkeit für Software-Engineering und die Modellierung von IT-Architekturen in Erwägung gezogen werden kann. Dadurch ließen sich die Vorzüge des CCM, wie Offenheit und Dynamik (siehe Unterkapitel 4.2), auf die Modellierung übertragen. Damit bestünde jederzeit die Möglichkeit, das Modell zu ändern und zu erweitern. Die Rede wäre in so einem Fall kaum noch von einem Standard, sondern von einer flexiblen und personalisierten Variante der Modellierung. Zu beachten wäre, dass die Darstellung der Information, die sich mit Hilfe eines Standards allgemein verständlich gestaltet, durch die Konzeptorientierte Inhaltsverwaltung nicht in einem Durcheinander ausartet.

1.2 Zielsetzung der Arbeit

Es soll zunächst untersucht werden, welche verbreiteten Möglichkeiten heutzutage in der Praxis und Forschung existieren, IT-Architekturmodelle zu beschreiben. Dabei sollen unterschiedliche Sprachen, Standards und Methoden betrachtet werden, die als Verfahren zur Software-Entwicklung und IT-Architekturbeschreibung verwendet werden.

Im konkreten zu beschreibenden und zu modellierenden Szenario soll eine Architekturbeschreibungssprache (engl. Achitecture Description Language, abgekürzt ADL) inklusive einer Methode nicht bei der Entwicklung einer neuen Software oder dem Errichten einer neuen IT-Architektur, sondern zum Modellieren einer schon bestehenden komplexen IT-Infrastruktur eingesetzt werden. Am Beispiel eines Finanzdienstleisters soll ein Datenmodell[2] mit Hilfe einer ADL aufgestellt und in einer Datenbank umgesetzt werden. Die Datenbank kann dann in Folge zum Management der IT-Architektur verwendet

[2]Das Ergebnis der Datenmodellierung ist ein Datenmodell. Unter Datenmodellierung ist die formale Beschreibung der Unternehmensdaten in einem Datenmodell zu verstehen. Das Datenmodell liegt meistens in einer Form vor, die aus Diagrammen und einer mehr oder weniger detaillierten textuellen Beschreibung unter Verwendung einer Architekturbeschreibungssprache erstellt worden ist.

werden. Sie ermöglicht im Gegensatz zu manuellen Lösungen auf der Basis von Modellierungssoftware, wie z.B. Rational Software Modeler, einen integrierten Ansatz.

Des Weiteren soll untersucht werden, ob es möglich ist, die klassische Darstellung eines Datenmodells in Form einer ADL durch die Variante in Form von Assets[3] und dem Conceptual Content Management zu ersetzen. Dabei soll sowohl auf die Elemente eines Datenmodells als auch auf die dem Modell zu Grunde liegenden Informationsquellen eingegangen werden.

1.3 Gliederung der Arbeit

Zunächst wird in Kapitel 2 eine Auswahl unterschiedlicher Standards und Methoden vorgestellt, die in der Wirtschaft und Forschung existieren. Es wird auf deren Notwendigkeit, Schwerpunkte und Einsatzbereiche eingegangen. In Kapitel 3 wird unter Verwendung eines Modellierungsstandards ein Modell entworfen, welches eine existierende IT-Infrastruktur beschreibt. Auf der Basis dessen wird im Rahmen dieser Arbeit eine Infrastrukturdatenbank implementiert, die ebenfalls in diesem Kapitel vorgestellt wird. Kapitel 4 beschäftigt sich mit der Konzeptorientierten Inhaltsverwaltung und der Eignung dieser für den Einsatz bei der Dokumentation und dem Entwurf von IT-Infrastrukturen. Das Kapitel 5 enthält eine abschließende Bewertung und den Ausblick.

1.4 Notation

Um den Inhalt der Arbeit übersichtlicher zu gestalten wurden je nach Thematik unterschiedliche Schriftarten verwendet. Aus Gründen der Lesbarkeit des Textes wurde bei allen Bezeichnungen jeweils auf die weibliche Form verzichtet und durchgehend nur die männliche Form verwendet. Selbstverständlich ist z.B. mit „Anwender“ auch die „Anwenderin“ gemeint.

Die folgende Notation ist in der Arbeit anzutreffen:

- *Kursiv*: Zitate, Abkürzungen und Fachbegriffe
- **Fett**: Überschriften und wichtige Schlagworte
- Sans Serif: Variablen aus Quellcode oder Modellen
-

```
1 Darstellung von Assetdefinitionssprache- und XML-Quellcode
```

[3] Unter einem Asset ist ein gekapselter Satz von Informationen und Erfahrungen zu verstehen, die wieder verwendet werden können und aus Effizienzgründen sogar verwendet werden sollten.

2 Standards und Methoden für die klassische Modellierung

Das IT-Architekturmanagement ist ein Prozess, der auf dem Weg zur Umsetzung unternehmensstrategischer Ziele durchlaufen werden sollte. Das ist eine aus langjähriger praktischer Erfahrung gewonnene Erkenntnis [IBM07a].Grundsätzlich trägt er dazu bei, die Gesamtkosten durch eine Reduzierung der Komplexität zu senken, die IT-Strategie zu verbessern und die Qualität der Kommunikation zu erhöhen. Dafür ist es notwendig, sich an Standards zu halten. Im Rahmen dieser Diplomarbeit wird allerdings kein IT-Infrastrukturmodell aufgestellt, mit der Absicht eine neue IT-Infrastruktur innerhalb eines Unternehmens aufzubauen. Der Schwerpunkt in diesem Fall liegt bei dem so genannten Reverse Engineering.

In dem Unterkapitel 2.1 wird auf die Verwendungsmöglichkeiten der Standards und Methoden eingegangen und basierend auf den unterschiedlichen Definitionen einer IT-Systemarchitektur eine für dieser Arbeit relevante Definition hergeleitet.

Der Reverse Engineering-Prozess und die traditionelle Methode namens Forward Engineering werden im Unterkapitel 2.2 näher vorgestellt und deren wesentlichen Unterschiede erläutert. In dem Unterkapitel 2.3 wird ein ausgewählter Satz von, in der Wirtschaft und der Forschung vorhandenen, Standards und Methoden vorgestellt.

2.1 Notwendigkeit und Verwendung

Um die Effizienz der Projekte und die Zufriedenheit der Kunden zu steigern, wird zum Einsatz von einheitlicher und allgemeinverständlicher Vorgehensweise bei der Entwicklung von Software und dem Modellieren von Systemen geraten. Die Verwendung standardisierter Verfahren sorgt für Sicherheit und Stabilität. Ein Unternehmen gewinnt dadurch an Unabhängigkeit und Freiheit, weil das Know-how zu standardisierten Technologien und Methoden leichter auf dem freien Markt zu finden ist. Beim Einsatz einer individuellen Vorgehensweise sollte nicht außer Acht gelassen werden, dass das Wissen lückenhaft dokumentiert und gepflegt sein und sich unter Umständen nur in den Köpfen der Mitarbeiter befinden kann. Die Folge davon ist meistens die Unentbehrlichkeit der Mitarbeiter, die sich letztendlich in den Kosten niederschlägt.

Aufgrund der Tatsache, dass sich eine Vielzahl von Experten mit den Standards beschäftigt, was von proprietären Lösungsansätzen nicht behauptet werden kann, sind die standardisierten Vorgehensweisen in der Regel viel ausgereifter und leistungsfähiger. Sie kommen aus der Praxis und sind für die Praxis gemacht und werden oft durch Werkzeuge unterstützt. Ohne Zweifel weisen auch die standardisierten Methoden Schwachstellen auf. Da sie allerdings meistens offen diskutiert werden, kann diesen mit entsprechenden Maßnahmen begegnet werden.

In der Software-Entwicklung wächst der Quellcode von Programmen nach und nach. Dieses führt zu Problemen, die unter dem Begriff „Software Crisis“ zusammengefasst werden. Eine Struktur und Planung sind von Nöten. Bei der Arbeit in einem Team ist es essentiell, sich mit Kollegen abzusprechen. Ein Projekt muss koordiniert werden, damit jeder Einzelne weiß, was er zu tun hat und die teilweise voneinander unabhängig entstandenen Ergebnisse ineinander greifen und zu einem Ganzen zusammengesetzt werden können. Die Koordination ist nicht nur innerhalb eines Entwicklungsteams oder den Systemmodellierern notwendig, sondern auch mit anderen Schichten des Unternehmens, die meistens eine „andere Sprache sprechen“. Es muss also eine Ebene eingeführt werden, die sowohl die Arbeit innerhalb eines Projektes erleichtert als auch zwischen den unterschiedlichen in das Projekt involvierten Schichten des Unternehmens, wie z.B. dem Management und den Entwicklern, vermittelt. Um diese Schwierigkeiten in den Griff zu bekommen, stellt eine ADL dem Projekt ein Instrument zur Seite.

Eine ADL kommt zum Einsatz, um eine Software vor der tatsächlichen Implementierung und eine sowohl schon vorhandene als auch die sich in der Planung befindende IT-Architektur und -Infrastruktur zu beschreiben. Dieses geschieht sowohl in graphischer als auch in textueller Form. Durch die vereinheitlichte, gemeinsame Sprache ist es für die neuen Projektmitarbeiter möglich, durch die vorhandenen Dokumente schnell auf den aktuellen Stand zu kommen und Projektentscheidungen und Vorgehensweisen klar nachzuvollziehen. Eine Methode schreibt die Vorgehensweise bei der Durchführung eines Projektes vor, um basierend auf den in ihr gebündelten langjährigen Erfahrungen ein Projekt möglichst innerhalb des vorgegebenen Zeitraumes und Budgets zu koordinieren. Unter anderem legt eine Methode eine ADL fest.

Auch unter den Standards existiert keine einheitliche Lösung. Eine Systemarchitektur ist eine Darstellung von Beziehungen zwischen den einzelnen Teilen des Systems. Es existiert keine strenge Definition darüber, aus welchen Aspekten sich eine Systemarchitektur zusammensetzen muss. Die vielfältige Auslegung einer Systemarchitektur führt dazu, das unterschiedliche ADLs von verschiedenen Gruppen entwickelt werden und die Unternehmen jeweils eigene Methoden entwickelt haben, um deren Systeme zu modellieren, die allerdings innerhalb der Unternehmen wenn nicht einheitlich, dann zumindest großflächig eingesetzt werden. Die unterschiedlichen ADLs und Methoden enthalten aber in der Regel einen ähnlichen Grundgedanken. Im Folgenden werden verschiedene Definitionen von verschiedenen Organisationen vorgeschlagen. Diese werden als Zitate in englischer Sprache verwendet:

- „*The fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.*“ [IEE00]
- „*The composite of the design architectures for products and their life cycle processes.*“ [IEE98]
- „*A representation of a system in which there is a mapping of functionality onto hardware and software components, a mapping of the software architecture onto the hardware architecture, and human interaction with these components.*“ [CMU07d]
- „*An architecture is the most important, pervasive, top-level, strategic inventions, decisions, and their associated rationales about the overall structure (i.e., essential elements and their relationships) and associated characteristics and behavior.*“ [OPE07]
- „*A description of the design and contents of a computer system. If documented, it may include information such as a detailed inventory of current hardware, software and networking capabilities; a description of long-range plans and priorities for future purchases, and a plan for upgrading and/or replacing dated equipment and software.*“ [The07b]

- „*A formal description of a system, or a detailed plan of the system at component level to guide its implementation.*“ [The07f]
- „*The structure of components, their interrelationships, and the principles and guidelines governing their design and evolution over time.*“ [The07f]

Die meisten Definitionen enthalten eine Darstellung eines Systems auf Komponentenebene mit den Beziehungen zwischen den Komponenten. Die Definition aus [CMU07d] berührt als einzige die Abbildung der Software auf die Hardware, lässt allerdings die Komponentendarstellung aus. Die Definition aus [The07b] enthält als erste, die ebenfalls in dieser Arbeit verfolgten Absichten, eine Ist-Aufnahme der Soft- und Hardware und der Netzwerkeigenschaften zu erstellen, mit der Absicht damit langfristige Pläne für den Einkauf neuer Betriebsmittel und die Aktualisierung bestehender Software zu entwerfen.

Die jeweiligen Definitionen schneiden an vielen Stellen die, in dieser Arbeit verfolgten, Absichten an. Allerdings vereint keine dieser Definitionen die kompletten Anforderungen in sich. In Form von Komponenten sollen nur die Infrastrukturanwendungen[1] dargestellt werden, d.h. die Geschäftsanwendungen werden nicht berücksichtigt. Des Weiteren wird eine Ist-Aufnahme der bestehenden Hardware und der darauf laufenden logischen Partitionen und der Netzwerke erstellt. Die Komponenten der Infrastrukturanwendungen werden als Service auf die Hardware bzw. die logischen Server abgebildet.

Unter Verwendung der bestehenden Ansätze wird in dieser Arbeit die Systemarchitektur einschränkend folgendermaßen definiert:

> „Die Architektur eines IT-Systems ist eine auf Komponenten und deren Beziehungen basierende Darstellung der Infrastrukturanwendungen, die auf die, durch eine Ist-Aufnahme der Hardware und der Netzwerke entstandene, Topologie eines Systems abgebildet werden.“

2.2 Reverse Engineering und Forward Engineering

Die traditionelle Methode der Entwicklung einer Technologie wird als Forward Engineering bezeichnet. Bei der Entwicklung werden unterschiedliche technische Konzepte und Abstraktionen implementiert. Im Gegensatz dazu beginnt das Reverse Engineering mit dem Endprodukt und arbeitet rückwärts. Das Ziel dieser Methode ist, eine Verbesserung der Dokumentation sowie der Arbeitsweise des ursprünglichen Produkts, in dem vorliegenden Fall einer IT-Infrastruktur, zu erreichen, indem die zu Grunde liegende Struktur aufgedeckt wird. Reverse Engineering als Methode dient nicht unbedingt einem bestimmten Zweck, ist aber ein Teil der wissenschaftlichen Verfahren und der technologischen Entwicklung.

Während des Reverse Engineering-Prozesses sind die Forscher und Entwickler in der Lage, die Stärken und die Schwächen eines Systems in Bezug auf die Leistung, Sicherheit und Kompatibilität zu untersuchen. Dieser erlaubt es, sowohl die Arbeitsweise zu erkennen als auch Bereiche des Systems zu entdecken, die zu einem ungewünschten Verhalten führen. Die Entwicklung eines besseren Designs, die Erweiterung und Verbesserung von Funktionsfähigkeiten existierender Produkte beginnt häufig mit dem Reverse Engineering.

[1] Infrastrukturanwendungen sind Anwendungen, die zum Management und zur Überwachung der IT-Infrastruktur eingesetzt werden, wie z.B. das in dem Kapitel 3.3 vorgestellte Server Resource Management.

In dieser Arbeit wird der Reverse Engineering-Ansatz verwendet, um eine IT-Infrastruktur abstrakt darzustellen und zu dokumentieren. Er kann allerdings unterschiedlich eingesetzt werden. Im Folgenden sind einige Beispiele des verschiedenen Gebrauchs des Reverse Engineering aufgeführt:

- detaillierte Kenntnis über die Funktionsweise eines Produkts
- Untersuchung und Korrektur von Fehlern und Beschränkungen innerhalb bestehender Systeme
- Untersuchung der Designprinzipien eines Produktes als ein Teil der Ingenieur-Bildung
- Erstellen von miteinander kompatibler Produkten und Systemen zwecks Zusammenarbeit und Austausch gemeinsamer Daten
- Beurteilung ausrangierter Produkte, um die Grenze deren Leistungsfähigkeit nachzuvollziehen
- Transformation unbrauchbarer Produkte in die Wiederverwendung dieser durch die Anpassung an neue Systeme und Plattformen

2.3 Standards und Methoden

In diesem Unterkapitel werden die in der Praxis und Forschung verbreiteten Standards und Methoden betrachtet, die als Verfahren zur Software-Entwicklung und Architekturbeschreibung verwendet werden. Dabei wird ein Satz von Architekturbeschreibungssprachen ausgewählt und etwas genauer vorgestellt. Die Auswahlkriterien sind z.B. der Bekanntheitsgrad wie bei der Unified Modeling Language (UML) und dem Rational Unified Process (RUP), Eignung zur praktischen Anwendung im Rahmen dieser Arbeit, wie z.B. der IBM Global Services Method (GSM) und dem Architecture Description Standard (ADS), Einsatzbereiche, wie z.B. der IEEE 1471 für softwareintensive Systeme oder das Reference Model for Open Distributed Processing (RM-ODP) für verteilte Systeme. Darüber hinaus wird ein kleiner Ausschnitt aus den ADLs mit einem geringeren Bekanntheitsgrad, aber einem bemerkenswerten Ursprung näher vorgestellt, wie z.B. UniCon von der Carnegie Mellon University und AADL von der Society of Automotive Engineers. Von der ausführlichen Vorstellung der Vorgehensmodelle, wie Wasserfall- oder V-Modell wird abgesehen. Eine Liste von, auch nicht in diesem Kapitel erwähnten, ADLs und Methoden ist im Anhang in der Tabelle A.1 zu finden.

2.3.1 Unified Modeling Language

Unified Modeling Language (UML) [Obj07] ist die wohl am weitesten verbreitete ADL, die sich langsam als Industriestandard herauskristallisiert. UML selbst wurde durch die Object Management Group (OMG) standardisiert. Die UML ist aus verschiedenen Sprachen entstanden und hat, wie die Abbildung 2.1 auf Seite 9 zeigt, eine lange Entwicklungsgeschichte hinter sich. Im Jahre 1997 kam die erste standardisierte Version der Sprache heraus, die in den darauf folgenden Jahren an vielen Stellen optimiert wurde und im Jahre 2004 offiziell unter UML 2.0 verabschiedet wurde.

Die UML besitzt eine hohe Bandbreite und deckt große und unterschiedliche Bereiche einer Anwendung oder eines Systems ab. Meistens werden die Möglichkeiten von UML nicht innerhalb eines Projektes komplett ausgeschöpft. Es wird nur ein, zur Lösung des bestehenden Problems benötigter, Satz an UML-Werkzeugen verwendet. Dieses setzt voraus, dass die Sprache modular aufgebaut ist, um nur die Teile auswählen zu können, die gerade relevant sind. Die Folge einer erhöhten Flexibilität ist die ebenfalls höhere Wahrscheinlichkeit, dass unterschiedliche UML-Tools sich auf unterschiedliche Bereiche

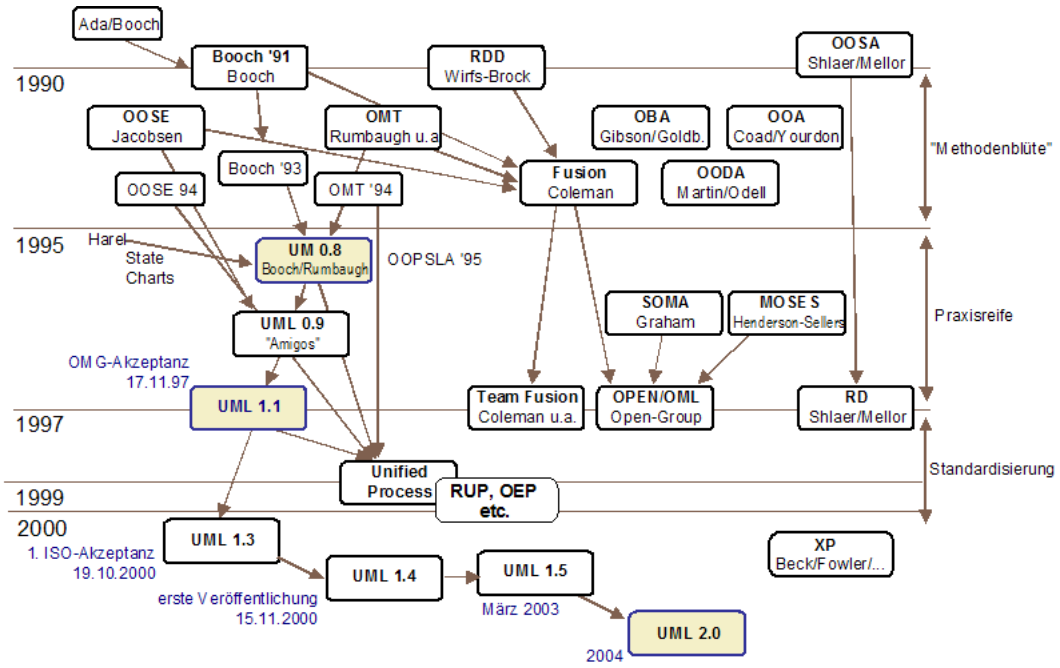


Abbildung 2.1: Historie der objektorientierten Methoden und Notationen [oos07a]

der Sprache konzentrieren und diese entsprechend unterstützen, was wiederum zu Austauschproblemen der UML-Modelle zwischen den einzelnen Tools führt. Es ist also eine Gradwanderung zwischen der Modularität und der Kompatibilität. Folglich ist die Anforderung, die an UML gestellt wird, ein Gleichgewicht zwischen diesen beiden Eigenschaften zu halten.

Die UML ist darauf ausgerichtet, Modelle in Form einer graphischen Notation zu erstellen. Durch eine graphische Beschreibung einer Software oder eines Systems wird ein wesentlich schnelleres Verständnis der Materie erreicht. Durch Modelle können in Abhängigkeit von der aktuellen Situation, wie z.B. der Zielgruppe, nur relevante Aspekte eines komplizierten realen Problems hervorgehoben und die weniger wichtige Teile vernachlässigt werden. Diese Vorgehensweise trägt zum besseren Verständnis des momentan wichtigen Ausschnitts des Systems bei.

Die UML bietet die folgenden 13 Arten von Modellierungsdiagrammen, die entweder zu den Struktur- oder den Verhaltensdiagrammen zählen:

Strukturdiagramme

- Klassendiagramm

  Dieses Diagramm beschreibt Klassen und ihre Beziehungen untereinander.

- Komponentendiagramm

  Der Fokus liegt hier auf den Komponenten sowie ihren Beziehungen und Schnittstellen.

- Kompositionsstrukturdiagramm

  Bei diesem Diagramm steht die Abbildung innerer Zusammenhänge einer komplexen Systemarchitektur sowie die Darstellung von Design Mustern im Vordergrund.

- Objektdiagramm

  Hier werden die Objekte, Assoziationen und Attributwerte zu einem bestimmten Zeitpunkt während der Laufzeit betrachtet.

- Verteilungsdiagramm

  Bei dieser Art von Strukturdiagrammen handelt es sich um Einsatzdiagramme, Knotendiagramme sowie Laufzeitumgebungen.

- Paketdiagramm

  Mit Hilfe dieses Diagramms werden Softwaresysteme in Untereinheiten gegliedert.

Verhaltensdiagramme

- Anwendungsfalldiagramm

  Diese Art von Verhaltensdiagrammen wird verwendet, um Beziehungen zwischen Akteuren und Anwendungsfällen darzustellen.

- Aktivitätsdiagramm

  Dieses Diagramm beschreibt Ablaufmöglichkeiten, die aus einzelnen Aktivitäten/Schritten bestehen.

- Kommunikationsdiagramm

  Das Ziel dieses Diagramms liegt darin, Beziehungen und Interaktionen zwischen den Objekten darzustellen.

- Sequenzdiagramm

  Hier steht der zeitliche Ablauf von Nachrichten zwischen den Objekten im Vordergrund.

- Zustandsdiagramm

  Dieses Verhaltensdiagramm zeigt eine Folge von Zuständen eines Objekts.

- Zeitverlaufsdiagramm

  Bei diesem Diagramm handelt es sich um ein Interaktionsdiagramm mit Zeitverlaufskurven von Zuständen.

- Interaktionsübersichtsdiagramm

  Ähnlich wie bei einem Aktivitätsdiagramm handelt es sich in diesem Fall um eine Übersicht über die Abfolgen von Interaktionen.

Die UML gehört zu den Sprachen, über die schon etliche Bücher, Whitepapers und Tutorials herausgebracht worden sind. Deswegen wird an dieser Stelle nicht auf weitere Details der Sprache eingegangen.

2.3.2 IBM Architecture Description Standard

Um methodisch vorgehen zu können und IT-Architektur Arbeitsergebnisse wiederverwendbar zu machen, war es notwendig, einen Standard zu definieren, an den sich alle IT-Architekten halten. Im Jahre 1998 wurde der Grundstein dazu gelegt, einen Standard innerhalb der IBM festzulegen. Dieser Standard sieht vor, die funktionalen und operativen Aspekte einer IT-Architektur vereinheitlicht darzustellen und steht seit 2002 als Version 2.0 zur Verfügung [Kah02].

IBM Architecture Description Standard (ADS) [Kah02] ist ein Derivat von UML und eine ADL von GSM, die im Unterkapitel 2.3.3 vorgestellt wird. UML wurde um die operationalen Aspekte eines Systems erweitert. Die Abbildung 2.2 auf Seite 12 stellt das Metamodell vom ADS dar und hebt dabei den funktionalen und den operationalen Aspekt hervor.

In großen Projekten ist eine Aufteilung der Verantwortung aus einem elementaren Grund unerlässlich. Es ist unmöglich, eine Person mit allen notwendigen Kenntnissen auszustatten, wie z.B. Technologien, Methoden und Tools. Darüber hinaus wird dadurch das Verständnis einer komplexen Aufgabe und eine geographisch verteilte Entwicklung gefördert. So werden die Aktivitäten eines Projektes in kleinere Blöcke unterteilt, mit denen sich unterschiedliche Arbeitsgruppen beschäftigen. ADS unterstützt die Aufteilung eines Systems. So wird dort zwischen den funktionalen und operationalen Aspekten unterschieden [Kah02].

- Funktionale Sicht

  Der funktionale Aspekt beschreibt die Funktion und die Interaktion der Softwarekomponenten eines Systems. Er drückt sich in Form mehrerer Komponenten aus und beschreibt sowohl deren statisches als auch dynamisches Verhalten mit Hilfe von z.B. „Component Relationship“-Diagrammen bzw. „Component Interaction“-Diagramen. Weitere Diagramme des funktionalen Aspektes sind:

  - Component Collaboration Diagram - stellt das dynamische Verhalten von Komponenten dar
  - Component Specification Diagram - stellt die statische Beziehung zwischen den Komponenten und deren Schnittstellen dar (sowohl angebotene als auch verwendete Schnittstellen)
  - Entity Class Diagram - stellt Entitäten, deren Attribute und Beziehungen dar
  - Interface Responsibility Diagram - stellt die Beziehungen zwischen den Komponenten, deren Schnittstellen und die durch diese verwalteten Datentypen dar

- Operationale Sicht

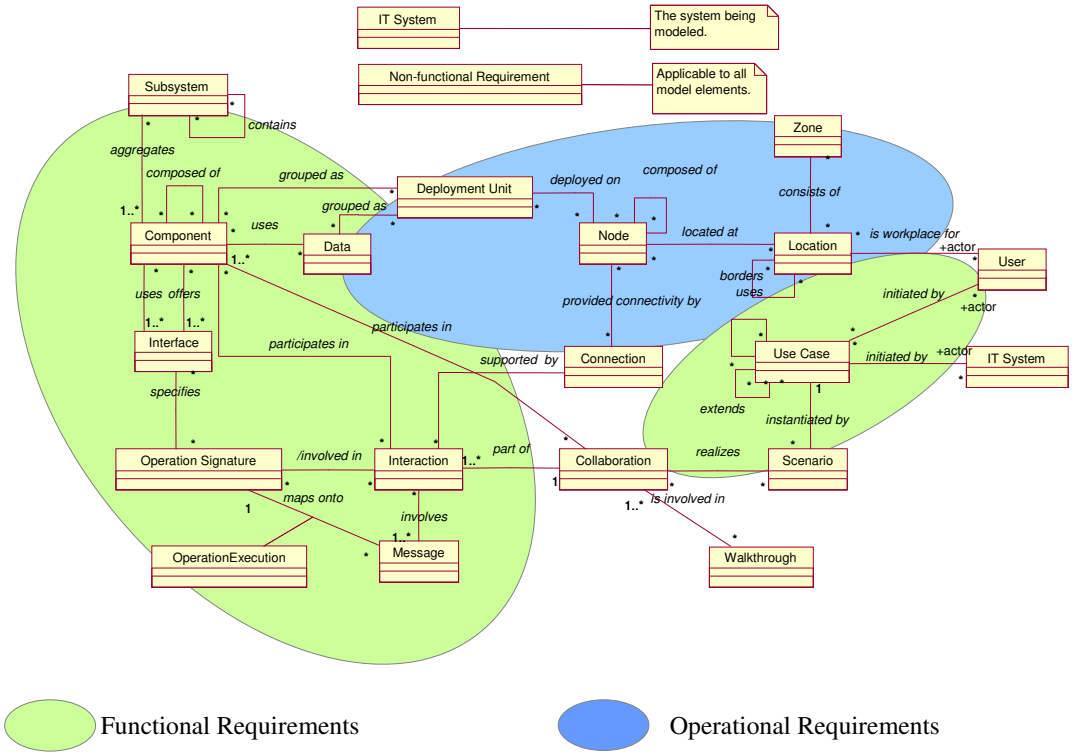


Abbildung 2.2: Vereinfachtes ADS-Metamodell mit funktionalen und operationalen Aspekten [Kah02]

Der operationale Aspekt beschäftigt sich mit der Verteilung der Komponenten über das komplette System und mit der geographischen Anordnung der Hardware, unter Berücksichtigung der erhobenen Bedingungen und Anforderungen, wie z.B. Performance und Verfügbarkeit. Darüber hinaus wird damit die Konfiguration des Systems dokumentiert. Die zu verwendende Hard- und Software, die Bandbreite, die Antwortzeit, Prozessorgeschwindigkeit, Festplattenkonfiguration usw. werden spezifiziert. Zur Darstellung eines Diagramms auf der operationalen Ebene werden neue Elemente eingeführt. In Abhängigkeit von der Zielgruppe kann zwischen unterschiedlichen Ebenen des operationalen Modells gewählt werden, die unterschiedlichen Detaillierungsgrad aufweisen:

- Conceptual Level
- Specification Level
- Physical Level

Ein operationales Modell kann sehr detailliert dargestellt werden. So gehören z.B. unmittelbar zu den Locations die Borders, zu deutsch Grenzen, durch die die Art der Verbindung (z.B. keine direkte Verbindung, niedrige Bandbreite und hohe Bandbreite) zwischen unterschiedlichen Lokationen dargestellt werden kann. Darüber hinaus existieren unterschiedliche Konventionen zur Darstellung von Zusatzinformationen zu z.B.

Nodes und DUs.

Mehr zu den funktionalen und operationalen Aspekten des ADS wird in Kapitel 3 vorgestellt und anhand von Beispielen näher gebracht. Detaillierte Information ist in [IBM02a] und [IBM02b] zu finden.

2.3.3 IBM Global Services Method

Die IBM Global Services Methode (GSM) [Ska03] ist eine Asset basierte Methode, in der die einst unterschiedlichen Methoden unterschiedlicher Servicebereiche, wie z.B. Software-Entwicklung, Unternehmensberatung und Outsourcing, konsolidiert wurden. Die Abbildung 2.3 stellt eine graphische Übersicht über die Bausteine der Methode dar.

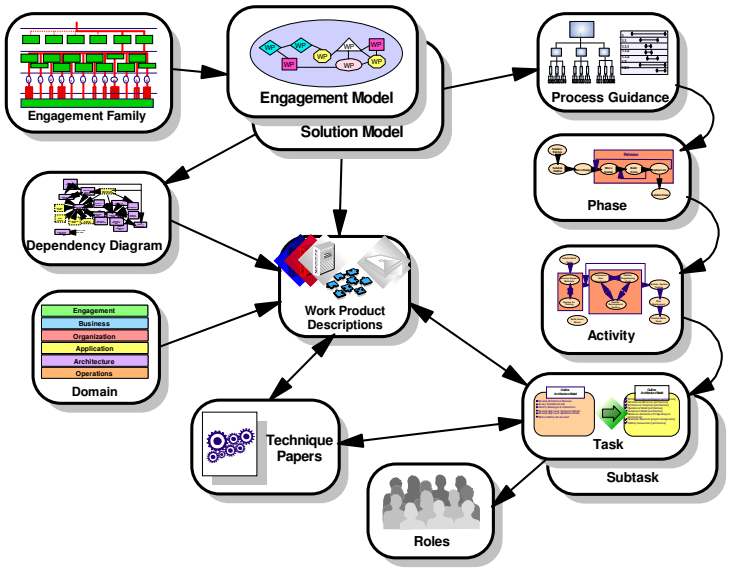


Abbildung 2.3: Überblick über die Bausteine der GSM [Ska03]

Die GS-Method stellt ein Framework („meta-model“) zur Verfügung, um den Methodeninhalt (Know-how) auf eine konsequente, zusammenhängende und integrierte Weise zu definieren, und stellt einen umfassenden Satz des Know-hows, welches für die IBM Fachleute auf der ganzen Welt als Basis zur Verfügung steht, um Kundenanforderungen in den Projekten zu definieren und umzusetzen.

Obwohl verschiedene Bereiche von IBM Global Services unterschiedliche Arten von Aufträgen durchführen und dem Kunden verschiedene Dienstleistungen bieten, greifen alle auf das schon vorhandene Depot der Arbeitsergebnisse (engl work product), der so genannten Assets, zurück oder erzeugen ähnliche Arbeitsergebnisse. Als es die GSM noch nicht gab, wurden diese Arbeitsergebnisse von jedem Business-Zweig des Unternehmens unabhängig voneinander definiert. Aufgrund der sich wiederholenden Prozesse und des dadurch mehrfachen Arbeitsaufwandes bestand an dieser Stelle ein nennenswertes Verbesserungspotential. Die GSM definiert also eine allgemeine Sprache und vordefinierte Bausteine, die unter Umständen nur teilweise an die neuen Anforderungen angepasst werden müssen und aus

denen unterschiedliche Modelle konstruiert werden können, was die Produktivität und die Effizienz stark fördert.

Eine Methode vereinheitlicht die Sprache, die im Projekt zur Kommunikation von Ergebnissen und Entscheidungen verwendet wird. Die IBM Global Services Methode verwendet z.B. die UML, erweitert um Definitionen für die Dokumentation von Architekturen. Ziel ist dabei, dass die verwendete Sprache möglichst eindeutig Ergebnisse festhält, und wenige Möglichkeiten zur Fehlinterpretation bietet. Für viele Teilaufgaben ist die Syntax zur Dokumentation fest vorgegeben.

In der GSM ist die Nutzung von Projekterfahrungen an mehreren Stellen fest vorgesehen. Dabei wird im Sinne eines „best practices“ Ansatz z.B. auf Referenzarchitekturen und Referenzlösungen zurückgegriffen. Durch die einheitliche Sprache und Dokumentation ist es einfacher, diese Ergebnisse auf ihre Verwendbarkeit hin zu prüfen und ggf. einzusetzen. In der Abbildung 2.4 sind die Kernpunkte der Methode dargestellt.

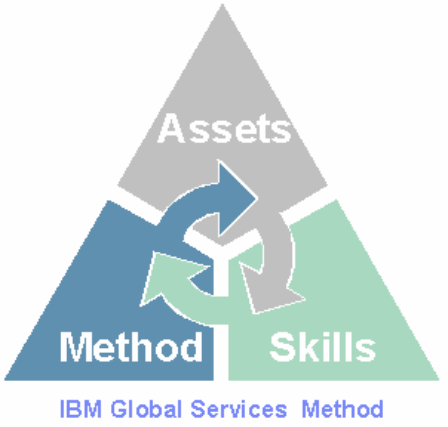


Abbildung 2.4: Der Kern der GSM [Ska03]

Die Methode wird auf den Projektbedarf zugeschnitten. Dieser Vorgang nennt sich „Tailoring“. In einem gemeinsamen Workshop wird zu Beginn des Projektes festgelegt, welche Teilergebnisse erstellt werden müssen. Danach folgen die Projektplanung und die Zuordnung von Aufgaben, sowie die Ressourcenplanung. Basierend auf einem gemeinsamen Framework, welches die Teilaufgaben beinhaltet, ist eine große Anzahl von unterschiedlichen Projekten realisierbar.

Durch die Definition von Engagement-Modellen, d.h. exemplarischen Projektplänen und Strukturierungen von Aufgaben in vielen Bereichen, werden die Projektleiter beim Aufsetzen eines Projektes unterstützt. Für jede Teilaufgabe ist genau beschrieben, wie, mit welchen Tools, mit welchem Fachwissen (eng. skill) und mit wie viel Aufwand diese zu erledigen ist. Mit den schon vorhandenen Kenntnissen und Assets lässt sich gerade der Aufwand relativ genau abschätzen.

In Abhängigkeit von dem gewählten Engagement-Model wird ein Projekt in Teilaufgaben mit zeitlicher Abhängigkeit dargestellt, wodurch die einzelnen Meilensteine sortiert und laut Zeitplan abgearbeitet werden, um das Projektziel zu erreichen. Diese Vorgehensweise ist auch unter dem Ausdruck „Work Breakdown Structure“ bekannt.

Durch die Strukturierung eines Projektes in Teilaufgaben wird ein Projekt handhabbar. Der Projektplan stellt dabei einen Prozess dar. In Abhängigkeit von der Prozesssteuerung werden die einzelnen Teilaufgaben kreiert. Der Prozess schreibt den zeitlichen und den inhaltlichen Rahmen für die Bearbeitung dieser Teilaufgaben vor.

Die Abhängigkeit der Teilaufgaben untereinander wird in Diagrammen dargestellt. Diese dienen zur Verdeutlichung der Aufgaben einzelner Mitarbeiter innerhalb des Projektes sowie die Auswirkungen von Verzögerungen und Versäumnissen.

2.3.4 ORACLE's Custom Development Method

Die Custom Development Method (CDM) [Pal07] von Oracle wurde ins Leben gerufen, um die Entwickler kundenspezifischer Software zu unterstützen. Die Methode kann von Kunden und Partnern von Oracle verwendet werden. Sie bietet einen umfangreichen Werkzeugsatz und gilt als erprobtes und durchstrukturiertes Verfahren zur Entwicklung von Softwarelösungen mit Hilfe der Oracle-Entwicklungstools. Mit der CDM ist es möglich, alle Phasen innerhalb eines Projektes abzudecken, von der Planung über die Durchführung bis zur Kontrolle. Die Handbücher und Tools für die Projektmanager und Projektmitglieder dienen der Unterstützung bei der Standardisierung und der Automatisierung eines Entwicklungsprozesses. Auch diese Methode basiert, wie die IBM GSM, auf einer langjährigen Erfahrung von Beratern, die weltweit für das Unternehmen tätig sind. Laut [Tho00] besteht die CDM aus:

- Online Guidelines und Handbüchern
- Workplan Templates
- Deliverable Templates

und wurde durch erfahrene Praktiker erstellt zum:

- Schätzen
- Managen
- Entwickeln
- Betreiben

von Unternehmensanwendungen. Die Abbildung 2.5 auf Seite 16 stellt die Methodenschritte von CDM und deren Einsatz in den Entwicklungsphasen eines Projektes dar.

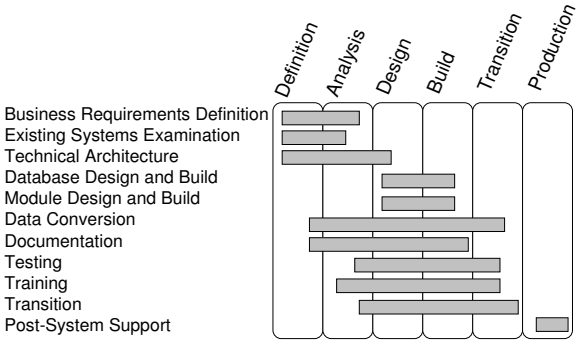


Abbildung 2.5: Ein klassisches Beispiel zur CDM [Tho00]

2.3.5 Rational Unified Process

Der Rational Unified Process (RUP) [Kru04] ist ein Software Engineering Prozess. Er stellt einen Leitfaden zur optimalen Verteilung von Aufgaben und Verantwortung innerhalb einer Entwicklungsorganisation zur Verfügung. Das Ziel ist auch bei diesem Verfahren, ein hochqualitatives Produkt innerhalb des vereinbarten Zeitraumes und Budgets dem Kunden zu liefern. RUP ist ein Produkt der Firma Rational Software, die seit 2002 Teil des IBM Konzerns ist.

Der RUP beschreibt die effektive Verwendung von Ansätzen auf dem Gebiet der Software-Entwicklung, die sich kommerziell bewiesen haben, innerhalb der Entwicklungsteams. Diese Ansätze werden aufgrund ihres Einsatzes durch erfolgreiche Unternehmen in der Industrie als „best practices“ bezeichnet. Der Prozess versorgt jedes Teammitglied mit Leitfäden, Vorlagen und Tool-Einleitungen, um unter anderem die Vorteile der folgenden „best practices“ zu nutzen:

- Iterative Software-Entwicklung

  Angesichts der Größe und des Anspruchs der heutigen Systeme ist es fast unmöglich, ein Produkt sequentiell zu entwickeln, d.h. die kompletten Anforderungen gleich zu Beginn zu erfassen, danach die vollständige Lösung zu implementieren, um diese im Anschluss ausführlichen Tests zu unterziehen. Sichere und wirkungsvollere Ergebnisse werden mit dem iterativen Vorgehen erzielt, welches durch das mehrfache Wiederkehren in die einzelnen Entwicklungsphasen zur Folge hat. Damit wird das Problem nach und nach verständlicher, die Zusammenhänge werden besser verstanden und durch die nachträgliche Verfeinerung der Lösung eine höhere Qualität erreicht. Beim RUP endet jede Iteration mit einer ausführbaren Version des Produktes, was dem zukünftigen Anwender die Möglichkeit gibt, noch während der Entwicklung sich die Zwischenergebnisse anzuschauen und diese zu testen und durch das Feedback die möglichen Abweichungen von den Anforderungen zu melden oder weitere, aus z.B. mittlerweile neu gewonnenen Erkenntnissen, Änderungswünsche zu äußern.

- Managen der Anforderungen

  In dem RUP ist festgehalten, wie die geforderten Funktionalitäten und Bedingungen festzulegen, zu organisieren und zu dokumentieren sind. Des Weiteren gibt es Richtlinien zum Verfolgen und Dokumentieren der Entscheidungen und zum Gewinnen und Austausch von Geschäftsanforderungen.

- Verwendung von auf Komponenten basierenden Architekturen

  Darunter verbirgt sich der Leitfaden zum Entwickeln der Software in Form von Komponenten mit klar definierten Funktionen. Der RUP bietet Lösungsansätze zur Definition von Architekturen, die die neuen aber auch die schon vorhandenen Komponenten verwenden.

- Visuelle Darstellung des Produktmodells

  Eine visuelle Abstraktion eines Problems ist mittlerweile essentiell für eine erfolgreiche Projektdurchführung. Auf diese Weise werden die Elemente und die Zusammenhänge des Systems anschaulich dargestellt. Durch entsprechende Toolunterstützung besteht die Möglichkeit, aus dem graphischen Modell den Quellcode zu generieren, ohne diesen manuell schreiben zu müssen. Der RUP bedient sich dabei der Unified Modeling Language (UML).

- Qualitätsprüfung des Produktes

  In diesem Abschnitt von RUP ist die Planung, Implementierung, Ausführung und Bewertung von Tests zu finden, die sich auf die Qualitätsmerkmale, wie Zuverlässigkeit, Funktionalität und die Effizienz der Anwendung und des Systems konzentrieren. Diese Tests sind in den Entwicklungsprozess integriert und werden nicht nachträglich separat durchgeführt.

- Überwachung der Änderungen

  In einem Umfeld, in dem Änderungen unvermeidlich sind, ist es wichtig diese im Griff zu behalten. Der RUP liefert Vorschläge zum Überwachen und Verfolgen der Änderungen. Er bietet außerdem einen Leitfaden zum Bilden von sicheren Arbeitsumgebungen für einzelne Mitarbeiter, um von Änderungen in anderen Bereichen isoliert zu bleiben.

Der komplette Prozess kann, wie in Abbildung 2.6 auf Seite 18 gezeigt, in zwei Dimensionen dargestellt werden. Auf der waagerechten Achse ist der zeitlichen Ablauf des Prozesses samt den einzelnen Phasen, Meilensteinen, Zyklen und Iterationen, die den so genannten dynamischen Part repräsentieren, zu sehen. Die senkrechte Achse repräsentiert die statische Sicht des Prozesses, in der die Tätigkeit, Artefakte, Mitarbeiter und die Workflows beinhaltet sind.

An dieser Stelle wird nicht detaillierter auf die einzelnen Elemente des Rational Unified Prozesses eingegangen. Zum Nachlesen empfiehlt sich unter anderem [Kru04] und [Rat98].

Im Folgenden wird noch ein wichtiges Architektur-Modell vorgestellt, welches ein Bestandteil von RUP ist und unter dem Namen „4+1 View Model“ bekannt ist. Wie der Name es schon sagt, konzentriert sich dieses Modell auf fünf Sichten einer Systemarchitektur:

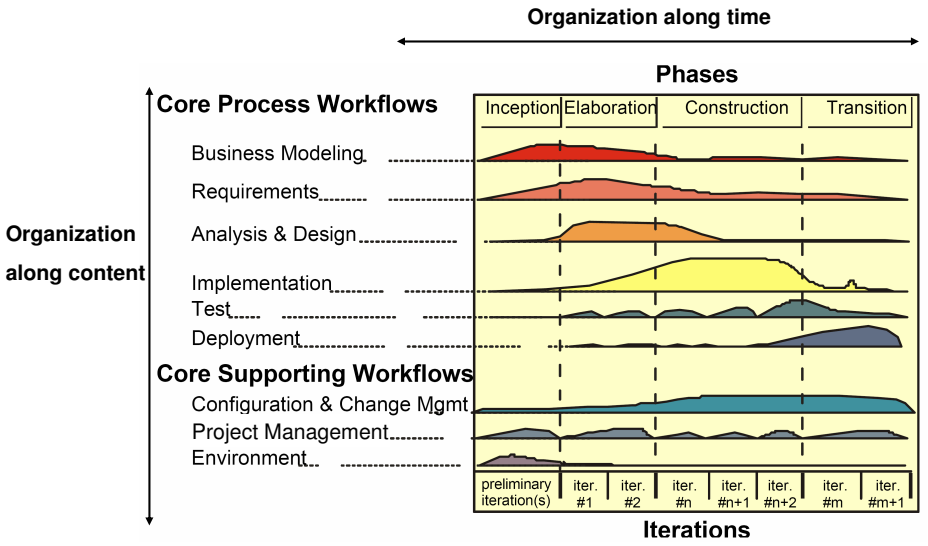


Abbildung 2.6: RUP Modellgraph [Rat98]

- Logische Sicht

  Die logische Sicht beschreibt in erster Linie die funktionalen Aspekte eines Systems, indem die Funktionalität eines Systems nach unterschiedlichen Schlüsselkriterien, die sich aus der Problemstellung ergeben, in Bereiche aufgeteilt wird und diese in Form von Objekten oder Objektklassen darstellt werden.

- Prozesssicht

  Die Prozesssicht beschreibt das dynamische Verhalten eines Systems mit dem Schwerpunkt auf die Kernprozesse. Unter den Kernprozessen sind diejenigen zu verstehen, die über synchrone und asynchrone Mechanismen kommunizieren, wie z.B. Aufruf einer entfernten Prozedur (engl. remote procedure call).

- Physische Sicht

  Die physische Sicht bildet die logische Sicht und die Prozesssicht auf die physische Infrastruktur des Systems ab. Da die physische Sicht eines großen Systems schnell unhandlich und unübersichtlich werden kann, wird hier zwischen verschiedenen Graden der Abstraktion einer Systemsicht unterschieden, von einer groben Übersicht, die nur die einzelnen Knoten des System anzeigt, bis zu einer detaillierten Sicht, in der die Anordnung der Prozesse auf den Knoten und die Kommunikationsmechanismen zwischen den Knoten dargestellt werden.

- Entwicklungssicht

  Der Fokus der Entwicklungssicht liegt bei der statischen Organisation der Systemsoftwarekomponenten innerhalb deren Einsatzbereiches. Zwecks besserer Übersicht und erhöhter Wiederverwendung wird die Software in Pakete unterteilt und unterschiedliche Subsysteme zusammengefasst.

Die Softwareentwickler können ihre Entscheidungen mit Hilfe dieser Sichten organisieren und diese mit ein paar ausgewählten Use Cases oder Szenarien graphisch darstellen, was unter der fünften Sicht des „4 + 1 View Modells“ zu verstehen ist.

Das „4+1 View Model“ gibt unterschiedlichen Interessenten die Möglichkeit, ihre eigenen Anforderungen zu verdeutlichen und diese zu optimieren. Es gibt unterschiedliche Herangehensweisen an das Modell. Die Systemingenieure können mit der physischen Sicht anfangen und mit der Prozess-Sicht fortsetzen. Die Endanwender, Kunden und Datenspezialisten würden wahrscheinlich zunächst die logische Sicht betrachten. Ein Projektmanager und ein Softwareentwickler würden das Modell in erster Linie aus der Entwicklungssicht betrachten.

Eine ausführliche Vorstellung des Modells ist unter anderem in [Kru95] zu finden.

2.3.6 Open Unified Process

Der Open Unified Process (OPENUP) [The07a] lehnt sich an den in dieser Arbeit schon vorgestellten Rational Unified Process an und ist ein Teil des Eclipse Process Frameworks (EPF) und wird von der Eclipse Foundation entwickelt. Da dieser Prozess die grundlegenden Eigenschaften des RUP besitzt, wie z.B. der „best practices“ Ansatz, iterative Software-Entwicklung, wird in dieser Arbeit nicht näher darauf eingegangen.

2.3.7 Object Engineering Process

Der Object Engineering Process (OEP) [OHJ+01] ist ein praxisorientierter und praxiserprobter Vorgehensleitfaden für die objektorientierte Software-Sntwicklung. Er ist seit ca. zehn Jahren ein im deutschsprachigen Raum verbreiteter Vorgehensleitfaden für agile[2] Softwareprojekte und die Entwicklung von Individualsoftware und lehnt sich strukturell an den Unified Process an. Entwickelt wird der OEP von der oose Innovative Informatik GmbH. Er ist ein moderner, iterativ-inkrementeler Entwicklungsprozess, dessen Fokus auf den dreischichtigen Architekturen liegt. Überwiegend wird OEP zur Entwicklung individueller, betrieblicher Informationssysteme verwendet und ist weniger für die Entwicklung von technischen Systemen und Standardsoftware geeignet. OEP ist kein universelles Vorgehensmodell, sondern ist auf eine bestimmte Architektur, Organisation und Betriebsumgebung zugeschnitten. Die Anpassung an andere Projekte geschieht nicht durch eine Konkretisierung abstrakter Beschreibungen, sondern durch das Entfernen der nicht benötigten und das Hinzufügen neuer relevanter Beschreibungen.

Wie in der Abbildung 2.7 auf Seite 20 dargestellt, weist OEP fünf Phasen auf, die samt ihrer Meilensteine den so genannten Makroprozess repräsentieren. Die Entwurfs- und die Konstruktionsphase werden iterativ abgewickelt und beschreiben einen so genannten Mikroprozess, welcher Analyse, Design, Implementierung und Test besteht.

[2] Agilität ist eine Arbeitseinstellung, ein Leitprinzip, das bei jeder Entscheidung eine Rolle spielt. Agile Prozesse schreiben nichts vor, sie postulieren nur „best practices“, aus denen die der Situation/Problemstellung angemessenen ausgewählt werden.

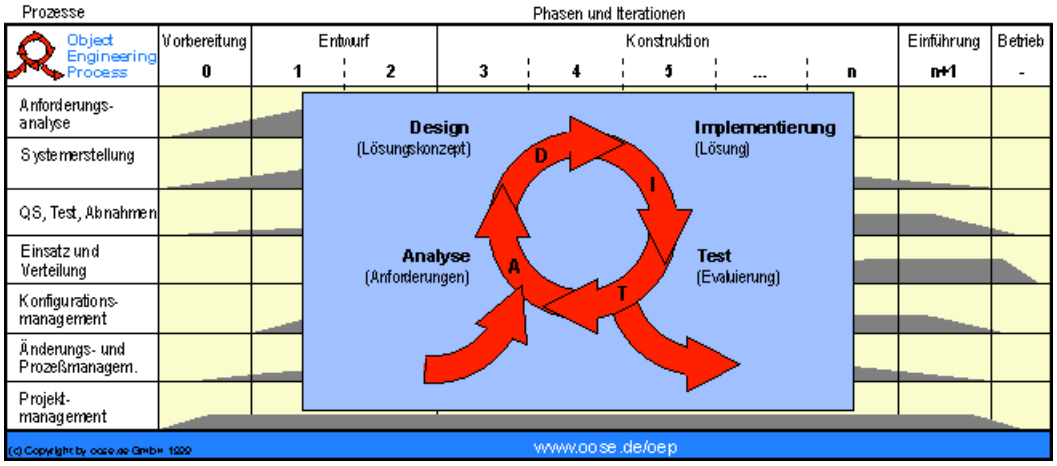


Abbildung 2.7: Phasen und Aktivitäten des OEP [oos07b]

Eine ausführliche Darstellung des Modells bieten [OHJ+01] und [OSKZ06].

2.3.8 IEEE 1471

IEEE 1471 (Recommended Practice For Architecture Description Of Software-Intensive Systems) [IEE00] ist der Entwurf einer zur Beschreibung von Software- und Systemarchitekturen empfohlenen Vorgehensweise. Besonders gut eignet sich diese in Verbindung mit den software-intensiven Systemen, d.h. Systemen, in denen die Software eine kritische Rolle in der Entwicklung, dem Einsatz und dem Heranwachsen eines Systems spielt, wie z.B. eingebettete Systeme und systems-of-systems. Der Standard wurde von der IEEE Architecture Working Group zwischen 1995 und 1998 entwickelt. Die wichtigsten Bestandteile des Standards sind:

- eine normative Sammlung von Definitionen zum Beschreiben von Architekturen, architektonischen Ansichten und architektonischen Veranschaulichungen
- ein konzeptueller Rahmen zum Einsatz dieser Definitionen in den unterschiedlichen Bereichen der Architekturbeschreibung für die Konstruktion, Analyse und die Entwicklung eines Systems
- ein Satz von Anforderungen, die an die Beschreibung einer Architektur gestellt werden

IEEE 1471 wird auf die Beschreibungen der Architekturen angewendet. Eine so genannte Architecture Description (AD) wird als IEEE 1471 konform bezeichnet, wenn diese die Anforderungen der Methode erfüllt. Als Beispiel erfüllt das im Unterkapitel 2.3.10 vorgestellter TOGAF laut [The07e] die Anforderungen des IEEE und war darüber hinaus ein Vorreiter beim Streben nach IEEE-Konformität. Die Anforderungen des IEEE sind unabhängig von jeglichen Architekturtechniken und können aus dem Grunde mit unterschiedlichen Methoden und Rahmenbedingungen zur Modellierung verwendet werden. Die Methode beschreibt und fordert keine Konformität von Systemen, Projekten, Prozessen, Methoden oder Tools. Dieses liegt in dem Zuständigkeitsbereich der jeweiligen Methoden und der diese praktizierenden Unternehmen [Hil00].

2.3.9 Reference Model For Open Distributed Processing

Wie der Name es schon verrät, ist das Reference Model For Open Distributed Processing (RM-ODP) [ISO95] ein Referenzmodell zum Entwickeln und Spezifizieren von verteilten Systemen. Seine Entwicklung begann 1988 und wird hauptsächlich von der International Organisation of Standardisation (ISO) und des International Telecommunications Union (ITU) vorangetrieben. Der Standard bezeichnet die wichtigsten Prioritäten für Spezifikationen von Architekturen und liefert ein minimales Set von Anforderungen und ein Objektmodell, um die Systemintegrität sicherzustellen. Es ist schwierig, ein komplettes, nicht triviales, verteiltes System mit der riesigen Menge an Information und Aspekten des Designs in einer einzigen Beschreibung zu erfassen. Außerdem ist die Entwicklung eines verteilten Systems eine komplexe Aufgabe. Der Entwicklungsprozess wird möglicherweise von unterschiedlichen Entwicklerteams realisiert, die zum Teil räumlich und zeitlich voneinander entfernt sind und unterschiedliche Aspekte des Systems bearbeiten. Dieses hat dazu geführt, dass unterschiedliche Sichten (engl. Views) auf das System entstanden sind, die nur einen bestimmten Teil der Gesamtanforderungen betrachten und deswegen eine geringere Komplexität aufweisen. Die einzelnen Sichten dürfen allerdings nicht völlig unabhängig voneinander betrachtet werden, um die Konsistenz zu bewahren, weil diese letztendlich das gesamte System darstellen. RM-ODP verwendet fünf Sichten [Kah02], die, wie die Abbildung 2.8 auf zeigt, den unterschiedlichen Phasen eines Entwicklungsprozesses zugeordnet werden können:

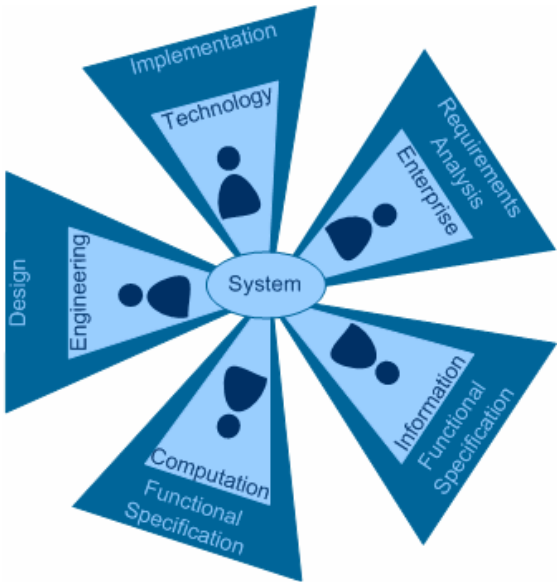


Abbildung 2.8: RM-ODP Viewpoints und Software-Engineering [CIS07]

- Unternehmenssicht

  Die Unternehmenssicht beschreibt die einzelnen Benutzergruppen des Systems, deren Anforderungen und Interaktionen sowohl untereinander als auch mit dem System. Sie beinhaltet Konzepte des Verwendungszwecks und des Systems, Strategien, die Richtwerte und Schranken ermitteln, und definiert die Systemanforderungen. Darüber hinaus werden die Geschäftsanforderungen und die Erfüllung dieser beschrieben.

- Informationssicht

  Der Fokus der Informationssicht liegt in der Semantik der Information und der Informationsverarbeitung. Damit werden die vom System verwaltete Information und die Struktur und der Typ der unterstützten Daten beschrieben.

- Funktionale Sicht

  Die Funktionale Sicht beinhaltet die funktionale Aspekte des Systems. Das komplette System wird in Komponenten und Verbindungen unter Berücksichtigung der Qualitätsanforderungen (QoS - Quality of Service) unterteilt. Es wird kein Bezug der funktionalen Komponenten zur Hardware des Systems hergestellt.

- Ingenieurssicht

  Die Ingenieurssicht betrachtet die Lokalisierung der physischen Bauteile des Systems und die Verteilung der funktionalen Komponenten auf diesen.

- Technologiesicht

  Die Technologiesicht beschäftigt sich mit der Auswahl der Technologie und Produkte, enthält die Spezifikation und die Konfiguration der Komponenten sowie die Konfiguration der physischen Knoten und der Verbindungen zwischen den einzelnen Komponenten. Des Weiteren sind unter diesem Gesichtpunkt Informationen zum Testen des Systems zu finden.

2.3.10 The Open Group Architecture Framework

The Open Group Architecture Framework (TOGAF) [The07d] ist, wie der Name es schon sagt, ein Produkt der Open Group und ist sowohl ein Standard zum Beschreiben als auch eine Methode zum Modellieren von Architekturen. Dieses beherbergt als Kern die Architecture Development Method (ADM).

Die ADM ist eine generische Methode zum Projektieren von Architekturen mit der Unterstützung der meisten System- und Organisationsanforderungen. Dennoch kommt es nicht selten vor, dass die ADM erweitert werden muss, um speziellen Anforderungen zu genügen. Vor der Verwendung der ADM ist es ratsam, die Methode hinsichtlich der Eignung zur Lösung des vorliegenden Problems zu prüfen und ggf. anzupassen, womit eine unternehmensspezifische ADM erzeugt wird.

Bei der Ausführung von ADM ist das Ziel eines IT-Architekten, nicht nur das Endergebnis in Form von einer organisationsspezifischen Architekturlösung zu erreichen, sondern ebenfalls das so genannte Kontinuum[3] des Unternehmens mit relevanten, wiederverwendbaren Bausteinen zu erweitern. Unter einem Kontinuum ist ein virtuelles Depot zu verstehen, welches alle sowohl im Unternehmen als auch in der Industrie vorhandenen Architektur-Asset-Modelle beherbergt, wie z.B. Muster, Dokumentationen von Architekturen und andere Artefakte. Diese Assets werden als vordefinierte Bausteine, die im Laufe mehrerer Projekte entstanden und optimiert worden sind, betrachtet und finden ihren Einsatz in den neueren Vorhaben.

Damit ist das Ziel eines Projektes neben der Problemlösung und Erfüllung der Anforderungen des Kunden, teilweise unter der Verwendung des Kontinuums, unter anderem das Unternehmen um weitere Assets reicher und damit in der Zukunft effektiver zu machen. Auch hier sind die ersten Schritte schwierig, weil der Inhalt des Asset-Depots zu Beginn nicht viel zu bieten hat, was wieder verwendet werden könnte, und erst nach und nach mit Erfahrung angereichert werden muss.

ADM ist eine iterative Methode über den kompletten Prozess sowohl zwischen den einzelnen Phasen als auch innerhalb der Phasen. Die Anzahl der Iterationen soll zu Beginn eines Projektes in Abhängigkeit vom Budget und Zeit festgelegt werden. Die Abbildung 2.9 auf Seite 25 zeigt sowohl die Basisstruktur der Methode als auch am Beispiel von „Technology Architecture“ die Unterteilung der Phasen des ADM-Zykluses in weitere Schritte. Im Allgemeinen weisen die einzelnen Phasen einen ähnlichen strukturellen Aufbau auf. Dieser drückt sich in Form von Eingaben (engl. input), Schritten (engl. step) und Ausgaben (engl. output) aus. Zur Darstellung von Modellen bedient sich die Methode der UML.

[3] Unter einem Kontinuum ist ein virtuelles Depot zu verstehen, welches alle sowohl im Unternehmen als auch in der Industrie vorhandenen Architektur-Assets - Modelle, Muster, Dokumentationen von Architekturen und andere Artefakte - beherbergt.

Im Folgenden werden die Phasen vorgestellt, die einen IT-Architekten durch die ADM führen sollen:

- Rahmen und Prinzipien

  In dieser Phase findet eine Identifizierung weiterer zu verwendender Modelle sowie eine Anpassung und Definition von Architekturprinzipien als Leitfaden bei der Entwicklung statt.

- Architekturvision

  In diesem Schritt werden der Umfang, der Fokus sowie die Geschäftsanforderungen und die Geschäftsbedingungen definiert. Außerdem werden die Genehmigungen für die Realisierung des Projektes beschafft.

- Geschäftsarchitektur

  Hier wird die Geschäftsarchitektur mit Geschäftsmodellen entwickelt. Zusätzlich findet eine Darstellung der Ist-Situation und des Ziel-Zustands sowie die Analyse der Abweichungen zwischen diesen statt.

- Informationssystemarchitektur

  In dieser Phase findet die Entwicklung einer Zielarchitektur inklusive der Anwendungen oder Datenmodelle, je nach Abhängigkeit von der Projektzuständigkeit, statt.

- Technologiearchitektur

  In diesem Schritt wird eine physische Architektur als Basis für nachfolgende Implementierungen errichtet.

- Möglichkeiten und Lösungen

  Es werden die bedeutenden Implementierungsprojekte identifiziert.

- Migrationsplanung

  Hier findet eine Schätzung der Abhängigkeiten, der Kosten sowie der Risiken und der Vorteile verschiedener Migrationsprojekte statt. Des Weiteren wird eine priorisierte Liste von Projekten erstellt, die eine Basis des detaillierten Implementierungs- und Migrationsplans darstellt.

- Implementierungssteuerung

  In diesem Abschnitt der ADM findet eine Überwachung der Kompatibilität mit der definierten Architektur und der Beziehungen der einzelnen Projekte statt.

- Management der Architekturänderung

  In dieser letzten Phase der ADM steht eine Planung zur Überwachung des Marktes hinsichtlich neuer Entwicklungen und Änderungen im Geschäftsumfeld im Vordergrund. Zusätzlich wird die Notwendigkeit eines neuen Entwicklungszykluses zur Einführung neuer Features bestimmt.

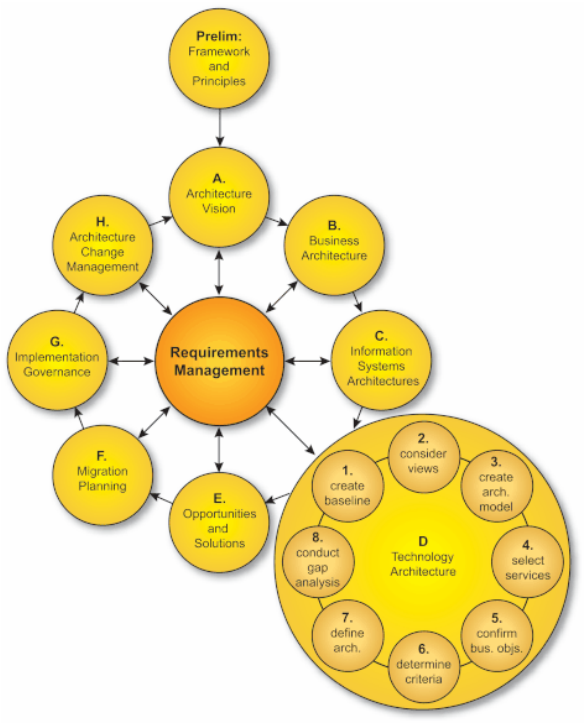


Abbildung 2.9: Architecture Development Zyklus [The07c]

2.3.11 UniCon

UniCon [CMU07b] ist eine an der Carnegie Mellon University entwickelte ADL mit dem Fokus auf die Komponenten und vor allem die Verbindungen zwischen diesen. Im Gegensatz zur traditionellen Entwicklung, die ein System als Menge interagierender Komponenten unter Vernachlässigung der Interaktionen betrachtet, liegt der Schwerpunkt dieser ADL bei den so genannten Konnektoren (engl. connector), die die Eigenschaft eines Systems wesentlich prägen.

Das Modell einer IT-Architektur, entstanden durch UniCon, besteht im Grunde aus den folgenden Elementen (siehe auch [CMU07c]):

- Komponente - Berechnungs- oder Datenspeicherungseinheit (siehe Abbildung 2.10)

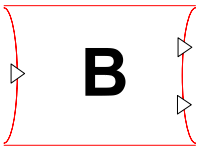


Abbildung 2.10: UniCon-Komponente

  Jede Komponente besitzt eine Schnittstelle (engl. interface), über die die Komponente verwendet werden kann und die die Komponente spezifiziert.

- Schnittstelle einer Komponente - Services (Nachrichten, Operationen, Variablen), die die Komponente bietet (siehe Abbildung 2.11)

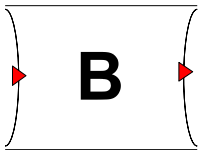


Abbildung 2.11: UniCon-Schnittstelle

Die Schnittstelle einer Komponente definiert den Typ der Komponente, die Eigenschaften dieser, wie z.B. Bedingungen, und die so genannten Players, die sichtbare, semantische Einheiten innerhalb einer Schnittstelle darstellen und über die Komponenten Services anbieten oder beziehen oder von den äußeren Zuständen und Ereignissen beeinflusst werden können. Offensichtlich sind die Players Elemente einer Schnittstelle, über die die Kommunikation zwischen den Komponenten eines Systems stattfindet. Eine Player-Definition enthält einen Typ und die folgenden Eigenschaften: Attribute Anforderungen und Bedingungen.

- Konnektor - Interaktion zwischen Komponenten (siehe Abbildung 2.12)

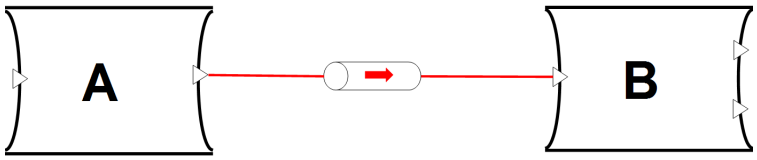


Abbildung 2.12: UniCon-Konnektor

Ein Konnektor ist das Element, welches die Regeln für die Kommunikation zwischen den Komponenten festsetzt. Er wird durch ein Protokoll spezifiziert. Durch ein Protokoll werden der Typ, die Eigenschaft und die Rolle des Konnektors definiert. Die Rolle beschreibt die Punkte, zwischen denen ein Konnektor vermittelt. Damit steht die Rolle unmittelbar mit den Players der Schnittschtelle in Kontakt. Mehr zu den Konnektoren ist im [SDZ95] zu finden.

2.3.12 Architecture Analysis & Design Language

Architecture Analysis & Design Language (AADL) [FGH06] ist eine von der Society of Automotive Engineers (SAE) entwickelte und im November 2004 veröffentlichte ADL. Sie findet ihren Einsatz in zahlreichen renommierten Unternehmen und Forschungszentren der Welt, European Space Agency (ESA), European Aeronautic Defense and Space (EADS), University of York, um nur einige zu nennen [CMU07a].

Die Sprache verwendet formale Modellierungskonzepte zur Beschreibung und Analyse von Systemarchitekturen in Form von verschiedenen Komponenten und den dazwischen liegenden Verbindungen. Sie beinhaltet die Abstraktion von Software, Hardware und Systemkomponenten zum Spezifizieren

von Realzeitsystemen, eingebetteten Systemen und Systemen mit einer hohen Zuverlässigkeitsanforderung. AADL ist sowohl eine textuelle als auch eine graphische Sprache. Die Abbildung 2.13 zeigt die unterschiedlichen Möglichkeiten der Verwendung von AADL.

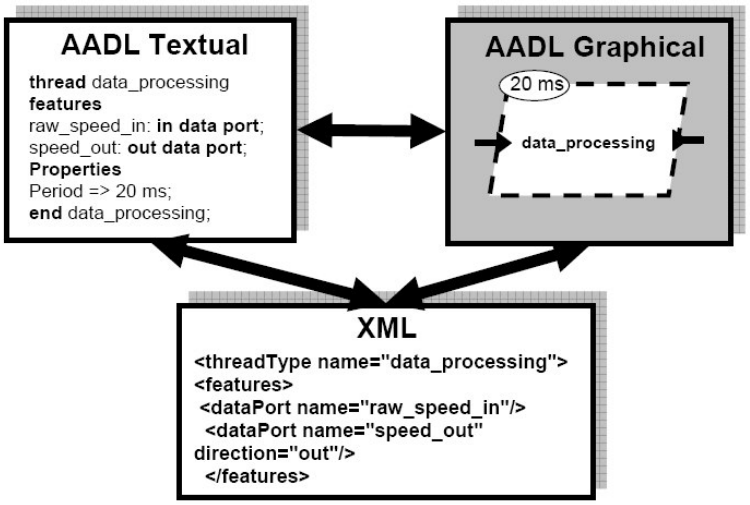


Abbildung 2.13: Die unterschiedlichen Darstellungen von AADL [FGH06]

Folgendes wird mit ADDL beschrieben:

- funktionale Schnittstellen der Komponenten, die in Form von Eingaben und Ausgaben ausgedrückt werden, und die leistungskritischen Aspekte der Komponenten, wie z.B. Zeitvorgaben
- Verteilung und Interaktion der Komponenten, Verbindung zwischen den Ein- und Ausgaben
- Dynamisches Verhalten der Laufzeitarchitektur

Die Sprache wurde entwickelt, um sich den Analysen der Laufzeitumgebungen anzupassen. Sie ist deswegen erweiterbar. Eine Erweiterung kann sich in Form einer neuen Eigenschaft und einer analysespezifischer Notation äußern, die mit einer Komponente in Verbindung gebracht werden kann. Mit Hilfe von Tools wird die automatische Generierung des Quellcodes und damit die Model Driven Architecture (MDA)[4] während des kompletten Lebenszyklusses eines Systems unterstützt.

2.4 Fazit

Verschiedene Schwerpunkte der Definitionen einer Systemarchitektur und damit eine nicht einheitliche Vorstellung davon führen dazu, dass viele unterschiedlichen Gruppen ihre eigenen Standards und Methoden entwickeln. Auf der Basis einiger vorhandenen Definitionen wurde eine für diese Arbeit zutreffende Definition hergeleitet. Es gibt keine einzige ADL, die alle möglichen Situationen abdeckt. Laut [Cle96] unterscheiden sich die ADLs:

[4]Model Driven Architecture bezeichnet einen Ansatz zur Entwicklung von IT-Lösungen, der auf einer klaren Trennung von Funktionalität und Technik beruht.

- im Umgang mit den Realzeitsystemen auf dem Architekturlevel. Schätzungsweise kann nur die Hälfte der Sprachen mit den harten Realzeitanforderungen, wie „deadlines“, umgehen und nur wenige mit den weichen Anforderungen, wie Prozessprioritäten.
- in der objektorientierten Vererbung. Die Beschreibung von hierarchischen Komponentenstrukturen stellt meistens kein Problem dar. Schwierigkeiten treten aber bei der objektorientierten Vererbung auf und bei den dynamischen Architekturen.
- in der Definition neuer Komponenten- und Beziehungstypen, der Definition neuer Erkenntnisse und in der Darstellung von Informationen, die nicht direkt die Architektur betreffen, wie z.B. Anforderungen und Testfälle.
- in der Konsistenz zwischen unterschiedlichen Artefakten, wie z.B. zwischen dem Architektur- und Komponentendesign
- in der Fähigkeit, zwischen den unterschiedlichen Architektursichten zu vermitteln, wenn diese Sichten gegeben sind

Viele ADLs unterscheiden sich in einigen Aspekten, es sind aber auch viele Gemeinsamkeiten vorhanden, die sich über einen großen Teil der ADL-Landschaft hinweg ziehen. Einen gemeinsamen Kern weisen auch die unterschiedlichen Methoden auf. Es gibt ADLs, die zum Teil dominanter als die anderen erscheinen, wie z.B. UML. Aber auch das variiert in Abhängigkeit von dem Einsatzgebiet. Für die Methoden scheint ein wichtiges Kriterium das Unternehmen zu sein, in dem die Methode teilweise entwickelt bzw. auf welches eine Methode angepasst worden ist, wie z.B. die IBM GSM oder die Oracle's CDM.

Zur Erstellung des IT-Modells in dieser Arbeit wurde der Ansatz des im Unterkapitel 2.2 vorgestellten Reverse Engineering verwendet. Der grundlegende Unterschied zu der traditionellsten Vorgehensweise, die unter der Bezeichnung Forward Engineering bekannt ist, liegt darin, dass beim Forward Engineering zunächst Anforderungen erhoben und Konzepte erstellt werden und das Produkt danach implementiert wird. Beim Reverse Engineering spielt sich die Vorgehensweise rückwärts ab. Es wird von einem Endprodukt ausgegangen, welches in seine Bestandteile zerlegt wird. Daraus lassen sich Strukturen ableiten. Da beide Vorgehensweisen bestimmter Muster bzw. Leitfäden, d.h. Methoden, bedürfen und die Struktur des, sei es zu implementierenden oder zu analysierenden, Systems beschrieben und dargestellt werden muss, sind die vorgestellten Standards und Methoden für beide Vorgehensweisen einsetzbar.

In dieser Arbeit werden bei der Modellierung und der folgenden Implementierung einige wichtige Aspekte der GSM aufgegriffen, auch wenn die GSM nicht durchgängig als Leitfaden dient:

- Verwendung von ADS als Beschreibungsprache
- Verwendung vorhandener Assets in Form von den schon vorhandenen IT-Architekturdatenbanken (engl. architecture repository)
- Verwendung vorhandener Erfahrung durch Kollegen

Der ADS, als IT-Architekturbeschreibungssprache, wurde für die Dokumentation einer IT-Architektur in Kapitel 3 aufgrund der Möglichkeit der Darstellung der operationalen Sicht eines Systems, einer Toolunterstützung dafür und des Einsatzbereichs ausgewählt. Auch der ADS ist kein Werkzeug, welches alles kann, aber auch keins, welches weit über die Fähigkeiten einiger anderer Standards hinausgeht. So sind Parallelen zwischen diesen festzustellen. Ein System aus unterschiedlichen Perspektiven

in Form von „viewpoints“ zu betrachten, scheint z.B. eine sehr verbreitete Vorgehensweise zu sein. Der „computational viewpoint“ von RM-ODP ähnelt z.B. sehr stark der funktionalen Sicht von ADS. Der „engineering viewpoint“ weist Ansätze der operationalen Sicht auf und der „technology viewpoint“ ist äquivalent dazu. Die logische Sicht des RUP enthält die statischen Ansätze des ADS. Die dynamischen Aspekte sind allerdings in der Prozesssicht enthalten. Wobei eigentlich der statische Ansatz im vorliegenden Fall auch schon ausreicht. Die physische- und die Entwicklungssicht sind ebenfalls in dem ADS wieder zu finden, auch wenn mit kleinen Einschränkungen. Die Verwandtschaft zu UML, zumindest in dem funktionalen Bereich, wurde schon im Unterkapitel 2.3.2 angesprochen. Auch in MODAF sind Analogien zu ADS und den in Kapitel 3 verfolgten Absichten zu finden, wie z.B. die Verwendung von UML bei der Modellierung und von XML Metadata Interchange (XMI) für den Austausch von Daten. XMI wird im Unterkapitel 3.4.2 zum Datenaustausch zwischen dem Modell und der graphischen Darstellung verwendet.

Es wäre abwegig, an dieser Stelle zu sagen, dass das IT-Modell nur mit dem ADS realisierbar ist. Da der ADS allerdings die Anforderungen im vorliegenden Fall erfüllt, zu den Modellierungsstandards von IBM gehört und sich eine Toolunterstützung für die beabsichtigte graphische Darstellung des operationalen Modells innerhalb des Unternehmens findet, fiel die Wahl auf den ADS.

3 Dokumentation einer IT-Infrastrukturarchitektur

Die IT-Industrie schlägt eine Vielzahl von Definitionen für die Architektur vor. Diese erwähnen an vielen Stellen die, in dieser Arbeit verfolgten, Absichten. Allerdings vereint keine dieser Definitionen die kompletten Anforderungen in sich. In Anlehnung an einige Definitionen wird im Unterkapitel 2.1 die Definition für die IT-Architektur, zutreffend für diese Arbeit, folgendermaßen hergeleitet:

> „Die Architektur eines IT-Systems ist eine auf Komponenten und deren Beziehungen basierende Darstellung der Infrastrukturanwendungen, die auf die, durch eine Ist-Aufnahme der Hardware und der Netzwerke entstandene, Topologie eines Systems abgebildet werden."

Viele Unternehmen befinden sich in einem Architekturdilemma. Aufgrund der wachsenden Globalisierung, steigenden Kostendrucks und starker Konsolidierung heiß umkämpfter Märkte sollte die Effizienz und der Nutzen der IT nicht außer Acht gelassen werden. Die Konzentration darf nicht nur auf dem reinen Selbstzweck der IT liegen. In [KS03] wird die Einstellung zur Unternehmensarchitektur folgendermaßen ausgedrückt: *„Die IT-Architektur eines Unternehmens ist nach unserer Erfahrung oft der neuralgischste Punkt und gleichzeitig das am häufigsten unterschätzte Investitionsgut."*

Eine sauber dokumentierte IT-Infrastrukturarchitektur könnte bei der Lösung von, den schon im Unterkapitel 1.1 angesprochenen, Problemen helfen.

In diesem Kapitel wird der Reverse Engineering-Ansatz angewendet, um die IT-Infrastrukturarchitektur eines Finanzdienstleisters unter Verwendung einer ADL und einiger Entwicklungswerkzeuge zu dokumentieren. Es werden einige Fragen geklärt, die im Zusammenhang mit der Modellierung gestellt werden sollten, und der Lösungsansatz sowie die Umsetzung diesen in einer IT-Infrastrukturdatenbank vorgestellt.

3.1 Anforderungen

Es sollten Überlegungen darüber angestellt werden, was dargestellt werden soll, welche Möglichkeiten vorhanden sind, wie z.B. Toolunterstützung, welche Alternativen, für die Lösung der vorliegenden Anforderungen, darüber hinaus möglich sind und welchen Nutzen ein bestimmter Lösungsansatz hat.

Benötigt wird eine Erfassung der sich im Einsatz befindenden Hardware und der vorhandenen Netzwerke, Darstellung von Infrastrukturanwendungen auf Komponentenebene und die Verteilung sowohl der Hardware auf die Netzwerke als auch der Anwendungen auf die Hardware. Die Vision ist, ein graphisches Modell in Form von Beziehungsdiagrammen der funktionalen und der operationalen Sichten.

Es gibt viele unterschiedliche Möglichkeiten bzw. Werkzeuge, mit denen ein System manuell modelliert werden kann. Zur Darstellung einer einmaligen Übersicht ist manuelles Zeichnen noch eine vertretbare Variante. Im vorliegenden Fall sollen, nach Bedarf, unterschiedliche Anwendungen in jeweils einem Komponentendiagramm und die Verteilung der Komponenten im System in einem Diagramm der operationalen Sicht dargestellt werden können. Da die Logik der Verknüpfungen immer gleich bleibt, bietet sich eine Automatisierung der Vorgehensweise an.

Eine IT-Infrastrukturdatenbank, die die Struktur der Anwendungen und anderer Systembestandteile beherbergt, stellt eine Lösungsmöglichkeit dar. Die in der Infrastrukturdatenbank enthaltene Struktur liefert die Basis für die graphische Darstellung.

Ein weiterer Vorteil der Infrastrukturdatenbank, im Vergleich zu den unterschiedlichen Modellierungstools, wie z.B. Rational Software Modeler (siehe auch Unterkapitel 3.4.1.4), ist der integrierte Einsatz. Somit wird es möglich, die für die Modellierung benötigte Information über die direkten Schnittstellen zu den bestehenden Asset- und Managementdatenbanken zu bekommen, um diese entsprechend für die benötigten Verknüpfungen innerhalb des Systems zu verwenden.

3.2 Mehrwert und Nutzen

In dem vorliegenden Fall soll die Infrastrukturdatenbank ein Werkzeug für den IT-Architekten darstellen. Die für ihn relevanten Elemente eines Systems (siehe Abbildung 3.8 auf Seite 42) werden strukturiert, inklusive aller Verbindungen zwischen diesen, in einer Datenbank abgelegt. Diese Information soll folgendermaßen verwendet werden:

- eine abstrakte Darstellung von Anwendungen auf der Komponentenebene
- eine einheitliche Dokumentation über die vorhandenen Ressourcen
- Möglichkeit des Abgleichs dessen, welche Tools sich auf welchen Servern befinden müssen und welche Tools sich tatsächlich dort befinden (Zusammenhang zwischen dem Komponentenmodell und dem operationalen Modell)
- Export der Information aus der Datenbank in einer strukturierten Form zwecks einer visualisierten Darstellung des Modells oder eines Ausschnitts daraus
- kompakte und abstrakte Systemdarstellung als Kommunikationsbasis vor allem mit dem Management, am besten graphisch, zwecks:
  - Fehleranalyse/Qualitätssicherung
  - Aquiseunterstützung
  - Schulung

3.3 Konzept

Auf der Basis des im Unterkapitel 2.3.2 vorgestellten, ADS wird in diesem Unterkapitel ein Modell für die IT-Infrastrukturdatenbank entworfen. Die Wahl des ADS als Modellierungssprache wurde unter Berücksichtigung der folgenden Faktoren getroffen:

- Möglichkeit der Darstellung der operationalen Sicht auf das System

- Toolunterstützung bei der Visualisierung des operationalen Modells
- typische ADLs und Methoden des Unternehmens IBM

Das Modell, welches die, im Unterkapitel 3.1 erhobenen, Anforderungen erfüllt, enthält nur einen Ausschnitt aus dem in der Abbildung 2.2 auf Seite 12 dargestellten ADS-Metamodell.

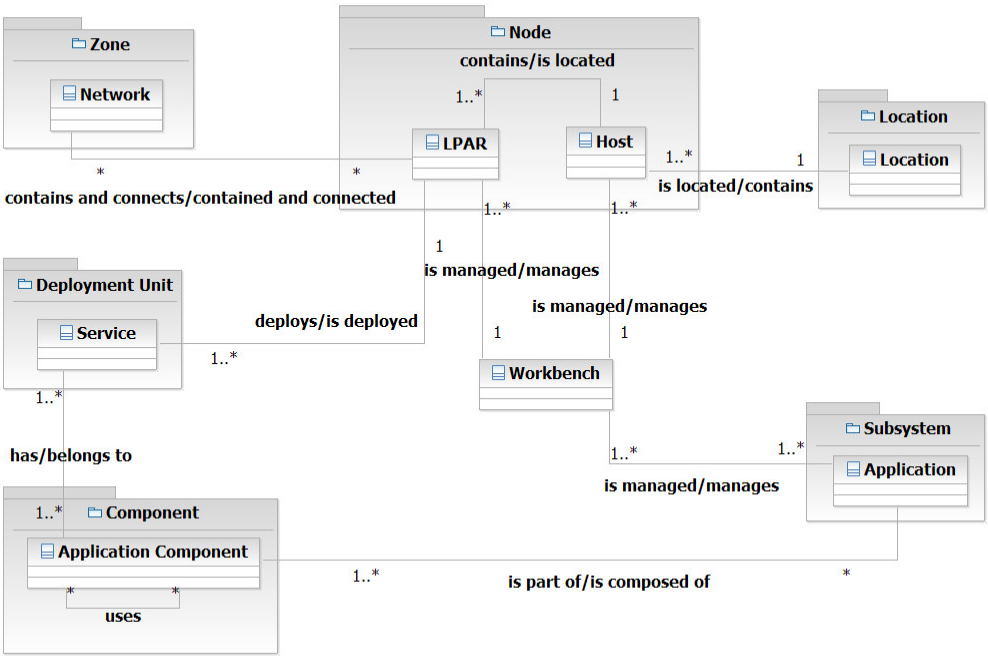


Abbildung 3.1: Das Modell der IT-Infrastrukturdatenbank

Ein wichtiges Merkmal des in dieser Arbeit erstellten Modells, welches in der Abbildung 3.1 zu sehen ist, und der implementierten Infrastrukturdatenbank, die im Unterkapitel 3.4.2 vorgestellt wird, ist die Darstellung der Anwendungen (im Modell als Application dargestellt) in Form von Komponenten (im Modell als Application Component dargestellt). Eine Komponente wird im ADS als eine modulare Einheit definiert, die eine bestimmte Funktion ausführt. Sie kann ein Teil beliebig vieler Applikationen sein, wie z.B. der Browser, der bei vielen Anwendungen die graphische Darstellung übernimmt. Die Komponenten sind untereinander verbunden, d.h. verwenden sich gegenseitig, und bilden auf diese Weise die kompletten Anwendungen.

Die Anwendungen dienen als Ausgangspunkt für das Reverse Engineering in dem vorliegenden Fall. Die restlichen Elemente ergeben sich indirekt daraus. Da die Anwendungen die Komponenten als deren Bestandteile gruppieren, wird dafür aus dem ADS-Metamodell das Element Subsystem verwendet. Ein Subsystem bedeutet im Sinne von ADS eine Gruppierung von Komponenten. Es setzt nicht voraus, dass die in diesem enthaltenen Komponenten den selben Service anbieten. Die Gruppierung kann aus folgenden Gründen erfolgen:

- Verteilung der Arbeit auf unterschiedliche Entwicklungsteams zur unabhängigen Entwicklung

- Organisation großer Systeme oder Anwendungen in kleinere Abschnitte zwecks besseren Verständnisses und Handhabung
- Identifikation eines Produktes (wie die Zusammenfassung von Komponenten zu einer Anwendung)

Die Subsysteme können sich überlappen, z.B. können einzelne Komponenten, wie der Browser unterschiedlichen Anwendungen, verschiedenen Gruppen angehören.

Die Anwendungen, auch Applikationen genannt, sind unterschiedlichen Charakters. Das die IT-Abteilung betreuende Projekt ist in mehrere Verantwortungsbereiche aufgeteilt. Jeder Bereich übernimmt entsprechende Aufgaben und verwendet bestimmte Applikationen zum Verrichten seiner Aufgaben. Solche Bereiche werden als Workbenches bezeichnet. Dazu wurde in dem ADS-Metamodell kein passendes Pendant gefunden. Somit wird an dieser Stelle vom ADS abgewichen. Die Workbenches existieren nicht nur in Verbindung mit Anwendungen, sondern ebenfalls im Zusammenhang mit der Hardware. So ist in dem entstandenen Modell die Workbench auch mit der logischen Partition (engl. logical partition, abgekürzt LPAR) und dem Host verbunden, die im weiteren Verlauf dieses Unterkapitels vorgestellt werden. In dem betreuenden Projekt gibt es Applikationen, die nur innerhalb eines Bereichs zum Einsatz kommen. Es gibt aber auch welche, die in mehreren Bereichen eine Rolle spielen, auch wenn diese nicht unbedingt demselben Zweck dienen. Es sind mit den einzelnen verantwortlichen Workbenches Interviews durchgeführt worden. Dabei ist unter anderem festgehalten worden, welche Applikationen im Einsatz sind, aus welchen Bausteinen diese sich zusammensetzen und wie diese geographisch verteilt sind. Anhand dieser Bausteine entstehen die Komponenten, die in ihrer Gesamtheit die wesentlichen Bestandteile und den Informations- und Datenfluss innerhalb einer Anwendung repräsentieren.

An dieser Stelle wird exemplarisch eine Anwendung namens Server Resource Management (SRM) vorgestellt, die in Form von Komponenten in einem Beziehungsdiagramm der funktionalen Sicht des ADS in der Abbildung 3.2 auf Seite 35 zu sehen ist. Die Anwendung bietet ein anschauliches Beispiel für die gerade beschriebenen Elemente Components und Applications des Modells und dient im weiteren Verlauf der Arbeit als Vorlage für unterschiedliche Beispiele.

Server Resource Management (SRM) ist im Wesentlichen ein Werkzeug zur Analyse von Effizienz (engl. performance) und Kapazität (engl. capacity) von IT-Systemen. Mit SRM können Analyseergebnisse von jedem UNIX und Windows Systemadministrator leicht eingesehen werden. SRM erfordert einen kleinen Agenten[1], der auf dem Server installiert wird und Berichte über die Daten aus der Vergangenheit und nahezu Realzeittendenzen der wichtigsten Server Ressourcen, wie z.B. CPU, Arbeitsspeicher und die Auslastung der Festplatten, erstattet. Diese patentierte Lösung übermittelt die Server-Daten an eine zentrale Stelle, von der aus die Berichte sowohl über das Intranet als auch Internet einsehbar sind (eine Anmeldung wird vorausgesetzt). Es dient folglich als ein zentrales Datenhandelszentrum (engl. data warehouse), welches sich auf die Effizienz- und Kapazitätsberichte der eingesetzten Server spezialisiert.

Die Komponenten sind dabei die wesentlichen Bestandteile von SRM innerhalb des Informationsflusses vom Server des Kunden, auf dem der SRM-Agent als Service ausgeführt wird, bis zur Darstellung des Berichts mit den gesammelten Daten zur Effizienz und Kapazität des Servers, was durch die Komponenten Browser Customer und Browser Employee dargestellt wird.

[1] Ein Agent ist ein automatisch in bestimmten, vorgegebenen Zeitabständen ausführbares Programm.

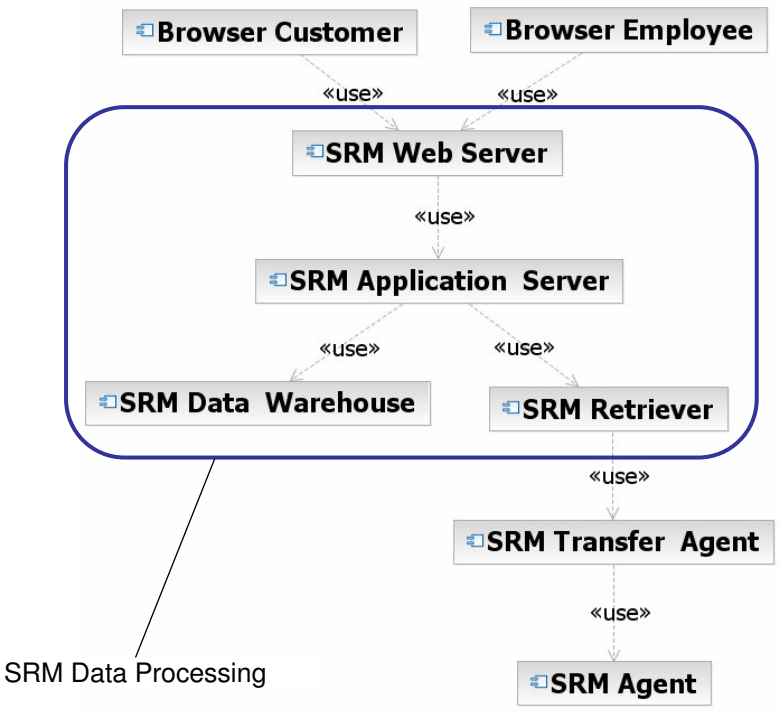


Abbildung 3.2: SRM Komponenten-Diagramm

Unmittelbar mit einer Komponente sind die Services verbunden. Im Sinne des ADS sind Services die so genannten Deployment Units (DUs) und werden als ein Mechanismus zur Unterstützung der Platzierung von Komponenten auf den Nodes verwendet. Eine DU kann nur einer Komponente zugewiesen werden, es können aber auch Komponenten in Abhängigkeit von den so genannten nicht funktionalen Anforderungen (engl. non-functional requirements)[2] in einer DU zusammengefasst werden. Eine DU kann sich nur auf einem Node befinden. Die Abbildung 3.2 zeigt einen Service namens SRM Data Processing, welcher in sich die vier Komponenten SRM Webserver, SRM Application Server, SRM Data Warehouse und SRM Retriever vereint.

Die Services werden also auf den LPARs ausgeführt. Folglich sind diese ebenfalls in dem Modell enthalten. Zur Darstellung von LPARs wird das Element Node aus dem ADS verwendet, welches eine abstrakte Hardwareplattform darstellt, auf der die DUs platziert werden können. Auf einer LPAR wird mindestens ein Service ausgeführt und ein Service verteilt sich nicht über mehrere LPARs.

Ein Node kann grundsätzlich aus mehreren anderen Nodes zusammengesetzt sein. Somit wird ein weiteres Element des Modells in der Abbildung 3.1 auf Seite 33 namens Host dem ADS-Node zugeordnet. Auf einem Host wird mindestens eine LPAR ausgeführt. Eine LPAR dagegen läuft auf genau einem Host.

Die geographische Verteilung einer Anwendung ist über die Components, die dazugehörigen Services, den LPARs als Plattform für die Services und den dazugehörenden Hosts, als Hardware, nachvollziehbar. Ein Host befindet sich nämlich an genau einem Standort, der so genannten Location. Zur Dar-

[2]Unter den nicht funktionalen Anforderungen sind z.B. folgende zu verstehen: Performance, Sicherheit und Fehlerhandhabung.

stellung einer Location dient die ADS-Location, die als geographischer Ort definiert wird. Angefangen von globalen Bereichen, wie z.B. Land und Stadt, werden die Locations auf bestimmte Plätze, wie z.B. Gebäude und Raum, herunter gebrochen.

Zur Darstellung der ebenfalls benötigten Netzwerke, die im Modell als Network bezeichnet werden und bestimmte IP-Bereiche darstellen, wird auf die Zonen des ADS zurückgegriffen. Laut ADS repräsentiert eine Zone in erster Linie einen Bereich im operationalen Modell, für den eine Reihe von nicht funktionalen Anforderungen definiert werden kann. Die Elemente einer Zone müssen zwar die Anforderungen dieser Zone erfüllen, können aber auch weitere nicht funktionale Anforderungen besitzen. Die einzelnen Elemente können also in mehreren Zonen mit unterschiedlichen Anforderungen enthalten sein. Laut ADS sind Zonen nicht unmittelbar mit den Nodes verbunden. In dem vorliegenden Modell wird an dieser Stelle vom ADS abgewichen und eine direkte Verbindung zu dem Host hergestellt. Eine LPAR kann sich in einem bis beliebig vielen Netzwerken befinden.

3.4 Implementierung

In diesem Unterkapitel wird nach der Einführung einiger verwendeter Tools und einer Programmiersprache eine IT-Infrastrukturdatenbank vorgestellt.

3.4.1 Entwicklungstools und Programmiersprachen

In diesem Unterkapitel werden die wesentlichen im Laufe der Arbeit verwendeten Tools und Programmiersprachen kurz vorgestellt. IBM Lotus Notes/Domino kommt als ein Datenbankmanagementsystem (DBMS) mit dem integrierbaren IBM Lotus Domino Designer, in dem LotusScript als Programmiersprache verwendet wird, zum Einsatz. Für die graphische Darstellung von Modellen wird der Rational Software Modeler (RSM) eingesetzt.

3.4.1.1 IBM Lotus Notes/Domino - Eine Groupware-Plattform

Lotus Notes/Domino war die erste kommerzielle Groupware-Plattform, die die nötige Marktreife erreicht hat. Aufgrund ihrer Komplexität und Vielfältigkeit fiel eine eindeutige Einordnung in eine Softwarekategorie anfangs sehr schwer und ist bis heute noch in der Fachliteratur umstritten. Lotus Notes/Domino vereinigt in sich viele Elemente aus dem Computer Supported Cooperative Work (CSCW)-Forschungsgebiet. Es ist ein auf E-Mail und Datenbankanwendungen basierendes Bulletin Board-System, das den kontrollierten Informationsaustausch zwischen Nutzern erlaubt.

Entwickelt in den frühen 80er Jahren als ein rechnergestütztes Unterrichtssystem, mit dem Professoren und Studenten weltweit kommunizieren können, verstand es der Student Raymond Ozzie den Grundgedanken dieser Idee aufzugreifen und in „Plato Notes“ umzusetzen. Zusammen mit Mitch Kapor (Gründer von Lotus Development Inc.), der ihn bei der Gründung seiner Firma Iris Associates Inc. unterstützte, wurde Notes weiterentwickelt. Das Ergebnis dieser Entwicklung war ein Datenbank-Betriebssystem für elektronische Mail, das im Stil des Client-Server-Modells eine Kommunikationsplattform aufbaut. Die exponentielle Entwicklung im IT-Bereich seit Beginn der 90er Jahre, hervorgerufen durch das World Wide Web, machte dem Produkt Notes seine Daseinsberechtigung streitig. Folgerichtig war ab der Version 4.6 die Annäherung von Notes an die Technologien des Internets. Um diesen

Schritt deutlich zu machen, wurde die Serverkomponente dieses Groupware-Systems in Domino umbenannt und der Client wurde nur noch Notes genannt. Zur Erhöhung seiner Popularität unterstützt Notes ab der Version 5 alle geläufigen Internet-Protokolle. Dem Entwickler wurde der im Unterkapitel 3.4.1.2 vorgestellte Domino Designer als professionelles Entwicklungswerkzeug für Notes Anwendungen zur Verfügung gestellt. In der Abbildung 3.3 ist der Notes Workspace der Version 7.0 dargestellt. Die aktuelle auf Eclipse basierte Verison ist 8.0 [IBM07b].

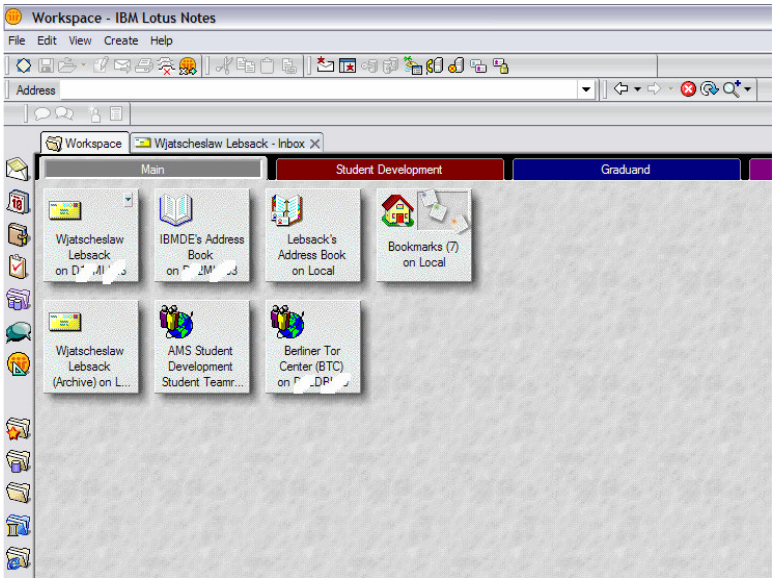


Abbildung 3.3: Lotus Notes Workspace

3.4.1.2 IBM Lotus Domino Designer

Mit Hilfe des in der Abbildung 3.4 auf Seite 38 dargestellten Domino Designer können neue Datenbanken erstellt sowie das Design vorhandener Datenbanken geändert werden, sofern es nicht geschützt ist. Der Domino Designer umfasst neben Funktionen zum Anlegen und Verändern der verschiedenen Gestaltungselemente wie Masken, Ansichten, Seiten, Rahmen usw. auch eine Entwicklungsumgebung, in der mittels Formelsprache (so genannter @Befehle und @Funktionen), LotusScript (mit Debugger), Java oder JavaScript programmiert werden kann.

3.4.1.3 LotusScript

LotusScript ist eine in Notes voll integrierte Programmiersprache. Es erlaubt den Programmablauf mittels Schleifenkonstrukten und Verzweigungsanweisungen zu steuern. Wie in C++ werden neben den objektorientierten Features auch rein prozedurale Sprachelemente zur Verfügung gestellt. So können eigene Funktionen und Subroutinen erstellt sowie eigene Klassen und Bibliotheken angelegt werden.

Was LotusScript auszeichnet, ist jedoch das bereitgestellte Klassenmodell. Es ermöglicht einen leichten Zugriff auf nahezu alle Objekte der Notes-Umgebung, auch auf solche, die nicht in der aktuellen Arbeitssitzung geladen sind (z.B. auf nicht geöffnete Datenbanken). Damit lassen sich Dokumente ohne Benutzeraktion bearbeiten oder neu erstellen.

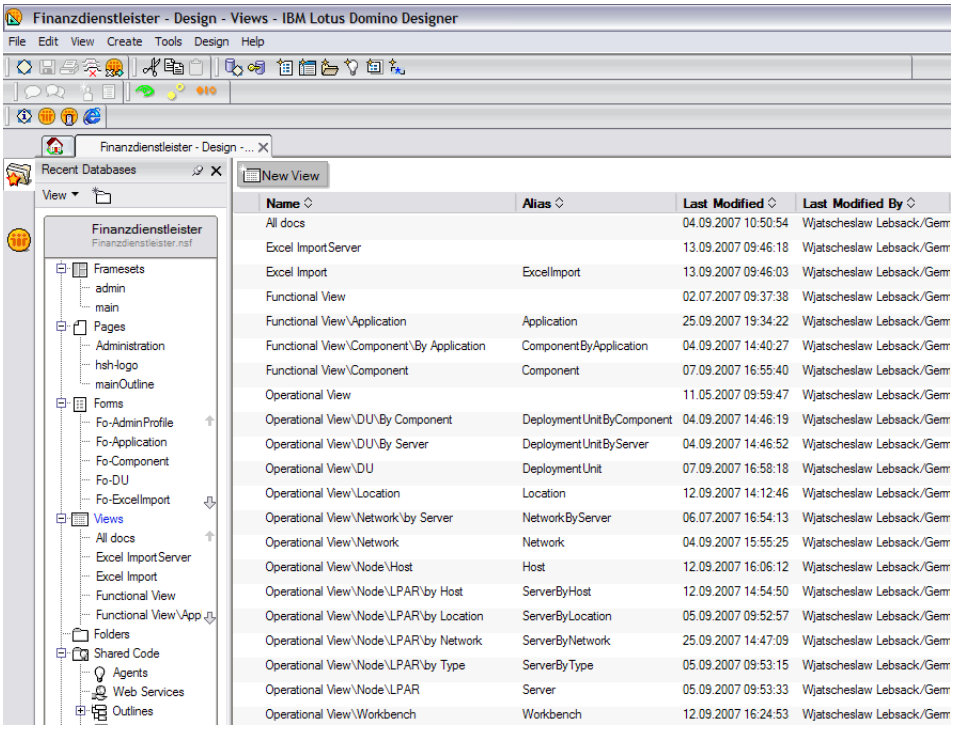


Abbildung 3.4: Lotus Notes Designer

Gilt es ein Script ausschließlich für die Notes-Client-Umgebung oder eine Applikation als Agent zu erstellen, so ist LotusScript gerade wegen seiner umfangreichen Klassenbibliothek grundsätzlich erste Wahl gegenüber den anderen integrierten Scriptsprachen. Insgesamt lässt sich sagen, dass LotusScript von der Notes-Entwicklungsumgebung voll unterstützt wird (dies trifft für Java oder JavaScript nicht zu). Umgekehrt ist das Klassenmodell von LotusScript genau an die Objektstruktur von Notes angepasst.

Eine Einschränkung für die Verwendung von LotusScript besteht jedoch: es wird nicht von Webbrowsern unterstützt. Für das Programmieren von Web-Ereignissen kann auf die seit der Version 5 integrierte Sprache JavaScript zurückgegriffen werden.

Die Sprache LotusScript selbst ist stark an Visual Basic angelehnt, was eine große Übersichtlichkeit und gute Strukturierbarkeit zur Folge hat. Ebenso entspricht die Bedeutung von Visual Basic für Microsoft Office der von LotusScript für Lotus Notes. Nach wie vor unerreicht bleibt jedoch das Klassenmodell von LotusScript für die Programmierung von Notes-Anwendungen. LotusScript ist insofern plattformunabhängig, als dass es für jedes Betriebssystem eingesetzt werden kann, welches von Lotus Notes/Domino unterstützt wird. Ein Beispiel des LotusScript-Codes ist im Anhang in der Abbildung B.1 auf Seite 75 dargestellt.

3.4.1.4 Rational Software Modeler

IBM Rational Software Modeler (RSM) ist ein UML-basiertes Werkzeug für visuelle Modellierung und Gestaltung zur Dokumentation und Übermittlung der verschiedenen Ansichten eines Systems. RSM ist eine Erweiterung der offenen Software-Entwicklungsumgebung Eclipse 3.2 und bietet eine umfassende Unterstützung für die Modellierung mit UML 2.1. Damit haben unterschiedliche Projektmitglieder,

wie z.B. IT-Architekten und Designer, die Möglichkeit, das System aus verschiedenen Perspektiven zu betrachten. Im Rahmen dieser Arbeit wird sowohl das in der Abbildung 3.1 auf Seite 33 gezeigte Gesamtmodell als auch die Komponentenmodelle, die, wie im Unterkapitel 3.4.2 beschrieben, automatisch generiert werden, und das operationale Modell visuell dargestellt. In der Abbildung 3.5 ist die Arbeitsfläche des RSM zu sehen.

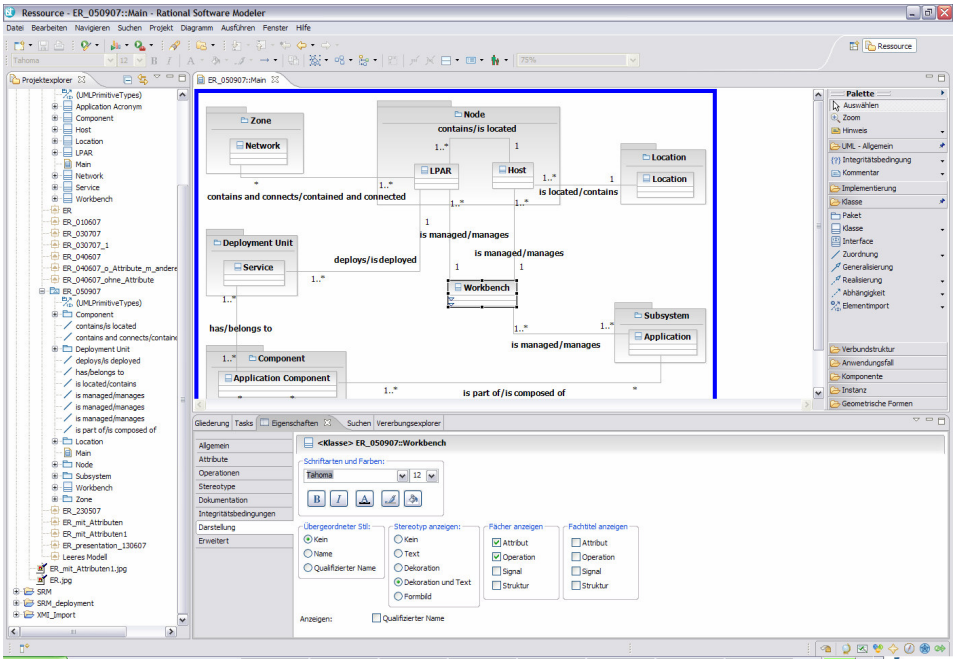


Abbildung 3.5: RSM Arbeitsumgebung

3.4.2 IT-Infrastrukturdatenbank

Auf der Basis des im Unterkapitel 3.3 vorgestellten Modells wurde eine IT-Infrastrukturdatenbank implementiert. Die Datenbank ist in dem für Lotus Notes typischen Stil entwickelt. Es existieren unterschiedliche Sichten, in denen die Unterschiedlichen Dokumente der jeweiligen Modellelemente angezeigt werden. Die Abbildung 3.6 auf Seite 40 stellt z.B. die Dokumente zu den Komponenten dar, die nach dem Anwendungsakronym sortiert sind. Auf diese Weise sind auch die restlichen in der Gliederung der Datenbank zu erkennenden Sichten, mit Ausnahme der Admin Area, aufgebaut. An der Aufteilung der Sichten in der Gliederung sind die funktionalen und die operationalen Aspekte des ADS und die Elemente des IT-Modells, inklusive deren Zugehörigkeit zu den beiden Aspekten, zu erkennen.

Die eigentlichen Dokumente sind im Stil des in der Abbildung 3.7 auf Seite 41 dargestellten LPAR-Dokumentes aufgebaut. Jedes Dokument enthält einen Satz von Attributen. In der Abbildung 3.8 auf Seite 42 sind alle Elemente des Modells mit deren Attributen dargestellt. Der Eintrag der jeweiligen Eigenschaften erfolgt zum Teil durch direktes Eintippen des Inhaltes aber auch durch ein Auswahlfenster. In dem Auswahlfenster werden, die in der Admin Area vordefinierten Auswahlmöglichkeiten

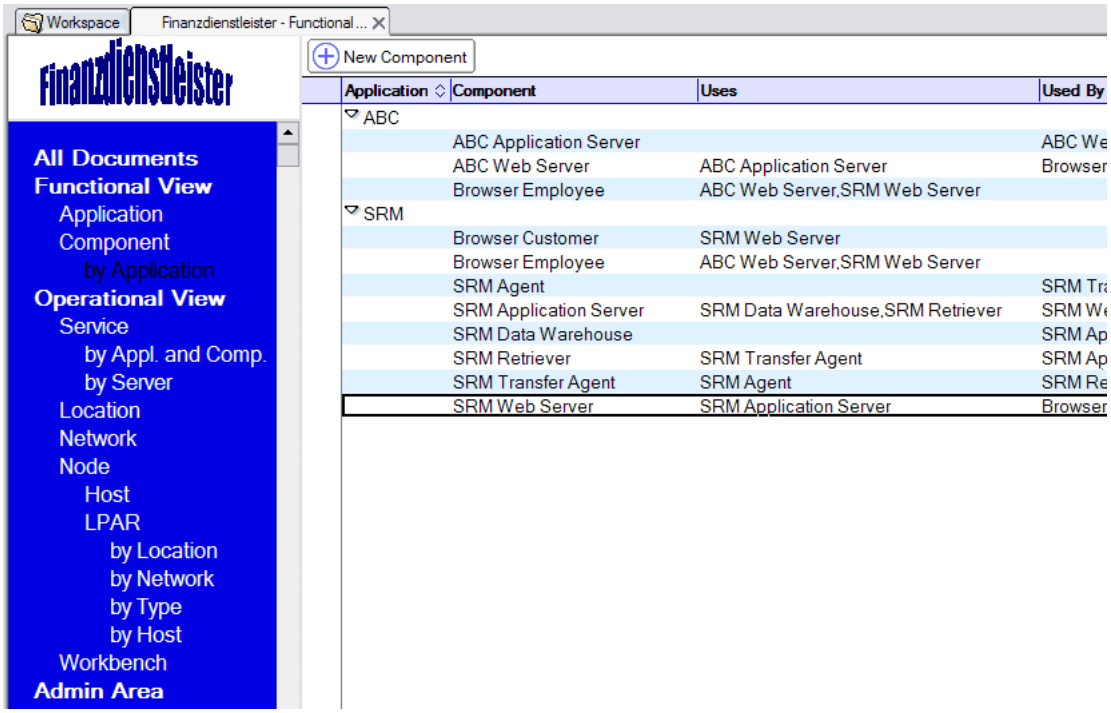


Abbildung 3.6: Übersicht zur IT-Infrastrukturdatenbank

angeboten. Über die „go to“- Schaltflächen wird zu den jeweiligen Dokumenten verlinkt. So wird z.B. aus dem LPAR-Dokument auf den Parent server, d.h. Host, oder eines der Netzwerke, in dem sich die LPAR befindet, zugegriffen. Oft enthält ein Dokument eine eingebettete Sicht, in der die mit dem entsprechenden Element in Beziehung stehenden Elemente angezeigt werden, wie z.B. bei der LPAR die darauf ausgeführten Services.

Die in der Abbildung 3.1 auf Seite 33 dargestellten Elemente lassen sich alle in der IT-Infrastrukturdatenbank in Form von Dokumenten ablegen. Es existieren ebenfalls alle Verknüpfungen zwischen den Elementen. Erwähnenswert ist die Möglichkeit der mehrfachen Verwendung von Elementen. Damit sind vor allem die Komponenten gemeint, die in unterschiedlichen Anwendungen zum Einsatz kommen, wie z.B. der Browser. Diese müssen nämlich nicht mehrmals angelegt werden. Über die mehrfache Zuordnung zu den unterschiedlichen Anwendungen, kann eine Komponente mehrfach verwendet werden. Dieses erleichtert die Pflege des Inhaltes. Änderungen werden in allen, diese Komponente verwendenden, Anwendungen gleichzeitig wirksam.

In der höchsten Ebene der Gliederung ist zusätzlich zu den beiden ADS-Aspekten eine bereits kurz erwähnte Admin Area zu sehen. Über diesen Bereich lassen sich, neben der Definition unterschiedlicher Parameter, die bei dem Ausfüllen der Felder innerhalb eines Dokumentes als Auswahl zur Verfügung stehen, Informationen importieren. Die Infrastrukturdatenbank lässt sich teilweise automatisch mit der Information aus den bestehenden Dokumenten, die im Moment noch eine Schnittstelle zwischen der IT-Infrastrukturdatenbank und den bestehenden Management-Datenbanken bilden, füllen. Dazu muss die für den Import vorgesehene Sicht ausgewählt werden, in der die zu importierenden Felder definiert sind. Unter Verwendung der unter Lotus Domino vorhandenen Funktion „Import“ werden die Daten importiert. Über das Bedienen der Schaltfläche zur Generierung des Inhalts, werden die entsprechenden Dokumente angelegt.

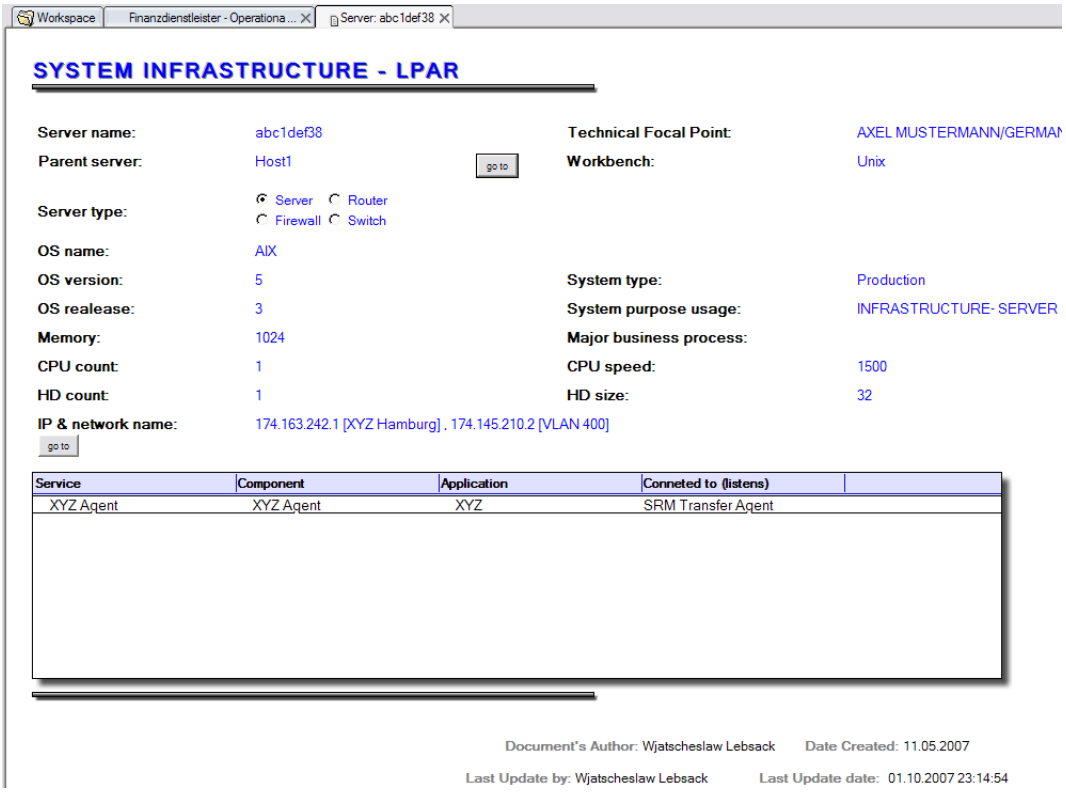


Abbildung 3.7: LPAR-Dokument aus der Infrastrukturdatenbank

Eine weitere in der Infrastrukturdatenbank enthaltene Funktionalität ist mit der Visualisierung der Systemausschnitte verbunden. Die Visualisierung ist zwar das Ziel, ist aber kein notwendiger Bestandteil dieser Diplomarbeit. Es erschien allerdings durchaus sinnvoll, diese zumindest teilweise zu implementieren, um einen großen Schritt in Richtung einer graphischen Darstellung des operationalen Modells zu veranschaulichen. Somit ist es möglich, das funktionale Modell aus der Datenbank zu exportieren, um ein Komponenten-Diagramm automatisch in dem RSM zu generieren. Aus der Sicht Application lassen sich die so genannten XML-Metadata-Interchange-(XMI)-Dateien generieren. Die XMI-Dateien bilden eine Schnittstelle zwischen der Infrastrukturdatenbank und dem RSM. Nach dem Auswählen von den zu exportierenden Applikationen und der Bedienung der Schaltfläche „XMI-File(s)“, deren Funktionalität mit LotusScript in den Abbildungen im Anhang ausgedrückt wird (die Abbildung B.1 auf Seite 75 leitet den gesamten Code ein), wird unter Verwendung der in der Datenbank abgelegten Beziehungen zwischen den Komponenten der jeweiligen Applikationen und unter Einhaltung der vom RSM geforderten Struktur eine XMI-Datei erzeugt. Diese wird im Anschluss im RSM importiert und das Modell unter Verwendung einer Funktion zur Visualisierung graphisch dargestellt. Der Inhalt der XMI-Datei zum im Unterkapitel 3.3 vorgestellten SRM ist in dem Code-Beispiel B.1 auf Seite 77 zu sehen.

Lotus Notes ist ein dokumentenorientiertes Datenbanksystem. Die existierenden Dokumente können vollkommen unabhängig voneinander verwaltet werden. Dokumente können gelöscht werden, ohne auf die bestehenden Beziehungen Rücksicht zu nehmen. Es fehlt die referenzielle Integrität, so wie diese aus den relationalen Datenbanken bekannt ist. Folglich sollte ein Entwickler dieses berücksichtigen und die entsprechenden Abhängigkeiten der Dokumente voneinander „manuell“ sicherstellen. Die Beziehungen zwischen den Elementen können über beliebige Attribute hergestellt werden. Dieses wird

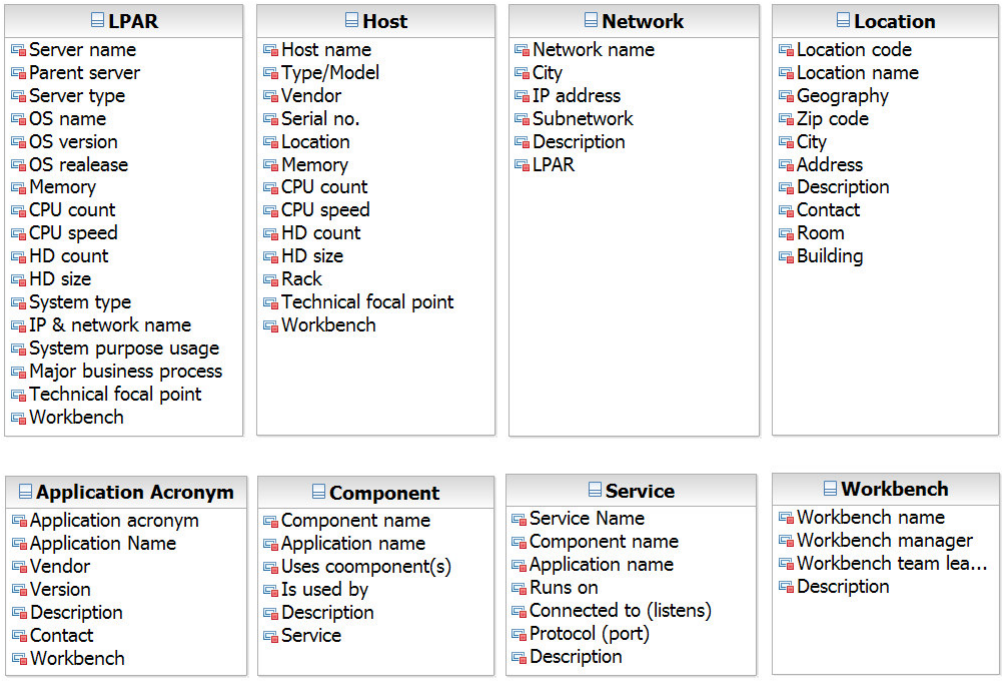


Abbildung 3.8: Elemente der Infrastrukturdatenbank mit Attributen

am Beispiel des Exports eines Komponentenmodells vorgestellt. Die Funktionsweise des Exports ist im Wesentlichen die folgende:
In der Datenbank geschieht die Verbindung zwischen den Komponenten über das Feld Uses component(s). Nach der Auswahl einer Anwendung, werden die dazu gehörenden Komponenten anhand des Feldes Application name gefunden. Danach wird Komponente für Komponente durchgegangen und das Feld Uses component(s) untersucht. Sind in dem Feld Einträge vorhanden, so werden diese in einer inneren Schleife abgearbeitet. Damit werden die Verbindungen zu den anderen Komponenten hergestellt. An dieser Stelle muss darauf geachtet werden, dass der Inhalt des Feldes Uses component(s) zu derselben Anwendung gehört. Dieses ist nämlich für die mehrfach verwendeten Komponenten wichtig, um nicht versehentlich eine Verbindung zu einer Komponente einer anderen Anwendung herzustellen. Ist diese Prüfung durchgeführt, können die Elemente und deren Verbindungen in der XML-Form abgelegt werden. Die IDs in der XMI-Datei werden aus den, innerhalb der kompletten Infrastrukturdatenbank eindeutigen, zu den Dokumenten gehörenden „NoteIDs“ generiert. Sonstige, obligatorische Bestandteile der XMI-Datei werden der Admin Area entnommen.

3.5 Fazit

Die Modellierung einer IT-Infrastruktur ist mit vielen Hürden verbunden. Die Modellierung einer bestehenden IT-Infrastruktur stellt auch keine Ausnahme dar. Die gesammelten Informationen aus den manuellen Assets- und Anwendungslisten sowie aus den Interviews müssen nachvollzogen werden, um daraus eine Struktur ableiten zu können. Die Struktur muss in eine allgemeinverständliche Form ge-

bracht werden. An dieser Stelle kommt im aktuellen Fall der ADS zum Einsatz. Die Verwendung einer Modellierungssprache bedarf einer gründlichen Auseinandersetzung damit. Die Anwendung der Architekturbeschreibungssprache auf die bestehenden Rahmenbedingungen stellt eine weitere Herausforderung dar. Bis ein Modell endgültig feststeht, wird es mehrmals verändert und mit der Zeit optimiert. Es wurde die Erfahrung gemacht, dass vor allem die letzten Details, in einem fast fertigen Modell, große Schwierigkeiten bereiten können.

Die Dokumentation der IT-Infrastruktur findet im aktuellen Fall in einer dokumentenbasierten Datenbank. Die fehlende referenzielle Integrität erfordert vom Entwickler ein hohes Maß an Kreativität und Aufmerksamkeit. Dennoch ist es möglich, komplizierte Verknüpfungen anzulegen und anspruchsvolle Suchanfragen zu definieren. Das komplette Modell wurde in der Lotus Notes Datenbank umgesetzt. Diese enthält die notwendige Struktur zur Generierung einer graphischen Darstellung sowohl eines funktionalen als auch eines operationalen Modells. Es ist möglich, eine Struktur für das Komponentendiagramm zur graphischen Darstellung der Infrastrukturanwendungen aus der Datenbank zu exportieren. Dieses stellt nicht unbedingt einen Bestandteil dieser Arbeit dar, ist allerdings ein großer Schritt zum eigentlichen Ziel, nämlich zur graphischen Darstellung der Modelle der operationalen Systemsicht. Der Export wurde zusätzlich implementiert, um zu verdeutlichen, dass die Idee einer Datenbank mit dem strukturierten Inhalt als Basis für die Visualisierung umsetzbar ist.

Die Schnittstelle zur Darstellung des operationalen Modells wird in Aussicht gestellt. Das operationale Modell kann noch nicht mit dem RSM dargestellt werden. Es befindet sich allerdings innerhalb der IBM ein anderes Tool in Entwicklung. Demnach sollte das operationale Modell demnächst entweder mit diesem Tool selbst oder mit den daraus entnommenen Funktionalitäten in dem RSM darstellbar sein. Sobald das Format für die Darstellung der operationalen Sicht bekannt ist, kann die IT-Infrastrukturdatenbank um die entsprechende Exportfunktion erweitert werden.

4 Conceptual Content Management Ansatz

Conceptual Content Management Systems (CCMS), zu deutsch Konzeptorientierte Inhaltsverwaltungssysteme, sind interaktive Systeme. Der Anwender hat die Möglichkeit, ein solches System nicht nur zu nutzen, sondern es gleichzeitig durch seine eigenen Beiträge zu erweitern. Der Grundgedanke des Conceptual Content Management (CCM) ist die Tatsache, dass Vieles auf der Welt nicht durch harte Fakten beschrieben und dargestellt, sondern in Abhängigkeit vom Betrachter unterschiedlich ausgelegt werden kann. Besonders stark ausgeprägt ist dieses Phänomen auf dem Gebiet der Geistes- und Kulturwissenschaften. Die Aussagekraft eines Bildes ist vielfältig. Die Politische Ikonographie[1] (PI) ist ausschlaggebend dafür gewesen, eine Konzeptorientierte Inhaltsverwaltung einzuführen und ein System dafür zu entwickeln.

4.1 Warburg Electronic Library

Die Warburg Electronic Library (WEL) ist ein solches CCMS und bietet eine Plattform zum Forschen und Nachschlagen in dem Material zur Politischen Ikonographie. Die WEL ist in einem Forschungsprojekt des Instituts für Softwaresysteme (STS) der Technischen Universität Hamburg-Harburg und der Forschungsstelle Politische Ikonographie des kunstgeschichtlichen Seminars der Universität Hamburg im Warburg-Haus entstanden [STS07b]. Dabei wurde der Bildindex der Politischen Ikonographie als multimediale digitale Bibliothek abgebildet.

Im Vordergrund der digitalen Bibliothek WEL steht, im Gegensatz zu Bilddatenbanken und multimedialen Archiven, nicht das Sammeln von Bildern in druckreifer Qualität, sondern einen Beitrag zur Erforschung und Vermittlung auf dem Gebiet der PI gesammelten Inhalte und Ideen zu leisten und unter Anderem einen Lernort darzustellen. Die Qualität der Bilder steht hier im Hintergrund. Vielmehr wird hier versucht, mit Hilfe entsprechender Bilder die unterschiedlichen Facetten des jeweiligen Konzepts der Politischen Ikonographie, das durch ein Schlagwort repräsentiert wird, begreifbar zu machen [STS07a]. Die Abbildung 4.1 auf Seite 46 stellt ein Bild dar, welches über die Schlagworte „Architektur“ und „Sonnenlandschaft“ zu finden ist, was die Möglichkeit unterschiedlicher Interpretationen des Inhalts verdeutlicht. Der Bildindex stellt in der WEL eine Sammlung von Verweisen auf Bilder dar. Die WEL enthält momentan im Wesentlichen Bilder und Textdokumente mit einigen Metainformationen. Nach und nach kommen andere mediale Dokumente dazu, wie z.B. Audio- und Videodokumente, um weitere Facetten der PI zu erfassen.

[1] Ikonographie ist ursprünglich ein Begriff der Altertumswissenschaften für die vergleichende Beschreibung von Bildnissen, meint heute in der Regel die Aufzeichnung von Themen- oder Motivtraditionen.

Abbildung 4.1: Ein Bild aus der WEL [STS07b]

4.2 Personalisierung und Publikation

Zur Unterstützung der subjektiven Wahrnehmung und Interpretation eines Inhaltes, die sich in den unterschiedlichen Verweisen zwischen den Assets niederschlagen, muss ein Inhaltsverwaltungssystem bestimmte Voraussetzungen erfüllen. In solch einem System muss jeder Anwender die Möglichkeit haben, seine Wahrnehmung auszudrücken. Er muss also einen separaten Zugriff auf die Daten besitzen, dessen Änderungen sich nicht auf den allgemein verwendeten Inhalt auswirken oder, besser gesagt, automatisch auswirken. Es handelt sich also an dieser Stelle um eine Art der Personalisierung. Diese darf aber nicht mit der Personalisierung heutiger Webapplikationen verwechselt werden, die im Wesentlichen die Designaspekte abdeckt.

Für jeden Anwender wird eine Kopie der Basisdaten angelegt, die er nach seinen Vorstellungen und zur Erfüllung seiner Aufgaben ändern kann. Es soll möglich sein, nicht nur inhaltliche Änderungen vorzunehmen, sondern auch strukturelle, z.B. durch das Anlegen oder Entfernen von Kategorien und eine Veränderung der Inhaltsstruktur durch Erstellen neuer Schemata des Inhalts. Durch das Löschen oder Einführen vor allem neuer struktureller Elemente, kann sich die personalisierte Variante des Systems sehr stark von dem ursprünglichen unterscheiden.

Des Weiteren wird gefordert, dass zwischen den Anwendern eine Kommunikation stattfinden kann. Die Information soll untereinander ausgetauscht werden können. Damit werden der Wissenserwerb und die Wissensvermittlung gefordert. Es sollen nicht nur die Erfahrungen der Anwender untereinander ausgetauscht werden können, sondern auch die neuen Erkenntnisse, die mit der Zeit gewonnen werden, in das System aufgenommen werden können, ohne dass am System umständlich programmiert werden muss, um dieses entsprechend anzupassen. Der unendliche Prozess des Reifens und des Wissenserwerbs soll durch das System mit so wenig wie möglich Aufwand unterstützt werden. Im Sinne der Konzeptorientierten Inhaltsverwaltung heißt das, dass jeder einzelne Anwender das System um neue Erkenntnisse durch die so genannte Veröffentlichung und die Integration seiner personalisierten Sicht in eine allgemeinere Sicht, bereichern kann, wie es in der Abbildung 4.2 auf Seite 47 dargestellt ist. Wie sich die Veröffentlichung letztenendes gestaltet und welche Schwierigkeiten sich dabei ergeben,

wird in [Seh04] erörtert. Das CCM soll also der Freiheit eines Anwenders beim Modellieren dienen. Es wird nach Offenheit und Dynamik verlangt, die im Folgenden vorgestellt und ebenfalls in [Seh04] näher erläutert werden. An dieser Stelle werden die Möglichkeiten der konventionellen Datenverwaltungssysteme mit ihrer starren Struktur mit den Grenzen ihrer Fähigkeit konfrontiert.

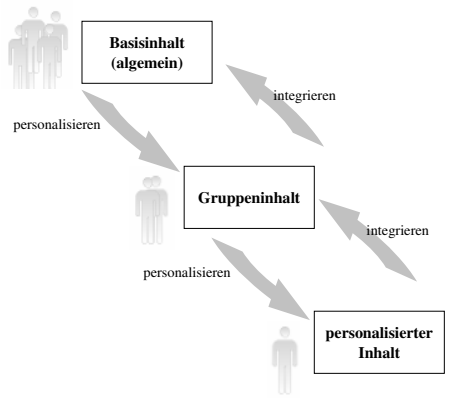


Abbildung 4.2: Personalisierung und Veröffentlichung des Inhalts

- Offenheit

  Die Offenheit eines Systems drückt sich in der Fähigkeit, durch die einzelnen Anwender des Systems veränderbar zu sein, d. h. dass eine Entität nicht fest definiert ist, sondern individuell beschrieben werden kann. Dieses wird durch die Assetdefinitionssprache unterstützt.

- Dynamik

  Die Architektur eines CCM und ein entsprechender Modellcompiler ermöglichen es, Änderungen am generierten System zur Laufzeit vorzunehmen und realisieren damit die Dynamik. Ist ein Modell oder ein Vorschlag, in einer personalisierten Sicht vorliegend, nach Meinung des jeweiligen Anwenders oder Entwicklers reif genug, um der Gemeinschaft präsentiert werden zu können, müssen die Systeme zur Laufzeit konfiguriert und kombiniert werden können.

4.3 Ein Beschreibungsmodell im Sinne der Konzeptorientierten Inhaltsverwaltung

In diesem Kapitel wird untersucht, ob die Konzeptorientierte Inhaltsverwaltung auf die Modellierung von IT-Systemen angewendet werden kann. Die in Kapitel 3 mit Hilfe der Konzepte des ADS beschriebene und mit deren graphischen Elementen visualisierte Darstellung einer IT-Infrastrukturarchitektur soll hier mir den Werkzeugen des CCM beschrieben werden. Des Weiteren soll das CCM den theoretischen Hintergrund des Standards enthalten, der weiterhin als Definitionsgrundlage für die Assets des CCM und deren Interaktion dienen soll. Als Datenverwaltungssystem wird an dieser Stelle, anders als in Kapitel 3, keine auf der Basis des ADS-Modells erstellte Datenbank verwendet, sondern ein Konzeptorientiertes Inhaltsverwaltungssystem.

Durch den Einsatz von CCM bei der Modellierung und Entwicklung von Soft- und Hardwaresystemen, könnten die Vorzüge des CCM, wie Offenheit und Dynamik, genutzt werden, ohne auf die zugrunde liegenden ADLs und die entsprechenden Methoden verzichten zu müssen, auch wenn deren Einsatz von dem derzeitigen zum Teil abweichen würde. Die ADLs würden nicht mehr von Anfang an in einer graphischen Darstellung münden. Hier wäre es eine Überlegung wert, aus einem CCMS die graphische Darstellung des Modells generieren zu lassen. Dieses müsste selbstverständlich dann auch in einer personalisierten Darstellung des Systems möglich sein. Somit hätte jeder Entwickler die Möglichkeit, das Basismodell, zwecks des Umgehens der während der Entwicklung festgestellten Schwierigkeiten in der Umsetzung, abzuändern und die Arbeit fortzusetzen. Die Vorgehensweise darf allerdings eine anschließende Präsentation der Änderung dem verantwortlichen IT-Architekten nicht ersetzen, denn sie würde sonst zu einer unkontrollierten und anarchistischen Entwicklung führen.

Die Methoden müssten teilweise an die Aspekte der Offenheit und der Dynamik und die sich damit geänderte Vorgehensweise bei der Projektdurchführung angepasst werden.

4.3.1 Domänen

Mit dieser Arbeit soll untersucht werden, ob das CCM zur Darstellung von IT-Architekturen geeignet ist. Deswegen wird hier als Grundlage ein mit dem Architecture Description Standard erstelltes Modell genommen, welches eine Infrastrukturarchitektur eines Finanzdienstleisters darstellt. Daraus soll exemplarisch ein Ausschnitt zur Darstellung mit den Werkzeugen des CCM verwendet werden.

Als erstes sollten die so genannten Domänen festgelegt werden, die in das CCMS aufgenommen werden sollen. Die Domänen sind Bereiche, die sich in ihren Eigenschaften grundlegend voneinander unterscheiden. In dem konkreten Modell dieser Arbeit werden die folgenden Domänen festgelegt (siehe auch die Abbildung 4.3 auf Seite 49):

- Anwendungsdomäne

  Eine Anwendungsdomäne gibt die Information über die genauen Elemente der IT-Infrastrukturarchitektur wieder, wie z.B. die Eigenschaften dieser und die Beziehungen der Elemente untereinander. Mit Ausnahme der untersten Ebene, der so genannten Blätter des Baumes, enthalten die Elemente dieser Domäne Beschreibungen, die gemeinsam eine Art textuelle Dokumentation der IT-Infrastrukturarchitektur ergeben.

- Domäne der Theorie

  In diesem Bereich wird der theoretische Hintergrund der Elemente der Anwendungsdomäne gepflegt. Damit ist die Theorie gemeint, die zum Modellieren verwendet wird, wie der ADS in diesem Beispiel. Hier wird allerdings nur eine grobe Beschreibung eines Elements geboten. So wird nicht gleich eine detaillierte und unter Umständen performanceintensive Darstellung aus der Datenbank geladen. Viele Fragen lassen sich oft durch eine kurze Erklärung beantworten. Die ausführliche Darbietung der ADS-Elemente obliegt der „Domäne der Theorie-Quelle“. Auf diese Weise lässt sich die weiter in der Arbeit vorgestellte Klassifikation in einem Objektsystem gut darstellen. Der objektorientierte und damit fest vorgegebene Inhalt einer Asset-Instanz lässt sich aus dieser Domäne definieren. So ist jede Instanz mit einem Minimum an Theorie versehen. Die detaillierte Variante würde an dieser Stelle beim Erzeugen jeder neuen Instanz in

der genannten Klassifikation unnötig Datenmengen erzeugen, was sich negativ auf die Performance des Systems auswirken würde.

- Domäne der Theorie-Quelle

  Die Theorie des ADS könnte wiederum in einer Form in dem CCMS abgelegt werden, die aus dem konventionellen CMS bekannt ist. Der Text wird in strukturierter Form, die in Sektionen unterteilt wird, dargestellt.

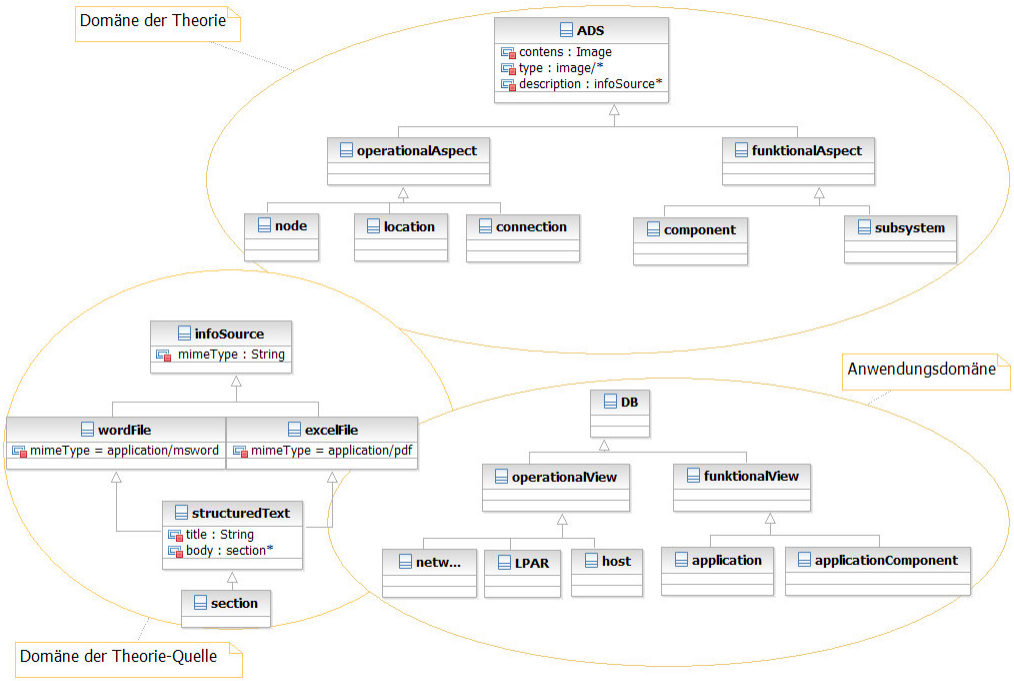


Abbildung 4.3: Beispiel für mögliche Anwendungsdomänen

Die Elemente der Domänen werden über Beziehungen miteinander verbunden. Ein Element der Infrastruktur ist nämlich ebenfalls ein Element im Sinne des ADS. Ein Element des ADS besitzt wiederum eine Quelle, die die benötigten Definitionen zur Verfügung stellt. Die Domänen unterliegen oft unterschiedlichen Zuständigkeitsbereichen, wie z.B. Entwicklungs- oder Modellierungsgruppen, die für die Beschreibung der Assets und deren Interaktion verantwortlich sind. Damit können die Assets verwendet bzw. wiederverwendet werden und brauchen nicht neu definiert zu werden, was stark zu einer höheren Effizienz beiträgt.

Die gerade mehrmals erwähnten Elemente unterschiedlicher Domänen werden im CCM in Form von Assets dargestellt, auf die im Folgenden näher eingegangen wird. Zum besseren Verständnis wird zunächst eine graphische Darstellung des Assets vorgestellt gefolgt von einer linguistischen. Die linguistische Darstellung baut auf einer Definitionssprache auf, die sich Assetdefinitionssprache nennt. Diese enthält noch weitere Konstrukte, die es einem Anwender ermöglichen, Assets nicht nur zu definieren, sondern auch zu ändern und zu löschen. Darüber hinaus bietet die Sprache eine semantische Unterstützung zum z.B. Definieren von Modellen, die im CCM die Domänen widerspiegeln und Durchsuchen des Inhalts von CCM.

4.3.2 Assets

Ein Asset beschreibt eine Entität und besteht aus zwei, voneinander nicht trennbaren, Teilen. Der erste Teil ist der so genannte Inhalt (engl. content) und der zweite das Konzept (engl. concept). Durch den Inhalt wird meistens ein medialer Teil des Assets repräsentiert. Dieses können unter anderem Bilder und Videos sein. Es ist allerdings ebenfalls möglich, z.B. einen String oder ein Bytearray in dem Inhaltstabschnitt darzustellen. In dieser Arbeit wird der Inhalt sich häufig aus dem String zusammensetzen, der bei den Asset-Instanzen den Namen der Instanz enthält. Der Konzept-Teil eines Assets enthält die charakteristischen Werte und Verweise auf andere Assets zur Herstellung von Beziehungen zwischen diesen. Die Abbildung 4.4 stellt ein Asset an einem Beispiel dar.

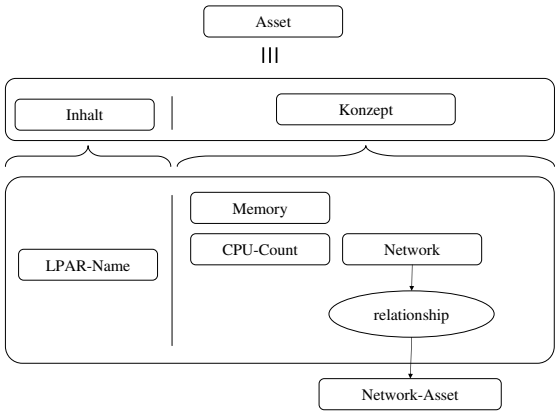


Abbildung 4.4: Darstellung von Assets

Wo die konventionellen Informationssysteme hinsichtlich der Offenheit und Dynamik an ihre Grenzen stoßen, weil sie üblicherweise auf einer Datenbank-Technologie basieren und ihre typischen Eigenschaften teilen, wie z.B. das statische Schema, kommt ein kozeptorientiertes Inhaltsverwaltungssystem als Alternative in Frage. Dieses muss demnach eine Besonderheit in seiner Architektur aufweisen.

Die Besonderheit drückt sich in der Assetdefinitionssprache aus. Auf diese wird in dem folgenden Abschnitt näher eingegangen.

4.3.3 Assetdefinitionssprache

Die Assetdefinitionssprache bildet die Grundlage von Konzeptorientierten Inhaltsverwaltungssystemen. Die graphische Darstellung eines Assets wurde im vorigen Abschnitt vorgestellt. Hier soll nun kurz auf eine linguistische Darstellung eingegangen werden. Zur Beschreibung von Entitäten in Abhändigkeit von einer persönlichen Sicht und Interpretation, wird eine Sprache verwendet, die auf Objekten basiert, so wie die objektorientierten Sprachen auf Variablen basieren. Die Assets basieren wie Objekte auf der Definition von Klassen, die deren Struktur beschreibt. Die Grunddarstellung eines Assets, die sich aus den schon bekannten Teilen zusammensetzt, wird mit dem Code-Beispiel 4.1 auf Seite 51 ausgedrückt.

```
class assetName {
        content ...
        concept ...
}
```

Listing 4.1: Asset Basisdarstellung

Zur Beschreibung unterschiedlicher Domänen lassen sich die einzelnen Assets zu den so genannten Modellen gruppieren. Da die Assets nur innerhalb dieser Modelle, die den Namensraum darstellen, angezeigt werden können, müssen diese importiert werden, um auch in anderen Modellen sichtbar zu sein. Das Code-Beispiel 4.2 zeigt, dass zur Angabe des theoretischen Hintergrunds zu einer LPAR, welche sich im, in dieser Arbeit behandelten, Modell in Form eines Node ausdrückt, das Einbinden des Modells ads von Nöten ist, welches das Asset node definiert.

```
model infrastructure
from ads import node
class LPAR refines operationalView {
        content contents: application/pdf
        concept characteristic lparName: String
                characteristic memory: int
                characteristic cpuCount: int
                characteristic serialNumber: String
                relationship parentServer: host
                relationship networks: network*
                relationship adsType: node
}
```

Listing 4.2: Modell und eine Asset-Klasse

Eine Entität lässt sich mit unterschiedlichen Mitteln beschreiben. Die immanenten Eigenschaften einer Entität werden in der Assetdefinitionsspache als Charakteristika bezeichnet und mit einem entsprechenden Schlüsselbegriff dargestellt. In dem Code-Beispiel 4.2 steht z.B. vor der Variable memory vom Typ int das Schlüsselwort characteristic. Dann gibt es die Beziehungen, die durch das Schlüsselwort relationship eingeleitet werden. Diese stellen, wie der Name es schon sagt, Beziehungen zwischen den Assets her. So besitzt eine Instanz der Klasse LPAR eine Verbindung zu einer Instanz der Klasse node. Durch das Anhängen eines Sterns „*“ an den Typ einer Beziehung wird ausgedrückt, dass das Attribut mehrere Assets referenziert, wie z.B. networks.

Im weiteren Verlauf dieser Arbeit werden weitere Fähigkeiten der Sprache in Verbindung mit der direkten Verwendung vorgestellt. Darüber hinaus ist eine detaillierte Darstellung der Sprache in [SSSW07] und [ScS04] zu finden.

4.3.4 CCMS Module

Ist ein Basismodell vorhanden, kann eine personalisierte Sicht daraus erzeugt werden. Dazu muss zunächst die Architektur des Systems etwas genauer angeschaut werden, um den technischen Zusammenhang zwischen den unterschiedlichen Sichten und Domänen zu verstehen.

Die Architektur besteht aus Komponenten. Diese werden in Module unterteilt, die die Basis einer domänenspezifischen Architektur für die dynamische Systemgenerierung darstellen und durch den Modell-Compiler erzeugt werden. Momentan wird zwischen den folgenden, in der Abbildung 4.5 dargestellten, fünf Modularten unterschieden (siehe [BSSS07]):

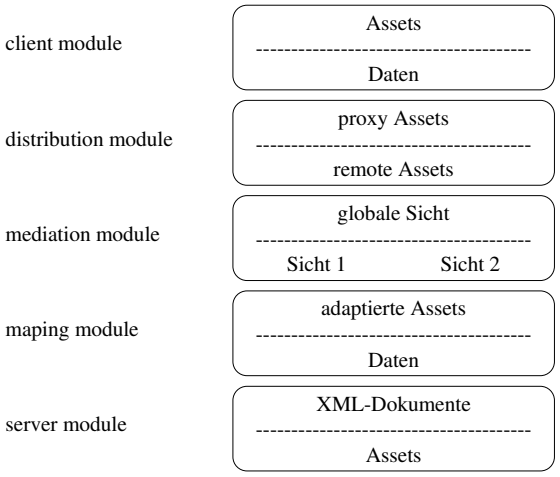


Abbildung 4.5: Module und Verbindungselemente

- Client module

  Damit wird ein Asset-Modell auf das Schema einer Datenbank abgebildet, in der die beschreibenden Daten eines Assets, wie Inhalt, Charakteristik und Beziehung, abgespeichert werden.

- Distribution module

  Durch den Einsatz einer Instanz dieser Modulart können die einzelnen Komponenten physikalisch unterschiedlich lokalisiert sein. Die Kommunikation zwischen diesen erfolgt dann über den Austausch von z.B. XML-Dokumenten.

- Mediation module

  Darüber erfolgt seitens des Anwenders der erste Kontakt mit den Komponenten. Die Anfragen werden an dieser Stelle so verteilt, dass zum Anlegen und Manipulieren von Objekten innerhalb der personalisierten Sicht sowohl auf den privaten als auch den öffentlichen Datenvorrat bzw. auf die Daten unterschiedlicher Domänen zugegriffen werden kann.

- Mapping module

  Deses Modul ist dafür zuständig, die Anfragen, die innerhalb einer personalisierten Sicht formuliert worden sind, in die Anfragen der Basisschichten umzuformulieren.

- Server module

  Über dieses Modul wird auf die Komponenten mit Hilfe der Standardprotokolle zugegriffen.

Die Zusammensetzung einiger Systemmodule wird in der Abbildung 4.6 präsentiert. Dabei werden die Anfragen geeignet auf die Systemmodule verteilt und die Ergebnisse wieder zusammengefügt. Über ein mediation module werden die Anfragen in Abhängigkeit von der Personalisierungsstrategie über die einzelnen Domänen, sowohl private als auch öffentliche, verteilt. Über ein client module wird der Inhalt aus einer Datenbank gelesen bzw. in diese geschrieben. Ein mapping module überführt die Struktur der Anfragen, die im Rahmen der personalisierten Sicht (AB)´ erstellt worden sind, in die der öffentlichen Schemata A und B.

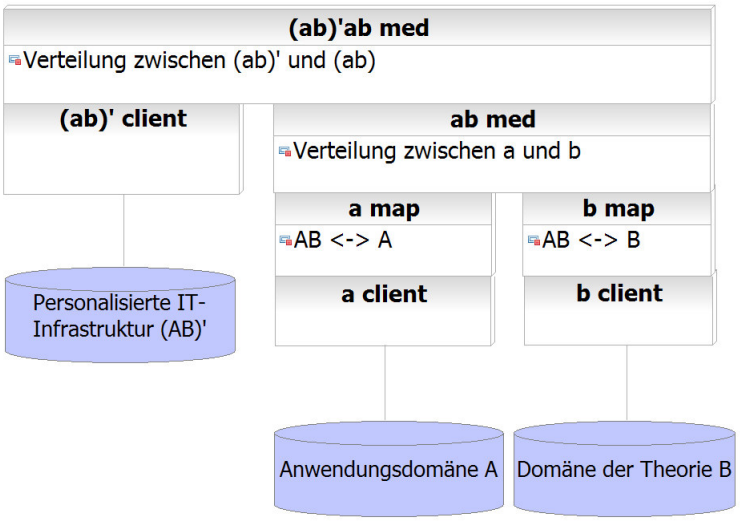


Abbildung 4.6: Interaktion der Systemmodule

4.3.5 Informationszugriff

Nach der Vorstellung einzelner Module eines CCMS wird im Folgenden der Aufbau eines Systems aus diesen Modulen und die Arbeit mit dem System mit Hilfe der Assetdefinitionssprache vorgestellt. Die Asset-Klasse lpar wurde in dem Code-Beispiel 4.2 auf Seite 51 eingeführt. Als Beispiel für das Anlegen einer Asset-Instanz wird im Code-Beispiel 4.3 auf Seite 54 eine Instanz namens musterLPAR vom Typ LPAR dargestellt.

```
let musterLPAR : Asset := create LPAR {
        lparName := "eineLPAR"
        memory := 1024
        cpuCount := 2
        serialNumber := "12B78D3"
        parentServer := hostName
        network := {networkName1, networkName2}
        adsType := adsNodeInstance
}
```

Listing 4.3: Beispiel einer Asset-Instanz

Oft wird gewünscht, ohne nach Elementen explizit suchen zu müssen, auf die in einem Asset erwähnten Referenzen zugreifen zu können, bzw. ist es für das vorliegende IT-Modell wichtig, dass diese Möglichkeit existiert. Die musterLPAR enthält z.B. Beziehungen zu dem parentServer namens hostName, zu den unterschiedlichen Netzwerken und sogar zu einer Instanz einer anderen Domäne. Die Beziehungspfade zwischen den referenzierten Assets sollten verfolgt werden können. Dieses geschieht, wie im Code-Beispiel 4.4 dargestellt mit Hilfe einer Suchanfrage, die mit dem Schlüsselwort lookfor eingeleitet wird. Gesucht wird nach allen Assets vom Typ network, die eine Beziehung zu der aktuellen LPAR haben, oder anders ausgedrückt, deren Beziehung namens LPAR mindestens die aktuelle LPAR referenziert. Das Ergebnis kann nun in Form von Links, die auf die gefundenen Assets verweisen, dargestellt werden und entsprechend gestaltet sich der Zugriff auf die einzelnen Netzwerke.

```
let networkOfLPAR : Asset* := lookfor network{
        LPAR >= {aktuelleLPAR.lparName}
}
```

Listing 4.4: Beipiel für die Suche nach Assets

Ist ein Element angelegt und besteht die Notwendigkeit , es zu verändern, kann dieses, wie im Code-Beispiel 4.5 gezeigt, mit dem Befehl modify getan werden.

```
modify musterLPAR {
        parentServer := einAndererNode
}
```

Listing 4.5: Verändern eines Assets

Die Personalisierung eines Assets unter Benutzung des Schlüsselwortes personalize ist im Code-Beispiel 4.6 auf Seite 55 gezeigt. Dort wird das Asset namens musterLPAR aus dem Bestand DB in den

Bestand des Anwenders kopiert, der das Ausführen des Befehls veranlasst hat. Das Veröffentlichen von Assets aus dem eigenen Bestand wird mit dem Schlüsselwort publish eingeleitet und ist ebenfalls in dem Code-Beispiel 4.6 dargestellt.

```
1 let myMusterLPAR : Asset := personalize musterLPAR from DB
2
3 publish myMusterLPAR to DB
```

Listing 4.6: Personalisieren und Publizieren von Assets

Die Assets werden in der Regel nicht gelöscht. Sie beschreiben Entitäten und solche hören, zumindest konzeptionell, nicht auf zu existieren. Selbst wenn eine Entität nicht mehr vorhanden ist, soll das Asset beschreiben, dass es diese mal gegeben hat und aus welchem Grunde sie nun nicht mehr existiert [Seh04]. Das Löschen wird dennoch ermöglicht, um z.B. die Assets aus einer personalisierten Sicht eines Anwenders entfernen zu können. Mit dem Schlüsselwort delete wird die Löschung vollzogen. Dabei besteht nicht nur die Möglichkeit, ein einziges Asset zu löschen, sondern auch Mengen davon, wie das Code-Beispiel 4.7 es zeigt. Durch eine Kombination mit dem Befehl lookfor lassen sich Assets auffinden und ausblenden.

```
1 delete musterLPAR
2
3 delete {musterLPAR_1, ..., musterLPAR_n}
```

Listing 4.7: Löschen von Assets

4.3.6 Klassifikationsarten

Nach der Einführung von Domains, der Assetdefinitionssprache und der unterschiedlichen Darstellungen von Assets, soll nun im Folgenden eine Klassifikation von Entitäten durch Assets eines Ausschnitts aus dem IT-Modell vorgestellt werden.

In Inhaltsverwaltungssystemen sind die folgenden drei Arten zur Beschreibung von Entitäten vorzufinden [Seh04]:

- basierend auf Klassen und Instanzen (bekannt aus der objektorientierten Programmierung)
- die Klassenzugehörigkeit wird basierend auf logischen Ausdrücken dynamisch bestimmt (bekannt aus beschreibungslogischen Systemen)
- eine Klassifikationsbeziehung wird auf Instanzebene dargestellt (Verwendung in Objektsystemen)

Die dritte Variante ist die, die in der WEL zum Einsatz kommt und für das IT-Modell dieser Arbeit verwendet werden könnte. Grundsätzlich gilt allerdings, dass in den Konzeptorientierten Inhaltsverwaltungssystemen alle drei Arten verwendet werden können. Die Art der Modellierung kann somit an die Art der Aufgabe angepasst werden. Darüber hinaus sind Kombinationen dieser möglich.

Die Objekte einer Klasse in der objektorientierten Modellierung werden beim Erzeugen fest einer Klasse zugewiesen. Eine dynamische Typänderung des Objektes ist nicht erlaubt, weil dann nicht mehr, gemäß der in einer Klasse definierten Eigenschaften, darauf zugegriffen werden kann. Für eine Klassifizierung über mehrere Domänen müssen die so genannten Potenzklassen gebildet werden. Die entsprechenden Aspekte der Domänen werden dadurch in einer Klasse vereint, die zur Erzeugung eines Objektes dient. In einem CCMS werden so Assets erstellt, die durch die Information mehrerer Domänen beschrieben werden. Wie in der Abbildung 4.7 zu sehen ist, ist zur Erzeugung der Asset-Instanz musterLPAR die Asset-Klasse LPAR als Potenz der Klassen node und operational view zu erstellen. musterLPAR enthält nicht nur die technische Information einer logischen Partition, sondern wird durch den ADS untermauert. Dadurch erfährt das Objekt seinen theoretischen Hintergrund und die Anordnung innerhalb des IT-Modells.

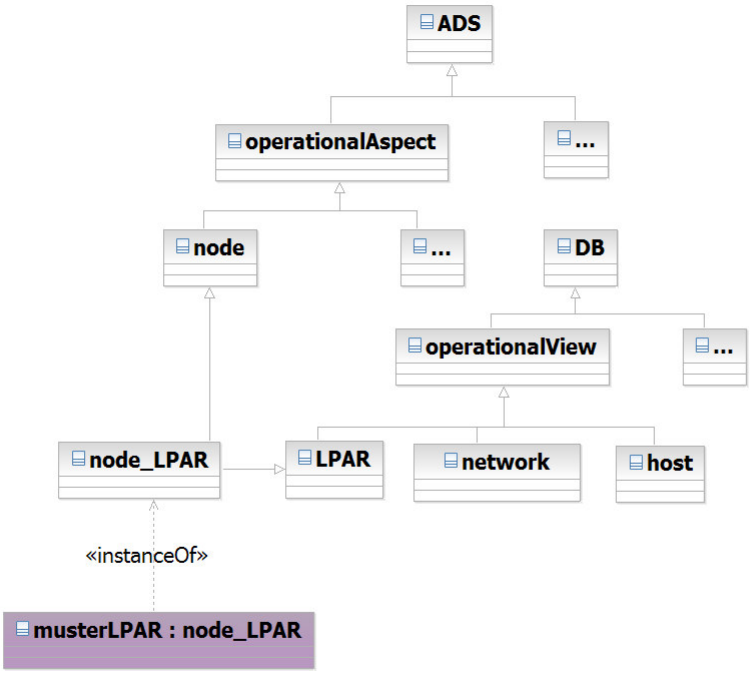


Abbildung 4.7: Beispiel für die Klassifikation in der objektorientierten Modellierung

In beschreibungslogischen Systemen wird eine Instanz nicht aus einer Klasse erzeugt. Es wird unabhängig von der Klassendefinition, allerdings mit Schlüsselelementen bzw. Attributen erzeugt, die eine dynamische Zuordnung zu den Klassen ermöglichen. Dementsprechend kann ein Objekt in Abhängigkeit von seinen Attributen erstens zu mehreren Klassen gehören und zweitens die Klassenzugehörigkeit ändern. Für die Klassifikation werden in den Asset-Klassen Werte für Attribute vorgegeben, was in der objektorientierten Modellierung nicht möglich ist. An dieser Stelle prüft ein System unter anderem die-

se Zugehörigkeit. Die Abbildung 4.8 stellt die Klassifikation in einem beschreibungslogischen System dar. Zum besseren Verständnis wird dort angenommen, dass für die unterschiedlichen Arten einer LPAR, wie Firewall, Switch und Router, auch unterschiedliche Asset-Klassen vorliegen. Tatsächlich sind es im Modell die Werte eines Attributs (siehe Abbildung 3.7 auf Seite 41).

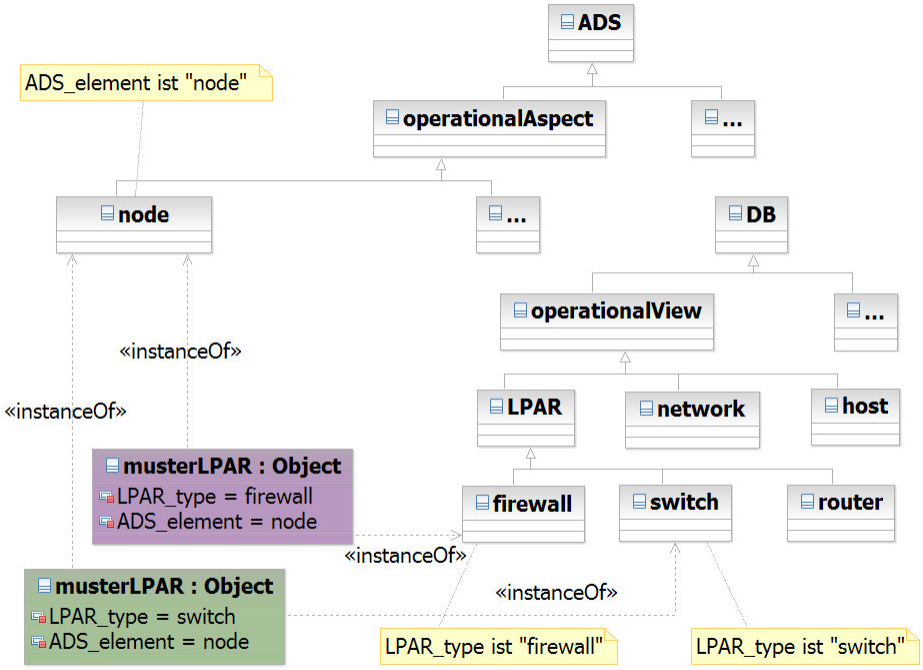


Abbildung 4.8: Beispiel einer inhaltsbasierten Klassifikation

Kann sich z.B. der Inhaltsteil eines Assets nicht ändern, wird die dritte Art der Klassifikation gewählt, die auf der Instanzebene durchgeführt wird. In der WEL wird das Medium, welches in der Regel ein Kunstwerk ist und seine Medienart nicht ändert, durch eine Asset-Klasse dargestellt. In dem vorliegenden IT-Modell bleibt z.B. der ADS-Typ einer LPAR immer gleich. Dieser heißt nämlich node. Die Beziehungen einer LPAR lassen sich unterschiedlich anlegen. Dadurch erfolgt die Klassifikation in einer anderen Domäne. Diese Konstellation ist in der Abbildung 4.9 dargestellt. Konkret erfolgt die Kategorisierung in der Domäne der Theorie objektorientiert durch das Anlegen von Subklassen. Die Instanzen dieser werden in Form von Assoziationen zu den Objekten der Anwendungsdomäne dynamisch klassifiziert. Der ADS-Zweig zum node mit den Sub- und Oberklassen sieht folgendermaßen aus: ADS -> operational aspect -> node.

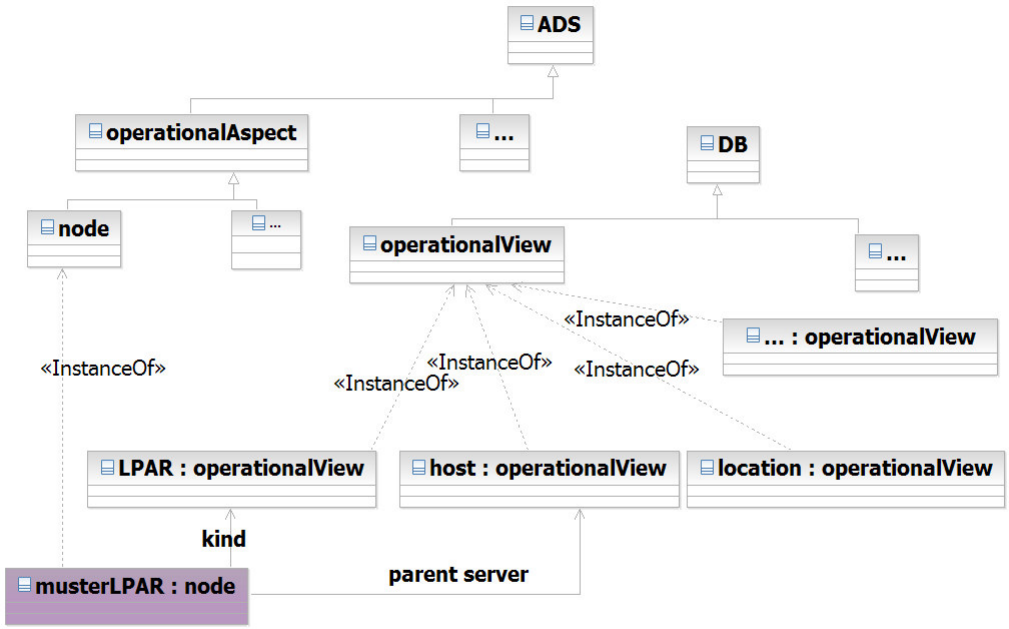


Abbildung 4.9: Beispiel für Klassifikation in einem Objektsystem

4.3.7 Schemagenerierung

In [BSCS07] wird ein Ansatz zur Generierung eines Schemas für ein CCMS vorgestellt. Dieser kann sowohl zur Weiterentwicklung eines CCMS als auch zur anfänglichen Konstruktion eines Systems verwendet werden. Der Ansatz nennt sich Asset Schema Inference Process (ASIP). Dieses ist ein weiterer Schritt zur Minimierung der Arbeit mit der Assetdefinitionssprache. Ein Problem liegt in einer semistrukturierten Notationsform vor. Damit werden Entitäten auf eine Weise beschrieben, die dem Inhalt und seinen Erklärungen einen konzeptorientierten Sinn verleiht (mehr dazu in [BSCS07]). Liegt solch eine Beschreibung einer Entität vor, kann diese erkannt und entsprechend verarbeitet werden. Auf diese Weise lassen sich Asset-Klassen erstellen. Dieser Prozess unterteilt sich in vier Phasen, die im Folgenden kurz vorgestellt werden (siehe auch Abbildung 4.10 auf Seite 60):

- Akquisition

  Es müssen Entitätsbeschreibungen in Form einer semistrukturierten Notation erstellt werden.

- Schemagenerierung

  Die Generierung eines Schemas ist eine vollautomatische Phase. Aus der formfreien Notation wird ein Schema erstellt.

- Feedback

  Das generierte Schema wird dem Anwender präsentiert. Dieser hat die Möglichkeit, Änderungsvorschläge zu machen und das Schema neu generieren zu lassen, wenn er mit dem aktuellen unzufrieden ist.

- Systemkonstruktion

  In dieser Phase wird das komplette System aufgebaut, inklusive der Integration der unter Umständen schon vorhandenen Teile des Systems.

Somit lässt sich mit dem ASIP unter Berücksichtigung der Anforderungen eines Anwenders, im Rahmen von Interaktionen mit diesem, aus einem semistrukturierten Text ein Schema für ein CCMS generieren, ohne manuell die Asset-Klassen anlegen zu müssen.

4.4 Einsatzmöglichkeiten

In diesem Unterkapitel werden die konkreten Einsatzmöglichkeiten der Konzeptorientierten Inhaltsverwaltung vorgestellt. Dazu wird zwischen den Möglichkeiten bei der Modellierung und Entwicklung und in der Handhabung unterschieden. Das Unterkapitel 4.4.1 geht zunächst auf die Modellierung mit mehreren beteiligten Fachkräften unter Koordination durch einen verantwortlichen IT-Architekten ein. Danach wird der eLearning Ansatz bei der Entwicklung kurz vorgestellt. Auf diesen folgt die Darstellung einer phasenübergreifenden Entwicklung, im CCM ist diese als domänenübergreifende Entwicklung zu verstehen. Das Unterkapitel 4.4.2 beschäftigt sich mit den Vor- und Nachteilen des CCM im Vergleich zu der im Unterkapitel 3.4.2 vorgestellten Lotus Notes Lösung hinsichtlich der Handhabung. Es beginnt mit den Möglichkeiten der Verwendung des Inhaltsteils eines Assets und wird mit den Verzweigungen innerhalb eines Systems fortgesetzt. Nach der Vorstellung einer Einsatzmöglichkeit, die im Wesentlichen auf die Personalisierung eines Systemausschnitts zurückzuführen ist, werden die Möglichkeiten

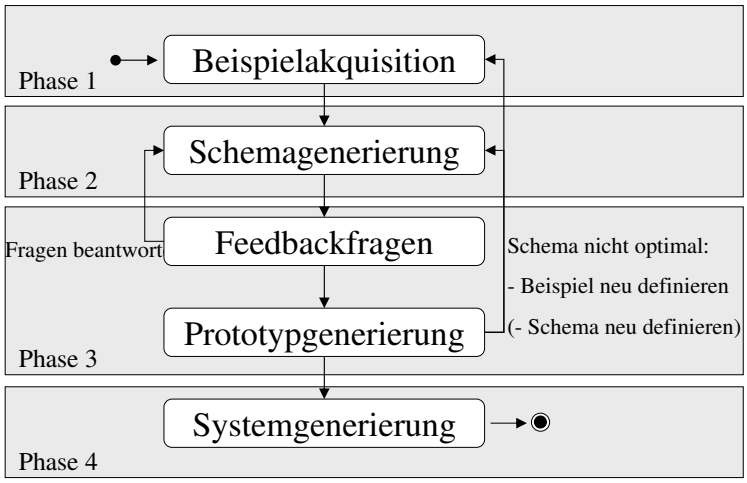


Abbildung 4.10: Übersicht zum ASIP [BSCS07]

der Initialisierung und der Wahrung der referentiellen Integrität vorgestellt. Eine kurze Diskussion über die graphische Oberfläche schließt dieses Unterkapitel ab.

4.4.1 Modellierung und Entwicklung

Das CCM bietet neue Möglichkeiten bei der Erstellung eines Modells. Damit kann z.B. das in Abbildung 3.1 auf Seite 33 dargestellte Modell von mehreren Modellierern gleichzeitig bearbeitet werden. Im Folgenden wird beschrieben, wie dieses konkret aussehen könnte.

Ein bestimmter IT-Architekt koordiniert bzw. überwacht die Modellierung, d.h. er ist derjenige, der die verantwortliche Rolle übernimmt und darüber entscheidet, welche Teile einer Publikation aus einer personalisierten Sicht in das Modell eingehen. Die Ausgangslage wäre entweder ein durch z.B. den verantwortlichen IT-Architekten selbst ausgearbeiteter grober Entwurf oder nur die Modellanforderung, d.h. es gäbe zunächst kein Basismodel, welches personalisiert werden könnte. Der einzelne IT-Facharchitekt bzw. Modellierer hat seine eigene Sicht und macht seine Vorschläge. Sobald er seinen Vorschlag für veröffentlichungswürdig hält, publiziert er diesen. Der verantwortliche IT-Architekt entscheidet über die Verwendbarkeit des vorgeschlagenen Entwurfs oder dessen Teile. Erkennt er akzeptable Ansätze, übernimmt er diese in das Basismodell. Wenn es noch kein Basismodell bis dato gegeben hat, kann der verantwortliche IT-Architekt den ersten von den Facharchitekten gemachten Entwurf komplett übernehmen und diesen als Ausgangslage verwenden.

Treten irgendwelche Änderungen am Basismodell auf, werden die einzelnen Facharchitekten darüber informiert. Für sie gilt nun das aktualisierte Basismodel als Grundlage, auf der sie ihre Modellierung fortsetzen. Folglich muss jeder Facharchitekt sein bis dahin entworfenes Modell der Basis anpassen oder sein Modell sofort veröffentlichen und den verantwortlichen IT-Architekten darüber entscheiden lassen, ob sein Modell ein besserer Vorschlag ist und dieses als Basis übernommen wird. Entscheidet

sich der verantwortliche IT-Architekt an dieser Stelle für eine Vereinigung der Ideen, ist der Grundstein für eine Teamarbeit mit Hilfe des CCM gelegt. Das Modell kann nun inkrementell, Publikation für Publikation, wie in Abbildung 4.11 dargestellt, verfeinert und optimiert werden. Die dabei Enstehenden Konflikte bzw. Widersprüche werden ständig durch den verantwortlichen IT-Architekten überwacht und beseitigt.

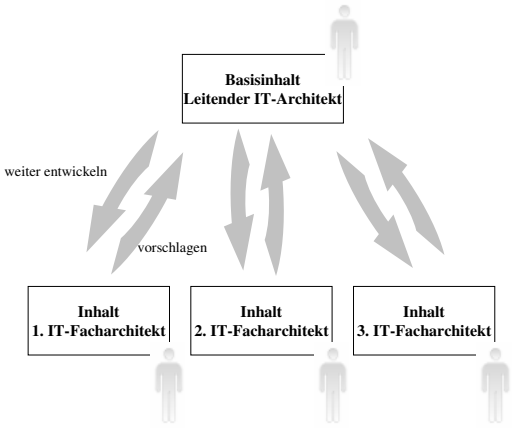


Abbildung 4.11: Modellentwicklung im Team

Die größten Änderungen bzw. der größte Aufwand treten zu Beginn der Modellierung auf, besonders wenn kein erster grober Entwurf seitens des verantwortlichen IT-Architekten vorliegt. Ist diese Phase überwunden, wird die Anzahl der Änderungen immer geringer, bis das endgültige Modell fertig gestellt ist.

An dieser Stelle muss erwähnt werden, dass man mit der Erstellung eines Modells gleichzeitig ein CCMS implementiert hat. Die benötigten Asset-Klassen mit deren Eigenschaften und Beziehungen sind damit ebenfalls definiert worden. Damit kann ein CCMS nun mit Inhalt gefüllt werden.

Des Weiteren kann beim Software- und System-Engineering der Ansatz des eLearnings verwendet werden. Ein CCMS kann hiermit als Wissensverwaltungssystem dienen. Mit jedem Projekt wächst der Inhalt eines CCMS und damit die Anzahl der Assets, die in einem weiteren Projekt wieder verwendet werden können. Durch die Personalisierung können in Abhängigkeit von den aktuellen Anforderungen gezielte Bereiche eines CCMS verwendet und durch die Publikation erweitert werden. Damit werden nach und nach Erfahrungen gesammelt und ausgetauscht. Mehr zum Einsatz von CCM beim eLearning ist im [SBS05] zu finden.

Eine weitere Möglichkeit des Einsatzes von CCM, ist die Praktizierung einer, wie in der Abbildung 4.12 dargestellt, domänenübergreifenden Entwicklung (siehe dazu [BSSS07]). Mit Hilfe der Systemmodule können unterschiedliche Projektmitarbeiter zur Optimierung der Ergebnisse auf unterschiedliche für sie relevante Phasen innerhalb eines Entwicklungsprozesses zugreifen. Die Ergebnisse der einzelnen Phasen werden in unterschiedlichen Datenbanken abgelegt, die die verschiedenen Domänen darstellen. Die Designer können z.B. auf die Ergebnisse der Analyse-Phase zurückgreifen. Damit wird mit Hilfe eines mediation module eine Brücke zwischen der Analyse und der Designphase geschlagen. Das mapping module, welches den Designer von „seiner“ Datenbank trennt, erweitert die Design-Assets um

die Beziehungen zu den Analyse-Assets. Analog sind die übrigen in der Abbildung 4.12 aufgeführten Phasenübergriffe zu deuten.

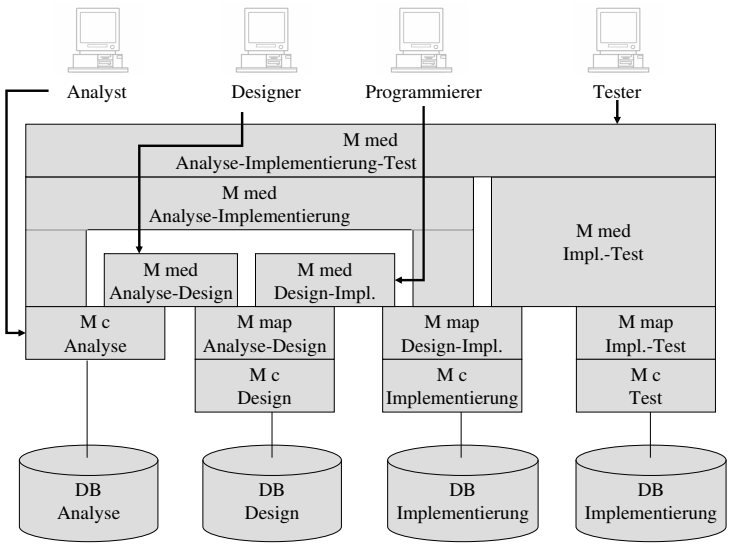


Abbildung 4.12: CCM für den Einsatz beim Softwareengineering [BSSS07]

4.4.2 Handhabung

In der Anwendung hätte ein CCM aufgrund der Personalisierung ebenfalls einige Vorteile. Neben weiterer Beziehungen oder Charakteristika besteht die Möglichkeit, größere zusätzliche Datenmengen innerhalb eines Assets zu speichern, wie z.B. Bilder oder Handbücher in Form von Dateien. Diese werden in dem Content-Teil eines Assets abgelegt. Eine Lotus Notes Datenbank bietet diese Möglichkeit rein technisch gesehen auch. Dieses muss allerdings bei der Entwicklung berücksichtigt werden, und der eingebettete Inhalt ist für alle, mit den entsprechenden Rechten ausgestatteten Anwendern, sichtbar.

Ist die Möglichkeit des Hinzufügens zusätzlicher Dateien bei der Entwicklung einer Lotus Notes Datenbank berücksichtigt worden, kann einem Dokument z.B. die Information zum theoretischen Hintergrund beigefügt werden. Demnach kann eine LPAR namens musterLPAR die Information zum ADS-Node enthalten. Allerdings gilt ein Eintrag solcher Art nur innerhalb eines bestimmten Dokuments. Den theoretischen Hintergrund an einer aus allen entsprechenden Dokumenten heraus zugreifbaren Stelle zu verwalten, ist unter Lotus Notes ebenfalls machbar. Damit ist eine Aufteilung in unterschiedliche Domänen realisierbar, kann allerdings nicht durch den Anwender geschehen.

Darüber hinaus gestaltet sich die Verzweigung zu anderen Elementen, wie z.B. von der musterLPAR zu networkName1 oder networkName2, in einem CCMS direkter und effektiver als in einer Lotus Notes Datenbank. Es wird über die im Browser angezeigten Links verzweigt, die automatisch mit Hilfe einer vordefinierten Suchanfrage, wie im Beispiel-Code 4.4 gezeigt, erstellt wird. In einem Lotus Notes Dokument gibt es nicht die Möglichkeit, ein Feld über eine automatische Suche mit Links zu anderen Dokumenten zu füllen. Folglich werden die Felder mit Schlüsselausdrücken belegt, die bei der Suche

nach den Dokumenten, auf die verzweigt wird, eingesetzt werden. Enthält das Feld mehr als einen Eintrag, wird der Umweg über ein Auswahlfenster gemacht, in dem der komplette Inhalt des z.B. Feldes Uses component(s) noch mal angezeigt wird, wie es in der Abbildung 4.13 dargestellt ist, und für die Verzweigung ausgewählt werden kann.

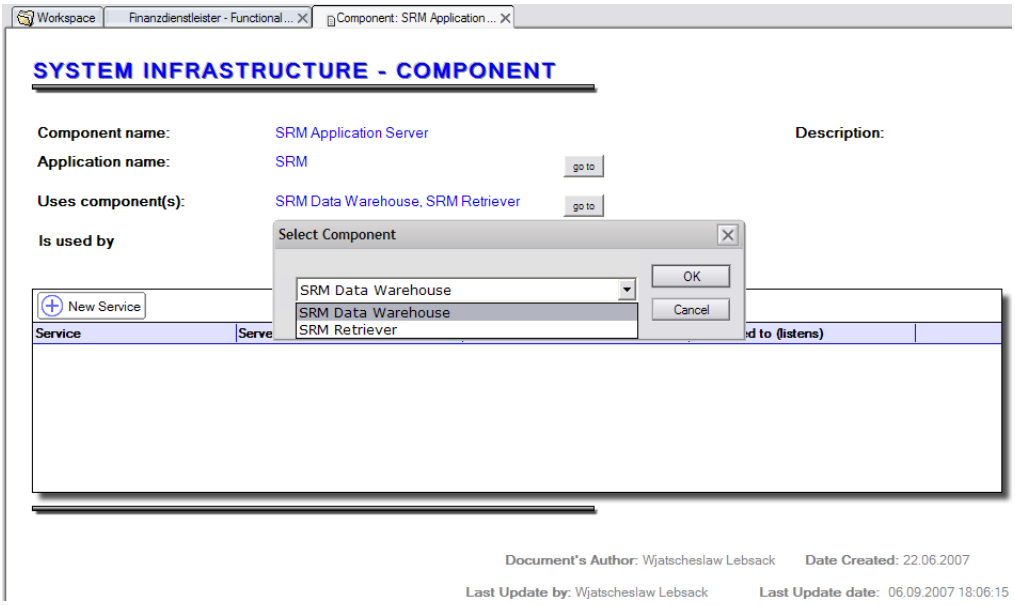


Abbildung 4.13: Verzweigung zu enem anderen Element

Eine weitere interessante Einsatzmöglichkeit besteht momentan in dem Projekt, in dem diese Arbeit entstanden ist. Die gesamte IT-Infrastruktur ist meistens in mehrere Bereiche unterteilt. Diese werden von den dafür zuständigen Teams, verwaltet. Die Verwaltung geschieht häufig in einer nicht einheitlichen Form. Die Informationen z.B. darüber, welche LPAR wie ausgestattet ist, wo diese sich befindet und wer der Ansprechpartner dafür ist, werden unterschiedlich aufbewahrt. Meistens sind es Text- oder Tabellenkalkulationsdateien. Nachdem die Information einmal daraus entnommen und in Form von Asset-Instanzen in einem CCMS abgelegt ist, könnte auf diese Quelldokumente in einem CCMS verzichtet werden, weil nun den einzelnen Teams ein entsprechender Bereich, d.h. eine personalisierte Sicht, zur Verfügung gestellt werden könnte, die sie zu pflegen hätten. Dieses Ziel der Abschaffung von jeglichen Dokumenten, die eine Schnittstelle darstellen, soll eine Fortsetzung der in Kapitel 3 vorgestellten Version einer IT-Infrastrukturdatenbank werden, indem eine Onlineschnittstelle zu den bestehen Asset- und Managementdatenbanken implementiert wird. Bei der aktuellen Lösung bleiben allerdings diese Quelldokumente zunächst ein wichtiger Bestandteil der Informationsverwaltung, weil die Gesamtheit sich nur mit einem großen Aufwand in separat zu verwaltende Abschnitte unterteilen lässt und die einzelnen Teams mit der Datenbank nicht arbeiten sollen. Damit wird die Kommunikation zwischen den Teams und dem Datenbank-Administrator/Anwender weiterhin über die externen Dateien laufen.

Bezüglich der Initialisierung eines Systems bietet eine Lotus Notes Datenbank einen höheren Grad an Automatismus. Der Inhalt kann nämlich aus den Dateien, die momentan noch als Schnittstelle zwischen der IT-Infrastrukturdatenbank und den bestehenden Asset- und Managementdatenbanken dienen, generiert werden. Diese Funktionalität muss allerdings im Designer bei der Entwicklung berücksichtigt werden. Beim CCM besteht mittlerweile, wie im Unterkapitel 4.3.7 vorgestellt, eine Möglichkeit, ein Schema für das System automatisiert zu generieren, allerdings werden auf diese Weise noch keine

Asset-Instanzen erstellt.

Ein großer Vorteil des CCM im Vergleich zu einer Lotus Notes Datenbank ist die Wahrung der referentiellen Integrität. Während diese in der Lotus Notes Lösung bis ins kleinste Detail vom Entwickler sicherzustellen ist, unterstützt das CCM das Löschen der Beziehungen auf das bereits gelöschte Element (siehe Abbildung 4.14). Das kaskadierte Löschen von Elementen ist unter Lotus Notes nur möglich, wenn dieses bei der Implementierung vorgesehen worden ist. In Abhängigkeit von den durch den Anwender gesetzten Beziehungen ist es, im Gegensatz zum CCM, nicht möglich. Das Löschen aller Netzwerke der im Code-Beispiel 4.3 auf Seite 54 angelegten Asset-Instanz musterLPAR erfolgt z.B. mit der im Code-Beispiel 4.8 auf Seite 65 angegeben Löschoperation.

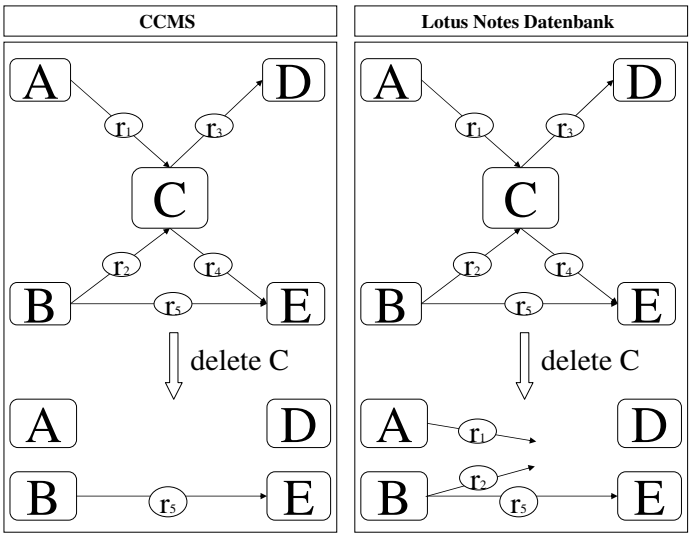


Abbildung 4.14: Integrität im CCM im Vergleich zu Lotus Notes

```
1 delete musterLPAR.network
```

Listing 4.8: Kaskadiertes Löschen von Assets

Ein großer Vorteil der Lotus Notes Lösung im Vergleich zum CCM ist die graphische Oberfläche und keine Voraussetzung jeglicher Programmierkenntnisse seitens der Anwender. Mit der graphischen Oberfläche ist nicht der Umgang mit einem System in der Rolle eines lesenden Anwenders zu verstehen. Eine graphische Oberfläche dieser Art wird im CCM über einen Browser realisiert. Was fehlt, ist ein Modellierungstool, um einem Anwender ein komplett programmierfreies Arbeiten beim Erstellen neuer Asset-Klassen aber auch beim Anlegen, Löschen, Suchen usw. von Asset-Instanzen zu bieten.

4.5 Fazit

Das CCMS und seine Daten sind so organisiert, dass sowohl die Systemstruktur als auch der Inhalt durch den Anwender in einer personalisierten Sicht manipulierbar sind, was in den konventionellen Inhaltsverwaltungssystemen nicht möglich ist. Dort besitzen die Daten eine feste Definition und ein unveränderliches Schema. Durch die Möglichkeit der Publikation wächst der Informationsgehalt. Durch die Aufteilung des Wissens in Domänen, deren Inhalt von dem entsprechenden Fachpersonal gepflegt und in Form von wiederverwendbaren Assets abgelegt wird, erhöht sich die Qualität der Information und die Effizienz einer Problemlösung.

Der Einsatz des CCM würde beim Software- und Systemengineering neue Möglichkeiten bieten. Meistens sind diese auf den Aspekt der gleichzeitigen Implementierung und Nutzung eines Systems zurückzuführen. Dadurch entwickelt sich das System immer weiter, ohne einen nennenswerten Programmieraufwand. Durch eine direkte Anpassung an den Anwender aufgrund des ständigen Eingriffs und der Optimierung durch den Anwender, kann eine hohe Qualität in der Bedienbarkeit erreicht werden. Es muss allerdings an dieser Stelle erwähnt werden, dass das CCM auch einige Hürden mit sich bringt. Unter anderem stellt die Pflicht des Anwenders, sich mit der Assetdefinitionssprache auseinandersetzen zu müssen, eine wesentliche Hürde dar, obwohl diese nur geringe Programmierkenntnisse voraussetzt. Eine graphische Modellierungsumgebung könnte die Arbeit mit einem CCMS insofern erleichtern, indem die Voraussetzung selbst minimaler Programmierkenntnisse beim Anwender entfallen würde. Im Allgemeinen wird noch sehr viel direkt mit dem Quellcode gearbeitet. Ein unterstützendes Tool wäre auch hier sehr hilfreich. Wird relativ zum Basismodell ein personalisiertes Modell erstellt, werden die danach durchgeführten Änderungen am Basismodell in der personalisierten Sicht nicht bekannt gegeben. Es darf zwar keine automatische Änderung an einer personalisierten Sicht vorgenommen werden, eine Mitteilung über die Änderungen am Basismodell wäre aber wünschenswert.

5 Schlussbetrachtung und Ausblick

In diesem Kapitel werden die im Laufe dieser Arbeit unternommenen Schritte zusammengefasst und aus diesen gewonnene Erkenntnisse vorgestellt. Ausführliche Bewertungen der Materie sind zum Teil am Ende der jeweiligen Kapitel zu finden. Die Arbeit wird mit einer kurzen Vorstellung der möglichen und offenen Punkten abgeschlossen.

5.1 Schlussbetrachtung

Ausgehend von der Schwierigkeit der Unternehmen, den Überblick über die sich im Einsatz befindende Hard- und Software zu behalten, und der Feststellung, dass sich daraus eine Menge von Problemen resultiert, wird im Rahmen dieser Diplomarbeit der Ansatz des Reverse Engineering gewählt und die IT-Infrastruktur eines Finanzdienstleisters in einem Modell dargestellt, welches unter Verwendung der „klassischen“ Vorgehensweise in einer Datenbank abgelegt wird. Darüber hinaus wird die Möglichkeit der Anwendung der Konzeptorientierten Inhaltsverwaltung (engl. Conceptual Content Management, abgekürzt CCM) als alternative Möglichkeit für eine strukturierte Dokumentation untersucht.

Der Modellierung geht eine Darstellung, der Übersicht der in der Wirtschaft und der Forschung vorhandenen Standards und Methoden, voran. Auf der Basis unterschiedlicher Definitionen einer IT-Systemarchitektur wird die Existenz vieler unterschiedlicher Standards und Methoden begründet und eine Definition für das Verständnis einer IT-Architektur in dieser Arbeit hergeleitet. Es gibt keinen Standard und keine Methode, die alle Situationen berücksichtigt. Allerdings sind in der Regel ähnliche Grundgedanken in den unterschiedlichen Standards und Methoden verankert. Es wird sowohl auf die Architekturbeschreibungssprachen als auch auf einige methodischen Vorgehensweisen der Unternehmen eingegangen, die meistens innerhalb der Unternehmen basierend auf langjähriger Erfahrung entstanden sind. Dabei wird die Bedeutung der Modellierung und einer strukturierten und, nach Möglichkeit, einheitlichen Darstellung der Modelle angesprochen. Auf die Eignung der Standards und ADLs für die Darstellung der zu realisierenden und der schon bestehenden Soft- und Hardwaresysteme wird ebenfalls eingegangen. Es wird das Reverse Engineering als Vorgehensweise vorgestellt und seine grundlegenden Unterschiede zu dem traditionellen Forward Engineering behandelt. Darüber hinaus werden der Ursprung und der Kerngedanke einzelner, ausgewählter Vorgehensweisen angesprochen.

Auf des Basis bestimmter Anforderungen und verfolgter Visionen wird unter Verwendung eines bestimmten Standards die IT-Infrastruktur eines Finanzdienstleisters in einem Modell abstrahiert. Auf der Basis dieses Modells entsteht eine Infrastrukturdatenbank, die die wesentlichen Elemente und die Verbindungen zwischen diesen beinhaltet und teilweise die Möglichkeit bietet, den Inhalt so aus der Datenbank wieder zu exportieren, dass ohne einen weiteren nennenswerten Aufwand eine graphische Darstellung der Systemausschnitte erreicht wird. Ein Import von Daten, aus den als Schnittstelle zwischen der Infrastrukturdatenbank und den bestehenden Management-Datenbanken dienenden Dateien, ist ebenfalls möglich. Hier wurde die Erfahrung gemacht, dass die Modellierung eines schon vorhandenen

Systems keinesfalls ein einfaches Unterfangen ist. Das letztendlich fertig gestellte, im Unterkapitel 3.3 dargestellte, Modell hat eine intensive Auseinandersetzung mit der vorhandenen Hardware, der in dem Projekt eingesetzten Anwendungen und deren Verteilung im System und der Verknüpfung der Server mit den dazugehörigen Netzwerken vorausgesetzt. Darüber hinaus ist eine gründliche Erforschung eines Standards samt seiner Eigenarten und Grenzen ein langwieriger Prozess. Es stellt sich ebenfalls als schwierig heraus, alle Wünsche und Visionen ohne weiteres zufrieden stellen zu können, wie z.B. die graphische Darstellung des operationalen Modells. Die Lösung dieses Problems wird in dieser Arbeit in Aussicht gestellt.

Die Konzeptorientierte Inhaltsverwaltung verfolgt die Fähigkeit der Menschen, deren Umfeld unterschiedlich zu interpretieren. Durch die Geistes- und Kulturwissenschaften, durch Fakten schwierig darzustellenden Gebiete, initiiert, werden die Idee und das Werkzeug des CCM auf die Darstellung und Entwicklung von Systemen angewendet. Es wird ein sich im Einsatz befindendes CCM vorgestellt und die wesentlichen Fähigkeiten und deren Basis angesprochen. Darauf basierend werden Modelle für die Dokumentation von IT-Infrastrukturen entworfen und der Einsatz der Modellierungssprache des CCM zur Darstellung von Systemelementen in Form von Assets und deren Handhabung getestet. Unter einem Asset ist ein gekapselter Satz von Informationen und Erfahrungen zu verstehen, die wieder verwendet werden können und aus Effizienzgründen sogar verwendet werden sollten. Darüber hinaus werden unterschiedliche Einsatzmöglichkeiten des CCM diskutiert und einige wesentlichen Unterschiede zu einer Lotus Notes Lösung vorgestellt. In diesem Kapitel kommt es zu der Schlussfolgerung, dass Projekte mit dem Einsatz von CCM zur Beschreibung von Modellen durgeführt werden könnten. Gerade bei der Dokumentation bestehender Systeme, unter Verwendung des Reverse Engineering-Ansatzes, könnte CCM in Frage kommen. Der Einsatz des CCM könnte beim Software- und Systemengineering neue Möglichkeiten bieten. Nicht zu vernachlässigen sind allerdings die noch vorhandenen Hürden, die ebenfalls in Aussicht gestellt werden. Dazu gehört in erster Linie die fehlende graphische Modellierungsumgebung.

5.2 Ausblick

In diesem Unterkapitel werden einige offene Punkte vorgestellt, die sowohl die Verwendung der in Kapitel 3.4.2 entworfenen IT-Infrastrukturdatenbank als auch den Einsatz des in Kapitel 4 vorgestellten CCM beeinträchtigen. Darüber hinaus wird ein Vorschlag zur Vereinigung beider Lösungsansätze gemacht.

Wie schon im Unterkapitel 3.5 erwähnt, stellt die Ungewissheit darüber, wann und mit welchen der beiden, im Moment hofnungstragenden, Tools das operationalen Modells darstellbar wird, eine Hürde bei der Generierung einer graphischen Darstellung der operationalen Sicht auf ein System dar. Eine wünschenswerte Darstellung wäre z.B. die zunächst manuell gezeichnete Sicht auf das Server Ressource Management in der Abbildung 5.1.

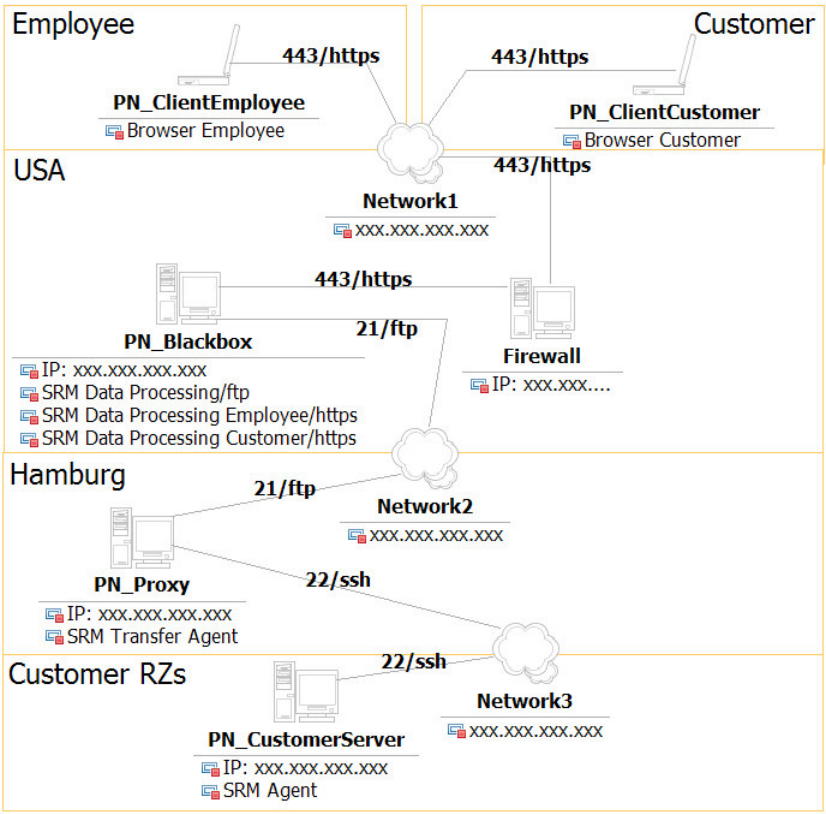


Abbildung 5.1: SRM operationales Modell

Ebenfalls wünschenswert ist eine direkte Verbindung zu den bestehenden Asset- und Managementdatenbanken. Auf diese Weise würde die Information direkt der Quelle entnommen werden. Die Schnittstellen in Form von Dateien könnten abgeschafft werden.

Ein weiterer, noch offener, Punkt ist eine Typisierung der Server. Durch eine Typisierung würde einem Server in Abhängigkeit von dem Typ ein Satz von Anwendungen, die auf diesem zu laufen haben, zugewiesen. Auf dieser Basis könnte ein Abgleich, zwischen dem was auf einem Server ausgeführt werden

sollte und was tatsächlich ausgeführt wird stattfinden. Es soll also mit einer Art Soll-Aufstellung von Anwendungen und dem Inhalt der Datenbank verglichen und beim Auftritt von Diskrepanzen darauf hingewiesen werden können.

Eine wesentliche Hürde beim CCM stellt die noch fehlende graphische Oberfläche dar und damit die Pflicht eines Anwenders, sich mit der Assetdefinitionssprache auseinandersetzen zu müssen. Unter der graphischen Oberfläche ist ein Modellierungstool zu verstehen. Damit würde dem Anwender ein komplett programmierfreies Arbeiten beim Erstellen neuer Asset-Klassen aber auch beim Anlegen, Löschen, Suchen usw. von Asset-Instanzen ermöglicht.

Es könnten Projekte mit dem CCM durchgeführt werden, um auch aus der Praxis heraus über die Eignung des CCM als alternative Modellierungsmöglichkeit zur „klassischen“ Modellierung und vor allem für das in dieser Arbeit besonders relevante Reverse Engineering, bei dem eine Ist-Aufnahme der IT-Infrastruktur erstellt wird, entscheiden zu können. Reverse Engineering ähnelt sehr stark der Vorgehensweise, die zum Erfassen des graphischen Materials der Politischen Ikonographie in der WEL verwendet wurde. Im Prinzip ist eine Digitalisierung, der schon vorhandenen und strukturierten Ressourcen, auch eine Art Reverse Engineering, welches eine Weiterentwicklung impliziert.

Eine zum größten Teil automatische Generierung einer graphischen, abstrakten Darstellung einer IT-Infrastrukturarchitektur ist unter anderem ein Ziel, welches durch eine Dokumentation eines IT-Systems realisierbar wird. Demnach müsste eine Möglichkeit gefunden werden, auch aus einem CCMS den Inhalt so zu exportieren, dass er ohne weiteren nennenswerten Aufwand in eine graphische Darstellung überführbar ist.

Eine weitere Überlegung wird in Verbindung mit den beiden Lösungsansätzen, dem CCM und der Lotus Notes Datenbank, angestellt. Und zwar könnte der Inhalt einer Lotus Notes Datenbank verwendet werden, um damit den Inhaltsteil eines Assets zu bilden.

A Aufstellung von Standards und Methoden

ADL/Standard/Methode	Quelle
AADL Architecture Analysis & Design Language	Feiler P. H., Gluch D. P. and Hudak J. J., *The Architecture Analysis & Design Language (AADL): An Introduction.*, Feb. 2006)
Acme	CMU-Carnegie Mellon University, *Acme*, `http://www.cs.cmu.edu/~acme` (letzter Zugriff am 16. Sept. 2007)
ADS IBM Architecture Description Standard	Kahan E., *Architecture Description Standard Version 2.0: Overview*, IBM Paper, Mai 2002 IBM Global Services, *ADS Operational Model Notation Version 2.0*, IBM Paper, Mai 2002 IBM Global Services, *ADS Funktional Model UML1 Notation*, IBM Paper, Mai 2002
Aesop	CMU-Carnegie Mellon University, *Aesop*, `http://www.cs.cmu.edu/afs/cs/project/able/www/aesop/aesop_home.html` (letzter Zugriff am 16. Sept. 2007)
BPMN Business Process Modeling Notation	Owen M. and RAJ J., *BPMN and Business Process Management, Introduction to the New Business Process Modeling Standard*, Popkin Software, 2003
C2	Medvidovic N., Rosenblum D. S. and Taylor R. N., *A Language and Environment for Architecture-Based Software Development and Evolution*, In Proceedings of the 21st International Conference on Software Engineering (ICSE99), Los Angeles, CA, May 1999
CDM ORACLE's Custom Development Method	Palindrome Solutions, Inc., *Project Management and Mentorship*, `http://www.palindrome-solutions.com/serv03.htm` (letzter Zugriff am 16. Sept. 2007) Thomaschewski A., *Vorstellung Oracle Consulting*, Oracle Präsentation, 2000
CODE	University Of Texas At Austin, *CODE*, `http://www.cs.utexas.edu/users/code` (letzter Zugriff am 27. Sept. 2007)
ControlH & MetaH	Honeywell Technology Center, *ControlH & MetaH*, `http://www.htc.honeywell.com/projects/dssa/dssa_tools.html` (letzter Zugriff am 27. Sept. 2007)

ADL/Standard/Methode	Quelle
Darwin	Magee J., Dulay N., Eisenbach S. and Kramer J., *Specifying Distributed Software Architectures*, In Proceedings of the Fifth European Software Engineering Conference (ESEC95), Barcelona, Sept. 1995
Demeter	Northeastern University, *Demeter: Aspect-Oriented Software Development*, `http://www.ccs.neu.edu/home/lieber/demeter.html` (letzter Zugriff am 27. Sept. 2007)
DoDAF Department of Defense Architecture Framework	Department of Defense USA, *DoD Architecture Framework Version 1.5, Volume I: Definitions and Guidelines*, DoDAF Specification, 23.04.2007
FDD Feature Driven Development	Nebulon Pty. Ltd., *FDD Processes*, `http://www.nebulon.com/articles/fdd/latestfdd.html` (letzter Zugriff am 20. Sept. 2007)
Gestalt	Schwanke R. W., Strack V. A. and Werthmann-Auzinger T., *Industrial Software Architecture with Gestalt*, Siemens Corporate Research, Inc., Paper, `http://www.sei.cmu.edu/architecture/IWSSD8.Gestalt.paper.html` (letzter Zugriff am 27. Sept. 2007)
GSM IBM Global Services Method	Skarits D.I. E., *IBM Global Services Method (IBM GSM)*, IBM Paper, 2003
IDEF Integrated Definition Methods	Knowledge Based Systems, Inc., *Integrated Definition Methods*, `http://www.idef.com/` (letzter Zugriff am 27. Sept. 2007)
IEEE 1471	IEEE, *IEEE Recommended Practice for Architectural Description of Software Intensive Systems (IEEE Std 1471-2000)*, IEEE Computer Society, New York, 2000
IUM Integrierte Unternehmensmodellierung	Frauenhofer IPK, *Integrierte Unternehmensmodellierung*, `http://www.ipk.fhg.de/geschaeftsfelder/um/ufp/unternehmensgestaltung/anwendungen` (letzter Zugriff am 27. Sept. 2007)
MBASE Model-Based Architecting & Software Engineering	USC-University of Southern California, *Model-Based Architecting & Software Engineering*, `http://csse.usc.edu/research/MBASE`
MetaH	Vestal S., *MetaH Programmers Manual, Version 1.09*, Technical Report, Honeywell Technology Center., Apr. 1996
MODAF Ministry of Defence Architecture Framework	Britannic Majesty's Government, *The MOD Architecture Framework Version 1.1*, 39237, `http://www.modaf.org.uk`
NAF NATO Architecture Framework	*Bin noch Auf der Suche nach einer Vernünftigen Quelle (Stand 28.09.07)*

ADL/Standard/Methode	Quelle
OEP Object Engineering Process	Oestereich B., Hruschka P., Josuttis N., Kocher H., Krasemann H. und Reinhold M., *Erfolgreich mit Objektorientierung Vorgehensmodelle und Managementpraktiken für die objektorientierte Softwareentwicklung 2.*, Oldenbourg, Auflage: 2., Feb. 2001 Oestereich B., Schröder C., Klink M. und Zockoll G., *OEP - oose Engineering Process - Vorgehensleitfaden für agile Softwareprojekte*, Dpunkt Verlag, 1. Auflage., Nov. 2006
OPENUP Open Unified Process	The Eclipse Foundation, *Open Unified Process*, `http://www.eclipse.org/epf/downloads/openup/openup_downloads.php` (letzter Zugriff am 16. Sept. 2007)
Rapide	Stanford University, *Rapide*, `http://pavg.stanford.edu/rapide` (letzter Zugriff am 16. Sept. 2007)
RESOLVE	Ogden W. F., Sitaraman M., Weide B. W. and Zweben S. H., *Part I: The RESOLVE Framework and Discipline A Research Synopsis*, Ohio State University, 01.10.1994
RM-ODP Reference Model For Open Distributed Processing	ISO, *RM-ODP*, 1995 • Teil 1 (Overview): ITU-T Rec. X.901 / ISO/IEC 10746-1 • Teil 2 (Foundations): ITU-T Rec. X.902 / ISO/IEC 10746-2 • Teil 3 (Architecture): ITU-T Rec. X.903 / ISO/IEC 10746-3 • Teil 4 (Architectural semantics): ITU-T Rec. X.904 / ISO/IEC 10746-4
RUP Rational Unified Process	Kruchten P., *The Rational Unified Process An Introduction*, Thirt Edition Addison-Wesley, Longman, Amsterdam, Auflage: 3., 6. Jan. 2004 Rational Software, *Rational Unified Process, Best Practices for Software Development Teams*, Rational Software Corporation White Paper, 1998 Kruchten P., *The 4+1 View Model of Architecture*, Paper published in IEEE Software 12 (6), pp. 42-50., Nov. 1995
SADL Structural Architecture Description Language	Moriconi M. and Riemenschneider R. A., *Introduction to SADL 1.0: A Language for Specifying Software Architecture Hierarchies*, Technical Report SRI-CSL-97-01, SRI International, March 1997
Scrum	Schwaber K., *What Is Scrum?*, Paper, 17. Sept. 2007
TOGAF The Open Group Architecture Framework	The Open Group, *The Open Group Architecture Framework (TOGAF 8.1.1 The Book)*, 38961, `http://www.opengroup.org/togaf` (letzter Zugriff am 16. Sept. 2007)

ADL/Standard/Methode	Quelle
UML Unified Modeling Language	Schäling B., *Der moderne Softwareentwicklungsprozess mit UML*, Version 1.01, 14. Feb. 2005, `http://www.highscore.de/uml/index.html` (letzter Zugriff am 16. Sept. 2007) Kecher C., *UML 2.0 - Das umfassende Handbuch*, Galileo Computing, Auflage: 2., Mai 2006
UniCon	CMU-Carnegie Mellon University, *UniCon Reference Manual*, `http://www.cs.cmu.edu/~UniCon/reference-manual/Reference_Manual_1.html` (letzter Zugriff am 16. Sept. 2007)
Weaves	Gorlick M. M. and Razouk R. R., *Using Weaves for Software Construction and Analysis*, In Proceedings of the 13th International Conference on Software Engineering (ICSE13), pages 23-34, Austin, TX, May 1991
Wright	CMU-Carnegie Mellon University, *The Wright Architecture Description Language*, `http://www.cs.cmu.edu/afs/cs/project/able/www/wright/index.html` (letzter Zugriff am 27. Sept. 2007)
XP Extreme Programming	Beck K., *Extreme Programming. Das Manifest*, Addison-Wesley; Auflage: 2, 15. Sent. 2000
Zachman Zachman Framework	ZIFA-Zachman Institute for Framework Advancement, *Zachman Framework*, `http://www.zifa.com` (letzter Zugriff am 20. Sept. 2007)

Tabelle A.1: Eine Aufstellung von ADLs, Standards und Methoden

B Export aus der Infrastrukturdatenbank

```
Mit dem folgenden Code werden XMI-Dateien generiert.
Sub Click(Source As Button)
    Dim session As New NotesSession              'Deklaration einer Session
    Dim ws As New NotesUIWorkspace               'Deklaration eines UIWorkspace
    Dim database As NotesDatabase                'Deklaration einer Datenbank
    Dim viewComponent As NotesView               'Deklaration einer Sicht fuer die Komponenten
    Dim viewComponentByName As NotesView         'Deklaration einer Sicht fuer die Komponenten sortiert nach Applikation
    Dim docCollection As NotesDocumentCollection 'Deklaration einer Dokumentensammlung fuer die Applikationen
    Dim docXMI As NotesDocument                  'Deklaration XMI-Dokumentes
    Dim docApplicationTmp As NotesDocument       'Deklaration eines Dokumentes zum Zwischenspeichern der aktuellen Applikation
    Dim rtItemXMIBody As NotesRichTextItem       'Deklaration eines Textfeldes fuer den Code
    Dim strXMI As String                         'Deklaration eines Strings zum Sammeln von Inhalt
    Dim profileXMI As NotesItem                  'Deklaration eines Profile-Dokumentes zum Auslesen des Headers und der ersten Modellzeile fuer die XMI-Datei
    Dim docProfile As NotesDocument              'Deklaration
    Dim firstComponent As NotesDocument          'Deklaration
    Dim secondComponent As NotesDocument         'Deklaration eines Feldes
    Dim firstComponentKey As String              'Deklaration
    Dim secondComponentItem As NotesItem         'Deklaration eines Feldes zum festhalten der Komponenten, zu denen verbunden wird
    Dim itemCheckApplication As NotesItem 'to check the connection to the component oft the right application
    Dim docCollectionComponent As NotesDocumentCollection
    Dim XMIID, XMIName,  XMIClientDependency, XMIClientDependencyComp As String


    Set database = session.CurrentDatabase                  'Initialisierung eine Datenbank
    Set viewComponentByName = database.GetView("Component") 'Auswahl der Komponenten-Sicht
    Set viewComponent = database.GetView("ComponentByApplication") 'Auswahl der Komponenten-Sicht sortiert nach Applikationen
    Set docProfile = database.GetProfileDocument("Fo-AdminProfile") 'Lades des Profile-Dokumentes mit den Header fuer das XMI-File
    Set docCollection = database.UnprocessedDocuments       'Initialisierung eine Sammlung mit ausgewaehlten Applikationen
    Set docApplicationTmp = docCollection.GetFirstDocument  'Auswahl der ersten Applikation aus der Sammlung

    'Eine Schleife zum Abarbeiten der kompletten Applakationen-Sammlung
    While Not(docApplicationTmp Is Nothing)
        Set docXMI = database.CreateDocument                'Ein Dokument fuer den XML-Code wird erzeugt
        docXMI.Form = "Fo-XMI"                              'Festlegung einer Maske fuer das XMI-Dokument
        Set rtItemXMIBody = docXMI.CreateRichTextItem( "Body" ) 'Erzeugen eines Feldes namens "Body" fue den XML-Code
```

Abbildung B.1: LotusScript-Code zur Erstellung einer XMI-Datei (1 von 3)

```
    strXMI = docProfile.GetItemValue("AdminProfileXMIHeader")(0)        'Der Header wird in den String eingelesen
    Call rtItemXMIBody.AppendText(strXMI)                               'Der Inhalt des Strings wird in das Feld "Body" geschrieben
    Call rtItemXMIBody.AddNewline(1)                                    'Dem Feld "Body" wird eine leere Zeile angehaengt
    'Die erste obligatorische Zeile des Models wird eingelesen
    strXMI = docProfile.GetItemValue("AdminProfileXMIComponent")(0) & docApplicationTmp.ApplicationAcronym(0) & """ name=""" & docApplicationTmp.ApplicationAcronym(0) &""">"
    Call rtItemXMIBody.AppendText(strXMI)                               'Der Inhalt des Strings wird in das Feld "Body" geschrieben
    Call rtItemXMIBody.AddNewline(2)                                    'Dem Feld "Body" wird eine leere Zeile angehaengt

    'Auswahl von Komponenten zu der entsprechenden Applikation
    Set docCollectionComponent = viewComponent.GetAllDocumentsByKey(docApplicationTmp.ApplicationAcronym(0))
    Set firstComponent = docCollectionComponent.GetFirstDocument        'Auswahl der ersten Komponenten

    'Eine Schleife zum Abarbeiten aller Komponenten zur entsprechenden Applikation
    While Not(firstComponent Is Nothing)
       Set secondComponentItem = firstComponent.GetFirstItem("ComponentUses") 'festhalten der Verbindungskomponenten

       'Generierung von IDs aus den NoteIDs der Komponenten-Dokumente
       XMIID = firstComponent.NoteID                                    'NoteID des ersten Komponenten-Dokumentes
       XMIName = firstComponent.ComponentName(0)                        'Name der ersten Komponente
       XMIClientDependency = XMIID                                      'NoteID des ersten Komponenten-Dokumentes fuer die Verbindung
       XMIClientDependencyComp = ""                                     'zunaechst leer

       'Wenn die aktuelle Komponente Verbindungen zu anderen Komponenten besitzt, geht man Komponente fuer Komponente durch
       'und generiert den entsprechenden XML-Code, sonst wird nur der Code zu der aktuellen Komponente generiert.
       If (secondComponentItem.ValueLength > 2)Then
          Forall v In secondComponentItem.Values
             Set secondComponent = viewComponentByName.GetDocumentByKey(v)
             'Es wird immer geprueft, ob die zweite Komponente zu derselben Applikation gehoert, wie die erste.
             'Dieses wird durchgefuert, um bei den Komponente mit der Zugehoerigkeit zu mehreren Applikationen
             'keine Verwechlung der Applikationen bei der Herstellung von Verbindungen sicherzustellen.
             Set itemCheckApplication = secondComponent.GetFirstItem("ComponentApplication")
             If(itemCheckApplication.Contains( docApplicationTmp.ApplicationAcronym(0)))Then
                XMIClientDependency = XMIID  & secondComponent.NoteID
                'Ein Leerzeichen wird nur dann eingesetzt, wenn es mehrere Verbindungen gibt.
                If(XMIClientDependencyComp = "")Then
                   XMIClientDependencyComp = XMIClientDependency
                Else
                   XMIClientDependencyComp = XMIClientDependencyComp & " " & XMIClientDependency
                End If
                strXMI = "<packagedElement xmi:type=""uml:Usage"" xmi:id=""" & XMIClientDependency & """ supplier=""" & secondComponent.NoteID & """ client=""" & XMIID & """/>"
                Call rtItemXMIBody.AppendText(strXMI)
```

Abbildung B.2: LotusScript-Code zur Erstellung einer XMI-Datei (2 von 3)

```
                Call rtItemXMIBody.AddNewline(1)
             End If
          End Forall
          'Die Verbindung zwischen zwei Komponenten wird generiert.
          strXMI = "<packagedElement xmi:type=""uml:Component"" xmi:id=""" &_
          XMIID & """ name=""" & XMIName & """ clientDependency=""" & XMIClientDependencyComp & """/>"
          Call rtItemXMIBody.AppendText(strXMI)
          Call rtItemXMIBody.AddNewline(1)
       Else
          strXMI = "<packagedElement xmi:type=""uml:Component"" xmi:id=""" &_
          XMIID & """ name=""" & XMIName & """/>"
          Call rtItemXMIBody.AppendText(strXMI)
          Call rtItemXMIBody.AddNewline(1)
       End If
       Set firstComponent = docCollectionComponent.GetNextDocument(firstComponent)
    Wend

    strXMI = "</uml:Model>"     'Ende des Modells
    Call rtItemXMIBody.AppendText(strXMI)

    'Der komplette XMI-Code wird in das Feld "Body" geschrieben.
    Call docXMI.CopyItem(rtItemXMIBody, "XMIBody")
    'Call ws.EditDocument(True, docXMI)

    'Der Inhalt des Body wird in eine Datei geschrieben
    Call createXMIFile(docXMI, docProfile, docApplicationTmp)
    'Uebergang zur Naechsten Applikation
    Set docApplicationTmp = docCollection.GetNextDocument(docApplicationTmp)
  Wend
  'Mitteilung ueber den Speicherort der XMI-Datei. Dieser wir aus dem Profile-Dokument gezogen.
  Messagebox "Successful created in the following directory:" & Chr(10) & Chr(13) & docProfile.AdminProfileXMIFileLocation(0)
End Sub
```

Abbildung B.3: LotusScript-Code zur Erstellung einer XMI-Datei (3 von 3)

```
1
2  <?xml version="1.0" encoding="UTF-8"?>
3  <uml:Model xmi:version="2.1" xmlns:xmi="http://schema.omg.org/spec/XMI/2.1"
4  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5  xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore"
6  xmlns:uml="http://schema.omg.org/spec/UML/2.1.1"
7  xsi:schemaLocation="http://schema.omg.org/spec/UML/2.1.1
8  http://www.eclipse.org/uml2/2.0.0/UML" xmi:id="SRM" name="SRM">
9
10   <packagedElement xmi:type="uml:Component" xmi:id="94A" name="SRM Agent"/>
11   <packagedElement xmi:type="uml:Usage" xmi:id="95294A" supplier="94A" psy191
         client="952"/>
12   <packagedElement xmi:type="uml:Component" xmi:id="952" name="SRM Transfer psy191
         Agent" clientDependency="95294A"/>
13   <packagedElement xmi:type="uml:Usage" xmi:id="99E952" supplier="952" psy191
         client="99E"/>
14   <packagedElement xmi:type="uml:Component" xmi:id="99E" name="SRM psy191
         Retriever" clientDependency="99E952"/>
15   <packagedElement xmi:type="uml:Usage" xmi:id="9AE9B2" supplier="9B2" psy191
         client="9AE"/>
16   <packagedElement xmi:type="uml:Component" xmi:id="9AE" name="SRM Web psy191
         Server" clientDependency="9AE9B2"/>
17   <packagedElement xmi:type="uml:Usage" xmi:id="9B29B6" supplier="9B6" psy191
         client="9B2"/>
18   <packagedElement xmi:type="uml:Usage" xmi:id="9B299E" supplier="99E" psy191
         client="9B2"/>
19   <packagedElement xmi:type="uml:Component" xmi:id="9B2" name="SRM psy191
         Application Server" clientDependency="9B29B6 9B299E"/>
20   <packagedElement xmi:type="uml:Component" xmi:id="9B6" name="SRM Data psy191
         Warehouse"/>
21   <packagedElement xmi:type="uml:Usage" xmi:id="12D29AE" supplier="9AE" psy191
         client="12D2"/>
22   <packagedElement xmi:type="uml:Component" xmi:id="12D2" name="Browser psy191
         Employee" clientDependency="12D29AE"/>
23   <packagedElement xmi:type="uml:Usage" xmi:id="12DA9AE" supplier="9AE" psy191
         client="12DA"/>
24   <packagedElement xmi:type="uml:Component" xmi:id="12DA" name="Browser psy191
         Customer" clientDependency="12DA9AE"/>
25 </uml:Model>
26
27 }
```

Listing B.1: Struktur des SRM inerhalb einer XMI-Datei

Literaturverzeichnis

[BSCS07] BOSSUNG, S., H.-W. SEHRING, H. CARL und J. W. SCHMIDT: *An Agile Process for the Creation of Conceptual Models from Content Descriptions*. In: *Proc. ADBIS*. Springer, 2007.

[BSSS07] BOSSUNG, S., H.-W. SEHRING, M. SKUSA und J. W. SCHMIDT: *Conceptual Content Management for Software Engineering Processes, Institut für Softwaresysteme (STS), Technische Universität Hamburg-Harburg (TUHH)*. Springer, 2007.

[CIS07] CIS/TU-BERLIN: *RM-ODP TU-Berlin - http://cis.cs.tu-berlin.de/Forschung/Projekte/neweconomy/lernmodule/interoperabilitaet-_rmodp/Output/html/a3f.html*. letzter Zugriff am 16. Sept. 2007.

[Cle96] CLEMENTS, P. C.: *A Survey of Architecture Description Languages, Software Engineering Institute, Carnegie Mellon University*. In: *Proc. IWSSD*. IEEE Computer Society, 1996.

[CMU07a] CMU-CARNEGIE MELLON UNIVERSITY: *AADL - http://www.aadl.info*. letzter Zugriff am 16. Sept. 2007.

[CMU07b] CMU-CARNEGIE MELLON UNIVERSITY: *UniCon - http://www.cs.cmu.edu/ UniCon*. letzter Zugriff am 16. Sept. 2007.

[CMU07c] CMU-CARNEGIE MELLON UNIVERSITY: *UniCon Reference Manual - http://www.cs.cmu.edu/ŨniCon/reference-manual/Reference_Manual_1.html*. letzter Zugriff am 16. Sept. 2007.

[CMU07d] CMU-CARNEGIE MELLON UNIVERSITY: *SEI Open Systems Glossary, Software Engineering Institute - http://www.sei.cmu.edu/opensystems/glossary.html*. letzter Zugriff am 20. Sept. 2007.

[FGH06] FEILER, P. H., D. P. GLUCH und J. J. HUDAK: *The Architecture Analysis & Design Language (AADL): An Introduction*. 2006.

[Hil00] HILLIARD, R.: *Impact Assessment of IEEE 1471 on The Open Group Architecture Framework, Integrated Systems and Internet Solutions, Inc. Concord*. 2000.

[IBM02a] IBM: *ADS Funktional Model UML1 Notation, IBM Paper*. 2002.

[IBM02b] IBM: *ADS Operational Model Notation Version 2.0, IBM Paper*. 2002.

[IBM07a] IBM: *Die Architekturmanagement-Disziplin als strategisches Führungsinstrument, 'IT-Strategie & Architektur'-Disziplinen zur Steigerung der Leistungsfähigkeit von IT-Dienstleistern, IBM-Paper*. 2007.

[IBM07b] IBM: *The History of Notes and Domino - http://www.ibm.com/developerworks/lotus/library/ls-NDHistory*. letzter Zugriff am 23. Sept. 2007.

[IEE98] IEEE: *IEEE Standard for Application and Management of the Systems Engineering Process, Institute of Electrical and Electronics Engineers*. 1998.

[IEE00] IEEE: *IEEE Recommended Practice for Architectural Description of Software Intensive Systems (IEEE Std 1471-2000), IEEE Computer Society*. 2000.

[ISO95] ISO: *Reference Model For Open Distributed Processing, Teil 1 (Overview): ITU-T Rec. X.901 / ISO/IEC 10746-1, Teil 2 (Foundations): ITU-T Rec. X.902 / ISO/IEC 10746-2, Teil 3 (Architecture): ITU-T Rec. X.903 / ISO/IEC 10746-3, Teil 4 (Architectural semantics): ITU-T Rec. X.904 / ISO/IEC 10746-4*. 1995.

[Kah02] KAHAN, E.: *Architecture Description Standard Version 2.0: Overview, IBM Paper*. 2002.

[Kru95] KRUCHTEN, P.: *The 4+1 View Model of Architecture, Paper published in IEEE Software 12 (6)*. Seiten 42–50, 1995.

[Kru04] KRUCHTEN, P.: *The Rational Unified Process An Introduction, Thirt Edition Addison-Wesley Longman, Amsterdam, Auflage: 3.* Addison-Wesley, 3. Auflage, 2004.

[KS03] KRÜGER, S. und J. SEELMANN-EGGEBERT: *IT-Architektur-Engineering - Systemkomplexität bewältigen und Kosten senken*. Galileo Computing, 2. Auflage, 2003.

[Obj07] OBJECT MANAGEMENT GROUP: *Unified Modeling Language - http://www.uml.org*. letzter Zugriff am 17. Aug. 2007.

[OHJ+01] OESTEREICH, B., P. HRUSCHKA, N. JOSUTTIS, H. KOCHER, H. KRASEMANN und M. REINHOLD: *Erfolgreich mit Objektorientierung Vorgehensmodelle und Managementpraktiken für die objektorientierte Softwareentwicklung 2.* Oldenbourg, 3. Auflage, 2001.

[oos07a] OOSE, INNOVATIVE INFORMATIK GMBH: *Historie der objektorientierten Methoden und Notationen - http://de.wikipedia.org/wiki/Bild:Oo-historie.png*. letzter Zugriff am 16. Sept. 2007.

[oos07b] OOSE, INNOVATIVE INFORMATIK GMBH: *Object Engineering Process - http://www.hst.fhso.ch/Archiv/1999/swe/SWEP99/oep/index.html*. letzter Zugriff am 16. Sept. 2007.

[OPE07] OPEN PROCESS FRAMEWORK REPOSITORY ORGANIZATION: *OPEN Process Framework (OPF) - http://www.opfro.org/index.html?Components/WorkProducts/ArchitectureSet/Architectures/-Architectures.html Contents*. letzter Zugriff am 20. Sept. 2007.

[OSKZ06] OESTEREICH, B., C. SCHRÖDER, M. KLINK und G. ZOCKOLL: *OEP - oose Engineering Process - Vorgehensleitfaden für agile Softwareprojekte*. Dpunkt Verlag, 1. Auflage, 2006.

[Pal07] PALINDROME SOLUTIONS, INC.: *Project Management and Mentorship - http://www.palindrome-solutions.com/serv03.htm*. letzter Zugriff am 16. Sept. 2007.

[Rat98] RATIONAL SOFTWARE: *Rational Unified Process, Best Practices for Software Development Teams, Rational Software Corporation White Paper*. 1998.

[SBS05] SEHRING, H.-W., S. BOSSUNG und J. W. SCHMIDT: *ACTIVE LEARNING BY PERSONALIZATION Lessons Learnt from Research in Conceptual Content Management, Institut für Softwaresysteme, Technische Universität Hamburg-Harburg (TUHH)*. 2005.

[ScS04] SCHMIDT, J. W. UND H.-W. SEHRING: *Beyond Databases: An Asset Language for Conceptual Content Management*. In: *Proc. ADBIS*. LNCS, Springer, Seiten 99-112, 2004.

[SDZ95] SHAWL, M., R. DELINE und G. ZELESNIK: *Abstractions and Implementations for Architectural Connections, Computer Science Department Carnegie Mellon University Pittsburgh*. 1995.

[Seh04] SEHRING, H.-W.: *Konzeptorientiertes Content Management: Modell, Systemarchitektur und Prototypen, 2, Dissertation, Technische Universität Hamburg-Harburg (TUHH)*. 2004.

[Ska03] SKARITS D.I., E.: *IBM Global Services Method (IBM GSM), IBM Paper*. 2003.

[SSSW07] SCHMIDT, J. W., H.-W. SEHRING, M. SKUSA und A. WIENBERG: *Subject-Oriented Work: Lessons Learned from an Interdisciplinary Content Management Project. In Advances in Databases and Information Systems*. LNCS, Seiten 3–26, 2007.

[STS07a] STS/TUHH UND WARBURG-HAUS HAMBURG: *Politische Ikonographie in Warburg Electronic Library - http://www.sts.tu-harburg.de/projects/WEL/ublick_idx.html*. letzter Zugriff am 16. Sept. 2007.

[STS07b] STS/TUHH UND WARBURG-HAUS HAMBURG: *Warburg Electronic Library - http://www.welib.de*. letzter Zugriff am 16. Sept. 2007.

[The07a] THE ECLIPSE FOUNDATION: *Open Unified Process - http://www.eclipse.org/epf/downloads/openup/openup_downloads.php*. letzter Zugriff am 16. Sept. 2007.

[The07b] THE NATIONAL CENTER FOR EDUCATION STATISTICS: *The National Center for Education Statistics glossary - http://nces.ed.gov/pubs98/tech/glossary.asp*. letzter Zugriff am 20. Sept. 2007.

[The07c] THE OPEN GROUP: *Architecture Development Cycle - http://www.opengroup.org/architecture/togaf8-doc/arch/chap03.html*. letzter Zugriff am 16. Sept. 2007.

[The07d] THE OPEN GROUP: *The Open Group Architecture Framework (TOGAF 8.1.1 'The Book'), Sep. 2006 - http://www.opengroup.org/togaf*. letzter Zugriff am 16. Sept. 2007.

[The07e] THE OPEN GROUP: *Developing Architecture Views Introduction - http://www.opengroup.org/public/arch/p4/views/vus_intro.htm*. letzter Zugriff am 20. Sept. 2007.

[The07f] THE OPEN GROUP: *The Open Group Glossary - http://www.opengroup.org/architecture/togaf7-doc/arch/p4/glossary/glossary.htm*. letzter Zugriff am 20. Sept. 2007.

[Tho00] THOMASCHEWSKI, A.: *Vorstellung Oracle Consulting, Oracle Präsentation*. 2000.

[Uni06] UNIVERSITÄT ERLANGEN NÜRNBERG: *Jahresbericht 2005 der Informatik, Arbeitsberichte des Instituts für Informatik Friedrich-Alexander-Universität Erlangen Nürnberg, Band 38, Nummer 5, Seite 137*. 2006.