Weiterentwicklung einer Ausführungsplattform für WWW-Applikationen

Diplomarbeit

Michael Rumpf

11. Januar 2000

Diplomarbeit

zur Erlangung des akademischen Grades

Diplom Informatiker

an der
Universität des Saarlandes

Fachbereich Informatik
Lehrstuhl für verteilte Systeme
Prof. Dr. Helge Scheidig

Thema

Weiterentwicklung einer Ausführungsplattform für WWW-Applikationen

Diplomand
Michael Rumpf, Güterstr. 51, 54295 Trier

Erklärung

Hiermit erkläre ich, daß ich die vorliegende Arbeit selbstständig und ohne unzulässige Hilfe angefertigt habe. Die verwendeten Literaturquellen sind im Literaturverzeichnis vollständig angegeben.
Die Diplomarbeit wurde noch keiner Prüfungsbehörde in gleicher oder in anderer Form vorgelegt.

Trier, den 11.01.2000

Michael Rumpf

Inhaltsverzeichnis

1 Einleitung **7**
1.1 Begriffe . . . 7
1.1.1 System . . . 7
1.1.2 WWW-Applikation . . . 8
1.1.3 Komponenten . . . 9
1.2 KnowledgeNow 1 . . . 10
1.3 Beispiel: CountryNet . . . 11
1.3.1 Benutzer . . . 11
1.3.2 Autor . . . 12
1.3.3 Administrator . . . 12
1.3.4 Entwickler . . . 13
1.4 Motivation . . . 13

2 Grundlagen **18**
2.1 Software Technik . . . 18
2.1.1 Software-Entwicklung . . . 19
2.1.1.1 Notationen . . . 20
2.1.1.2 Werkzeuge . . . 20
2.1.2 Software-Management . . . 20
2.1.3 Software Qualitätssicherung . . . 21
2.1.4 Software-Wartung und Pflege . . . 21
2.2 Technologien . . . 22
2.2.1 ISO-Standards . . . 22
2.2.1.1 C++ . . . 22
2.2.1.2 Standard Template Library - STL . . . 23
2.2.1.3 SGML . . . 23
2.2.1.4 SQL . . . 24
2.2.2 W3C-Standards . . . 24
2.2.2.1 HTML . . . 24
2.2.2.2 XML . . . 25
2.2.2.3 DOM . . . 28
2.2.2.4 Stylesheets . . . 29
2.2.3 IETF-Standards . . . 32

2.2.3.1 HTTP 33
2.2.3.2 CGI 33
2.2.4 OMG-Standards 33
2.2.4.1 OMA 34
2.2.4.2 CCM 38
2.2.4.3 UML 39
2.2.5 Java-Standards 40
2.2.5.1 Java 40
2.2.5.2 EJB 40
2.2.5.3 JDBC 41
2.2.5.4 Java Servlets & JSP 41
2.2.5.5 J2EE APM 41
2.2.6 Microsoft 42
2.2.6.1 ODBC 42
2.2.6.2 COM 42
2.2.6.3 MTS 43
2.2.6.4 DNA 43
2.2.7 Sonstige 43
2.2.7.1 Applikations-Server 43
2.2.7.2 PC-Docs/Fulcrum SearchServer 44

3 KN1 - Ist-Analyse 45
3.1 Projektstruktur 45
3.2 Funktionsweise 47
3.3 HTML-Templates 47
3.4 Repositories 54
3.4.1 Format-Repository 54
3.4.2 SQL-Repository 58
3.5 Datenbankanbindung 59
3.6 Index-Server 59
3.6.1 SGML-TextReader 59
3.6.2 Highlighting 59
3.7 Zeichensatzkonvertierung 60
3.8 Zusammenfassung 60

4 KN 2 - Anforderungen 62
4.1 Motivation 62
4.2 Use-Cases 62
4.2.1 Administration 62
4.2.1.1 Administrator 63
4.2.1.2 Datenlieferant 63
4.2.2 Software-Entwicklung 64
4.2.2.1 Anwendungs-Entwicklung 64
4.2.2.2 Komponenten-Entwicklung 64

4.2.2.3 Kern-Entwicklung 64
4.3 Projektmanagement 64
4.4 System-Anforderungen 65
4.4.1 Erweiterbarkeit 65
4.4.2 Wiederverwendbarkeit 65
4.4.3 Offenheit 65

5 KN 2 - Architektur 66
5.1 KN2 Application Programming Model 66
5.1.1 Klienten 67
5.1.2 Web-Server 68
5.1.3 Komponenten-Laufzeitumgebung 69
5.2 Komponenten 72
5.2.1 Document Object Model 75
5.2.2 Search Network 75
5.2.3 Fulcrum Repository 84

6 KN 2 - Implementation 87
6.1 Klienten 87
6.2 Web-Server 87
6.2.1 HTTP-Server 87
6.2.2 Servlet-Engine 88
6.2.3 JSP-Engine 88
6.2.4 XSL 89
6.3 Laufzeitumgebung 89
6.3.1 VisiBroker 91
6.4 Komponenten-Bibliothek 93
6.4.1 DOM 95
6.4.1.1 Expat 96
6.4.2 SearchNetwork 98
6.4.2.1 ACE 98
6.4.3 Fulcrum Repository 100
6.5 Anwendungsbeispiel 101
6.6 Produkt-Struktur 107
6.7 Fazit 107

7 Middleware 109
7.1 Definitionen 109
7.2 Klient-Server 110
7.3 Eigenschaften 111
7.4 Bewertung 112
7.4.1 DOM 113
7.4.2 SearchNetwork 113
7.4.3 Fulcrum 113

7.5 Fazit 113

8 Zusammenfassung 115

9 Ausblick 117
9.1 Trends 117
9.2 Verbesserungsvorschläge 119
9.3 Empfehlung 119

A IDL 121

B Soft- und Hardware 133
B.1 Software 133
B.1.1 Windows 133
B.1.2 Solaris 134
B.1.3 General 134
B.2 Hardware 135
B.2.1 Windows 135
B.2.2 Solaris 135

Literaturverzeichnis 136

Tabellenverzeichnis 143

Abbildungsverzeichnis 144

Kapitel 1

Einleitung

In diesem Kapitel wird das Ziel der Diplomarbeit erläutert. Dabei werden zuerst Begriffe wie System, WWW-Applikation und Komponente erklärt, da sie im weiteren Verlauf von essentieller Bedeutung sind. Anschließend wird anhand eines Beispiels die typische Art von Anwendungen dargestellt, die mit dem zu analysierenden System bisher erstellt wurden. Abschließend werden die Hintergründe dieses Projektes aufgeführt.

Die Betreuung der Arbeit teilte sich wie folgt auf. Die Firma AASaM stellte die technische Ausrüstung in Form von Hard- und Software zur Verfügung (s. Anhang B). Die Universität betreute die CORBA-Aspekte und achtete auf die ingenieurmäßige Durchführung der Arbeit.

1.1 Begriffe

1.1.1 System

Der Begriff des Systems hat nach [Bal96] folgende Bedeutung: „Ein System ist ein Ausschnitt aus der realen oder gedanklichen Welt, bestehend aus Gegenständen (z.B. Menschen, Materialien, Maschinen und Produkten) und den darauf aufbauenden Strukturen“. Systemkomponenten oder Subsysteme sind Teile eines Systems, die untereinander in Beziehung stehen. Ein Systemelement ist eine nicht weiter zerlegbare Systemkomponente bzw. ein Subsystem.

Ein Software-System ist also eine Software, die aus Systemelementen besteht. Systemsoftware dagegen bezeichnet Software, die für den Betrieb und die Wartung einer speziellen Hardware entwickelt wurde. Der Begriff Computersystem oder DV-System bezeichnet die Kombination von Anwendungssoftware, Systemsoftware und Hardware. Ein technisches System ist die Kombination von Computersystemen und anderen technischen Einrichtungen. Das organisatorische System ist die Gesamtheit der Benutzer und Anwender, die sowohl mit dem technischen System, als auch untereinander

Rollenbeziehungen eingehen. In diesem Kontext definiert [Bal96] den Begriff Informationssystem als ein Computersystem mit sonstigen technischen Einrichtungen.

Der Begriff Auskunftssystem oder Kiosk-System wird hier als Bezeichnung für eine ganz spezielle Form von Informationssystemen verwendet. Diese zeichnen sich dadurch aus, daß der Benutzer nur Daten abfragen kann. Er ist nicht in der Lage Daten mit Hilfe von Formularen in die Anwendung einzubringen und so die Datenbasis zu erweitern.

1.1.2 WWW-Applikation

Der Begriff „WWW-Applikation“ bezeichnet die Klasse von Applikationen, deren generalisierbare Anwendungslogik auf einem WWW-Server liegt und das WWW als standardisiertes Benutzerinterface verwendet. Diese Art der Partitionierung einer Anwendung ist vergleichbar mit Terminal/Mainframe-Anwendungen, wobei das WWW, im Gegensatz zum Terminal, benutzerfreundlicher und leistungsfähiger ist. Das WWW ist ein klassisches Hypermedia- bzw. Hypertext-System. Hypermedia ist ein Begriff für Hypertext, der nicht auf reinen Text begrenzt ist, sondern auch Bilder, Video und Musik enthalten kann. Häufig werden diese beiden Begriffe aber als Synonyme verwendet.

Das WWW war anfänglich auf solche Hypertext-Systeme beschränkt. Die ursprüngliche Idee des WWW war es, wissenschaftliche Dokumente über ein Netzwerk verfügbar zu machen. Erst später wurde dieses System um Multimedia-Aspekte wie Grafik erweitert. Die ersten WWW-Anwendungen waren reine Auskunftssysteme. Der Benutzer hatte nur sehr wenige Möglichkeiten zur Interaktion.

Heute beschränkt sich das WWW jedoch nicht auf solche typischen Auskunftssysteme, sondern es etabliert sich mehr und mehr als plattformunabhängiges *grafisches Benutzerinterface (GUI)* und verdrängt in vielen Bereichen die diversen plattformspezifischen GUIs. Lediglich Anwendungen mit besonderem Visualisierungsaufwand, wie z.B. CAD- oder DTP-Anwendungen sind scheinbar noch nicht von dieser Entwicklung betroffen.

Als Beispiel können hier die neueren Microsoft Betriebssysteme erwähnt werden. Dort wird der WWW-Browser mittlerweile zu allen möglichen Visualisierungsaufgaben verwendet. Das führt dazu, daß ein Benutzer sich leichter in neuen Applikationen orientieren kann, da er mit der Oberfläche bereits vertraut ist.

Dieses Prinzip des Ease-Of-Use wurde schon bei der Einführung von grafischen Oberflächen im Allgemeinen angegeben. Mit dem WWW erfährt es eine logische Fortsetzung auf beliebige Betriebssysteme.

Eine weitere Tendenz in diesem Bereich ist das Verschwinden sog. Plattform-unabhängiger, grafischer Schnittstellen-Bibliotheken. Diese wurden im klassischen Client-Server-Umfeld eingesetzt, wenn die Klienten-Anwendung

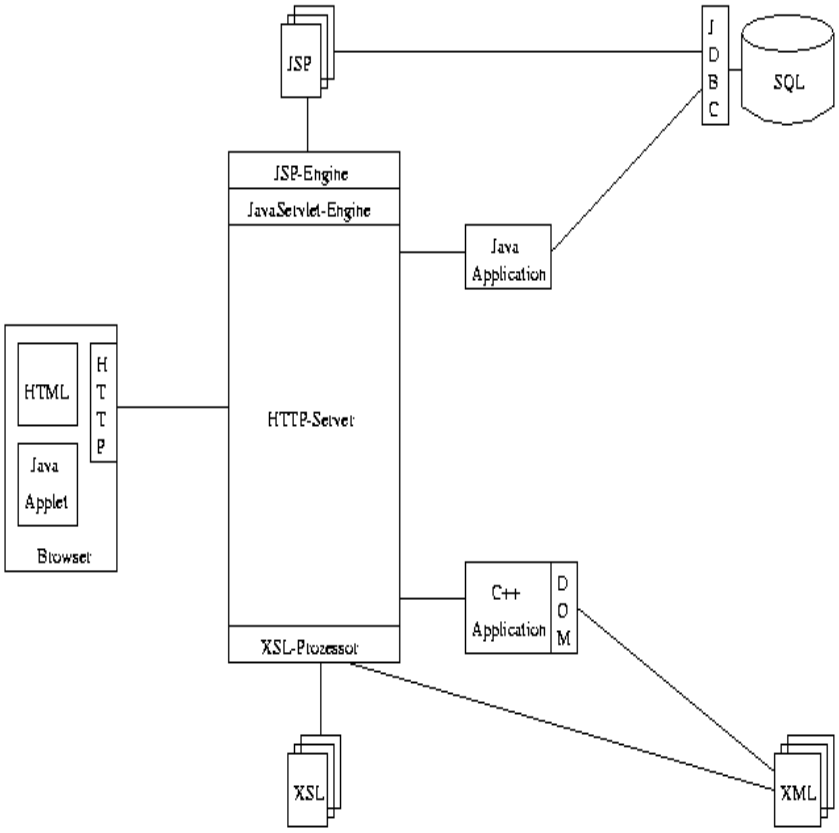

Abbildung 1.1: Beispielarchitektur einer WWW-Applikation

in einer heterogenen Betriebssystem-Landschaft eingesetzt werden sollte.

Abbildung 1.1 zeigt ein Architekturbeispiel einer WWW-Applikation. Die verschiedenen Technologien werden zum besseren Verständnis im Zusammenhang gezeigt. Eine Beschreibung der Technologien bietet Kapitel 2.

1.1.3 Komponenten

Der Begriff Komponente wird in dieser Arbeit in zwei verschiedenen Bedeutungen verwendet.

Eine Komponente wird im Duden als Bestandteil eines Ganzen bezeichnet. In diesem Fall jedoch ist es eine funktionale Einheit, die sich vom Gesamtsystem als getrennt betrachten läßt. Diese „fachlichen Komponenten“ [Skr99] werden in der Analyse und der Architekturbeschreibung einfach als Komponenten bezeichnet.

In der UML wird der Begriff der Komponente erst dann verwendet,

wenn die Implementierungssicht modelliert wird [BRJ99]. Dort haben Komponenten die Bedeutung von Implementierungskomponenten, meist in der Form von ausführbaren Dateien oder Bibliotheken. Wird der Begriff Komponente mit dieser Bedeutung verwendet, dann wird sie hier durch den Begriff „UML-Komponente“ oder „Implementierungs-Komponente“ als solche kenntlich gemacht.

1.2 KnowledgeNow 1

KnowledgeNow 1 (KN1) ist eine Ausführungsplattform für WWW-Applikationen und wurde von der Firma Software and Methods (SaM) S.A., Luxemburg, von 1995 bis 1997 entwickelt. Im Sommer 1997 wurde die Firma SaM von Arthur Andersen Luxemburg übernommen und die Abteilung *ArthurAndersen Software and Methods (AASaM)* gegründet, in der seitdem die Entwicklung an KnowledgeNow weitergeführt wird.

Die Applikationslogik dieser WWW-Anwendungen wird über das *Common Gateway Interface (CGI)* (s. Abschn. 2.2.3.2) mit einem WWW-Server verbunden. Zusätzlich gibt es eine Reihe von funktionalen Modulen, die alle in NetDiff, der „Container-Applikation“, zusammengebunden sind. Der Begriff „Container-Applikation“ spiegelt die Tatsache wider, daß mehrere funktional nicht zusammenhängende Module in dieser Applikation zusammengefaßt sind.

Bei NetDiff handelt es sich um eine, aus den Teilen „Netzwerk“ und „Diffusion“ zusammengesetzte, Bezeichnung. Das Wort Diffusion wurde im Sinne von Durchdringung verwendet und deutet daraufhin, daß während einer Suche das Netzwerk „durchdrungen“ werden muß, um an die gewünschten Informationen zu gelangen. Dies spielt auf die Funktionalität der Volltext-Suche an, welche eine der wichtigsten funktionalen Module von KN1 darstellt. Das Wort Diffusion trifft aber nicht den Kern dieser Suchfunktion, da eine Diffusion im allgemeinen einen passiven Prozeß bezeichnet[1]. Die Suche, wie sie in NetDiff implementiert ist, stellt jedoch einen aktiven Prozeß dar. Außerdem beschreibt der Begriff NetDiff nur eine der Funktionaliäten. Weiterhin existieren noch die folgenden Module: Datenbankzugriff, Index-Server Zugriff (s. Abschn. 2.2.7.2), SGML-Light Parser (s. Abschn. 2.2.1.3) und Interpreter für die KnowledgeNow Skriptsprache. Der Zusatz Light in der Bezeichnung des Parsers deutet lediglich daraufhin, daß nicht alle SGML-Eigenschaften unterstützt werden. Die Skriptsprache hat die Aufgabe, die verschiedenen funktionalen Module miteinander zu verbinden und dadurch die Applikationslogik bereitszustellen. Der Interpreter ist Teil der Laufzeitumgebung für die Skriptsprache.

Zusatzmodule, wie der SGML-TextReader (s. Abschn. 3.6.1), eine Bibliothek zur SGML-Erweiterung eines Index-Servers, ein ActiveX-Kontroll-

[1] DUDEN der deutschen Sprache: Durchdringung, Zerstreuung

element (grafisches Interface zur entfernten Administration der Applikation) und die Container-Applikation NetDiff selbst, werden unter der übergeordneten Bezeichnung KnowledgeNow 1 zusammengefaßt.

1.3 Beispiel: CountryNet

Dieser Abschnitt beschreibt ein Anwendungsbeispiel, welches auf Basis von KN1 implementiert wurde. Dieses Beispiel soll dem Leser einen konkreten Eindruck verschaffen, welche Art von Anwendungen mit KN1 entwickelt wurden. Es dient nicht dazu, die Probleme der Architektur darzustellen, denn auf Basis des hier zu entwerfenden Nachfolgesystems könnte sicherlich eine identische Applikation entwickelt werden. Dem Benutzer wäre es aufgrund der Funktionalität unmöglich festzustellen, ob die Anwendnung auf KN1 oder dem Nachfolger basiert.

Es handelt sich bei CountryNet um das einzige, unter Produktionsbedingungen laufende System auf Basis von KN1. Es gibt zwar noch weitere Anwendungen, die mit KN1 implementiert wurden, hierbei handelt es sich aber meistens um Prototypen, die nur zur Demonstration für einen bestimmten Kunden entwickelt wurden.

Im weiteren Verlauf der Arbeit wird immer wieder auf dieses Beispiel Bezug genommen.

CountryNet ist ein WWW-Auskunftssystem für Auswanderer und Geschäftsreisende, die einen Umzug bzw. eine Reise in ein anderes Land vorbereiten. Insgesamt stehen Informationen zu 84 Ländern zur Verfügung. Es handelt sich dabei um einen kostenpflichtigen Dienst, dessen inhaltlicher Teil in Zusammenarbeit mit Arthur Andersen (AA) Chicago, Craighead Inc. und The Economist Intelligence Unit (EIU) verwaltet wird. Die Fachabteilungen von AA Chicago tragen Informationen zu den Teilgebieten Steuern, Einwanderung und Allgemeines zusammen. Craighead Inc. ist spezialisiert auf Informationen und Hilfestellungen rund um die Verlagerung des Standortes und EIU steuert Wissen im Bereich Ökonomie und Politik bei (s. Abb. 1.2).

In den folgenden Abschnitten wird die Anwendung aus Sicht der unterschiedlichen Benutzer beschrieben.

1.3.1 Benutzer

Der Anwender hat nach der Anmeldung (s. Abb. 1.2) und der Auswahl eines Landes (s. Abb. 1.3) die Wahl zwischen den Themengruppen: Umsiedelung, Ökonomie und Politik, Steuern, Einwanderung und Allgemeines. Diese Oberthemen sind wiederum in Unterthemen aufgeteilt, die im linken Teil des Browsers angezeigt werden. Wählt der Benutzer eines dieser Unterthemen aus, dann wird der entsprechende Textbaustein aus einem SGML-Dokument extrahiert und im Hauptfenster, auf der rechten Seite, angezeigt (s. Abb. 1.4).

Für die Suche über alle SGML-Dokumente existiert ein eigenes Formular, das über den Link „Search“, im unteren Teil der Seite, erreicht werden kann (s. Abb. 1.5). Gibt der Anwender dort einen oder mehrere Suchbegriffe ein, so werden diese in einer Anfrage an den Index-Server weitergeleitet und die Fundstellen in Form einer Ergebnisseite angezeigt (s. Abb. 1.6). Der Benutzer erhält zu jeder Fundstelle den Titel und den Untertitel des Dokuments angezeigt.

Hinter den Links „Profiles“ bzw. „Create/Select Profile“ verbirgt sich die Möglichkeit für den Benutzer, eine Vorkonfiguration der Einstiegsseite vorzunehmen. Diese Funktionalität beschränkt sich darauf, einen Ursprung und ein Ziel vorzudefinieren, so daß die lästige Auswahl der Länder in den beiden Listenfeldern entfällt.

Der Verweis „Links“ verzweigt zu einer Seite, die eine statische Liste verwandter Informationen bereithält.

Der Verweis „Hilfe“ gibt allgemeine Hinweise zur Benutzung der WWW-Anwendung und spezielle Hinweise zur Feinabstimmung der Suchfunktion.

1.3.2 Autor

Die SGML-Dokumente werden von ca. 60 Autoren mit Hilfe einer SGML-Erweiterung für MicrosoftWord[2] geschrieben, anschließend vom Administrator gesammelt, um anschließend mit diesen Texten eine Neuindizierung durchzuführen. Der SGML-Editor gibt für den Autor ein bestimmtes Format vor, die *Document Type Definition (DTD)* (s. Abschn. 2.2.1.3). Der Autor bekommt vom Editor automatisch eine Struktur vorgegeben und muß diese dann mit Inhalt füllen. Dies garantiert, daß das Dokument ohne Adaptionen vom Index-Server (s. Abschn. 2.2.7.2) verarbeitet werden kann.

1.3.3 Administrator

Der Administrator hat als einziger Zugang zur Administrationsschnittstelle der Anwendung (s. Abb. 1.7). Dort kann er den Status aller Dokumente (Schalter: SGML Documents) im System herausfinden und den Indizierungsvorgang starten (Schalter: Update All). Außerdem bietet das System an, die statischen Dokumente einzubringen, die nicht in die Volltext-Suche integriert werden. So z.B. die allgemeinen Nachrichten, die Seite mit den Links bzw. die Hilfe-Seite (Schalter: Other Resources). Ein weiterer Verweis (Schalter: Log File) verzweigt zur Statusdatei des Indizierungsvorgangs. Dort ist für jedes SGML-Dokument aufgeführt, ob die Indizierung erfolgreich war.

Das Hochladen der SGML-Dokumente auf den Server, auf dem die Anwendung liegt, erfolgt mit Hilfe eines gewöhnlichen FTP-Programms.

Der Indizierungsvorgang wird vom Administrator mit Hilfe einer HTML-Schnittstelle zu einem CGI-Skript initiiert. Der Vorgang dauert, bei einer

[2] SGMLAuthor AddOn for MS-Word

Anzahl von 418 Dateien und einem Volumen von insgesamt 21 MB, etwa 25 Minuten.

1.3.4 Entwickler

Obwohl in Marketingdokumenten KN1 unter anderem als Entwicklungsumgebung propagiert wird, existieren für den Entwickler einer WWW-Applikation in KN1 keine entsprechenden Werkzeuge. Er kann die HTML-Seiten zwar mit einem HTML-Editor erstellen, allerdings ist dem Editor die Skriptsprache unbekannt, da es sich um proprietäre Funktionen handelt.

Es gibt in KN1 keine Unterstützng für die Fehlersuche. Der Entwickler ist gezwungen auf einfache Methoden, wie z.B. Fehlerinformationen in die HTML-Seite auszugeben, zurückzugreifen.

Der Zugriff auf den Index-Server erfolgt in einem proprietären SQL-Dialekt (s. Abschn. 2.2.1.4) über ODBC (s. Abschn. 2.2.6.1). Zur Fehlersuche auf dieser Ebene läßt sich das unter Microsoft-Windows verfübgbare ODBC-Tracing verwenden. Dieses wird über den ODBC-Manager aktivitert und protokolliert jeden ODBC-API Aufruf sowie das entsprechende Ergebnis in einer Text-Datei.

Der *SearchServer* Index-Server der Firma PC-Docs/Fulcrum (s. Abschn. 2.2.7.2) stellt die Funktionalität der Volltext-Suche zur Verfügung.

Der Index-Server arbeitet nach einem relationalen Modell, wobei in einer Zuordungsvorschrift definiert wird, welche Teile eines Dokuments welchem Attribut zugeordnet werden. Das Erstellen dieser Vorschrift fällt in den Aufgabenbereich des Entwicklers.

Das Dokumentformat SGML wird von diesem Index-Server nicht direkt unterstützt, deshalb kommt ein sogenannter SGML-TextReader zum Einsatz, der als Bestandteil von KN1 entwickelt wurde. Dieser wandelt die SGML-Dokumente automatisch in das Format des Index-Servers um.

1.4 Motivation

In der Software-Entwicklung ist ein Trend hin zu WWW-Applikationen zu verzeichnen. Dieser Trend begründet sich zum einen in der Verbreitung des Browsers als standardisierte Darstellungsplattform. Zum anderen bildet das Internet, bezogen auf Netzwerk-Aspekte, eine einheitliche technische Basis. Dies macht es für Firmen interessant verteilte Applikationen anzubieten, da sie sich z.B. auf TCP/IP als gemeinsames Protokoll verlassen können. Allgemein kann gesagt werden, daß die Internettechnologien sich mittlerweile zum Standard in vielen Unternehmen entwickelt haben. Der Einsatz der Technologie im Internet liefert den besten Beweis dafür, daß sie für sehr große Installationen geeignet ist.

Bei Betrachtung von KN1 fällt auf, daß die technologische Unterlage veraltet ist und mit den aktuellen Anforderungen im Internet nicht mehr

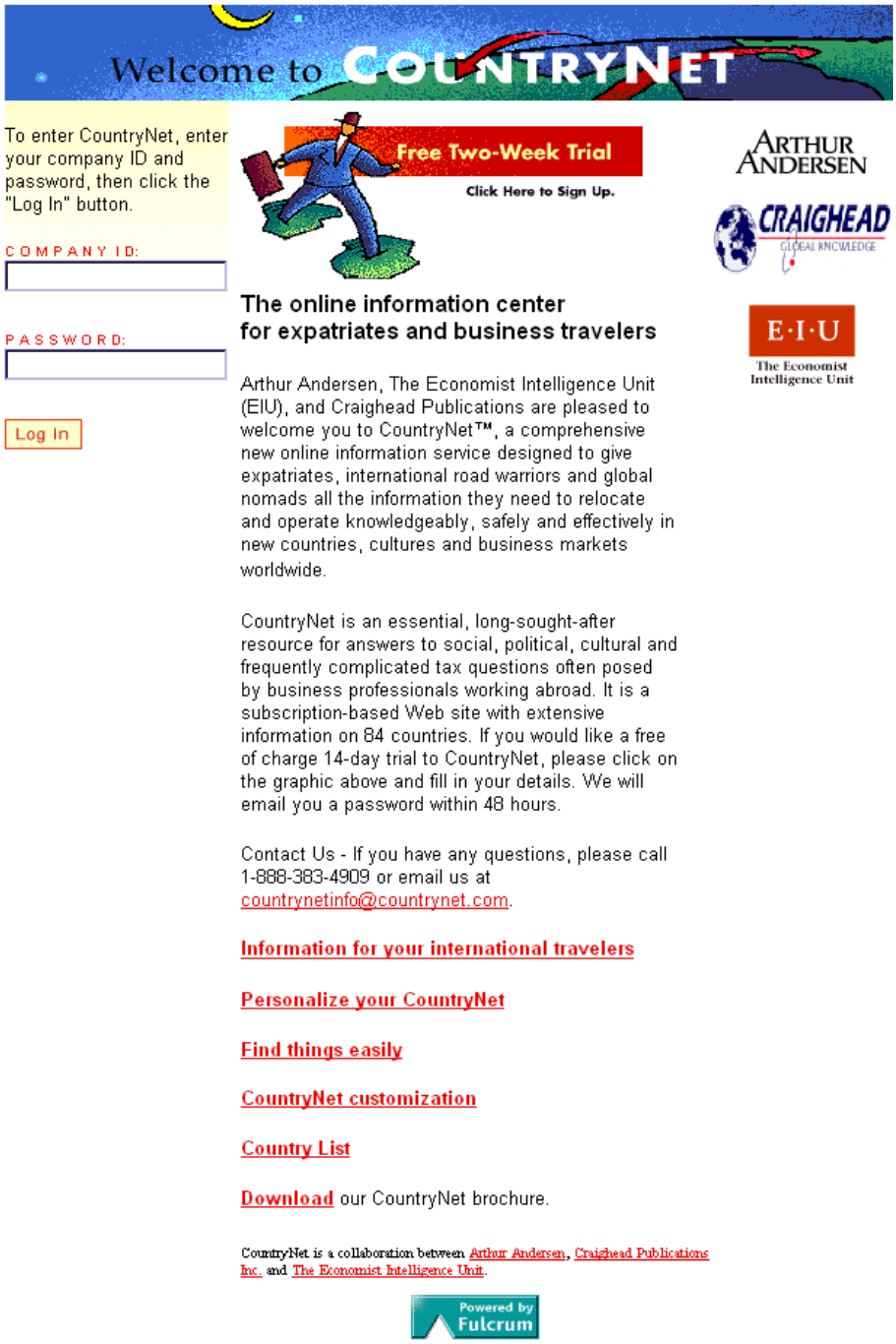

Abbildung 1.2: CountryNet Homepage

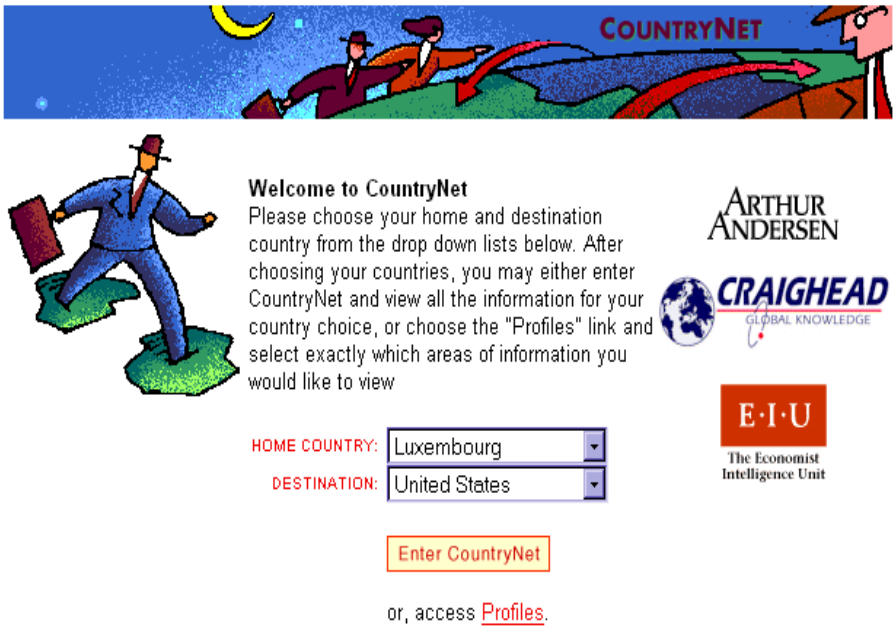

Abbildung 1.3: CountryNet Länderauswahl

Abbildung 1.4: CountryNet Navigationsseite (Relocation/Key Facts)

Search by Keyword

Search for:
Housing

Location:
All documents
Destination country: United States of America

Search

CountryNet employs a very powerful search engine. You may search for terms within the selected country, or across the entire set of countries. The engine will support simple keyword searches; however, the tips below will yield much more efficient and productive search results.

When searching for:	Type:
A specific Word	The Word
A specific Phrase	"The Phrase" in quotes
A set of words that are close in proximity	[The Words] in square brackets
A specific word or a synonym of the word	Word+ The word followed by a + sign

Abbildung 1.5: CountryNet Anfrageseite

Search Result

Your query was : Housing
Query Result: 203 references

India (IN)	Country General Information - India Finding a home
United Kingdom (GB)	Country General Information - the United Kingdom Living in the United Kingdom - 1
Australia (AU)	Country General Information - Australia Living in Australia - 1
Japan (JP)	Country General Information - Japan Finding a home
Chile (CL)	Country General Information - Chile Finding a home
Indonesia (ID)	Country General Information - Indonesia Finding a home

Next

Abbildung 1.6: CountryNet Suchergebnisse

Abbildung 1.7: CountryNet Administration

konkurrieren kann. Die Zielsetzung dieser Arbeit lautet deshalb:
Ersetzung der veralteten Technologien und Verbesserung der Architektur, um schneller und damit flexibler auf neue Technologien reagieren zu können.

Dazu wird in Kapitel 2 zunächst eine Übersicht über die in dieser Arbeit erwähnten Technologien gegeben. Jeder Abschnitt enthält dabei Verweise auf Quellen, um tiefer in diese Materie einzusteigen. Anschließend wird mit der Schwachstellen-Analyse des bestehenden Systems begonnen. Aus dieser Analyse lassen sich dann die Anforderungen an das Nachfolgesystem KN2 definieren. Auf Basis dieser Anforderungen wird im Kapitel 5 eine Architektur vorgestellt. Die Umsetzung dieser Architektur wird in Kapitel 6 beschrieben. Im Kapitel 7 wird auf den Begriff Middleware eingegangen und KN2 zu diesem in Beziehung gesetzt. Den Abschluß bildet eine Zusammenfassung und ein Ausblick. Im Anhang ist die Schnittstellendefintion von KN2 als IDL-Quelltext und eine Liste der verwendeten Hard- und Software abgedruckt.

Kapitel 2

Grundlagen

In diesem Kapitel werden die Software-Technik Aspekte des Projekts und die für das Projekt relevanten Technologien vorgestellt.

2.1 Software Technik

Der Begriff Software-Technik wird als Synonym zum englischen Ausdruck Software-Engineering verwendet. In [Bal96] findet sich folgende Definition:

Zielorientierte Bereitstellung und systematische Verwendung von Prinzipien, Methoden, Konzepten, Notationen und Werkzeugen für die arbeitsteilige, ingenieurmäßige Entwicklung und Anwendung von umfangreichen Software-Systemen.

Der Begriff zielorientiert bezeichnet in dieser Definition die Berücksichtigung von Ressourcen, wie Zeit, Kosten und Qualität. Hinter dem Begriff Kosten verbergen sich mehrere Einflußfaktoren, wie z.B. Anzahl der Mitarbeiter, benötigte Lizenzen für die Werkzeuge und Schulungen auf den Werkzeugen.

Prinzipien sind Grundsätze des Handelns. Methoden sind planmäßige Verfahren zur Erreichung eines Ziels. Unter dem Begriff Konzepte, oder Entwürfe, versteht man verschiedene Herangehensweisen an einen bestimmten Sachverhalt. Notationen sind Darstellungsformen eines Sachverhalts und Werkzeuge sind Hilfsmittel zur Unterstützung von Prinzipien, Methoden, Konzepten und Notationen.

Das Adjektiv arbeitsteilig macht in der Definition eine Aussage darüber, wie der Arbeitsvorgang in Teile zerlegt wird, um den Vorgang der Software-Entwicklung überschaubar zu machen. Es handelt sich um eine klassische Top-Down Methode, wobei in diesem Fall aber nicht das zu lösende Problem in Teilprobleme, sondern der Vorgang an sich partitioniert wird. Ingenieurmäßig bedeutet, daß es sich um einen rationalen Vorgang, ohne künstlerische Einflüsse handelt.

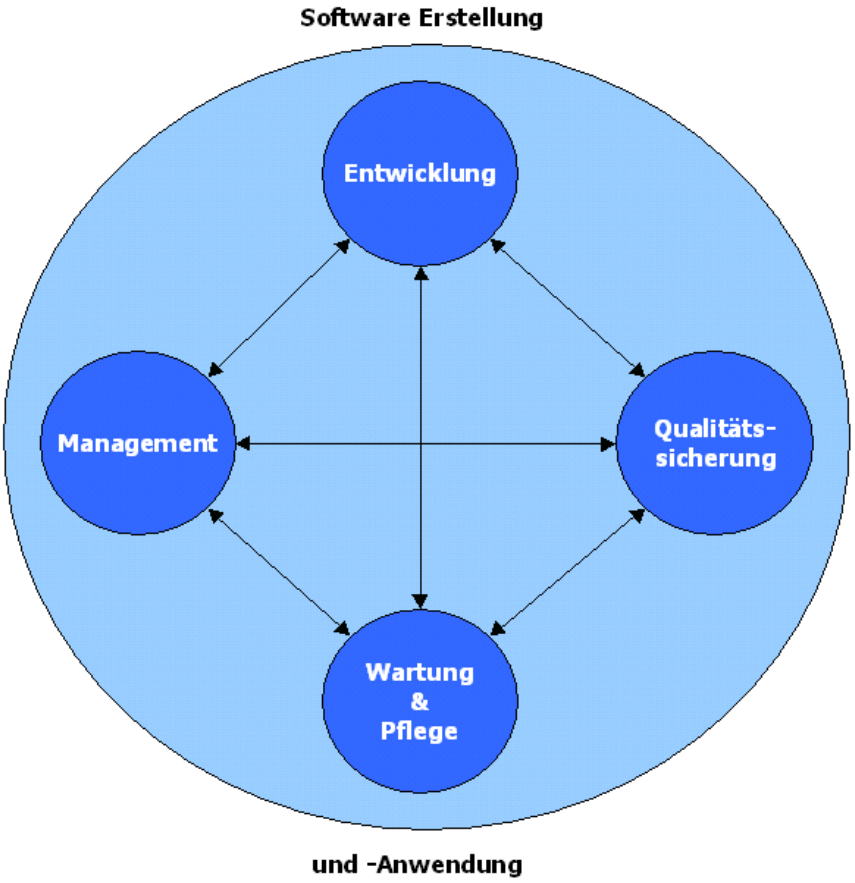

Abbildung 2.1: Teildisziplinen der Software Technik

In Abbildung 2.1 sind die Teildisziplinen der Software-Technik aufgeschlüsselt. Die Ausrichtung von oben nach unten zeigt die Nähe der Teildisziplin entweder zur Erstellung oder zur Anwendung der Software.

2.1.1 Software-Entwicklung

Der Begriff Software-Entwicklung beinhaltet die Schritte zur Durchführung des Entwicklungsprozesses. Diese umfassen Planung (Planning), Analyse (Analysis), Architektur (Architecture), Entwurf (Design), Implementation, Integration und Test. Software-Entwicklung ist ein sehr komplexer Prozeß, der von vielen unterschiedlichen Faktoren beeinflußt wird. Um diese Komplexität zu beherrschen, werden strukturierte Software-Entwicklungsmethoden verwendet. Diese strukturierten Methoden schreiben unter anderem die Notationen vor, mit deren Hilfe die Entwickler über die einzelnen Schritte dis-

kutieren, als auch die Werkzeuge, die sie bei ihrer Tätigkeit unterstützen.

2.1.1.1 Notationen

Als Notation wird in dieser Arbeit die *Unified Modeling Language (UML)* (s.a. 2.2.4.3) eingesetzt. Diese Sprache stellt die Vereinigung der populärsten Notationen der letzten Jahre dar und ist seit 1997 ein offiziell anerkannter OMG-Standard.

2.1.1.2 Werkzeuge

Als UML-Modellierungswerkzeug wird Rational Rose 98i Enterprise Edition von der Firma Rational eingesetzt. Dieses bietet in der Enterprise Edition die Möglichkeit zum CORBA-IDL Roundtrip-Engineering. Außerdem besitzt die Firma Arthur Andersen mit der Firma Rational einen Kooperationsvertrag. Das Programm ist das offizielle Standard Modellierungswerkzeug von Arthur Andersen weltweit.

2.1.2 Software-Management

Das Software-Management bildet den übergeordneten Rahmen, der die einzelnen Teildisziplinen der Software-Technik integriert.

Ein Prozeß- oder Vorgehensmodell gibt Auskunft darüber, wie die einzelnen Schritte der Software-Entwicklung zueinander in Beziehung stehen, wo die Qualitätssicherung in diesen Schritten ansetzt und welche Wartungsaufgaben durchgeführt werden müssen, um diese Qualitätssicherungsaspekte zu garantieren.

Da das Projekt in einer inkrementellen Art und Weise durchgeführt wird, eignet sich natürlich ein inkrementelles Prozeßmodell am besten, um dieses Vorgehen in geordnete Bahnen zu leiten.

Der *Rational Unified Process (RUP)* ist ein solches inkrementelles Prozeßmodell. Der Vorteil des RUP ist, daß er sich an fast beliebige Projektgrößen anpassen läßt. Andere Prozeßmodelle, wie z.B. das V-Modell, fordern detaillierte Dokumente nach jedem Schritt. Diese Forderung führt zu einer Bürokratisierung des gesamten Projektes. Dies kann bei sehr großen Projekten durchaus von Vorteil sein. Bei kleineren Projekten ist dies eher hinderlich, da die Entwickler in einem solchen Fall mehr mit der Erstellung der Dokumentation beschäfigt sind, als mit der Entwicklung selbst. Außerdem ist das V-Modell ein sehr statisches Modell, da der Übergang von einer Stufe auf die nächste nur dann möglich ist, wenn die vorherige vollständig abgeschlossen ist. Es ist also bei einem solchen Modell sehr schwierig, den Prozeß an neue oder geänderte Anforderungen anzupassen.

Der RUP teilt sich in die Phasen Anfang (Inception), Ausarbeitung (Elaboration), Konstruktion (Construction) und Übergang (Transition). Diese

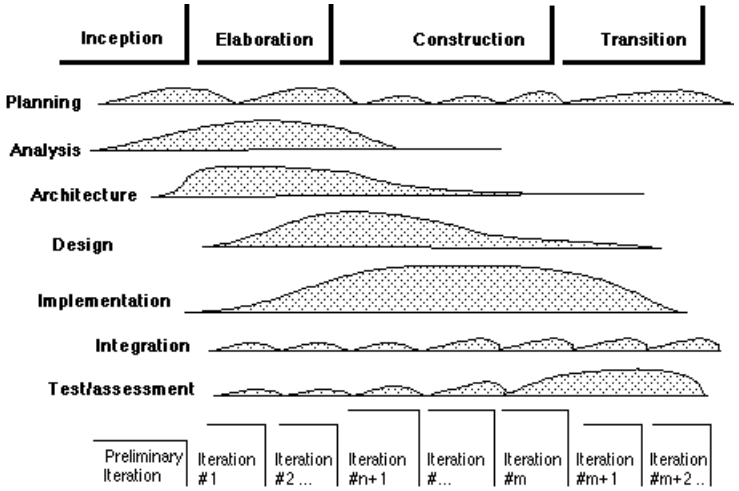

Abbildung 2.2: Der Rational Unified Process (RUP)

Schritte haben unterschiedliche Gewichtung in den einzelnen Phasen (s. Abb. 2.2). Eine umfassende Einführung in den RUP gibt [Kru99].

2.1.3 Software Qualitätssicherung

Die Qualitätssicherung beschränkt sich nicht nur auf die Bewertung des endgültigen Softwaresystems, sondern sie ist eng mit dem Prozeßmanagement verbunden. Die Qualität einer Software wird unter anderem von der Dokumentation bestimmt. Ein Fehlen wirkt sich somit negativ auf die Qualität der endgültigen Software aus. Eine Änderung ohne Dokumentation nimmt mehr Zeit in Anspruch, als eine Änderung bei der man auf eine umfassende Dokumentation zurückgreifen kann. Die Qualitätssicherung gewährleistet unter anderem, daß im Laufe des gesamten Prozesses die Resultate aus den festgelegten Meilensteinen, unter Erfüllung der Qualitätskriterien, eingehalten werden.

2.1.4 Software-Wartung und Pflege

Die Software-Wartung und Pflege adressiert die Schritte, die nach der erfolgreichen Auslieferung an den Kunden durchgeführt werden.

Ein Software-System weist ein hohes Maß an Dynamik bzgl. der Anforderungen seitens der Anwender auf. Deshalb muß ein Software-System nach der Auslieferung immer wieder an die Bedürfnisse der Anwender angepaßt werden. Es können aber auch Fehler auftreten, die in den Tests nicht lokalisiert werden konnten. Im einfachsten Fall sind dies Implementationsfehler. Im schlimmsten Fall handelt es sich um Designfehler. Bei Fehlern der letzten

Kategorie sind oft grundlegende Änderungen an der Architektur notwendig. Generell gilt die Regel, je später ein Fehler erkannt wird, um so teurer ist es ihn zu beheben.

Alle diese nachträglichen Adaptionen und Fehlerkorrekturen sind unter dem Begriff Wartung und Pflege zusammengefaßt.

2.2 Technologien

Im diesem Abschnitt erfolgt eine kurze Einführung in die Standards und Technologien, des Projektes. Zusätzlich erfolgt ein Überblick über den Prozeß unter dem die beschriebenen Standards entwickelt wurden. Dieses Kapitel hat unter anderem die Aufgabe eines Glossars. Ich habe versucht alle Technologien abzuhandeln, die in der Arbeit von Bedeutung sind.

2.2.1 ISO-Standards

Die *International Standardization Organization (ISO)* ist ein internationales Normierungsgremium mit Sitz in der Schweiz.

Bei den Mitgliedern aus 130 Ländern handelt es sich um nationale Gremien, welche in Abhängigkeit von der wirtschaftlichen Stärke ihres Landes ein Stimmrecht besitzen, um an Abstimmungen teilnehmen zu können.

Der Standardisierungsprozeß läuft dabei wie folgt ab: Eine Anfrage nach Standardisierung eines bestimmten Sachverhalts kann von beliebiger Stelle an die ISO herangetragen werden. Wird ihr stattgegeben, übergibt sie den Sachverhalt an das betreffende technische Komitee. Diese Komitees setzen sich aus Mitgliedern der ISO und damit aus Vertretern der nationalen Inistitutionen zusammen. Die Komitees erstellen ein *Working-Draft*, das anschließend von allen ISO-Mitgliedern und anderen Interessenten kommentiert werden kann. Auf Basis dieser Kommentare wird das Dokument zu einem *Standard-Draft* verfeinert, um schließlich in einer Abstimmung in einen vollwertigen ISO-Standard übernommen zu werden [ISO].

2.2.1.1 C++

Die C++ Programmiersprache ist im ISO-Standard ISO/IEC 14882-1998 [ISO98] definiert.

Der Standard umfaßt mittlerweile nicht nur die reine Programmiersprache, sondern er definiert auch eine *Standard Bibliothek* [Str99]. Diese Bibliothek besteht einerseits aus allgemeinen C-Funktionen, wie z.B. strcpy() und fopen(), und der *Standard Template Library (STL)*.

2.2.1.2 Standard Template Library - STL

Die *Standard Template Library (STL)* basiert auf der Arbeit von Alexander Stepanov und Meng Lee [SL95].

Ziel war es eine allgemeine C++-Bibliothek von Container-Klassen und Algorithmen zu erstellen. Dabei wurde besonderen Wert auf Effizienz, einfache Handhabung und eine fundierte theoretische Basis gelegt. Im Jahre 1994 beschloß das ISO-Komitee zur Standardisierung von C++ die STL als festen Bestandteil in die C++-Standard-Bibliothek aufzunehmen.

Die Container-Klassen der STL sind: list, vector, dequeue, queue, set multiset, map, multimap, stack und priority queue. Die Algorithmen der Bibliothek umfassen die Aufgaben Suchen, Sortieren, Verknüpfen, Kopieren und Tranformieren.

Das Problem der STL Implementationen, die bei fast jedem Compiler mitgeliefert werden, ist die mangelnde Konformität zum Standard bzw. die fehlerhafte Implementation. Diese Tatsache macht eine Verwendung in einem Multiplattform-Projekt, in dem verschiedene Compiler eingesetzt werden, sehr schwierig. Aushelfen kann hier eine Plattform- und Compiler-unabhängige Implementation mit dem Namen *STLport* [STL]. Diese Implementation zeichnet sich außerdem dadurch aus, daß sie in kommerziellen Anwendungen frei eingesetzt werden darf.

2.2.1.3 SGML

Die *Standard Generalized Markup Language (SGML)* ist im ISO-Standard ISO 8879:1986 definiert [ISO86].

Der Begriff Markup wurde aus dem Bereich des Korrekturlesens übernommen, wo fehlerhafte Stellen eines Dokuments mit Bemerkungen in einer speziellen Syntax, den Tags, markiert wurden.

Daraus entwickelte die ISO die Sprache SGML. Großer Wert wurde darauf gelegt, daß sich Darstellung und Struktur nicht vermischen. Die Struktur, bzw. der Typ eines Dokuments wird mit Hilfe einer *Document Type Definition (DTD)* festgelegt. Die DTD definiert, in welcher Reihenfolge Tags vorkommen dürfen und wie diese geschachtelt werden können. Außerdem werden die Attribute definiert, die ein Tag haben kann. Ein validierender SGML-Parser prüft beim Lesen eines SGML-Dokuments die Gültigkeit bezüglich der angegebenen DTD. Entspricht das Dokument nicht der DTD, bricht der Parser mit einer Fehlermeldung ab.

Aufgrund der hohen Komplexität des Standards hat sich diese Technologie noch nicht auf breiter Ebene durchsetzen können. Der Aufwand hat sich bisher nur für Unternehmen bezahlt gemacht, die einen enormes Aufkommen an Daten haben, die in unterschiedlichen Darstellungsformen benötigt werden.

2.2.1.4 SQL

Die *Structured Query Language (SQL)* ist eine Sprache zur Abfrage von Datenbanken, die auf einem relationalen Modell basieren. Sie wurde zum ersten Mal 1987 (SQL, SQL-87) zu einem Standard erhoben. Im Jahre 1992 wurde dieser Standard grundlegend überarbeitet und ist seitdem in ISO/IEC 9075:1992 (SQL2, SQL-92) definiert [ISO92]. Zur Zeit befindet sich SQL3 im Standardisierungsprozeß und es wird Ende 1999 bzw. Anfang 2000 mit der Verabschiedung dieser Standardrevision gerechnet. SQL3 wird als entscheidendes Merkmal Objekte unterstützen.

2.2.2 W3C-Standards

Das *W3 Consortium (W3C)* [W3C] ist 1994 von der *Universität Keio*, dem *Massachusets Institute of Technology (MIT)* und dem *Institute National de Recherche en Informatique et en Automatique (INRIA)* mit dem Ziel gegründet worden, die Aktivitäten bezüglich des WWWs zu koordinieren.

Das Konsortium teilt sich in drei organisatorische Bereiche auf, die für die verschiedenen W3C Technologien verantwortlich sind.

- User-Interface Domain: HTML, CSS, XSL
- Technology and Society Domain: RDF, PICS, P3P
- Architecture Domain: XML, HTTP, HTTP-NG

Der Standardisierungsprozeß geht von der *Requirement-Definition*, über ein *Working-Draft (WD)* und eine *Proposed-Recommendation (PR)*, bis zur endgültigen *Recommendation (REC)*.

In der Liste der Architecture-Domain besitzt das *HyperText Transfer Protokoll (HTTP)* einen Sonderstatus, da der Standard nicht vom W3C, sondern von der *Internet Engineering TaskForce (IETF)* entwickelt wird. Das W3C, als Mitglied in der IETF, übernimmt jedoch eine entscheidende Rolle bei der Gestaltung neuer HTTP-Spezifikationen.

2.2.2.1 HTML

Die *Hypertext Markup Language (HTML)* ist die universelle Sprache für den Austausch von Hypertext-Dokumenten über das World Wide Web.

Die erste Version der Sprache geht auf das Jahr 1990 zurück und wurde im *European Laboratory for Particle Physics (CERN)* als Anwendung von SGML entwickelt. In den darauffolgenden drei Jahren wurde sie auf verschiedene Arten erweitert (HTML+, 1993). Im Jahr 1995 schrieb die IETF mit der Version 2.0 [BLC95] zum ersten Mal fest, was im WWW zu diesem Zeitpunkt üblich war. Ein Vorschlag für HTML3.0 [Rag95] sah

umfangreiche Neuerungen vor, wurde aber in den Standardisierungsgremien nicht akzeptiert. HTML2.0 und die bis dahin eingeführten Erweiterungen, sowie einige Vorschläge aus HTML3.0 mündeten schließlich in der Version 3.2 [PR-97], einem offiziellen W3C-Standard. HTML4.0 [RHJ98] erweitert diesen Standard z.B. in den Bereichen Internationalisierung, Tabellen, Rahmen, Style-Sheets und Drucken. Der Standard definiert drei DTDs (`strict.dtd`, `loose.dtd` und `frameset.dtd`), mit deren Hilfe ein HTML-Autor die Standard-Konformität prüfen kann. Version 4.01 [RHJ99] verbessert die Text-, Multimedia- und Hyperlink-Eigenschaften von Version 4.0. Bis dahin stellt HTML eine Anwendung von SGML dar. Die Proposed Recommendation XHTML1.0 [R$^+$99] ist eine Neufassung von HTML4.0 als Anwendung der *Extensible Markup Language (XML)* (s. Abschn. 2.2.2.2). Dies verspricht HTML mit den Möglichkeiten von XML und ebnet mittelfristig den Weg für die Einführung der neuen Technologien XML und XSL, der *Extensible Stylesheet Language* (s. Abschn. 2.2.2.4).

2.2.2.2 XML

XML ist eine vereinfachte Form von SGML und liegt seit Februar 1998 als Recommendation in der Version 1.0 vor.

Den Prognosen zufolge soll sich XML zur zweiten großen Revolution des Internets, nach dem WWW, entwickeln.

Als größter Vorteil von XML wird angeführt, wie auch schon bei SGML, daß die Daten und die Darstellung nicht miteinander vermischt sind. Verglichen mit SGML fehlen XML einige, weniger benutzte, komplexe Details. Eine genaue Übersicht findet man in [Cla97].

Ein XML-Dokument bezeichnet man als *gültig* bzgl. seiner DTD, wenn es den Regeln dieser DTD entspricht. Ein XML-Dokument heißt *wohlgeformt*, wenn zu jedem Start-Tag auch ein Stop-Tag gehört. Die Festlegung auf eine bestimmte Menge von Tags bezeichnet man auch als *Anwendung* von XML.

In Abbildung 2.3 ist eine Übersicht einiger Anwendungen und deren Beziehungen untereinander dargestellt. *MathML* ist eine XML-Anwendung zur Beschreibung von mathematischen Formeln. *Scalable Vector Graphics (SVG)* ist eine Sprache zur Beschreibung von Vektorgrafiken. Die *Structured Multimedia Integration Language (SMIL)* dient der Definition von Multimedia Präsentationen im Internet. Das *Resource Definition Framework (RDF)* stellt eine Sprache zur Beschreibung von Ressourcen zur Verfügung. Darauf aufbauend ist die *Platform for Internet Content Selection (PICS)* ein Mechanismus zur Etablierung von Zugangskontrollen für WWW-Seiten. Die *Platform for Privacy Preferences (P3P)* ist eine Sprache zur Beschreibung privater Daten.

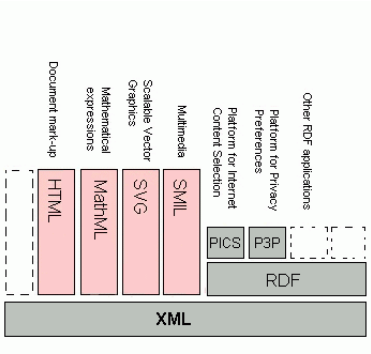

Abbildung 2.3: Anwendungen von XML

Die Arbeit des W3C charakterisiert sich dadurch, daß ständig neue Anwendungen von XML entwickelt werden, um möglichst viele proprietäre Formate auf eine XML-Basis zu bringen. Sollte dies gelingen, dann könnte sich XML als Anwendungs-übergreifendes Basisdatenformat durchsetzen und so zu einer Harmonisierung der Formate im elektronischen Datenaustausch führen.

Beispiel:

```
<?xml version="1.0"?>
<!DOCTYPE mail SYSTEM "mail.dtd">
<mail>
  <recipients>
    <recipient>mrumpf@yahoo.com</recipient>
  </recipients>
  <subject>Test</subject>
  <sender>harald.mustermann@freemail.com</sender>

  <body>
  Dies ist eine Test-Mail!

  Harald
  </body>
</mail>
```

Eine mögliche DTD für diese E-Mail kann wie folgt definiert sein:

```
<!ELEMENT mail (recipients,sender,subject,body)>
<!ELEMENT recipients (recipient)+>
<!ELEMENT recipient (#PCDATA)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

Die erste Zeile der DTD sagt aus, daß der `mail`-Tag die folgenden Kinder hat: `recipients`, `sender`, `subject` und `body`. Die Abwesenheit eines Kardinalitätsoperators „*“ (≥ 0), „+“ (≥ 1) oder „?“ (0..1) bedeutet, daß diese Kinder genau einmal vorkommen müssen. Der `recipients`-Tag besitzt ein Kind-Tag, der einmal vorkommen muß, aber auch mehr als einmal vorkommen kann. Die letzten vier Zeilen sagen aus, daß die Tags `recipient`, `sender`, `subject` und `body` reinen Text enthalten.

2.2.2.3 DOM

Das *Document Object Model (DOM) Level 1* ist eine API-Spezifikation für den Zugriff auf HTML- und XML-Dokumente.

Die Spezifikation besitzt seit Oktober 1998 den Status einer Recommendation [H+98] und für den Level 2 befindet sich ein Entwurf, als Working-Draft, gerade in der Arbeitsphase [H+99]. Level 2 erweitert Level 1 um APIs für: Style-Sheets, Events, Filter, Iteratoren und Bereiche.

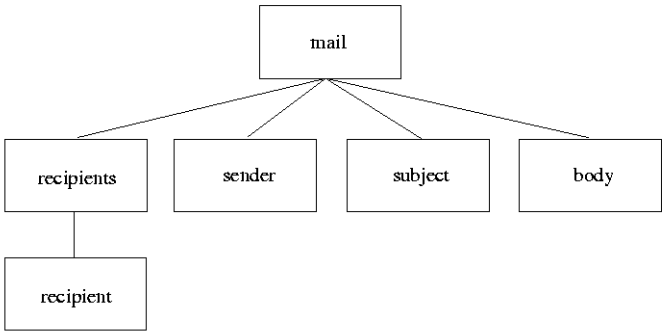

Abbildung 2.4: Baumdarstellung der XML E-Mail

Mit dem DOM-API ist es nun möglich, z.B. das `subject` der Mail auf folgende Art zu extrahieren:

```
// Document-Objekt (doc) instantiieren. Dieser Schritt wird
// nicht durch die DOM-API Level 1 Spezifikation erfasst und
// haengt damit von der Implementation ab.
//
//  "doc = createInstance()"

// Wurzelknoten
mail_node  = doc.documentElement;

// alle Kinder von main (recipients, subject, sender, body)
mail_nodes = mail_node.childNodes;

// zweites Kind ist das subject
subject_node = mail_nodes.item(1);

// Wert des Knotens ist 'Test'
value = subject_node.nodeValue;
```

2.2.2.4 Stylesheets

Die *Extensible Stylesheet Language (XSL)* liegt zum Zeitpunkt des Schreibens als Proposed Recommendation [Dea99] vor und ist damit kurz davor den Status einer Recommendation zu erhalten. Für *Cascading Style Sheets (CSS)* existieren Recommendations zum Level 1 [LB99] und Level 2 [LBLJ98]. Eine Beschreibung des in der Diskussion befindlichen Working-Drafts zum Level 3, findet man unter [CSS].

Bei Cascading Style Sheets handelt es sich um eine reine Formatierungssprache. Sie kann benutzt werden, um HTML- oder XML-Dokumente für die Anzeige auf einem bestimmten Medium zu formatieren.

Die Extensible Stylesheet Language bietet darüber hinaus Eigenschaften zur Transformation von XML-Dokumenten. Der Verarbeitungsprozeß bei Anwendung der XSL auf ein XML-Dokument ist also zweistufig. In der ersten Stufe wird die Baumdarstellung des Quelldokuments transformiert. In der zweiten Stufe werden dann auf den transformierten Baum die Formatierungsregeln, ähnlich denen der CSS, angewendet.

Formal ist XSL eine Kombination der *Document Style Semantics and Specification Language (DSSSL)* [ISO96] und CSS mit einer XML-basierten Syntax. Die Bezeichnungen für die beiden Stufen lauten *XSL Transformation (XSLT)* [Cla99] und *XSL Formatting Objects (XSL FO)* [Dea99]. In Abbildung 2.5 werden die verschiedenen Zusammenhänge zwischen XML, HTML, CSS und XSL gezeigt [LB98].

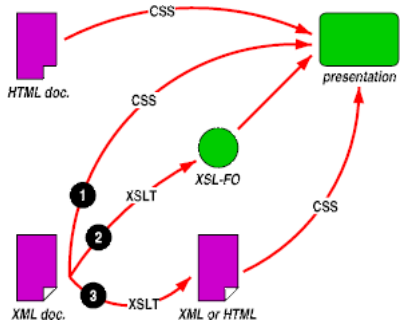

Abbildung 2.5: Kombinationsmöglichkeiten von CSS und XSL

Beispiel:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/mail">
    <HTML><BODY>
      <TABLE>
        <TR><TD>Absender  :</TD>
            <TD><xsl:value-of select="sender"/></TD>
        </TR>
        <TR><TD>Empfaenger:</TD>
            <TD><xsl:value-of select="recipients/recipient"/></TD>
        </TR>
        <TR><TD>Betreff   :</TD>
            <TD><xsl:value-of select="subject"/></TD>
        </TR>
        <TR><TD>Text      :</TD>
            <TD><xsl:value-of select="body"/></TD>
        </TR>
      </TABLE>
    </BODY></HTML>
  </xsl:template>
</xsl:stylesheet>
```

Dieses XSL-Stylesheet, angewendet auf die Mail im XML-Format, ergibt das folgende Ergebnis als HTML-Dokument (s. Abb. 2.6):

```
<HTML><BODY>
  <TABLE>
    <TR><TD>Absender  :</TD>
        <TD>harald.mustermann@freemail.com</TD>
    </TR>
    <TR><TD>Empfaenger:</TD>
        <TD>mrumpf@yahoo.com</TD>
    </TR>
    <TR><TD>Betreff   :</TD>
        <TD>Test</TD>
    </TR>
    <TR><TD>Text      :</TD>
        <TD>Dies ist eine Test-Mail!

Harald</TD>
    </TR>
  </TABLE>
</BODY></HTML>
```

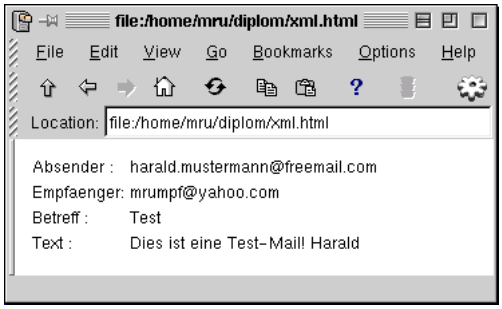

Abbildung 2.6: HTML-Ausgabe der mit XSL konvertierten E-Mail.

2.2.3 IETF-Standards

Die *Internet Engineering TaskForce (IETF)* ist ein 1986 gegründetes Konsortium, dessen Aufgabe die Koordination der kurz- und mittelfristigen technischen Belange des Internets ist.

Im Jahre 1992 wurden die *Internet Society (ISOC)* als Dachorganisation für das *Internet Architecture Board (IAB)*, die IETF, die *Internet Research Task Force (IRTF)* und die *Internet Assigned Number Authority (IANA)* gegründet [IET].

Die IETF richtet nach Bedarf *Working-Groups (WG)* zu bestimmten Themen ein, die auf den folgenden neun Bereichen operieren: *Applications Area (app)*, *Internet Area (int)*, *IP: Next Generation Area (ipng)*, *Network Management Area (mgt)*, *Operational Requirements Area (ops)*, *Routing Area (rtg)*, *Security Area (sec)*, *Transport Area (tsv)* und *User Services Area (usv)*.

Es gibt keinen formalen Abstimmungsprozeß innerhalb der WGs. Die Teilnehmer einer WG entscheiden zwar, ob und wann ein Dokument einen Status erreicht hat um veröffentlicht zu werden, aber theoretisch kann jeder an diesem Meinungsbildungsprozeß und damit an der Entscheidung selbst teilnehmen.

Die Arbeit einer WG resultiert entweder in einem *Internet Draft (ID)* oder in einem *Request For Comment (RFC)*. Ein ID ist eine Art Arbeitsdokument, welches sechs Monate Gültigkeit besitzt. Ein RFC ist wiederum in vier weitere Kategorien aufgeteilt: *Historic*, *Experimental*, *Informational* und *Standards Track*. Den Status *Historic* besitzt eine Dokumentation, die aus historischen Gründen erstellt wurde. Ein *Experimental* RFC beschreibt eine experimentelle Arbeit, die nicht Teil eines funktionsfähigen Dienstes ist. *Informational* beschreibt Dinge, die in Zusammenhang mit dem Internet stehen, aber nicht unter dem Einfluß einer WG sind. Diese RFCs werden von Autoren, außerhalb der IETF erstellt. Der Typ *Standards Track* bezeichnet RFCs, die sich auf dem Wege zu einem wirklichen Standard befinden. Die letzte Gruppe wird noch einmal unterteilt in: *Proposed Standard*, *Draft Standard* und *Internet Standard*. Ein *Proposed Standard* hat eine Gültigkeit zwischen 6 und 24 Monaten und muß anschließend entweder einen höheren Status erhalten, überarbeitet oder verworfen werden. Ein *Draft Standard* zeichnet sich dadurch aus, daß es mehrere, voneinander unabhängige Implementation gibt, die das Konzept stützen, das in diesem RFC dokumentiert ist. Dieser Draft Standard besitzt eine Gültigkeit zwischen 4 und 24 Monaten. Der *Internet Standard* ist die letzte Stufe und stellt die endgültige Fassung eines RFC dar. Die Gültigkeit des RFC ist unendlich, kann aber zu einem späteren Zeitpunkt in den Status *Historic* überführt werden.

Eine genauere Beschreibung der Struktur der IETF und deren Beziehung zu anderen Organisationen findet man in [IET]

2.2.3.1 HTTP

HTTP ist ein Protokoll für kollaborative und verteilte Hypermedia Informationssysteme und fügt sich auf der Applikationsebene in das TCP/IP-Schichtenmodell ein.

Das Protokoll existiert in der Version 0.9 seit 1990, wobei die offiziellen Standardisierungsbemühungen erst Ende 1994 gestartet und im Juni 1999 in den Status eines Internet Draft Standards erhoben wurde [BL+99].

Das Protokoll ist in HTTP-Servern und den Browsern implementiert und schafft dadurch die Basis für den Austausch von HTML-Dokumenten über das Internet.

2.2.3.2 CGI

Das *Common Gateway Interface (CGI)* ist eine Schnittstelle, um Programme in einen HTTP-Server einzubinden und dem Benutzer über das HTTP-Protokoll zugänglich zu machen. Das Interface sollte ursprünglich dazu dienen, Datenbanken an das Internet anzubinden. Das CGI-Programm übernimmt dabei die Rolle eines *Gateway* zwischen dem Internet und einer Datenbank.

Die Spezifikation CGI/1.1 wird von der Software Development Group der *National Super Computing Agency (NCSA)* verwaltet [CGI]. Seit November 1997 gibt es Bestrebungen diese Spezifikation in eine *Informational RFC* zu überführen. CGI ist ein System, das seine Praxistauglichkeit bereits unter Beweis gestellt hat. Es wurde bisher nicht von der IETF entwickelt. Erst die Version 1.2 soll, unter der Obhut der IETF, in einen tatsächlichen *Internet Standard* münden.

2.2.4 OMG-Standards

Die *Object Management Group (OMG)* wurde im April 1989 von 11 Firmen als „non-profit“ Organisation gegründet. Mittlerweile hat die OMG mehr als 800 Mitglieder. Die Ziele der OMG sind[1]:

1. *to promote a single object-oriented applications integration environment based on appropriate industry standards;*

2. *to promote frameworks for compatible and independent development of applications;*

3. *to enable coordination among applications across heterogeneous networked systems in a multinational, multilingual environment;*

4. *to adopt a core of commercially available specifications of these frameworks and to promote international market acceptance and use;*

[1] Auszug aus: [OMG99], 1.2 Purpose

5. *to actively influence the future direction and development of these adopted specifications; and*

6. *to foster the development of tools and applications that conform to and extend these frameworks and to provide a mechanism for certifying compliance with the adopted specifications.*

Die OMG besteht aus drei organisatorischen Einheiten, dem *Architecture Board (AB)*, dem *Platform Technology Committee (PTC)* und dem *Domain Technology Committee (DTC)*. Das AB kontrolliert die beiden Komitees bzgl. der Konformität ihrer Arbeit zur OMA (s. Abschn. 2.2.4.1) und untersucht die Resultate ihrer Arbeit bezüglich einer Überschneidung mit anderen Spezifikationen. Jede dieser Einheiten arbeitet mit ihren *Task Forces (TFs)*, *Special Interest Groups (SIGs)* und *Working-Groups (WGs)* zusammen. Das *Board of Directors (BOD)* bildet das oberste Gremium, das über die endgültige Annahme von Standards entscheidet. Die technischen Komitees unterstützen die Direktoren in technischen Belangen und geben Empfehlungen bezogen auf die Technologien. SIGs vertreten die Interessen gegenüber den Direktoren.

Es gibt acht Mitgliedschaften in der OMG: *Contributing-*, *Domain-*, *Platform-*, *Influencing-*, *Government-*, *Auditing-*, *Analyst-* und *University-Member*. Die ersten drei haben den größten Einfluß auf den internen Prozeß. Den geringsten Einfluß hat die *Analyst*-Mitgliedschaft, da ihr keine aktive Teilnahme an Abstimmungen gewährt wird.

Eine Standardisierung wird durch eine *Request For Proposal (RFP)* ausgelöst. Innerhalb einer Task-Force werden daraufhin Spezifikationen erarbeitet und in einem offenen Prozeß diskutiert. Dieser Prozeß dauert in der Regel 12-15 Monate und an dessen Ende steht eine Abstimmung auf Ebene der TCs. Diese Abstimmung gilt als Empfehlung auf Integration der Spezifikation in die OMA. Das BOD entscheidet abschließend auf Annahme dieser Empfehlung. Einen schnelleren Weg zu einem Standard bieten die *Request For Comment (RFC)*.

Einen umfassenden Einblick in die Struktur und den Standardisierungsprozeß der OMG vermittelt [OMG99] und [OMG97].

2.2.4.1 OMA

Die *Object Management Architecture (OMA)* ist ein Architektur-Framework für verteilte Objekte [OMA97]. Die OMA besteht aus Schnittstellen- und Protokoll-Spezifikationen.

Die OMA spezifiziert fünf von vier identifizierten Komponenten:

- Der *Object Request Broker (ORB)* ist das Zentrum der gesamten Architektur. Er stellt die grundlegenden Objekt-Kommunikationsmechanismen bereit.

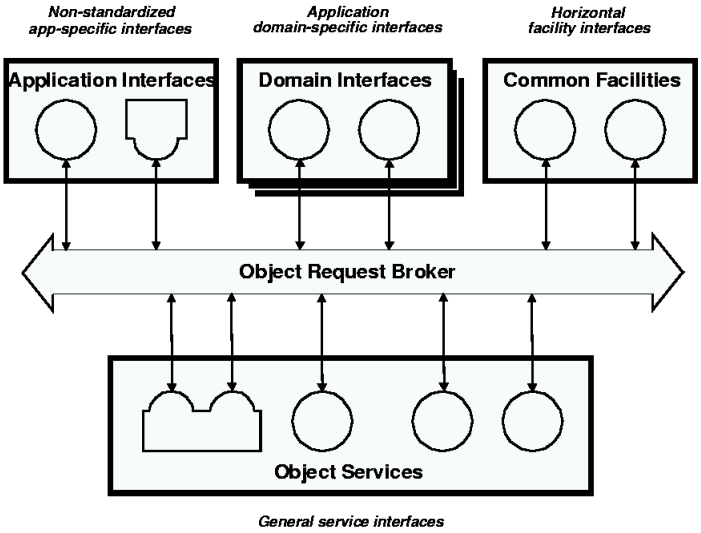

Abbildung 2.7: Referenzmodell der Object Management Architecture

- *Object Services (OS)* sind systemnahe Dienste, wie z.B. Sicherheits-, Transaktions- und Namensdienste.
- Die *Common Facilities (CF)* sind spezielle, anwendungsnahe Dienste, wie z.B. Drucker-, Datenbank-, und E-Mail-Dienste.
- Bei den *Domain Services (DS)*, auch *Domain Interfaces (DI)* genannt, handelt es sich um Spezifikationen von Schnittstellen für ein bestimmtes Anwendungsgebiet. Z.B. Finanzen, CAD, Telekommunikation, Gesundheitswesen und Flugverkehr.
- Die *Application Objects (AO)* werden von der OMA zwar identifiziert, aber nicht spezifiziert. Die AOs sind Anwendungsobjekte, die die CFs, DSs und OSs verwenden, um sie zu einer Anwendung miteinander zu verbinden.

In Abbildung 2.7 ist das Referenzmodell der OMA zu sehen. Dieses Modell verdeutlicht die Beziehungen zwischen den oben vorgestellten Einheiten der OMA[2].

Ein *Object Framework* ist eine Kollektion von kooperierenden Objekten, die gemeinsam einen Dienst für eine bestimmte Anwendung oder Anwendungs-Domäne verrichten [OMA97].

[2] s.a. 2.2.6.4 und 2.2.5.5

ORB Die Schnittstellen des ORB als zentrale Komponente der OMA, werden durch die *Common Object Request Broker Architecture (CORBA)* definiert [COR99a]. Der ORB ist für die gesamte Kommunikation zwischen zwei Objekten zuständig. Dazu gehören das Verpacken der Funktionsparameter, die Kommunikation mit den unteren Protokollebenen und die Synchronisation mit dem entfernten Objekt. Für den Anwendungsentwickler, der den entfernten Dienst in Anspruch nimmt, erfolgt diese Kommunikation vollkommen transparent.

Object Services Die *Object Services (OS)* beinhalten Dienste, die von der Anwendung weitestgehend unabhängig sind [COR98d]

- Naming Service
  Der Naming Service ist die zentrale Instanz zur Speicherung von Referenzen auf Objekte und dient in einem Object-Framework als zentraler Verzeichnisdienst.

- Event Service
  Der Event Service arbeitet entweder nach dem Push- oder dem Pull-Prinzip. Die Verbindung zwischen Quelle und Ziel kann sowohl synchron, als auch asynchron sein.

- Life Cycle Service
  Der Life Cycle Service folgt dem Factory-Pattern. Die Operationen Erzeugen, Bewegen und Zerstören werden von einem Factory-Objekt zur Vefügung gestellt.

- Persistent Object Service
  Der Persistent Object Service bietet Schnittstellen an, um den persistenten Zustand eines Objektes zu verwalten.

- Transaction Service
  Der Transaction Service bietet Unterstützung sowohl für flache, als auch für verschachtelte Transaktionen und kann Anwendungen sowohl nach dem prozeduralen, als auch nach dem objekt-orientierten Modell integrieren. Transaktionsdienste können über Netzwerkgrenzen zusammenarbeiten.

- Concurrency Control Service
  Der Concurrency Control Service koordiniert simultane Zugriffe von Klienten auf gemeinsame Resourcen. Er unterstützt verschiedene Lock-Ebenen.

- Relationship Service
  Der Relationship Service kann Beziehungen und Rollen zwischen Objekten ausdrücken. Bedingungen können für diese Beziehungen ein-

gerichtet werden und Verletzungen dieser Bedingungen werden durch Exceptions angezeigt.

- Externalization Service
  Die Schnittstellen des Externalization Service definieren Protokolle und Konventionen, wie Objekte in und aus einen Datenstrom konvertiert werden können.

- Query Service
  Der Query-Service erlaubt die Durchführung von Abfragen auf Kollektionen von Objekten und soll auf existierenden Standards wie SQL-92 oder OQL-93 basieren.

- Licensing Service
  Der Licensing Service dient zum Schutz von geistigem Eigentum. Er ermöglicht es Lizenz-Mechanismen in Anwendungen zu integrieren.

- Property Service
  Dieser Dienst erlaubt es, dynamisch Kollektionen von Name-Wert Paaren mit Objekten zu verknüpfen und bietet Funktionen an, die Operationen auf dieser Kollektion durchführen.

- Time Service
  Der Time Service bietet nicht nur Funktionen für Zeitberechnungen an, sondern er kann auch dazu verwendet werden, zeitgesteuerte Nachrichten zu erzeugen.

- Security Service
  Der Security Service übernimmt die Aufgaben Identifikation, Authentifizierung, Authorisierung, Zugangskontrolle und Verschlüsselung.

- Object Trader Service
  Der Trader Service kann automatisch Dienste vermitteln. Dazu werden die verfügbaren Dienste im Trader registriert und mit bestimmten Kriterien attributiert. Für die Suche nach einem bestimmten Dienst übermittelt der Klient die Such-Kriterien, die den gewünschten Dienst charakterisieren. Findet der Trader einen oder mehrere Dienste, so liefert er diese an den Klienten zurück.

- Object Collection Service
  Der Collection Service stellt allgemeine Datenstrukturen, wie z.B. list, set, queue, binary, tree und stack zur Verfügung.

Common Facilities Die Common Facilities sind ebenfalls CORBA-Objekte. Die von ihnen angebotenen Funktionen besitzen jedoch nicht die Universalität wie die OSs. Es wird zwischen *Horizontal Common Facilities* und

Vertical Market Facilities unterschieden, wobei diese Trennung auch wieder eine Spezialisierung darstellt. Die *Horizontal Common Facilities* umfassen: *User Interface Facility*, *Information Management Facility*, *System Management Facility* und *Task Management Facility*. Diese Dienste können in verschiedenen Anwendungsgebieten verwendet werden. Die *Vertical Market Facilities* sind dagegen wesentlich spezieller und können in nur wenigen Anwendungen eingesetzt werden. Identifiziert sind bisher: *Imagery*, *Information Superhighway*, *Manufacturing*, *Distributed Simulation*, *Oil and Gas Industry Exploration and Production*, *Accounting*, *Application Development* und *Mapping*. Eine Erläuterung dieser Dienste findet man in [COR98b].

Domain Services Die Domain Services zeichnen sich durch die Spezifikationen von Schnittstellen und Empfehlungen für ganze Anwendungsfelder aus. Zur Zeit sind die Bereiche Medizin [COR99c], Finanzen [COR98c], Fertigung [COR99b] und Telekommunikation [COR98e] definiert. Die Domain-Spezifikation CORBAMed für den medizinischen Bereich umfaßt zum Beispiel ein Personen-Identifikationsdienst und ein Abfrage-Dienst für medizinische Lexika.

Application Objects Die Application Objects werden im Rahmen der OMA zwar identifiziert, aber nicht spezifiziert. Die AOs greifen auf CFs, DSs und/oder OSs zu, um die gewünschte Anwendungs-Funktionalität zu implementieren.

2.2.4.2 CCM

Das *CORBA Component Model (CCM)* ist ein Framework für CORBA-Komponenten.

Das Modell umfaßt die Definition, Code-Generierung, Zusammenstellung und Installation von CORBA-Komponenten. Es handelt sich dabei zwar noch nicht um einen offiziellen Standard, allerdings befindet sich die überarbeitete Spezifikation in der abschließenden Abstimmungsphase und die letzte Frist läuft am 14. Februar 2000 aus. Mit Implementationen ist daher nicht vor dem zweiten Quartal 2000 zu rechnen. Die Joint-Revised-Submission Vol. I-III ([CCM98] mit [CCM99a] und [CCM99b], [CCM99c], [CCM99d]) enthält die vollständige Beschreibung des Komponentenmodells. Das CCM wird in Kapitel 5 genauer beschrieben.

Das Ziel des Modells ist es CORBA auf eine Stufe mit COM und EJB zu heben. CORBA beschränkt sich bisher auf die Spezifikation der Kommunikation zwischen den Objekten. Bisher fehlt ein Rahmenwerk, das auch Installation, Zusammenstellung und Lebenszyklus umfaßt. Dieses Rahmenwerk soll CORBA-Komponenten eine ähnlich Entwicklung wie COM-Komponenten in der Microsoft Welt ermöglichen. Dort ist es möglich, binäre Komponenten zu kaufen und diese in eigene Anwendungen zu integrieren.

Das CCM wird, im Gegensatz zu anderen Modellen, sogar Unterstützung für Analyse und Design umfassen[3]. Dies wird es in Zukunft möglich machen, aus UML-Diagrammen der Design-Phase direkt CORBA-Komponenten zu erzeugen.

2.2.4.3 UML

Die *Unified Modeling Language (UML)* entstand 1995 bei der Rational Software Corporation aus der *Unified Method (UM)*. Die UM war eine Vereinigung der *Booch*-Methode von Grady Booch [Boo93] und der *Object Modeling Technique (OMT)* von James Rumbaugh [RBP+93]. Ende 1995 stieß Ivar Jacobson zu Rational Software und integrierte seine Methode *Object Oriented Software Engineering (OOSE)* [Jac94] in die Bemühungen um die Vereinheitlichung der verschiedenen Software-Entwicklungs-Methoden. Daraus entstand Version 0.9 der UML. Im Januar 1997 wurde Version 1.0 der UML zum ersten Mal bei der OMG eingereicht und die überarbeitete Version 1.1 wurde im November 1997 zum offiziellen OMG Standard (s. Abb. 2.8). Nach mehreren Korrekturen und Erweiterungen wurde im November 1999 Version 1.3 von der OMG verabschiedet [Kob99]. Außerdem beschäftigt sich die ISO zur Zeit mit der UML und plant diese in einem schnellen Verfahren zu einem ISO-Standard zu machen. Die vollständige Beschreibung der aktuellen UML Version liefert [BRJ99].

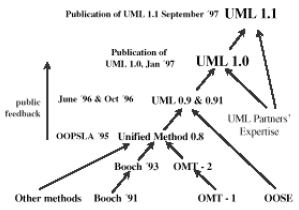

Abbildung 2.8: Zeitliche Entwicklung der UML

[3] „The OMG has issued and is planning on issuing several RFPs (as part of the Business Object Initiative) whose goal is to integrate these analysis and design activities with the CCM.“ [CCM98], 3-19

2.2.5 Java-Standards

Sun hat den *Java Community Process* etabliert, um Erweiterungen an der Java-Plattform in der Öffentlichkeit zu diskutieren und voranzutreiben. Dieses Prozeßmodell verleiht der Entwicklung neuer Spezifikationen Standard-ähnlichen Status. Die obere Leitung liegt dabei im *Process Management Office (PMO)* bei Sun Microsystems. Sun erlaubt es Firmen, Organisationen und Privatleuten dieses Prozeßmanagement in Anspruch zu nehmen und ist damit bei der Entwicklung einer neuen API-Spezifikation behilflich. Der Standardisierungsprozeß wird vom Vorschlag für eine Spezifikation bis zur finalen Version in 11 Stufen eingeteilt. Eine detaillierte Beschreibung des Java Community Process findet man in [Jav98].

2.2.5.1 Java

Für die Java Plattform, wie die Sprache und die Ansammlung von APIs von Sun genannt wird, existieren drei Partitionierungen: *Java 2 Enterprise Edition (J2EE)*, *Java 2 Standard Edition (J2SE)* und *Java 2 Micro Edition (J2ME)*.

Die Enterprise Edition beinhaltet zusätzlich zur Standard Edition eine Reihe von APIs und ein übergeordnetes Integrationsmodell für Unternehmenslösungen. Die APIs sind unter anderen *Enterprise Java Beans (EJB)*, *Java ServerPages (JSP)* und *Java DataBase Connectivity (JDBC)*.

Die Standard Edition enthält die Laufzeitumgebung (Virtual Machine) und die Basis APIs, wie sie überwiegend schon im JDK1.1 enthalten sind. Die Standard-Edition wird auch als Client-Version bezeichnet.

Die Micro Edition ist angepaßt auf die Anforderungen von mobilen Geräten, wie z.B. Mobiltelefone, Organizer oder Smart-Cards.

Ein Einführung in die Konzepte und Technologien der Enterprise Edition erfolgt in [Sun99b].

2.2.5.2 EJB

Bei den JavaBeans handelt es sich um die erste Version eines Frameworks für Java-Komponenten. JavaBeans umfaßt jedoch nur die Komponenten selbst und erlaubt einen festgelegten Zugriff von externen Anwendungen auf den Zustand eines Beans. Diese Vorgaben erfassen aber nicht die Installation, Zusammenstellung und Code-Generierung. Deshalb wurde ein um diese Merkmale erweitertes Komponentenmodell mit dem Namen *Enterprise JavaBeans (EJB)* eingeführt. EJB ist somit ein Framework für Java-Komponenten und unterstützt die Definition der Komponenten, die Code-Generierung, die Zusammenstellung und die Installation. EJB ist ein zentraler Bestandteil der J2EE Plattform und verfolgt die Ziele:

- Entwicklung verteilter Unternehmenslösungen auf Basis von Java und damit Unterstützung des „Write Once, Run Everywhere“ Prinzips.

- Interoperabilität zwischen Komponenten unterschiedlicher Hersteller.
- Abstraktion von „Low-Level APIs“ als Hilfe für den Business-Process Entwickler.
- Berücksichtigung des gesamten Lebenszyklus einer Anwendung, also Entwicklung, Installation und Laufzeit.
- Integration von „Legacy-Applikationen“.
- Kompatibilität zu CORBA.

Zur Erreichung dieser Ziele teilt die Spezifikation die Aufgabengebiete im Entwicklungs- und Installationsprozeß in sechs verschiedene Rollen auf: *Enterprise Bean Provider*, *Application Assembler*, *Deployer*, *EJB Server Provider*, *EJB Container Provider* und *System Administrator*. Die Aufgaben und Produkte dieser Rollen werden in der Spezifikation festgelegt, um die Interoperabilität zwischen den verschiedenen Parteien zu garantieren, da diese Einheiten physikalisch voneinander getrennt sein können. Eine präzise Beschreibung der Spezifikation mit Beispielen findet man in [MH99].

2.2.5.3 JDBC

Java DataBase Connectivity (JDBC) ist ein objekt-orientiertes API für den Zugriff auf relationale Datenbanksysteme. Dieses API hat sich im Java-Bereich zum Standard entwickelt und es sind mittlerweile JDBC-Treiber für jedes kommerzielle Datenbanksystem erhältlich. Als Vorbild diente *Open Database Connectivity (ODBC)* von Microsoft, das jedoch auf einem prozeduralen API basiert.

2.2.5.4 Java Servlets & JSP

Java Servlets sind eine Art Server-seitige Applets. Sie werden über ein URL-Naming Schema angesprochen und verhalten sich für den Klient wie eine normale Web-Ressource. *Java Server Pages (JSP)* ist ein Server-seitiges System zur dynamischen Erzeugung von HTML Seiten [Sun99a]. Dazu können entweder Java-Fragmente (Scriptlets) in eine HTML-Seite eingebettet werden oder JavaBeans mit Hilfe von XML-Tags aus der Seite heraus angesprochen werden. Beim Zugriff auf die Seite werden die darin enthaltenen Anweisungen abgearbeitet und das Ergebnis vom HTTP-Server an den Klienten zurückgeliefert (s. Abschn. 6.2.3). Eine JSP wird von der JSP-Engine in ein Java Servlet übersetzt.

2.2.5.5 J2EE APM

Das *J2EE Application Programming Model (APM)* [J2E99] ist ein Modell, das die Beziehungen zwischen den verschiedenen Java-APIs definiert und

damit eine Integrationsplattform für Unternehmen zur Verfügung stellt. Sie ist unmittelbar vergleichbar mit der Windows DNA (s. Abschn. 2.2.6.4) und der OMA (s. Abschn. 2.2.4.1).

2.2.6 Microsoft

Die Technologien in diesem Abschnitt sind proprietäre Technologien, da sie nicht in einem öffentlich beeinflußbaren Prozeß spezifiziert wurden. Bestenfalls hat Microsoft nachträglich die Spezifikation freigegeben (s. 2.2.6.2).

2.2.6.1 ODBC

Bei *Open Database Connectivity (ODBC)* handelt es sich um eine von Microsoft eingeführte prozedurale API für den Zugriff auf Datenbanken. Diese Schnittstelle stellt eine Abstraktionsebene über den Hersteller-spezifischen Schnittstellen dar. Die API hat sich mittlerweile zu einem Industriestandard entwickelt, da jeder Datenbank-Hersteller ODBC-Treiber mit seiner Datenbank ausliefert. Auch unter UNIX hat sich diese Schnittstelle etabliert, obwohl dort keine zentrale Instanz zur Administration der verschiedenen Treiber existiert. Unter Windows übernimmt dies der ODBC-Manager, mit dessen Hilfe Verknüpfungen zu Datenbanken ohne Programmieraufwand hergestellt werden können. Die ODBC-Konformität beschränkt sich beim Einsatz in Nicht-Windows Umgebungen auf die Ebene des APIs.

2.2.6.2 COM

Das *Component Object Model (COM)* ist die Komponententechnologie von Microsoft und seit Windows NT3.5/95 als integraler Bestandteil in diesen Produkten enthalten. Die COM-Technologie steht in direkter Konkurrenz zu EJB und CORBA und weist viele Parallelen zu diesen auf [W$^+$97]. Microsoft hat die Spezifikation von COM zwar nachträglich freigegeben [COM95], es täuscht jedoch nicht über die Tatsache hinweg, daß sie nicht unter einem Standardprozeß erstellt wurde.

Ursprünglich ohne Verteilungsaspekte definiert, wurde COM, durch die zunehmenden Popularität von Netzwerktechnologien, im Jahre 1996 um ein Netzwerkprotokoll erweitert. Diese objekt-orientierte Erweiterung, *Object Remote Procedure Call (ORPC)*, basiert auf DCE RPC [DCE] und ist bekannt unter dem Namen *Distributed COM (DCOM)* [K$^+$].

Die neuste Revision *COM+* wird mit Windows 2000 eingeführt. Dabei versucht COM+ weitere Eigenschaften von CORBA zu übernehmen und diese mit der Windows Architektur in Einklang zu bringen. So stellt der *Microsoft Transaction Server (MTS)* das Gegenstück zum CORBA Transaktionsdienst dar. Events unterstützen in COM+ zum ersten Mal das Pull-Modell. Fehlerbehandlung versucht man an den C++-Standard anzugleichen, indem

man den bisherigen Typ des Rückgabewerts `HRESULT` durch Exceptions ersetzt. Mehrfachvererbung auf Interface-Ebene ist in Zukunft möglich. Und die Programmiersprachen Unabhängigkeit wird weiter vorangetrieben. COM war sehr stark mit C++ verbunden und wurde sogar in einigen Publikationen als ein „besseres C++“ bezeichnet [Box98]. Eine Einführung in COM+ findet man in [Kir97].

2.2.6.3 MTS

Der *Microsoft Transaction Server (MTS)* ist ein Produkt, das einen Transaktionsdienst für COM-Objekte, eine Verbindungsverwaltung für Datenbanken und ein CICS-Integrationsmodul für IBM-Mainframe-Rechner anbietet. Erst durch dieses Produkt wird das Komponentenmodell COM mit EJB vergleichbar [MTS].

2.2.6.4 DNA

Die *Distributed interNet Applications Architecture (DNA)* bildet den Rahmen für die Integration aller Microsoft Produkte mit dem Ziel eine umfassendes Modell für Unternehmens-Lösungen anzubieten. Die J2EE und die OMA sind damit unmittelbar mit der Windows DNA vergleichbar [Tho99]. Der entscheidende Nachteil der DNA ist die Verschlossenheit gegenüber Standards und die Beschränkung auf die Windows-Plattform.

Marktanalysten zufolge erfolgte die Einführung der DNA nicht proaktiv, sondern reaktiv. Dies ist vermutlich darauf zurückzuführen, daß die DNA eine 3-Ebenen Architektur vorschlägt. Ein solches Modell verlagert aber die Applikationslogik vom Desktop auf den Server und stellt daher die Beschränkung auf ein bestimmtes Betriebssystem, wie z.B. MS Windows, in Frage.

2.2.7 Sonstige

In den folgenden Abschnitten werden Produkte, Architekturen und Projekte beschrieben, die keine Standards sind bzw. nicht auf Standards basieren. Sie werden erwähnt, weil sie in dieser Arbeit von Bedeutung sind.

2.2.7.1 Applikations-Server

Der Begriff Applikations-Server bezeichnet eine Kategorie von Produkten. Ganz allgemein ist ein Applikations-Server eine Ausführungsplattform für Applikationslogik. Durch diese Festlegung lassen sich aber selbst Betriebssysteme in die Kategorie der Applikations-Server einfügen, da auch sie eine Ausführungsplattform für Applikationslogik darstellen.

Um die Menge an Systemen einzuschränken, ist festgelegt worden, daß ein Applikations-Server über bestimmte Basisdienste verfügen muß, die ein

Betriebssystem normalerweise nicht bietet. Dazu gehören Transaktionen, Last-Balancierung, Ausfallsicherheit und Datenbankzugriff. Außerdem bauen die meisten Applikations-Server auf einem Applikationsmodell auf, das eine Trennung zwischen Applikationslogik und Präsentationslogik etabliert.

Die Festlegung, was einen Applikations-Server ausmacht, orientiert sich jedoch nicht an einer allgemeingültigen Definition, sondern ist ein Schnitt durch die populärsten am Markt erhältlichen Systeme. Es handelt sich also um eine Definition anhand von Beispielen und nicht um eine grundlegende Architekturdefinition. Die Quellen im Internet bzgl. Applikations-Server scheinen den gleichen Ursprung zu haben. Dieser Schluß liegt nahe, da alle Artikel bei der Erklärung der Bestandteile eines Applikations-Servers eine gewisse Ähnlichkeit aufweisen (s.a. [Mat99], [Ben99], [Fel], [Wri99]).

Im Zuge von EJB und CCM, die einen Rahmen für Server-seitige Ausführungsplattformen definieren, könnte sich der Begriff Applikations-Server in Zukunft konkretisieren [RW99].

2.2.7.2 PC-Docs/Fulcrum SearchServer

Der PC-Docs/Fulcrum SearchServer ist eine Server-Applikation, die einen Volltext-Suchdienst anbieten. Sie verfügt über eine relationale Abfragesprache und erlaubt Abfragen über eine ODBC-API. Die relationale Abfragesprache weicht in einigen Punkten von Standard-SQL ab und wird deswegen als *SearchSQL* bezeichnet. Der Server speichert die indizierten Daten in einem speziellen Format. Um beliebige Datenformate indizieren zu können, muß daher eine Transformation über einen sog. *TextReader* erfolgen.

Die genauen Eigenschaften des Search-Servers können in [Ful96] und [PC-] nachgelesen werden.

Kapitel 3

KN1 - Ist-Analyse

Die Analyse von KN1 ist Teil der Anforderungsdefinition und gehört zur Definitionsphase eines Software-Entwicklungsprozesses. Der Funktionsumfang und die Architektur des zu erweiternden Systems werden analysiert, um darauf aufbauend präzise Anforderungen an das Nachfolge-System formulieren zu können.

3.1 Projektstruktur

Eine KN1-Anwendung hat folgende typische Verzeichnisstruktur:

- **Data**
  Im Data-Verzeichnis liegen die SGML-Dokumente, die mit Hilfe des Index-Servers indiziert werden und in einer Anwendung die Datenbasis für die Volltext-Suche bilden.

- **Docs**
  Im Verzeichnis Docs liegen die statischen HTML-Seiten und die dazugehörigen Bilder. Meistens befindet sich hier nur eine Datei, da nur die erste Seite statisch ist. Dieses Verzeichnis muß als Basisverzeichnis (`DocumentRoot`) in der HTTP-Server Konfiguration angegeben werden.

- **ExecStdCgi**
  In diesem Verzeichnis befindet sich das CGI-Programm `NetDiff.exe` und die dazugehörige Konfigurationsdatei `NetDiff.ini`. Dieses Verzeichnis muß als CGI-Verzeichnis (`cgi-bin`) in der HTTP-Server Konfiguration angegeben werden.

- **HttpServerStdCgi**
  Hier ist der verwendete HTTP-Server abgelegt. Es handelt sich um

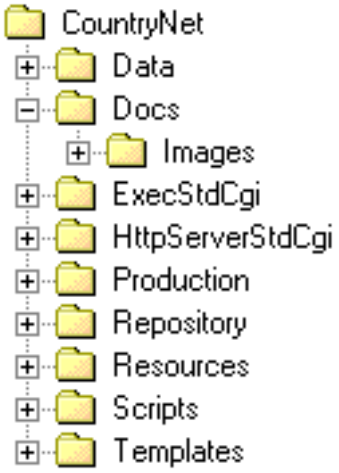

Abbildung 3.1: CountryNet Projektstruktur

einen besonders schlanken Server, der nur wenig Speicherplatz in Anspruch nimmt[1].

- **Production**
  Dieses Verzeichnis ist das Dokument-Arbeitsverzeichnis des Systemverwalters. Hier werden die SGML-Dokumente abgelegt und mit Hilfe der Programme aus dem Scripts-Verzeichnis indiziert.

- **Repository**
  In diesem Verzeichnis befindet sich das Format- und das SQL-Repository. Die Repositories werden als Textdatei vom Entwickler erstellt, mit einem Tool kompiliert und in das Verzeichnis Resources kopiert.

- **Resources**
  Hier befinden sich weitere Dateien, die vom CGI-Programm zur Ausführung benötigt werden.

- **Scripts**
  Im Scripts-Verzeichnis liegen die Hilfsprogramme zur Erstellung des Index. Eine Indizierung der Dokumente wird durchgeführt, wenn neue Dokumente hinzukommen oder alte entfernt bzw. verändert werden.

- **Templates**
  Dieses Unterverzeichnis enthält die HTML-Templates des Projekts und ist damit die physikalische Repräsentation des Template-Repositories (s. Abschn. 3.3). Einen Verweis auf dieses Verzeichnis befindet sich in der Konfigurationsdatei `NetDiff.ini`.

[1] Xitami, www.imatix.com

3.2 Funktionsweise

Der typische Ablauf einer KN1-Anwendung stellt sich wie folgt dar: Nach der Verbindungsaufnahme mit dem HTTP-Server bekommt der Benutzer die oft einzige, statische HTML Seite der Anwendung geliefert. In dieser Seite können HTML-Formulare (POST-Request) oder Verweise (GET-Request) enthalten sein, die das CGI-Programm `NetDiff.exe` referenzieren. Verschiedene Parameter , werden von `NetDiff.exe` akzeptiert, die den internen Ablauf beeinflussen. Die folgende Liste zeigt eine Auswahl dieser Parameter.

NPG (NextPaGe) – HTML-Template, auf dem die zu erzeugende Seite basieren soll.

CM (CharacterMap) – Zeichensatzkodierung der zu erzeugenden Seite.

LINKLEFT – HTML-Tag zur Einleitung einer Markierung.

LINKRIGHT – HTML-Tag zum Abschluß der Markierung.

NPG ist der einzige Pflichtparameter, da auf Basis des damit angegebenen HTML-Templates die nächste HTML-Seite erzeugt wird. Beim Aufruf des CGI-Programms mit z.B. `NPG=login.html` lädt die Laufzeitumgebung das Template `login.html`, führt die dort eingebetteten Befehle aus und sendet das Ergebnis anschließend zum Klienten zurück.

Der interne Ablauf von `NetDiff.exe` stellt sich wie folgt dar:
Nach dem Start wird eine Symboltabelle aufgebaut. Dort werden sämtliche Umgebungsvariablen und Parameter eingetragen. Im zweiten Schritt wird die Datei `htmlnot.tag` geladen. Diese Datei enthält eine Konvertierungstabelle, die vom SGML-Parser zur Umwandlung der SGML-Tags in HTML-Tags verwendet wird. Im nächsten Schritt wird das durch den Parameter NPG spezifizierte Template aus dem Template-Verzeichnis geladen und an den Parser übergeben. Der Parser führt die im Template eingebetteten Befehle aus und erstellt die Ausgabeseite. Wenn die Parameter CM oder LINK (LINKLEFT und LINKRIGHT) übergeben wurden, so wird im Anschluß an das Parsen noch die Zeichensatzkonvertierung und die Markierung der Schlüsselwörter durchgeführt. Der letzte Schritt ist dann die Ausgabe der erzeugten Datei, das dazu führt, daß der HTTP-Server sie zum Klienten sendet (s. Abb. 3.3-3.5).

3.3 HTML-Templates

Die Templates bestehen aus statischem HTML und darin eingebetteten Anweisungen. Der Entwickler dieser Templates kann die HTML-Version beliebig wählen, er muß dabei nur entscheiden, welche Browser unterstützt werden sollen.

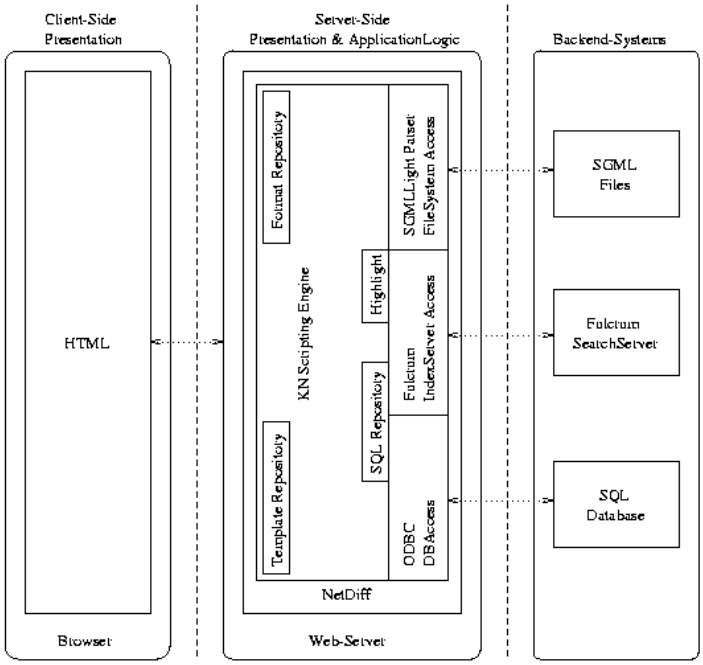

Abbildung 3.2: KN1 - Architektur

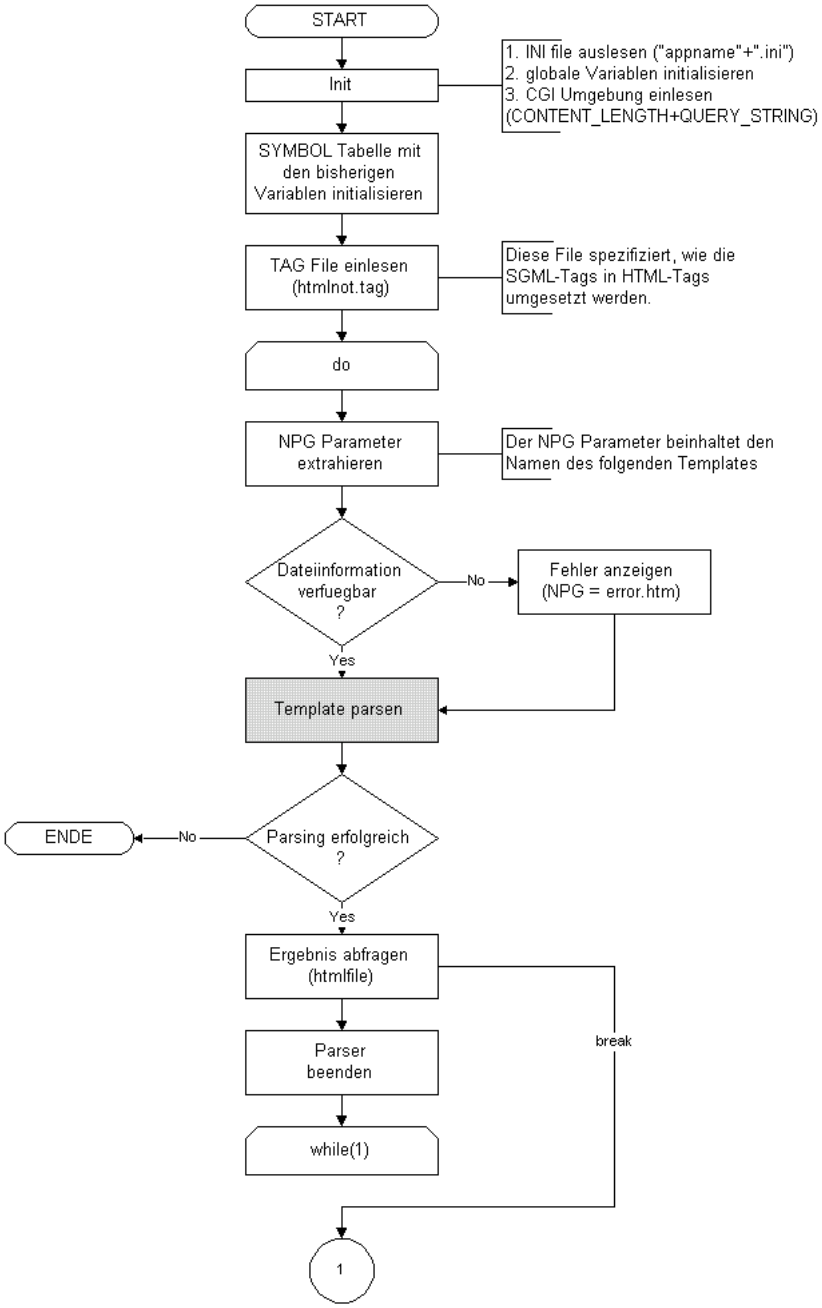

Abbildung 3.3: Ablaufdiagramm der NetDiff-Hauptschleife (Teil 1)

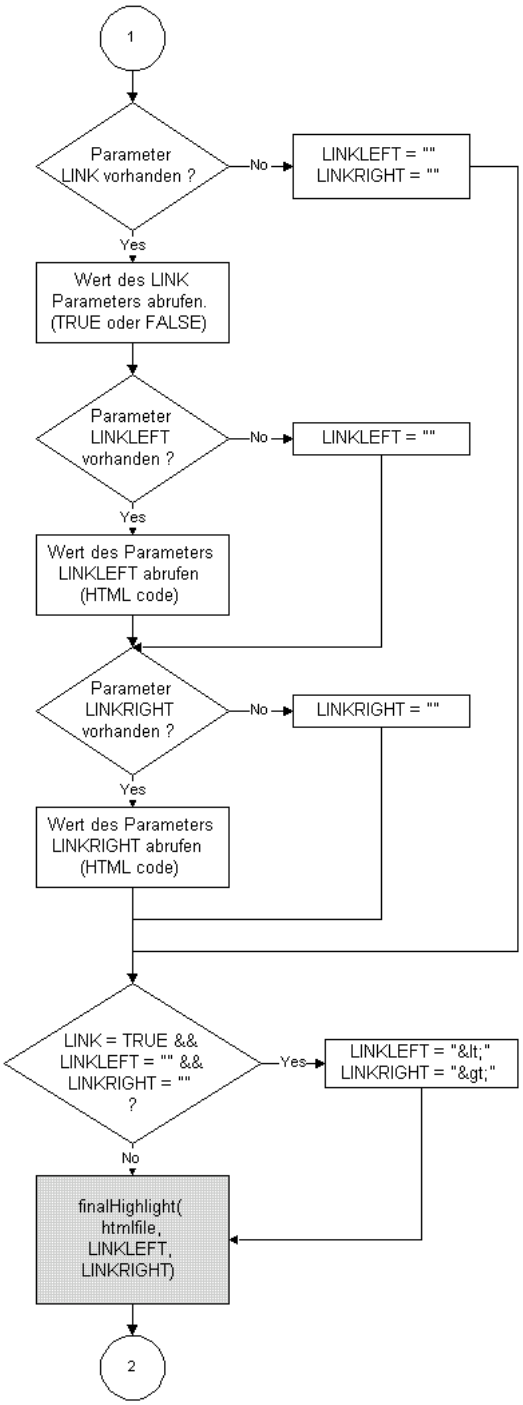

Abbildung 3.4: Ablaufdiagramm der NetDiff-Hauptschleife (Teil 2)

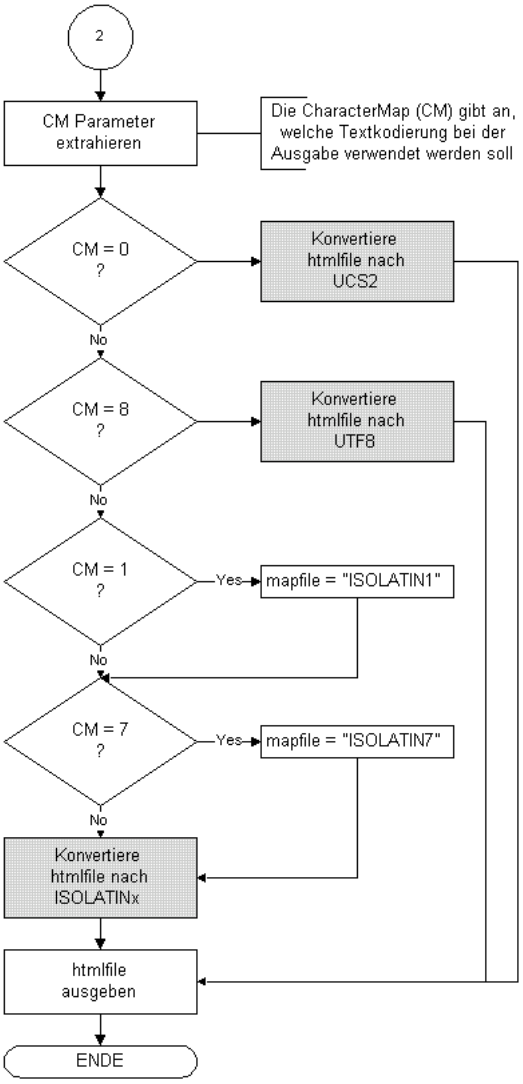

Abbildung 3.5: Ablaufdiagramm der NetDiff-Hauptschleife (Teil 3)

```
<HTML>
<?FORMAT(FMT="CHECK_EMPTY")>
<HEAD><TITLE></TITLE></HEAD>
<BODY BGCOLOR="#FFFFFF">

<FORM NAME=SEARCH METHOD=GET ACTION="/cgi-mort/netDiff.exe" onSubmit="return DatenCheck()">
<TABLE CELLPADDING=0 CELLSPACING=0 BORDER=0>
<TR>
        <TD>
        <BR>Search for:
        <BR><INPUT TYPE=TEXT NAME="NOTICE" SIZE=40 MAXLENGTH=255 onChange="NumCheck(this)">
        <P>Scope:
        <BR><INPUT TYPE=RADIO NAME=SQL-EXEC VALUE="SQL_GEN_ALL_BOOL" CHECKED>Entire document
        <INPUT TYPE=RADIO NAME="SQL-EXEC" VALUE="SQL_TITLE_BOOL">Title
        <P>Find documents that contain:
        <BR><input type="radio" name=s value="o"> At least one of the <i>keys</i> (boolean <b>or</b>)
        <input type="radio" name=s value="a" checked> All <i>keys</i> (boolean <b>and</b>)
        <P>Consider keys to be:
        <BR><input type="radio" name=w value="s" checked> Substrings
        <input type="radio" name=w value="w"> Complete words
        </TD>
</TR><TR>
        <TD ALIGN=RIGHT COLSPAN=2>
                <BR><INPUT TYPE=IMAGE SRC="/Images/ButtonSearch.gif" BORDER=0>
        </TD>
</TR>
<?HIDDEN(NAME='LNG',VALUE=LNG)>
<?HIDDEN(NAME='PPG',VALUE="1stbool.htm")>
<?HIDDEN(NAME="NPG",VALUE="1stbool.htm")>
<?HIDDEN(NAME='CM',VALUE=CM)>
<?HIDDEN(NAME='CP',VALUE=CP)>
<?HIDDEN(NAME='ST',VALUE='0')>
<?HIDDEN(NAME='IPK',VALUE='null')>
<?HIDDEN(NAME='User',VALUE=User)>
<?HIDDEN(NAME='Session',VALUE=Session)>
<INPUT TYPE=HIDDEN NAME=HIGHLIGHT_START VALUE="<?DFORMAT(DFMT="<?\"<FONT COLOR=\\\"#FF00FF\\\"><STRONG>\".urlEncode()>")>">
<INPUT TYPE=HIDDEN NAME=HIGHLIGHT_END VALUE="<?DFORMAT(DFMT="<?\"</FONT></STRONG>\".urlEncode()>")>">
</TABLE>
<TABLE CELLSPACING=0 CELLPADDING=0 WIDTH="100%">
<TR>
        <TD><?FORMAT(FMT="FULCRUM_TUNBIB")></TD>
</TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```

Abbildung 3.6: HTML-Template (Anweisungen markiert)

Die Einbettung hat folgende Syntax:

`<?FUNCTION(PARAMETER)>`

Diese Funktionsaufrufe können an beliebiger Stelle im Template stehen. Die einzige Einschränkung ist, daß die resultierende Datei eine gültige HTML-Datei sein muß. In Abbildung 3.6 ist das Beispiel eines solchen HTML-Templates zu sehen.

Es folgt eine Liste mit den Befehlen, die in die HTML-Templates eingefügt werden können:

`void AFFECT(NAME=var, VALUE=val)`

Führt eine Variablenzuweisung durch, wobei die Variable mit dem Namen `var` mit dem Wert `val` initialisiert wird.

Parameter:

`var` – Name der Variablen.

`val` – Wert, mit dem die Variable initialisiert wird.

`void CONCAT(CONCAT1=var, CONCAT2=val)`

Hängt `val` an den Inhalt der Variablen `var` an.

Parameter:

`var` – Name der Variablen.

`val` – Anzuhängender Text.

`string DFORMAT(DFMT=fmtcmd)`

Führt einen Format-Repository Befehl aus (s. Abschn. 3.4.1) und fügt das Ergebnis in die Ausgabe ein.

Parameter:

`fmtcmd` – Format-Repository Befehl.

Rückgabe:

Das Ergebnis des Format-Repository Befehls.

`string FORMAT(FMT=fmtkey)`

Fügt einen Eintrag aus dem Format-Repository in die Ausgabe ein.

Parameter:

`fmtkey` – Format-Repository Schlüssel.

Rückgabe:

Format-Repository Eintrag.

`string GETNOTICE(INIT=file, FMTNOT=tag)`

Fügt einen Unterbaum eines SGML-Dokuments in die Ausgabe ein.

Parameter:

`file` – Name der SGML-Datei.

`tag` – Name eines SGML-Tags.

Rückgabe:

Teilbaum des SGML-Dokuments.

`string HIDDEN(NAME=name, VALUE=value)`

Fügt ein verstecktes HTML-Eingabefeld in die Ausgabe ein (`<INPUT TYPE="HIDDEN" NAME="name" VALUE="value">`).

Parameter:

`name` – Name des Eingabefelds.

`value` – Wert des Eingabefelds.

Rückgabe:

HTML-Eingabefeld.

`string SQL-EXEC(SQL-EXEC=k, H-SQL=h, F-SQL=f, ELEM-SQL=b)`

Führt eine SQL-Abfrage aus und fügt die Ergebnistabelle als HTML-Tabelle in die Ausgabe ein. Dabei werden Einträge aus dem Format-Repository verwendet, um das Aussehen des Tabellenkopfes, des Fußes und des Rumpfs festzulegen.

Parameter:

`k` – SQL-Repository Schlüssel.

`h` – Format-Repository Schlüssel des Tabellenkopfes.

`f` – Format-Repository Schlüssel des Tabellenfußes.

`e` – Format-Repository Schlüssel des Tabellenrumpfs.

Rückgabe:

HTML-Tabelle.

3.4 Repositories

`NetDiff.exe` verwendet zwei Repositories, das Format- (`formrep`) und das SQL-Repository (`sqlrep`). Beide Repositories bestehen aus Einträgen, die über einen Schlüssel angesprochen werden (Dictionary).

Sowohl im Format-, als auch im SQL-Repository ist `&VARNAME;` die Syntax zum Einfügen einer Variablen.

3.4.1 Format-Repository

Die Syntax eines Eintrags im Format-Repository ist:

```
<ENTRY KEY=NAME>BODY</>
```

`NAME` – Schlüsselname des Eintrags.

`BODY` – Rumpf des Eintrags.

Die Kommandos des Format-Repository werden, im Gegensatz zu den Templates, in einer objektorientierten Notation geschrieben:

```
Ergebnis = Objekt.Methode(Parameterliste).
```

Es existieren die folgenden Objekte: `notice`, `dict`, `TXTOBJ` und `TRUTHVAL`. Das Objekt `notice` kapselt ein SGML-Dokument und ist eine Instanz des SGML-Parsers. Das Objekt `dict` steht für das Format-Repository selbst. Ein Text-Objekt `TXTOBJ` kann ein Stringliteral oder eine Stringvariable sein. `TRUTHVAL` ein Befehl, der zu einem Wahrheitswert evaluiert.

`TXTOBJ dict.atKey(KEY)`

Fügt einen Format-Repository Eintrag an der Stelle des Befehls ein. Bei diese Selbstreferenzierung wird der Rückgabewert nicht evaluiert, es handelt sich um einen einfachen „Kopiervorgang“.

Parameter:

`KEY` – Eintragsschlüssel.

Rückgabe:

Text des Format-Repository Eintrags.

`VOID notice.init(FILENAME)` und `TXTOBJ notice.getInfo(TAGNAME)`

`notice.init()` instantiiert das `notice`-Objekt und initialisiert es mit dem angegeben SGML-Dokument. Die Methode `getInfo` extrahiert den Teilbaum ab dem Tag `TAGNAME`. Die Datei `htmlnot.tag` enthält die Tabelle, anhand der die SGML-Tags in HTML-Tags umgewandelt werden.

Parameter:

`FILENAME` – Dateiname des SGML-Dokuments.

`TAGNAME` – Name des Tags, ab dem der Teilbaum eingesetzt wird.

Rückgabe:

Teilbaum ab dem angegebenen Tag.

`TXTOBJ TXTOBJ.scanHtml()`

Ersetzt in einem Objekt die Sonderzeichen durch die äquivalenten Entities in HTML. Die Ersetzung wird dabei durch eine Datei mit dem Namen `htmlfilt.tab` gesteuert.

Rückgabe:

Text, in dem die Sonderzeichen durch die entsprechenden HTML-Entities ersetzt wurden.

`TXTOBJ TRUTHVAL.cond(TRUE_TXT, FALSE_TXT)`

Liefert in Abhängigkeit des Wahrheitswertes `TRUTHVAL` entweder den Text `TRUE` oder `FALSE` zurück.

Parameter:

`TRUE_TXT` – Text, der bei `TRUE` in die Ausgabe eingesetzt wird.

`FALSE_TXT` – Text, der bei `FALSE` eingesetzt wird.

Rückgabe:

In Abhängigkeit vom Wahrheitswert entweder der Text `TRUE_TXT` oder `FALSE_TXT`.

`TRUTHVAL TXTOBJ.isEqual(TXTOBJ)`

Vergleicht die beiden Strings miteinander und liefert als Ergbnis den Wahrheitswert des Vergleichs zurück.

Parameter:

`TXTOBJ` – Bezeichnet das zu vergleichende Text-Objekt.

Rückgabe:

Das Ergebnis des Vergleichs.

`TRUTHVAL TXTOBJ.includes(TXTOBJ)`

Durchsucht den Text nach einem Teilstring.

Parameter:

`TXTOBJ` – Teilstring, nach dem gesucht werden soll.

Rückgabe:

`TRUE`, wenn der String gefunden wurde, `FALSE` andernfalls.

`TXTOBJ TXTOBJ.replace(SEARCH, REPLACE)`

Sucht und ersetzt einen Teilstring.

Parameter:

`SEARCH` – Text, der zu ersetzen ist.

`REPLACE` – Neuer Text.

Rückgabe:

Text, in dem die Ersetzung durchgeführt wurde.

`TXTOBJ TXTOBJ.urlEncode()`

Führt eine URL-Kodierung durch.

Rückgabe:

URL-kodierter Text.

Es folgt ein Auszug aus dem Format-Repository. Dabei handelt es sich um Format-Einträge für die `SQL-EXEC`-Funktion.

```
<ENTRY KEY=H_SQL_DEFAULT>
Your query was : <B>&NOTICE;</B><br>
<?OCC_NB.isEqual("0").cond("No matching found", "
<B>Query Result: </B>"+ OCC_NB + " references<BR>") >
<TABLE WIDTH="100%" BORDER="1" CELLSPACING="2"></>

<ENTRY KEY=ELEM_SQL_DEFAULT>
<TR><TD VALIGN="Top" ALIGN="Center">&RELEV;</TD>
    <TD>&STATE;</TD>
    <TD><A HREF = "<?dict.atKey("CatDiffCgi")>
    User=&User;&Session=&Session;&LNG=&LNG;&NPG=notbsen.htm&
    CM=&CM;&CP=&CP;&PPG=&PPG;&RELEV=&RELEV;&IPK=&IPK;&CNETKEY=
    <?CNETKEY.urlEncode()>
    <?LINKLEFT.cond("&LINKKLEFT="+LINKLEFT,"")>
    <?LINKRIGHT.cond("&LINKRIGHT="+LINKRIGHT,"")>
    <?VCC.cond("&VCC="+VCC.urlEncode(),"")>
    <?VCCR.cond("&VCCR="+VCCR.urlEncode(),"")>
    <?MARKER.cond("&MARKER="+MARKER.urlEncode(),"")>">
    &CREDITNAME;</A>
    </TD>
```

```
</>

<ENTRY KEY=F_SQL_DEFAULT>
</TABLE>
<?ST.isEqual("-1").cond("", "<BR><BR>
<INPUT TYPE=SUBMIT NAME=" +
SQL-EXEC + " VALUE=Next>")>
</>
```

3.4.2 SQL-Repository

Im SQL-Repository können häufig benötigte SQL-Anfragen gespeichert werden. Wie in ODBC üblich wird die Verbindung zur Datenbank über einen Datenquellennamen hergestellt. Die Syntax eines Eintrags ist:

```
<ENTRY KEY=NAM>DSN;SQL;CHN;</>
```

`NAM` – Schlüsselname des Eintrags.

`DSN` – ODBC DataSourceName.

`SQL` – SQL-Abfrage, die auf der Datenquelle auszuführen ist.

`CHN` – ODBC-Kanal.

Im SQL-Repository sind, im Unterschied zum Format-Repository, nur Variablen definiert. Der ODBC-Kanal dient der Performance-Optimierung, wird aber in der CountryNet-Anwendung nicht verwendet und deshalb auf Null gesetzt.

Im folgenden Beispiel handelt es sich um eine Anfrage an den Index-Server, der mit dem DSN `FULCRUM/COUNTRYNET` im Betriebssystem registriert ist.

```
<ENTRY KEY=SQL_GEN_ALL>
FULCRUM/COUNTRYNET;
SET RELEVANCE_METHOD '&RM;';0;
SELECT  vcc_rules() AS VCCR,
        marker_list() AS MARKER,
        match_vcc_list() AS VCC,
        RELEVANCE() AS RELEV,
        MKEY,
        STATE,
        FT_CID AS IPK,
        NAME AS CREDITNAME
    FROM SCONOTICE
    WHERE GLOBAREA IS_ABOUT '&NOTICE;' &WHERE;
```

```
    ORDER BY &ORDER; ;0;
</>
```

Zuerst wird ein Konfigurationsparameter des Index-Servers gesetzt (`SET`) und anschließend eine Abfrage ausgeführt (`SELECT`).

3.5 Datenbankanbindung

Als Datenbankanbindung wird das ODBC-API in der Version 1 verwendet, da der SearchServer nur diese Version unterstützt. ODBC-Treiber höherer Versionen sind im Allgemeinen abwärts-kompatibel, so daß auch Treiber der Version 3 noch die Funktionen der Version 1 zur Verfügung stellen.

3.6 Index-Server

Als Index-Server kommt der PC-Docs/Fulcrum SearchServer zum Einsatz. Zusätzlich wurde ein SGML-TextReader entwickelt, der den SearchServer, um die Fähigkeit mit SGML-Dokumenten umgehen zu können erweitert. Das Highlighting (s. Abschn. 3.6.2) basiert außerdem auf einer speziellen Funktionalität des SearchServers.

3.6.1 SGML-TextReader

Der SGML-TextReader implementiert die vom Server vorgegebenen Schnittstellen und ist somit in der Lage SGML-Dokumente in das interne Format umzuwandeln. Über die Konfiguration des TextReaders kann das Verhalten während des Indizierungsvorgangs beeinflußt werden. Dadurch ist es zum Beispiel möglich eine Liste von Stop-Wörtern anzugeben. Die Stop-Wörter dienen dazu Wörter ohne Informationsgehalt aus dem Text herauszufiltern. Jeder TextReader besitzt seine eigenen andere Konfigurationsoptionen.

3.6.2 Highlighting

Beim Highlighting handelt es sich um die Markierung der Such-Begriffe in den gefundenen Dokumenten. Dieser Mechanismus ist eng mit einem speziellen Merkmal des SearchServers verbunden. Nach einer Suche ist es möglich von jedem gefundenen Dokument eine Liste von Positionsangaben zu erhalten. Diese Positionsangaben sind die Stellen des Dokuments, an denen die gesuchten Begriffe vorkommen. Dadurch ist es, besonders in reinen Textdokumenten, möglich durch Einfügen bestimmter Informationen in den Text eine Markierung der Wörter zu erwirken.

Bei SGML-Dokumenten, die zur Darstellung in HTML umgewandelt werden, läßt sich durch Einfügen z.B. von `<B>` vor und `</B>` nach dem Suchbegriff diese Markierung erreichen.

3.7 Zeichensatzkonvertierung

Dieses Merkmal dient der Internationalisierung einer Anwendung. Durch den `CM`-Parameter kann eine Zeichensatzkodierung für die Ausgabeseite angegeben werden. `NetDiff.exe` wandelt dann das Dokument in den angegebenen Zeichensatz um. Dieses Funktion wird in CountryNet nicht verwendet, da es sich um eine einsprachige Anwendung handelt und somit keine Notwendigkeit einer Zeichensatzkonvertierung besteht.

3.8 Zusammenfassung

Die folgende Auflistung faßt die identifizierten Probleme von KN1 zusammen:

CGI Die Verwendung der CGI Schnittstelle stellt die größte Einschränkung bezüglich Performance und Skalierbarkeit dar. Wie aus den vorherigen Abschnitten zu entnehmen ist, wird mit jedem Aufruf eine neue Symboltabelle angelegt. Die Templates und die Repositories müssen neu geparst und eine Datenbankverbindung muß erneut hergestellt werden. All diese Punkte addieren sich zu dem betriebssystemseitigen Overhead der Prozeßerzeugung für das CGI-Modul. Da das CGI-Modul in C++ implementiert ist, entfällt zumindest das aufwendige Laden eines Interpreters.

Modularisierung Die ganze Funktionalität steckt im Modul `NetDiff.exe`. Damit wird die Wartbarkeit und die Erweiterbarkeit in höchstem Maße erschwert. Bei jeder Änderung muß das Modul neu kompiliert und der Web-Server angehalten werden, um die neue Version zu installieren.

Ein weiterer Punkt ist die Vernachlässigung einzelner Komponenten. Als Beispiel ist die Zeichensatzkonvertierung und die Kanal-Wiederverwendung bei ODBC zu nennen. Die Programmteile sind noch in `NetDiff.exe` enthalten, allerdings liegen sie brach und werden nicht mehr verwendet.

Skriptsprache Die Skriptsprache ist keine vollwertige Sprache im Sinne einer Programmiersprache. Dem Entwickler stehen nur einige wenige Funktionen zur Verfügung. Dieser geringe Umfang und die unterschiedliche Syntax in den Templates und den Repositories läßt vermuten, daß der Umfang nach Bedarf erweitert wurde.

Dokumentation Die fehlende Dokumentation macht KN1 so gut wie unbrachbar, da nur der KN1-Entwickler selbst in der Lage ist seine Anwendung zu benutzen. Anderen Entwicklern würde der Einstieg nur über den Quelltext oder unter Anleitung gelingen. Da jedoch die Quelltexte nur sehr spärlich dokumentiert sind und da der ursprüngliche

Entwickler die Firma Anfang 1999 verlassen hat, stellen sich die beiden Möglichkeiten als unbrauchbar heraus.

Entwicklungsumgebung Eine Entwicklungsumgebung fehlt in KN1 vollständig. Es ist noch nicht einmal ein Syntax-Highlighting Schema für einen Editor, wie z.B. den Emacs, verfügbar. Die HTML-Templates und die darin eingebetteten Kommandos werden mit einem einfachen Text-Editor entwickelt. Diese spartanische Unterstützung für den Entwickler, ist mit dem Komfort aktueller Entwicklungsumgebungen nicht mehr zu vergleichen.

Projektmanagement Die Entscheidung das KN1-System nur noch in der Projektarbeit einzusetzen und nicht mehr grundlegend weiterzuentwickeln war der größte Fehler. Dies führte zum heutigen Versionschaos.

Ein weiteres Phänomen in diesem Zusammenhang, ist die Aufspaltung des Systems in Zwei Quelltext-Basen. Ein weiteres Projekt der Firma AASaM ist das MiniDoc-Projekt. Dabei handelt es sich um ein Intranet für den europäischen Gerichtshof in Luxemburg zur Verwaltung aller Dokumente auf Basis von SGML. Mit der ursprünglichen Zielsetzung, KN1 als Entwicklungswerkzeug zu positionieren, wurde das Projekt auch auf Basis von KN1 begonnen. Sämtliche Änderungswünsche des MiniDoc-Teams an KN1 stießen, wegen Zeitmangel des KN1-Entwicklers, auf Ablehnung. Die Konsequenz daraus war, daß man im MiniDoc-Projekt den KN1-Kern nach und nach durch eigene Module ersetzte, um die Abhängigkeit von KN1 aufzulösen. Es bildeten sich somit zwei funktional identische Systeme, jedoch mit unterschiedlicher Basiskomponente. Diese Entwicklung rief einen doppelten Aufwand für die gleiche Funktionalität hervor.

Kapitel 4

KN 2 - Anforderungen

Dieses Kapitel definiert die Anforderungen an das Nachfolgesystem KN2. Die Anforderungen basieren auf den Ergebnissen der Analyse aus dem letzten Kapitel, den durchgeführten Marktanalysen und den identifizierten Use-Cases.

4.1 Motivation

Das übergeordnete Ziel dieses Projekts ist es, eine Laufzeitumgebung für WWW-Applikationen zur Verfügung zu stellen. Diese Laufzeitumgebung soll anfänglich intern eingesetzt werden, um aufbauend auf den Erfahrungen eine kommerzielle Version zu entwickeln. Für den Anwendungsentwickler sollen eindeutig Business-Prozesse im Vordergrund stehen, weniger die technischen Details der Infrastruktur. Die Architektur soll außerdem so offen gestaltet werden, daß zukünftige Erweiterungen, z.B. die Sicherheits-Aspekte im E-Commerce, einfach möglich sind.

4.2 Use-Cases

Um zielgerichtete Lösungen für die verschiedenen Benutzer zu definieren, ist die Identifikation der Use-Cases von entscheidender Bedeutung. Dabei muß auf eine strikte Trennung geachtet werden, um die unterschiedlichen Anforderungen der einzelnen Benutzergruppen klar herausstellen zu können. Use-Cases spiegeln immer eine bestimmte Sicht der Dinge wider. Die folgenden Anwendungsfälle stellen die für KN2 relevanten Fälle dar.

4.2.1 Administration

Der Anwendungsfall Administration kann in die Fälle System-Administration und Datenpflege aufgeteilt werden (s. Abb. 4.1).

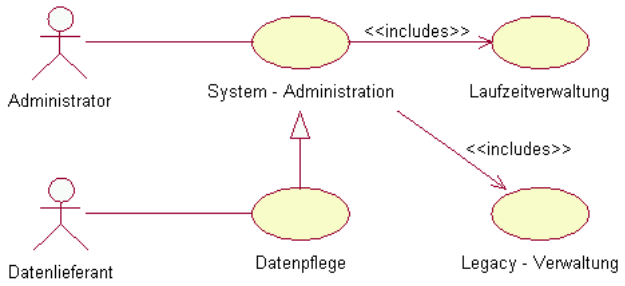

Abbildung 4.1: UML Use-Case: Administration

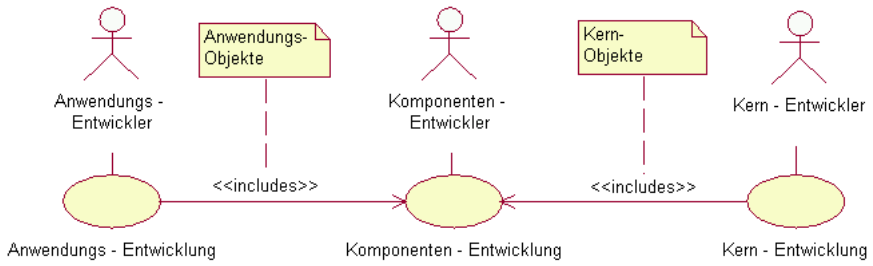

Abbildung 4.2: UML Use-Case: Software-Entwicklung

4.2.1.1 Administrator

Der Anwendungsfall des Administrators beinhaltet die Aufgaben Laufzeitverwaltung und Legacy-Verwaltung. Unter Laufzeitverwaltung sind alle Aufgaben zusammengefaßt, die mit dem Laufzeitsystem der WWW-Applikation in Verbindung stehen. Legacy-Verwaltung bezeichnet alle Systeme, die in irgendeiner Form an die WWW-Applikation angebunden werden.

4.2.1.2 Datenlieferant

Ein Datenlieferant ist der Benutzer einer WWW-Applikation, der die Anwendung mit Daten jeglicher Art versorgt. Dabei handelt es sich auch um einen Aspekt der Administration. Denn der unmittelbare Datenbestand, auf dem die Anwendung basiert, hängt von dieser Benutzergruppe ab.

Am Beispiel von CountryNet sind die Datenlieferanten eine Gruppe von Autoren, die SGML-Dokumente zu bestimmten Themen verfassen. Ganz allgemein wird der einzelne Benutzer der Applikation als Datenlieferant angesehen. Er erstellt Dokumente in MS-Word und bringt diese in die Anwendung ein.

4.2.2 Software-Entwicklung

Der Anwendungsfall der Software-Entwicklung wird in die Unterfälle Anwendungs-Entwicklung, Komponenten-Entwicklung und Kern-Entwicklung aufgeteilt (s. Abb. 4.2). Die horizontale Ausrichtung des Diagramms zeigt gleichzeitig die Abhängigkeit der einzelnen Entwickler voneinander. Ohne die Laufzeitumgebung, die vom Kern-Entwickler implementiert wird, können weder der Komponenten-, noch der Anwendungs-Entwickler ihre Arbeit verrichten. Ohne eine Bibliothek von leistungsfähige Komponenten, kann der Anwendungs-Entwickler den Entwicklungsprozeß nicht verkürzen. Alle drei Entwickler tragen zur Erweiterung der Komponenten-Bibliothek bei.

4.2.2.1 Anwendungs-Entwicklung

Die Anwendungs-Entwicklung zeichnet sich dadurch aus, daß der Entwickler über eine Bibliothek von Komponenten verfügt. Diese setzt er geeignet zusammen, um die gewünschte Anwendungs-Funktionalität zu erreichen. Er kann die Komponentenbibliothek auch selbstständig erweitern, dabei bewegt er sich aber immer auf einer abstrakten Ebene. Er kommt nicht mit den technischen Details der Komponenten-Entwicklung in Kontakt.

4.2.2.2 Komponenten-Entwicklung

Im Fall der Komponenten-Entwicklung erstellen ein oder mehrere Entwickler Software-Komponenten mit einem gewissen Maß an Allgemeingültigkeit. Diese Allgemeingültigkeit wird von der Anwendungsdomäne bestimmt. So benötigt man für E-Commerce Anwendungen andere Komponenten, als beispielsweise für Workflow-Anwendungen.

4.2.2.3 Kern-Entwicklung

Der Kern-Entwickler arbeitet an der Laufzeitumgebung selbst und implementiert die Basisdienste, wie z.B. Transaktions- und Sicherheits-Dienst.

4.3 Projektmanagement

Aus den Erfahrungen, die im Zusammenhang mit KN1 gemacht wurden, muß eine klare Trennung zwischen Kern-Entwicklern und allen anderen Anwendern, also Anwendungs-Entwicklern, Komponenten-Entwicklern und Administratoren erfolgen. Dies ist der wichtigste Punkt, den es bzgl. der Projektorganisation und des Managements zu erreichen gilt. Sollte dieses Ziel nicht erreicht werden, so kann mit Sicherheit gesagt werden, daß sich das Szenario der Projekt-bezogenen Weiterentwicklung wiederholen und damit zu einer Aufweichung der klaren Trennung zwischen Kern- und Anwendungsentwicklern führen wird.

4.4 System-Anforderungen

Eine grundlegende Forderung ist, für Benutzer des Systems alles so einfach wie möglich zu gestalten.

4.4.1 Erweiterbarkeit

Das System soll auf Erweiterbarkeit ausgerichtet sein. In einer ersten Stufe wird sich die Entwicklung von Applikationen auf Basis von KN2 auf Auskunftssysteme konzentrieren. In einer zweiten Stufe sollen z.B. auch E-Commerce-Anwendungen mit erweiterten Sicherheitsaspekten unterstützt werden.

4.4.2 Wiederverwendbarkeit

Wie die Erfahrungen aus anderen Projekten zeigt, wurde das Rad in jedem Projekt neu erfunden, da der Software-Entwicklungsprozeß nicht auf Wiederverwendung ausgerichtet war. Deshalb sollte in zukünftigen Projekten der Idealzustand angestrebt werden, so daß ein Anwendungsentwickler, je nach Funktionalität der zu entwickelnden Applikation, im Idealfall nur noch eine Menge von Komponenten miteinander verbinden muß.

4.4.3 Offenheit

Die Offenheit des System soll nach allen Seiten angestrebt werden.

In KN1 wurde versucht, eine Datenbanktransparenz zu erreichen. In KN2 soll diese Transparenz auch tatsächlich etabliert werden.

Die Beschränkung auf wenige Betriebssystem-Plattformen wird im Umfeld großer Unternehmen, mit heterogener IT-Landschaft, als Einschränkung angesehen. Deshalb ist die Forderung nach Betriebsystemunabhängigkeit wichtig, um das System für solche Bereiche interessant zu machen.

Das System soll außerdem durch Implementationen in beliebigen Programmiersprachen erweiterbar sein.

Kapitel 5

KN 2 - Architektur

In diesem Kapitel wird die globale Architektur des KN1 Nachfolgers beschrieben.

Die Architektur wird durch ein Anwendungsmodell beeinflußt. Deshalb konzentriert sich die folgende Beschreibung auf das Anwendungsmodell und die darin enthaltenen Einheiten, wie Laufzeitsystem, Präsentationsmodule und Komponenten.

5.1 KN2 Application Programming Model

Das Anwendungsmodell definiert die Design- und Implementationsrichtlinien für Applikationen. Dazu gehören die Separation zwischen Applikationslogik und Präsentationslogik, sowie die Partitionierung der Applikationslogik in funktionale und binäre Komponenten.

Beim *KnowledgeNow2 Application Programming Model (KN2 APM)* (s. Abb. 5.2) handelt es sich um das *Java2 Enterprise Edition Application Programming Model (J2EE APM)* (s. Abb. 5.1), das jedoch um die folgenden Punkte erweitert bzw. modifiziert wurde:

- Einer Applikation auf der Seite des Klienten wird der direkte Zugriff auf die Ebene der Business-Objekte erlaubt, ohne daß der Umweg über die Präsentationsschicht gemacht werden muß. Dies ist sinnvoll, da entweder die Präsentationslogik in dieser Art von Applikationen bereits integriert ist oder gar keine GUI verwendet wird.

- Die Präsentationsebene wird um einen Server-seitigen XSL-Prozessor erweitert. Die XSL-Technologie fördert die Trennung zwischen Daten und Inhalt und folgt damit dem vom Anwendungsmodell geforderten Separationsprinzip.

- An Stelle von EJB wird das *CORBA Component Model (CCM)* verwendet. Dies wird bestimmt durch die Forderung nach beliebiger Wahl

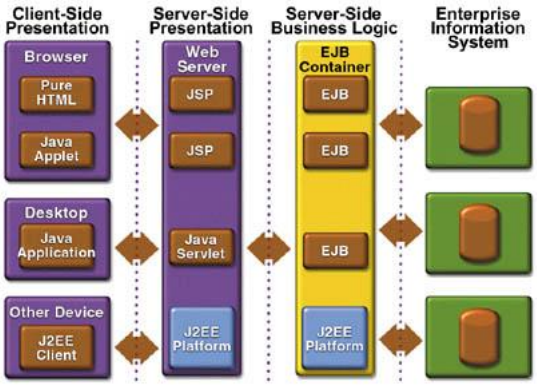

Abbildung 5.1: J2EE Application Programming Model (J2EE APM)

der Implementationsprache. EJB unterstützt nur Java als Implementationssprache. Dagegen erlaubt das CCM jede Sprache, für die es eine Abbildung auf CORBA-IDL gibt.

- Das Modul „Other Devices“ fällt weg, da eine Unterscheidung zwischen mobilen Geräten und Desktop-Rechnern in diesem Zusammenhang keinen Sinn macht. Ein mobiles Gerät verfügt entweder über einen, wenn auch abgespeckten, Browser oder die Applikation auf dem mobilen Gerät greift nur auf die Dienste zu und verwendet ihr eigenes GUI.

Der Grund für das J2EE APM als Grundlage der Architektur ist, daß weder die OMA, noch das CCM ein Anwendungsmodell für WWW-Applikationen anbieten. Die OMA definiert zwar die *User-Interface Facility (UIF)*[1]. Allerdings bietet die UIF kein umfassendes Programmiermodell an, sondern sie identifiziert Objekte, die mit dem GUI des Klienten in Verbindung stehen.

5.1.1 Klienten

Das Anwendungsmodell unterscheidet zwischen Applikationen, die Applikationslogik auf der Klienten-Seite enthalten (Fat-Client) und Browsern, welche die Klienten-Plattform nur zur Präsentation verwenden (Thin-Client). Applikationen können mit oder ohne Benutzer-Interface ausgestattet sein. Die letzte Kategorie ist nicht auf den typischen Browser eines Arbeitsplatzrechners beschränkt, sondern sie beinhaltet auch mobile Geräte, wie zum Beispiel Mobiltelefone. Natürlich muß der Art des Klienten seitens der Anwendung Rechnung getragen werden, da es z.B. keinen Sinn macht große Bilddateien an Mobiltelefone zu versenden.

[1] [COR98b], Last Update: 14. November 1995

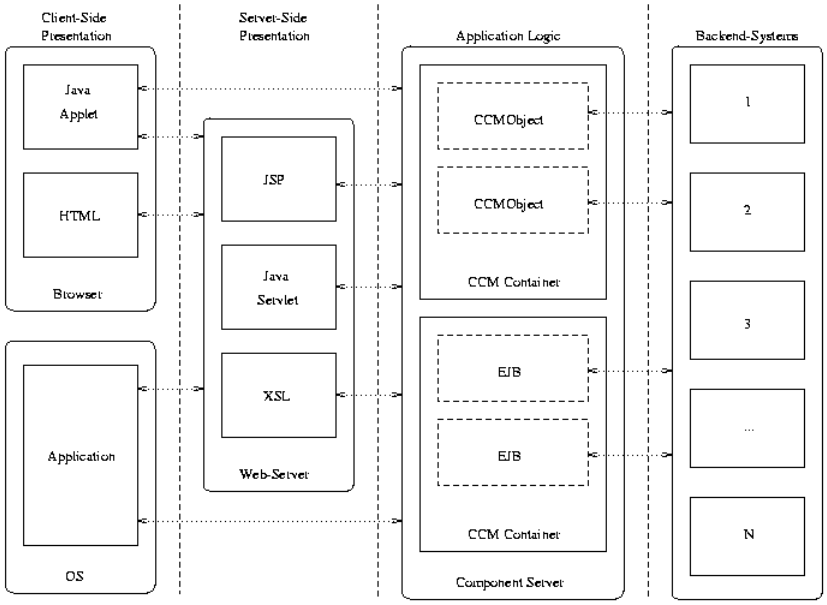

Abbildung 5.2: KN2 Application Programming Model (KN2 APM)

5.1.2 Web-Server

Hier wird der Begriff Web-Server, im Gegensatz zu HTTP-Server, verwendet, da der Server nicht nur reines HTML, sondern auch Java-Servlets und JSP ausführen kann. Zusätzlich wurde der Server um einen XSL-Prozessor erweitert, um unabhängig von der XSL-Implementation des Klienten, XSL Stylesheets anwenden zu können.

Java-Servlets können als Server-seitige Applets betrachtet werden. Der Zugriff auf Servlets erfolgt nach einem URL-Schema. Gegenüber dem Klienten erscheinen die Servlets wie jede andere Ressource. Ein Servlet ist eine Java-Klasse und erbt von einer der abstrakten Basisklassen:

- `javax.servlet.GenericServlet`
- `javax.servlet.HttpServlet`

Das `GenericServlet` bietet allgemeine Unterstützung für beliebige Protokolle. Um einen Request zu behandeln, muß lediglich die `Service()`-Methode implementiert werden. Bei einem Request wird dann diese Methode aufgerufen und kann so auf die Anfrage reagieren. Die Klasse `HttpServlet` ist eine Spezialisierung des GenericServlet und bietet Unterstützung für das HTTP-Protokoll. Um z.B. auf einen GET-Request reagieren zu können, muß die `doGet()`-Methode überschrieben werden.

Java ServerPages stellen in gewissem Sinne eine Vereinfachung der `HttpServlets` dar. Bei einer Java ServerPage handelt es sich um eine HTML-Datei mit eingebetteten Java Anweisungen. Diese Seite wird von der JSP-Engine in ein Servlet übersetzt und bei einem Request wird dieses Servlet ausgeführt. Gute JSP-Engines erkennen automatisch Veränderungen am JSP-Quelltext und führen die Rekompilierung nur beim jeweils ersten Zugriff nach der Änderung durch. Dies hat zum Vorteil, daß folgende Anfragen schneller behandelt werden können, da das Servlet bereits kompiliert ist. Die Kompilierung ist sehr einfach. Die HTML-Tags werden in `out.println()`-Anweisungen eingefaßt und die Java-Anweisungen werden direkt in die `doGet`- oder `doPost`-Methode des Servlet-Quelltexts übernommen.

Als Richtlinie für die Verwendung von Java-Servlets und JSP gibt das J2EE APM an, daß in solchen Komponenten nur Präsentationslogik implementiert werden soll. Unter keinen Umständen sollte dort Applikationslogik enthalten sein, da dies eine Aufweichung des Separationsparadigmas zur Folge hätte.

5.1.3 Komponenten-Laufzeitumgebung

Das CCM ist ein Rahmenwerk für die Ausführung von Software-Komponenten. Eine solche Laufzeitumgebung muß über präzise definierte Schnittstellen verfügen, damit Komponenten verschiedener Autoren problemlos zusammenarbeiten.

Das CCM teilt die Komponenten in zwei Klassen ein:
einfache- (basic) und erweiterte- Komponenten (extended). In die Kategorie der einfachen Komponenten fallen EJB und CORBA-Objekte, die nicht nach dem CCM entwickelt wurden. Dies stellt eine Möglichkeit dar, existierende CORBA- oder EJB-Objekte in die CCM-Laufzeitumgebung einzubringen, ohne Veränderungen an der Komponente durchführen zu müssen. Allerdings bleibt es den einfachen Komponenten verwehrt, auf die erweiterten Funktionalitäten der Laufzeitumgebung zuzugreifen.

Das CCM teilt auch die Klienten einer Komponente in zwei Kategorien: Komponenten-bewußt (component-aware) und unbewußt (component-unaware). Ein Klient ist Komponenten-bewußt, wenn er weiß, daß es sich um eine Komponente nach dem CCM handelt. Ist dies der Fall, dann kann er natürlich von den erweiterten Eigenschaften Gebrauch machen. Ist dies nicht der Fall, dann erscheint ihm die Komponente wie ein gewöhnliches CORBA-Objekt.

Der Rest des Abschnitts gibt einen groben Überblick über die Erweiterungen des CCM gegenüber CORBA.

Die CCM-Komponente erbt von einem neuen Meta-Typ CCMObject, welcher den Typ CORBA::Object um verschiedene Ports (Anschlußmöglichkeiten) erweitert. Zu diesen Anschlußmöglichkeiten zählen Facets, Recep-

tacles und Events. Facets sind zusätzliche Schnittstellen einer Komponente. Ein Komponenten-bewußter Klient kann mit Hilfe des Equivalent-Interface zwischen den verschiedenen Facets navigieren. Mit Hilfe der Receptacles ist es möglich, die Beziehungen zwischen Komponenten zu beschreiben. Damit lassen sich 1:0, 1:1 oder 1:N Beziehungen darstellen. Die Ereignisse teilen sich in die Unterkategorien Sources und Sinks. Es gibt weiterhin zwei verschiedene Typen von Quellen, Emitter und Publisher. Das Emitter-Modell erlaubt beliebig viele Sender, aber nur einen Empfänger. Das Publisher-Prinzip erlaubt nur einen aktiven Sender, aber viele verschiedene Empfänger.

Jede Komponente verfügt zur Laufzeit über eine Home-Interface. Diese Schnittstelle stellt die Lebenszyklus-Funktionen eines bestimmten Komponenten-Typs zur Verfügung. Ein Home-Interface wird für jeden Komponenten-Typ innerhalb eines Containers erzeugt.

Das *Component Implementation Framework (CIF)* stellt das Programmiermodell zur Erstellung von Komponenten-Implementationen bereit. Dazu verfügt es über eine deklarative Sprache, die *Component Implementation Definition Language (CIDL)*. In dieser werden die Komponenten und Home-Interfaces definiert. Vom CIDL-Compiler wird verlangt, daß automatisch Quelltexte erzeugt werden, so daß der Programmierer nur noch die Applikationslogik einfügen muß. CIDL ist eine Obermenge der *Persistent State Definition Language (PSDL)*, welche die Definition von Beziehungen zwischen dem internen Zustand und dem persistenten Datentyp einer Komponente erlaubt.

Als Executor wird das Programmfragment bezeichnet, welches das Verhalten der Komponente implementiert. „Home Executor“ bezeichnet das Fragment, das das Home-Interface implementiert. Das CIF kann automatisch Standardimplementationen dieser Executors anbieten. Ein Executor unterteilt sich in die Kategorien *monolithic* und *segmented*. Eine monolitische Komponente wird immer vollständig aktiviert bzw. deaktiviert. Für eine segmented Komponente können die einzelnen Segmente getrennt aktiviert bzw. deaktiviert werden.

Das *Container Programming Model* teilt die Laufzeitumgebung des Servers in einzelne *Container* auf. Jeder Container unterstützt Komponenten einer bestimmten *category*. Gültige Kategorien sind: Service, Session, Process, Entity, EJBSession, EJBEntity and Empty. In die Service-Kategorie fallen zustandslose Komponenten. In der Session-Kategorie sind Komponenten mit transientem Zustand enthalten. In die Process-Kategorie fallen Komponenten, die einen Zugriff auf Datenquellen kapseln. Die Komponenten der Entity-Kategorie teilen sich die Verantwortung über den Datenzugriff mit dem Klienten. Die beiden EJB-Kategorien beinhalten die entsprechenden EJB-Komponenten. Die Empty-Kategorie bietet keine Hilfen bzgl. Zustandsmanagement, sie kann zur Implementation weiterer Benutzer-definierter Kategorien verwendet werden. Die Laufzeitumgebung stellt innerhalb des Modells interne, externe und Container-APIs zur Verfügung. Die internen und

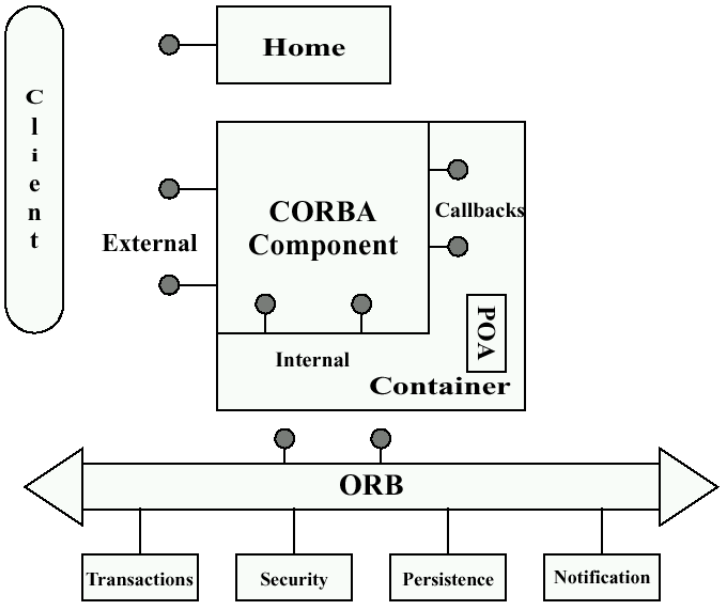

Abbildung 5.3: CCM Container Model

die Container-APIs werden vom Komponenten-Entwickler verwendet. Unter den externen APIs versteht man die Schnittstelle der Komponente zum Klienten (s. Abb. 5.3).

Die Container werden durch einen *Component-Server* kontrolliert. Ein Komponenten-Server beinhaltet neben der Container-Verwaltung noch die CORBA Object Services Transaction, Security, Persistence und Notification (s. Abb. 5.4). Der Notification ist eine Art Event-Service, der jedoch mit hohem Datenvolumen mittels Event-Filter umgehen kann. Das *CORBA Usage Model* definiert die Beziehungen der Container mit CORBA. Dabei unterscheidet es zwischen *stateless*, *conversational* und *durable*. Die Fähigkeiten einer Komponente mit Transaktionen umzugehen und die Persistenz zu handhaben untergliederen sich in self-managed und conatiner-managed.

Ein weiterer Punkt, der vom CCM abgedeckt wird, ist das Packaging und Deployment. Komponenten und die dazugehörigen Dateien werden in sog. Paketen zusammengefaßt. Eine Datei im XML-Format innerhalb dieses Pakets gibt Auskunft über den Inhalt und die Abhängigkeiten zu anderen Paketen. Dadurch ist es möglich automatische Installationen von Paketen durchzuführen. Eine Installation wird mit einer Reihe von Objekten durchgeführt. Das wichtigste Objekt davon ist die AssemblyFactory. Diese Factory muß auf jedem Rechner instantiiert sein, auf dem eine Installation von Komponenten möglich sein soll. Diese Factory erzeugt das Assembly-Objekt. Dieses wiederum ist dafür verantwortlich einen Component-Server und darin

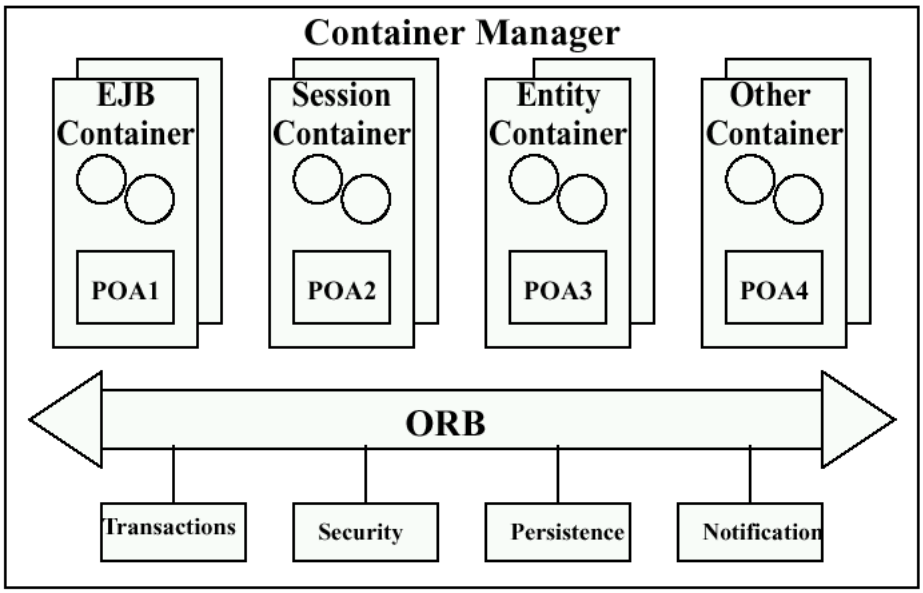

Abbildung 5.4: CCM Component Server

einen Component-Container der entsprechenden Kategorie zu instantiieren. Der Container erzeugt daraufhin das Component-Home und schließlich die Komponente selbst.

In Tabelle 5.1 ist die Zuordnung der KN2 Use-Cases zu den Rollen in EJB und den Prozessen im CCM zu sehen. Der Grund für die Auflistung der Rollen des EJB-Modells ist, daß EJB eine personalisierte Sicht der Aufgaben vertritt. Dementsprechend sind auch die Bezeichnungen der Rollen gewählt. Diese personalisierte Sicht entspricht den identifizierten KN2 Use-Cases und dient hier der besseren Zuordnung zwischen den Aufgaben und den Rollen. Das Fehlen einer entsprechenden Rolle für Server und Container deutet darauf hin, daß in KN2 die Ausführungsplattform nur ein Hilfsmittel ist, auf der die Komponenten ablaufen. Sie spielt eine untergeordnete Rolle und kann bei Bedarf ausgetauscht werden.

5.2 Komponenten

Das Ziel im Zusammenhang mit Komponenten ist die Erstellung einer Bibliothek, die sich beliebig durch neue Komponenten erweitern läßt. Um diese kontinuierliche Erweiterung der Bibliothek zu erreichen, muß in Zukunft ein Komponenten-orientierter Software-Entwicklungsprozeß angestrebt werden. Dieser hat zum Ziel Komponenten zu identifizieren und diese als eigenständige Komponenten zu implementieren.

EJB	**CCM**	**KN2**
Bean Provider	Declaration Implementation Packaging	Komponenten-Entwickler
Application Assembler	Assembly	Anwendungs-Entwickler
Deployer	Deployment Installation	Anwendungs-Entwickler Administrator
Server Provider	Server	/
Container Provider	Container	/
System Administrator	/	Administrator

Tabelle 5.1: Vergleich der Rollen der verschiedenen Objektmodelle mit den KN2 Use-Cases

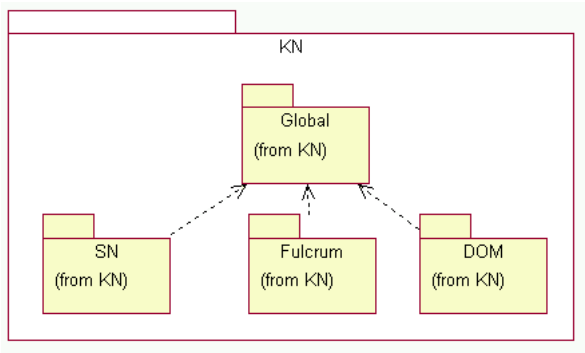

Abbildung 5.5: UML-Package-Diagramm: KN

Ein erstes Ziel bei der Festlegung der zu implementierenden Komponenten ist die KN1-Funktionalität. Ein Anwendungs-Entwickler soll mit KN2 Anwendungen in der Art entwickeln können, wie es mit KN1 möglich war. Dazu müssen in KN2 die gleichen Funktionalitäten zur Verfügung stehen.

Die folgenden Komponenten beruhen also auf den in KN1 identifizierten Funktionalitäten. Abbildung 5.5 zeigt das UML Package-Diagramm der Komponenten-Bibliothek. Das Paket Global beinhaltet allgemeine Klassen, die in den anderen Paketen verwendet werden (s. Abb. 5.6).

In jedem Klassendiagramm kommt jeweils eine Factory-Klasse vor, die vom Basis-Datentyp BaseFactory abgeleitet ist. Diese Factory-Klasse wird im Kapitel 6 erläutert, da es sich dabei um Details der Implementation handelt.

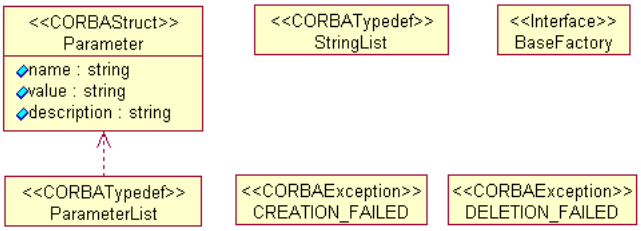

Abbildung 5.6: UML-Klassendiagramm: Global

Paket	Global	
Klasse	Parameter	
Stereotyp	Struct	
Beschreibung	Die Parameter-Klasse entspricht dem CORBA Name-Value-Pair Datentyp (NVP). Sie wurde jedoch um das Attribut description erweitert.	
Attribute	`name`	Name des Parameters
	`value`	Wert des Parameters
	`description`	Textuelle Beschreibung

Paket	Global
Klasse	ParameterList
Stereotyp	Typedef
Beschreibung	CORBA sequence für die Parameter-Klasse.

Paket	Global
Klasse	StringList
Stereotyp	Typedef
Beschreibung	CORBA sequence für den CORBA-Datentyp string.

5.2.1 Document Object Model

Die Document Object Model (DOM) Komponente ist die Implementation des W3C DOM API. Da der Fokus des Systems auf XML liegt, bietet die DOM-Komponente auch nur XML-Untertstützung (DOM Core Level 1). Die HTML-Erweiterungen (DOM HTML Level 1) werden nicht implementiert. Die DOM-Komponente soll immer dann eingesetzt werden, wenn der Anwendungs-Entwickler auf XML-Daten zugreifen muß. Auf die Klassen des DOM wird hier nicht genauer eingegangen. Die genaue Beschreibung kann der DOM Spezifikation [H+98] entnommen werden. Die DOMFactory, die von der Spezifikation nicht abgedeckt ist, wird im nächsten Kapitel beschrieben.

5.2.2 Search Network

Das Konzept der Informationssuche wird in KN2 wieder aufgegriffen, nachdem es in KN1 zum reinen Datenbankzugriff verkommen war. KN2 verfügt deshalb über eine SearchNetwork-Komponente (SN), welche die transparente Suche auf mehreren Datenquellen erlaubt.

Das Problem bei einem solchen Ansatz ist es, die Zugriffsschnittstellen der vielen Datenquellen zu vereinheitlichen. Eine solche Abstraktion von spezifischen Besonderheiten stellt aber zugleich auch eine Einschränkung auf den kleinsten gemeinsamen Nenner dar. Es muß also eine Schnittstelle gefunden werden, die sich auf möglichst viele Datenquellen anpassen läßt. Diese Vorgehensweise wird durch die Kombination des „Proxy-Patterns“ und des „Facade-Patterns“ abgedeckt, welche den weiteren Überlegungen zugrunde liegen.

Das Proxy-Pattern bietet einen Ansatz für die Repräsentation von entfernten Objekten in einem lokalen Rahmen. Dies bedeutet für das SearchNetwork, daß für jede Datenquelle, die angesprochen werden soll, ein lokaler Stellvertreter, also ein Proxy, eingesetzt wird.

Das Facade-Pattern bietet eine Musterlösung für die Vereinheitlichung von Schnittstellen. Dazu wird ein übergeordnetes Interface verwendet, das von den Spezifika der untergeordneten Interfaces abstrahiert. Im Falle des SearchNetworks kann das Interface auf möglichst viele Datenquellen angepaßt werden.

Diese Kombination der Entwurfsmuster führt zu der folgenden Lösung: Ein SearchNetwork ist ein Verbund von Datenquellen. Die Festlegung der Kriterien für die Einrichtung eines Verbundes bleibt dem Anwendungs-Entwickler und dem Administrator überlassen. Jede Datenquelle in diesem Verbund wird durch ein Proxy-Objekt repräsentiert. Das Proxy-Objekt verfügt, als Ergebnis des Facade-Patterns, über die einheitliche Schnittstelle.

Ein weiteres Entwurfsmuster löst das Problem die verschiedenen Proxy-Instanzen eines Verbundes anzusprechen. Das „Mediator-Pattern“ bietet ei-

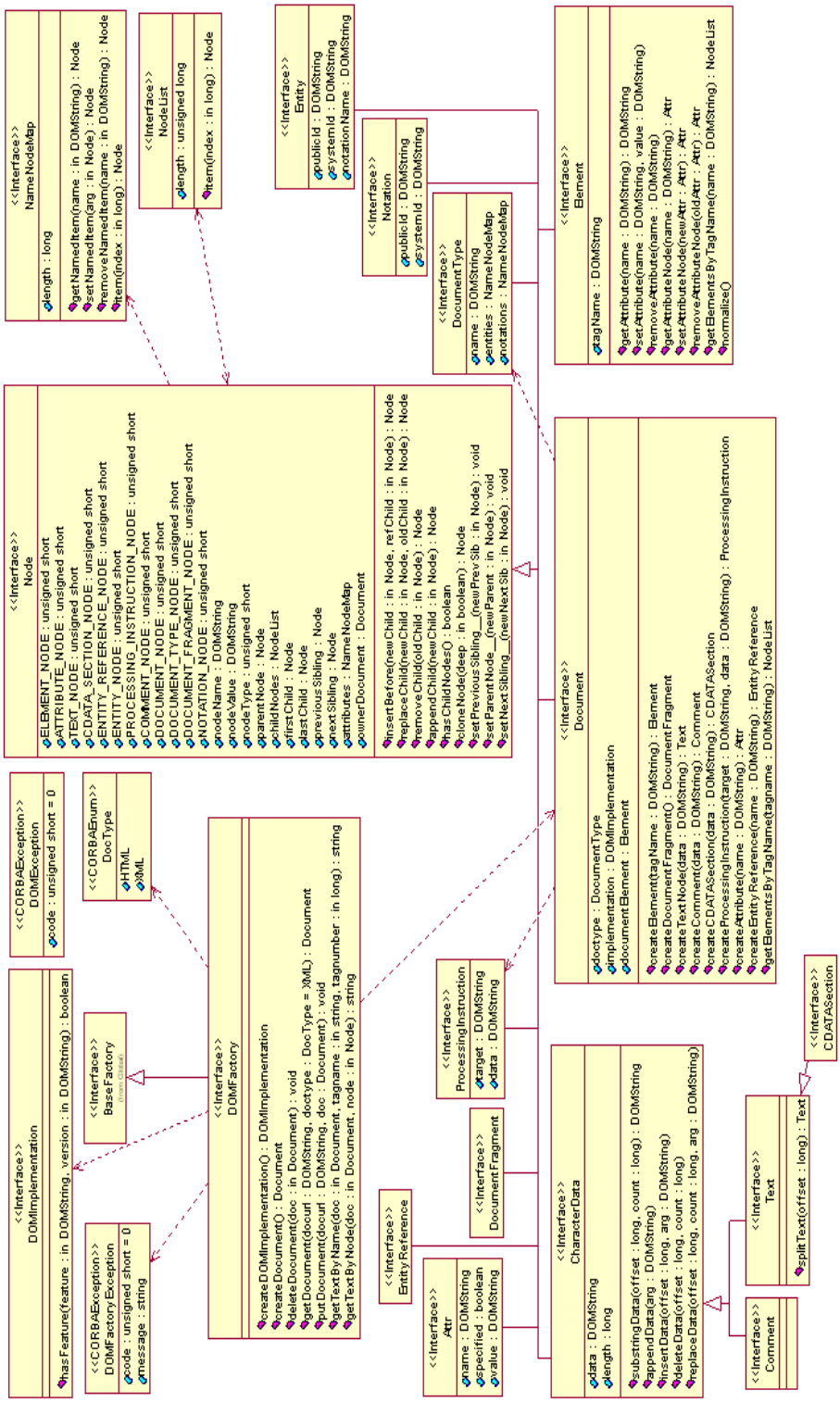

Abbildung 5.7: UML-Klassendiagramm: DOM

ne zentrale Instanz beim Zugriff auf eine Gruppe von Objekten. Im Falle des SearchNetworks verwaltet ein Manager-Objekt eine Gruppe von Proxy-Objekten. Dazu verfügt der Manager über die gleiche Schnittstelle wie die Proxy-Objekte. Der Zugriff auf einen solchen Verbund erfolgt über das Manager-Objekt, das die Anfrage an die Teilnehmer des Verbundes weiterleitet und die Kommunikation koordiniert. Das hat für einen Klienten des SearchNetworks den Vorteil, daß er sich nicht mit diesen Koodinationsaspekten auseinandersetzen muß, sondern nur mit dem Mediator des Netzwerks (Manager) kommuniziert.

Der Unterschied zwischen den Schnittstellen SimpleSearch und SQLSearch ist, daß das SimpleSearch-Interface nur das kleinste gemeinsame Vielfache darstellt. Es orientiert sich dabei an den Meta-Suchmaschinen im Internet[2]. Ein solches Interface stellt sich als ungeeignet heraus, wenn auf SQL-Datenbanken zugegriffen werden muß. Deshalb bietet das SQLSearch-Interface ein spezielleres Interface für SQL-Datenbanken.

[2] www.metacrawler.com

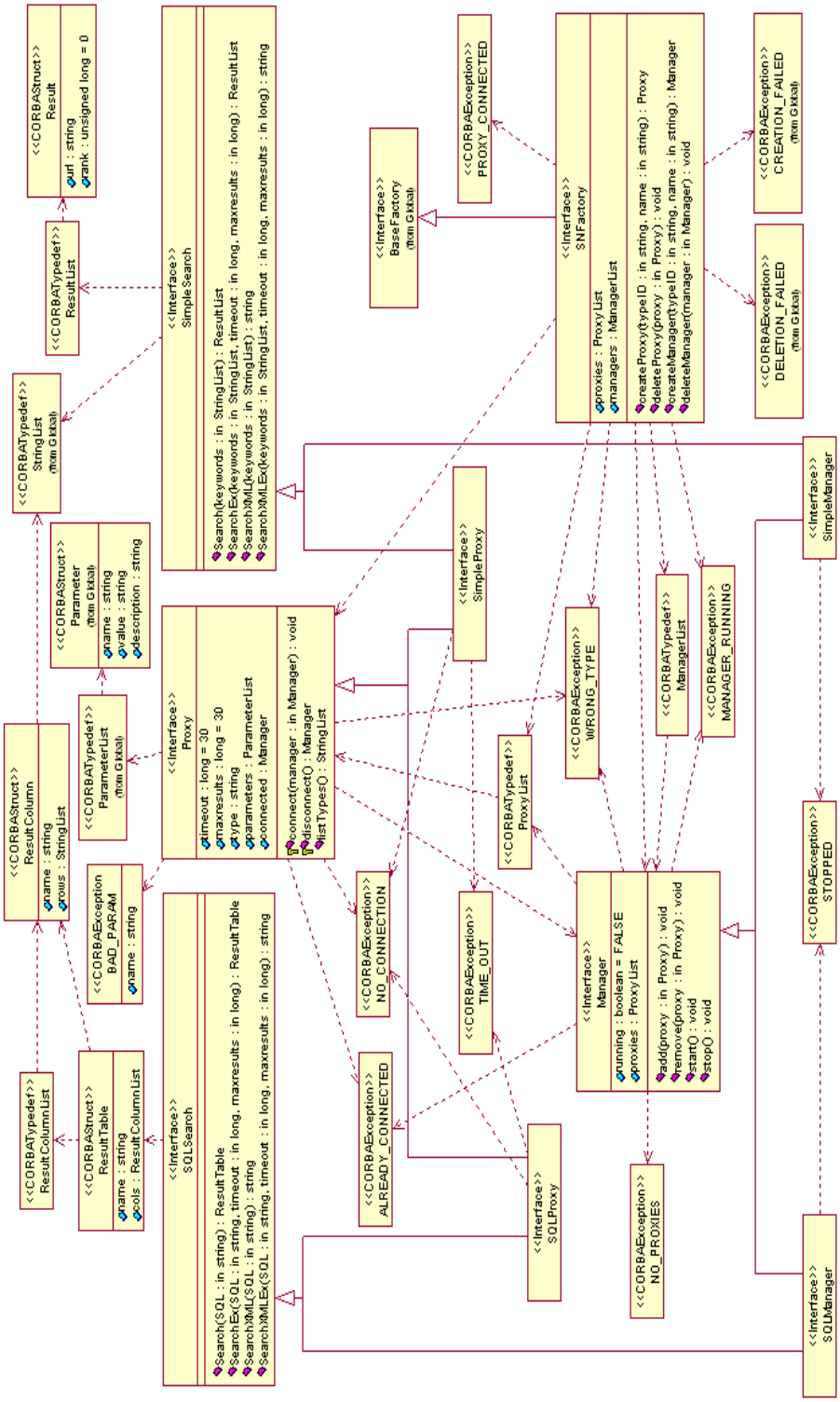

Abbildung 5.8: UML-Klassendiagramm: SearchNetwork

Klasse	Proxy	
Stereotyp	Interface	
Beschreibung	Das Proxy Interface bietet Zugriff auf die administrativen Funktionen eines Proxy-Objekts.	
Attribute	`timeout`	Dauer in Sekunden, die eine Anfrage an die Datenquelle maximal dauern darf.
	`maxresults`	Maximale Anzahl von Ergebnissen, die eine Datenquelle zurückliefern darf.
	`type`	Konfigurationstyp des Proxies. Dieser Typ legt fest, welches Interface der Proxy anspricht.
	`parameters`	Liste der Konfigurationsparameter dieses Proxies.
	`connected`	Referenz auf das Manager-Objekt, mit dem dieser Proxy verbunden ist.
Methoden	`connect`	Stellt die Verbindung zwischen diesem Proxy und dem Manager her. Das Zugriffsattribut dieser Methode ist privat, da die add-Funktion des Managers verwendet werden soll, um Proxy und Manager miteinander zu verbinden. Diese ruft dann implizit die connect-Methode des zu verbindenden Proxies auf. Es wird eine WRONG_TYPE Exception ausgeworfen, wenn der Konfigurationstyp der zu verbindenden Objekte nicht übereinstimmt. Falls der Proxy bereits mit einem Manager verbunden ist, wird ein erneuter Versuch mit einer ALREADY_CONNECTED Exception quittiert.
	`disconnect`	Trennt die Verbindung zwischen diesem Proxy und dem Manager-Objekt. Aus den gleichen Gründen wie die connect-Methode ist auch das Zugriffsattribut der disconnect-Methode privat. Falls der Proxy nicht verbunden ist wird eine NO_CONNECTION Exception beim Aufruf dieser Methode ausgeworfen.
	`listTypes`	Liste von Strings, die Auskunft über die möglichen Konfigurationstypen geben.

Klasse	ProxyList	
Stereotyp	Typedef	
Beschreibung	Die ProxyList ist eine CORBA sequence vom Typ Proxy.	

Klasse	Manager	
Stereotyp	Interface	
Beschreibung	Das Manager Interface bietet Zugriff auf die administrativen Funktionen eines Manager-Objekts.	
Attribute	`running`	Zustand des Managers, aktiv oder inaktiv.
	`proxies`	Liste der mit diesem Manager verbundenen Proxies.
Methoden	`add`	Fügt einen Proxy zur internen Liste hinzu. Eine WRONG_TYPE Exception wird ausgeworfen, wenn der Konfigurationstyp der zu verbindenden Objekte nicht übereinstimmt.
	`remove`	Entfernt einen Proxy aus der internen Liste. Beim Versuch den letzten Proxy aus einem aktiven Manager zu entfernen wird eine MANAGER_RUNNING Exception ausgeworfen.
	`start`	Versetzt den Manager in den aktiven Zustand, d.h. Anfragen von Klienten werden in diesem Zustand akzeptiert. Diese Methode ist nur erfolgreich, wenn die Liste der Proxies nicht leer ist. Andernfalls wird eine NO_PROXIES Exception ausgeworfen.
	`stop`	Versetzt den Manager in den inaktiven Zustand.

Klasse	Result	
Stereotyp	Struct	
Beschreibung	Result ist eine Struktur zur Repräsentation eines Suchergebnisses.	
Attribute	`url`	URL des Suchergebnisses.
	`rank`	Bewertung der Relevanz des Suchergebnisses. Rank ist Null, falls die Suchmaschine keinen solchen Wert zurückliefert.

Klasse	ResultList	
Stereotyp	Typedef	
Beschreibung	Die ResultList ist eine CORBA sequence vom Typ Result.	

Klasse	SimpleSearch	
Stereotyp	Interface	
Beschreibung	Das SimpleSearch Interface stellt das übergeordnete Interface (Facade-Pattern) für die einfache Suche zur Verfügung.	
Methoden	`Search`	Sucht nach den übergebenen Schlüsselwörtern und liefert eine Liste mit den gefundenen Ergebnissen zurück.
	`SearchEx`	Wie die Search-Funktion, nur überschreiben die timeout und maxresults Parameter die Attribute timeout und maxresults des Proxy-Interfaces.
	`SearchXML`	Wie die Search-Funktion. Das Ergebnis wird jedoch als XML-Dokument zurückgeliefert.
	`SearchXMLEx`	Wie die SearchEx-Funktion. Die Rückgabe erfolgt jedoch im XML-Format.

Klasse	ResultColumn	
Stereotyp	Struct	
Beschreibung	ResultColumn ist eine Struktur zur Repräsentation einer Spalte der Ergebnistabelle.	
Attribute	`name`	Name der Spalte.
	`rows`	Inhalt des Spaltenvektors.

Klasse	ResultColumnList	
Stereotyp	Typedef	
Beschreibung	Die ResultColumnList ist eine CORBA sequence vom Typ ResultColumn.	

Klasse	ResultTable	
Stereotyp	Struct	
Beschreibung	ResultTable ist eine Struktur zur Repräsentation der Ergebnistabelle.	
Attribute	`name`	Name der Tabelle.
	`cols`	Liste von Spaltenvektoren.

Klasse	SQLSearch	
Stereotyp	Interface	
Beschreibung	Das SQLSearch Interface stellt das übergeordnete Interface (Facade-Pattern) für die Suche auf SQL-Datenbanken zur Verfügung.	
Methoden	`Search`	Führt das übergebene SQL-Kommando aus und gibt die Ergebnistabelle zurück.
	`SearchEx`	Wie die Search-Funktion, nur überschreiben die timeout und maxresults Parameter die Attribute timeout und maxresults des Proxy-Interfaces.
	`SearchXML`	Wie die Search-Funktion. Das Ergebnis wird jedoch als XML-Dokument zurückgeliefert.
	`SearchXMLEx`	Wie die SearchEx-Funktion. Die Rückgabe erfolgt jedoch im XML-Format.

Klasse	SimpleProxy
Stereotyp	Interface
Beschreibung	Das SimpleProxy-Interface vereinigt durch Mehrfachvererbung die administrativen Funktionen des Proxy- und die Such-Funktionen des SimpleSearch-Interfaces. Die Such-Funktionen werfen eine NO_CONNECTION Exception aus, wenn ein Klient versucht direkt einen Proxy anzusprechen und dieser nicht mit einem Manager verbunden ist. Die Methoden werfen außerdem eine TIMEOUT Exception aus, wenn die Anfrage an die Datenquelle nicht im vorgegebenen Zeitraum abgeschlossen werden konnte.

Klasse	SimpleManager
Stereotyp	Interface
Beschreibung	Das SimpleManager-Interface vereinigt durch Mehrfachvererbung die administrativen Funktionen des Manager- und die Such-Funktionen des SimpleSearch-Interfaces. Im inaktiven Zustand endet eine Anfrage im Auswurf einer STOPPED Exception.

Klasse	SQLProxy
Stereotyp	Interface
Beschreibung	Das SQLProxy-Interface vereinigt durch Mehrfachvererbung die administrativen Funktionen des Proxy- und die SQL-Funktionen des SQLSearch-Interfaces. Die Such-Funktionen werfen eine NO_CONNECTION Exception aus, wenn ein Klient versucht direkt einen Proxy anzusprechen und dieser nicht mit einem Manager verbunden ist. Die Methoden werfen außerdem eine TIMEOUT Exception aus, wenn die Anfrage an die Datenquelle nicht im vorgegebenen Zeitraum abgeschlossen werden konnte.

Klasse	SQLManager
Stereotyp	Interface
Beschreibung	Das SQLManager-Interface vereinigt durch Mehrfachvererbung die administrativen Funktionen des Manager- und die SQL-Funktionen des SQLSearch-Interfaces.

5.2.3 Fulcrum Repository

Das Fulcrum Repository (FR) ersetzt die administrativen Schnittstellen zum Fulcrum SearchServer aus KN1. Dort gibt es diverse Shell-Skripte, die den Indizierungsvorgang kontrollieren. Diese Indizierungsskripte wurden während einer COM-Evaluation zwar durch mehrere COM-Objekte ersetzt, allerdings sollte in KN2 auf COM, aufgrund der bekannten Nachteile, verzichtet werden.

Das Fulcrum-Repository ist ein Wrapper-Objekt für den Fulcrum SearchServer. Zum Betrieb eines FR-Objekts muß der SearchServer, inkl. JDBC-Treiber[3], korrekt installiert sein. In Abhängigkeit vom Dokumentformat der zu indizierenden Dokumente muß außerdem der entsprechende TextReader installiert sein.

Die administrativen Aufgaben bezüglich des FRs sind: Erstellen und Löschen eines Repositories, Konfiguration des Repositories mit Hilfe von Konfigurationsdateien und Verwalten der darin gespeicherten Dokumente.

Ein Repository ist eine Ansammlung von zusammengehörigen Dokumenten und entspricht einem Index im SearchServer. Sollen in einer Anwendung mehrere Indizes verwendet werden, dann müssen verschiedene Repositories angelegt werden. Sollte eine Aufteilung eines Index aufgrund seiner Größe sinnvoll sein, so können mehrere Repositories erstellt werden. Die Repositories könnten durch diese Partitionierung auch auf verschiedene Rechner verteilt werden, wobei die Anwendung jedoch die Koordination eines verteilten Repositories übernehmen muß. Eine automatische Verteilung, wie im Falle der SearchNetwork-Komponente ist in dieser Version nicht geplant. Eine solche Erweiterung würde sich jedoch mit Hilfe der Facets des CCM nachträglich leicht hinzufügen lassen, ohne dabei bestehende Installationen zu beeinträchtigen.

[3] enthalten in „Fulcrum SearchBuilder for Java“

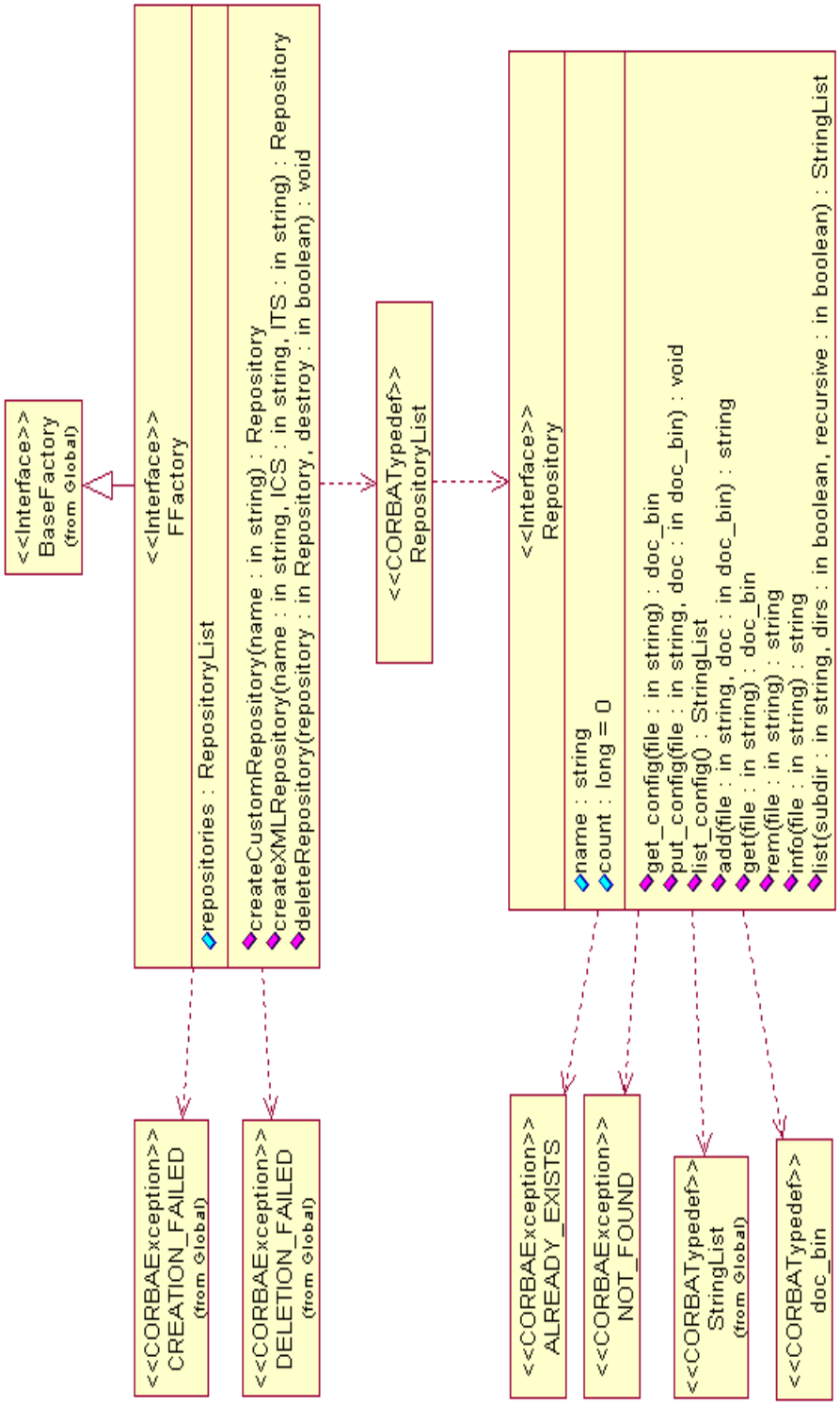

Abbildung 5.9: UML-Klassendiagramm: Fulcrum

Klasse	Repository	
Stereotyp	Interface	
Beschreibung	Das Repository kapselt die administrativen Funktionalitäten des Fulcrum SearchServers.	
Attribute	`name`	Name des Repositories.
	`count`	Anzahl der Dokumente im Repository.
Methoden	`get_config`	Liest eine Konfigurationsdatei aus dem Repository.
	`put_config`	Schreibt eine Konfigurationsdatei in das Repository.
	`list_config`	Gibt eine Liste aller Konfigurationsdateien zurück.
	`add`	Fügt dem Repository ein Dokument hinzu. Wirft eine ALREADY_EXISTS Exception aus, wenn das Dokument bereits existiert.
	`get`	Liest ein Dokument aus dem Repository, ohne dieses zu löschen. Wirft eine NOT_FOUND Exception aus, wenn das Dokument nicht existiert.
	`rem`	Entfernt ein Dokument aus dem Repository. Wirft eine NOT_FOUND Exception aus, wenn das Dokument nicht existiert.
	`info`	Liefert den Zeitpunkt der letzten Indexierung eines Dokuments. Wirft eine NOT_FOUND Exception aus, wenn das Dokument nicht existiert.
	`list`	Gibt eine Liste aller Dokumente im Repository zuück. Wirft eine NOT_FOUND Exception aus, wenn das Verzeichnis nicht existiert.

Klasse	RepositoryList
Stereotyp	Typedef
Beschreibung	Die RepositoryList ist eine CORBA sequence vom Typ Repository.

Klasse	doc_bin
Stereotyp	Typedef
Beschreibung	Der doc_bin Typ ist eine CORBA sequence vom Typ byte. Er dient zur binären Übertragung von Dokumenten.

Kapitel 6

KN 2 - Implementation

In diesem Kapitel wird die Umsetzung des im Kapitel 5 beschriebenen Architekturmodells dargestellt.

Das CCM stellt sich als äußerst umfangreich und komplex dar. Eine Implementation würde sich nur mit einer großen Entwicklungsabteilung sinnvoll realisieren lassen. Eine präzisere Aussage läßt sich nicht treffen. Denn der Umfang hängt davon ab, auf welche existierenden Module zurückgegriffen werden kann und wieviele noch implementiert werden müssen.

Aufgrund der Austauschbarkeit der Laufzeitumgebung liegt der „Wert“ von KN2 nicht in der Laufzeitumgebung, sondern in den Komponenten, die in dieser Umgebung ablaufen. Die Beschreibung der Implementation konzentiert sich in diesem Kapitel daher auch weniger auf die Laufzeitumgebung, als auf die Komponenten selbst.

6.1 Klienten

In Bezug auf die Software seitens des Klienten werden keine Vorgaben gemacht, da dies immer von der zu erstellenden Anwendung abhängt. Als Klient kommt also jede Software in Betracht, die sich entweder an die Präsentationsschicht oder an die Laufzeitumgebung bzw. die darin ablaufenden Komponenten direkt binden kann.

6.2 Web-Server

Der Web-Server unterteilt sich in HTTP-Server, Servlet-Engine, JSP-Engine und XSL-Prozessor.

6.2.1 HTTP-Server

Nach einer Analyse der verfügbaren HTTP-Server fiel die Wahl auf Apache (www.apache.org). Die Gründe werden in der folgenden Liste dargestellt:

- Weite Verbreitung
  (www.netcraft.com/survey)
- Open-Source
  (www.apache.org/LICENSE.txt)
- Umfangreicher Industriesupport
  (xml.apache.org/ack.html)
- Multi-Plattform-Verfügbarkeit
  (www.apache.org/dist/binaries)
- Erweiterbarkeit durch Modul-Architektur
  (SSL, PHP, Perl, JServ, ...)
- Integration von Java und XML-Technologien
  (java.apache.org, xml.apache.org)

6.2.2 Servlet-Engine

Die meisten HTTP-Server können Java-Servlets in Form eines optionalen Moduls, der sog. Servlet-Engine integrieren. Es gibt nur wenige Server, wie z.B. der „Java Web Server“ von Sun, die Servlets direkt unterstützen. Für den Apache bietet sich die JServ Servlet-Engine an, da es sich dabei um ein Apache-Projekt selbst handelt und deshalb die Installation sehr einfach durchzuführen ist[1]. Außerdem unterliegt die JServ-Engine ebenfalls der Apache-Lizenz und ist dadurch frei verfügbar.

6.2.3 JSP-Engine

Neben JSP wurden auch andere Server-seitige Skriptsprachen evaluiert, wie z.B. Embedded Perl und PHP3[2]. Der Nachteil dieser Sprachen ist, daß sie nicht standardisiert sind und es ist abzusehen, daß JSP hier eine Konsolidierung einleiten wird. Die JSP Spezifikation sieht ein Element vor, das die in der Seite verwendete Sprache kennzeichnet. Im Augenblick bezieht sich die Spezifikation zwar nur auf Java, allerdings existiert bereits eine Implementation, die auch Server-seitiges JavaScript zuläßt. JSP schlägt in der Spezifikation also nicht nur Java als bevorzugte Sprache vor, sondern es wird ein Rahmen etabliert, der es möglich macht andere Compiler in diesen Rahmen einzubringen.

Bei der Wahl der JSP-Engine fiel die Entscheidung auf die freie GNUJSP Implementation[3]. Diese Engine zeichnet sich durch die Unterstützung der JSP 1.0 Spezifikation aus. Es werden außer der Apache/JServ-Kombination

[1] java.apache.org
[2] www.cpan.org, www.php.org
[3] www.klomp.org/gnujsp

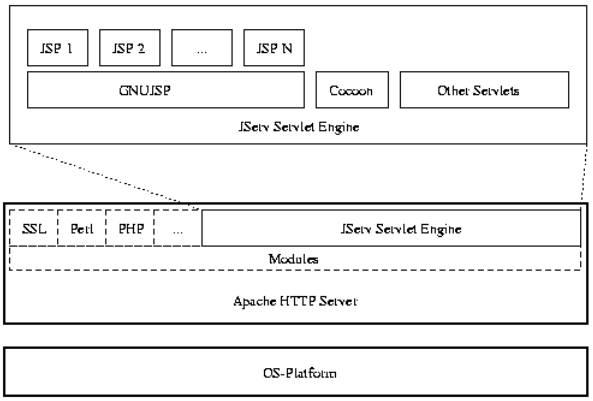

Abbildung 6.1: Apache Architektur-Diagramm

weitere HTTP-Server und Servlet-Engines unterstützt. Die JSP-Engine erkennt automatisch, ob die JSP-Seite modifiziert wurde und kompiliert diese bei Bedarf neu.

Seit Oktober 1999 hat Sun ihre Referenz-Implementation für JSP (JavaServer Web Development Kit (JSWDK)) dem Java-Apache Projekt unter der Apache Lizenz zur Verfügung gestellt[4]. Damit wird in Zukunft eine JSP Implementation unmittelbar mit dem Apache Server verknüpft sein, was die Integration von Java und Apache sicherlich weiter fördern wird.

6.2.4 XSL

Als XSL-Prozessor auf Server-Seite bietet sich das Cocoon-Servlet an, welches ebenfalls aus dem Java-Apache Projekt stammt. Cocoon implementiert das XSL Working-Draft vom 21. April 1999 [Dea99]. Außerdem wurde am 16. November 1999 ein weiteres Unterprojekt, das Apache XML-Projekt gestartet. Die Firmen IBM (xml4j, xml4c, xml4perl), Lotus (LotusXSL) und Sun (ProjectX) stellen die Implementationen ihrer Parser dem XML-Apache Projekt frei zur Verfügung. Das Ziel dieser Spenden ist die nahtlose Integration in den Apache Server. Auch das Cocoon Projekt wird in diese Anstrengungen integriert werden. Es ist zu erwarten, daß der Apache Server dadurch zu einer Server-Plattform mit umfassender XML-Unterstützung wird.

6.3 Laufzeitumgebung

Eine vollständige Implementation des CCM würde das Optimum darstellen. Allerdings ist die Umsetzung dieses komplexen Modells sehr umfangreich

[4]Codename Tomcat, jakarta.apache.org

CCM	**KN2**
Basic- und Extended-Components	CORBA-Objekte
Component Categorie: Session	Transiente Objekt-Referenz
Component Categorie: Entity	Persistene Objekt-Referenz
Ports	/
Component-Server und Component-Container	Betriebssystem
Core-Services (Transaction, Security, Persistence, Notification)	/
Packaging and Deployment	/
Home-Executor	Factories und VisiBroker Smart Agent Architecture

Tabelle 6.1: Vergleich der KN2 Implementation mit dem CCM

und mit den beschränkten Ressourcen in diesem Projekt nicht zu realisieren. Deswegen wird ein anderer Weg gewählt, der nur die für KN2 wichtigen Elemente umfaßt. Die ersten vollständigen CCM-Implementationen werden Mitte bis Ende 2000 erwartet.

In Tabelle 6.1 werden die Funktionalitäten des CCM mit der tatsächlichen Implementation in KN2 gegenübergestellt.

In KN2 werden im Gegensatz zum CCM nur CORBA-Objekte unterstützt. Diese entsprechen den Basic-Components im CCM.

Ports werden von KN2 nicht unterstützt, da es sich dabei um eine Erweiterung des CORBA Standards handelt und dazu die CORBA-Implementation angepaßt werden müßte.

Als Einheit, in der die CCM-Komponenten ablaufen, schreibt die Spezifikation den Component-Server vor. Der Component-Server beinhaltet einen oder mehrere Component-Container. Die Component-Container enthalten schließlich die eigentlichen Komponenten. In KN2 gibt es keine übergeordnete Einheit. Das Betriebssystem bildet hier den entsprechenden Rahmen.

Im CCM werden die Basisdienste Transaction, Security, Persistence und Notification zwingend vorgeschrieben. Die KN2-Laufzeitumgebung ist eine vergleichsweise lose Ansammlung von Komponenten. Die bisher mit KN1 entwickelten Anwendungen benötigten z.B. keinen Transaktionsdienst. Deshalb wird in der Laufzeitumgebung ein solcher Dienst auch nicht unbedingt gefordert. Das System ist aber aufgrund dieser losen Kopplung sehr wohl in der Lage, bestehende Implementationen des COS Transaction zu integrieren und so den Komponenten verfügbar zu machen.

Das CCM definiert einen präzisen Ansatz für die Bildung ausführbarer Einheiten (Packaging) und die anschließende Installation in der Laufzeitum-

gebung (Deployment). KN2 bedient sich hier dem traditionellen Ansatz, der durch den bisherigen CORBA-Standard und das Betriebssystem vorgegeben ist. Die Implementation orientiert sich am Factory-Pattern [GHJV94] und stellt dadurch die Lebenszyklus-Operationen zur Verfügung. Damit lassen sich Objekte auf entfernten Rechnern aktivieren und deaktivieren. Die Factories und die Komponenten müssen auf den Rechner kopiert werden, auf dem sie zum Einsatz kommen sollen. Die Factories müssen anschließend vom Administrator gestartet werden, damit über diese die entfernte Aktivierung und Deaktivierung der Komponenten gesteuert werden kann.

Das UML-Komponenten-Diagramm in Abbildung 6.2 zeigt eine vollständige Installation der Laufzeitumgebung. Mit einer solchen Implementation auf einem Rechner ist es möglich, jede beliebige Komponente auf diesem Rechner zu instantiiern. Es ist aber auch denkbar, daß nur die Factory eines bestimmten Typs auf einem Rechner gestartet wird. Dadurch würden sich auf diesem Rechner natürlich auch nur Komponenten des entsprechenden Typs starten lassen. Das Ziel einer solchen Aufteilung ist es, daß sich beliebig neue Versionen der Komponenten installieren lassen, ohne daß die Factories jedesmal neu gestartet werden müssen.

6.3.1 VisiBroker

Als ORB wurde der VisiBroker von Inprise ausgewählt. Dieser zeichnet sich besonders durch seine Geschwindigkeit gegenüber anderen kommerziellen ORBs, wie z.B. Orbix von IONA, aus [COR98a] [Gro98]. Die Erweiterungen, die das CCM am CORBA Standard durchführt, lassen sich aufgrund dieser Wahl nicht implementieren, da weder die Quelltexte des ORBs, noch die der Zusatzprogramme verfügbar sind. Aus diesem Grund findet eine Umsetzung des CCMs nur auf funktionaler Ebene statt.

Der VisiBroker verfügt über eine Art automatischen Verzeichnisdienst, „Smart Agent Architecture“ genannt. Dieser Dienst wird von „OSAgents“ zur Verfügung gestellt. Diese OSAgents verwalten eine interne Liste aller in ihrem Netzwerksegment verfügbaren Objekte. Ein VisiBroker CORBA-Objekt sendet während der Initialisierung TCP/IP Broadcast-Messages auf Port 14000 (Defaulteinstellung) aus, um so einen OSAgent ausfindig zu machen. Ist kein OSAgent verfügbar, fährt das Objekt mit der normalen Initialisierung fort. Meldet sich ein OSAgent, registriert sich dieses Objekt mit dem OSAgent und wird ab sofort in diesem Verzeichnis der verfügbaren Objekte aufgelistet. Eine Übersicht aller Objekte erhält man von dem Programm `osfind`. Es ist sogar möglich, diesen Verzeichnisdienst über die Subnet-Grenzen hinweg auszudehnen. Jeweils ein OSAgent eines Segments muß die Information, wo der OSAgent des jeweils anderen Segments zu finden ist, in einer Datei erhalten. Mit dieser Information ist es den beiden OSAgents dann möglich, die internen Daten auszutauschen und so die Objekte über Segmentgrenzen hinaus verfügbar zu machen. Das Modul `locserv`

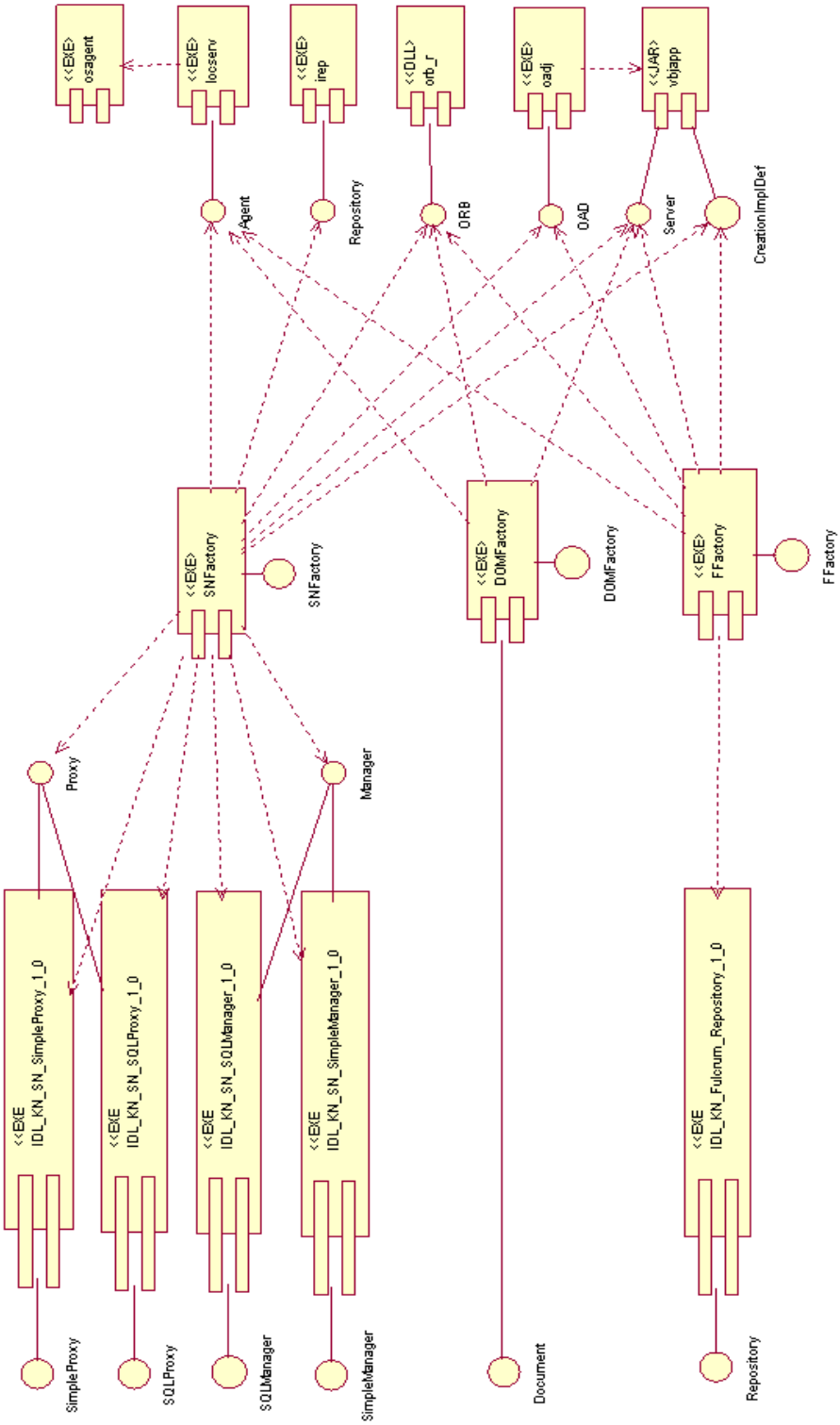

Abbildung 6.2: UML Komponenten-Diagramm der Laufzeitumgebung

stellt die CORBA-Schnittstellen zu diesem Verzeichnisdienst bereit.

Der *Object Activation Demon (OAD)* ist die VisiBroker Implementation des CORBA Implementation-Repository. Der OAD wird eingesetzt, um die automatische Objektaktivierung und Objektdeaktivierung bereitzustellen. Damit eine Instanz automatisch aktiviert bzw. deaktiviert wird, muß sie mit dem OAD registriert werden. Dieser sorgt dann dafür, daß die Instanz bei einer Anfrage automatisch geladen und gestartet wird. Beim Start übergibt der OAD eine Reihe von Parameter an die zu startende Instanz, darunter der Name des Objekts und die Benutzer-definierten Kommandozeilenparameter.

Das VisiBroker *Interface-Repository (IR)* ist eine Standard-konforme Implementation des CORBA Interface-Repositories. Mit dem Kommandozeilen-Programm `irload` wird beim Start der Laufzeitumgebung die Datei `KN.idl` in das Interface-Repository geladen. `KN.idl` enthält die vollständige Schnittstellen-Definition sowohl der Laufzeitumgebung (Factories), als auch der Komponenten.

6.4 Komponenten-Bibliothek

Alle Factories müssen vom Basis-Datentyp BaseFactory abgeleitet werden. Diese Maßnahme ermöglicht das Auffinden von Komponenten-Factories. Ein Klient kann mit Hilfe des Interface-Repositories (IR) herausfinden, von welchen Interfaces ein anderes abgeleitet wurde. Da die Komponenten-Bibliothek in UML-Packages aufgeteilt ist, muß der Klient in allen Sub-Packages, hier SN, DOM und Fulcrum, nach einem Interface suchen, welches von BaseFactory abgeleitet ist. Dadurch ist es möglich alle Komponenten-Factories in der Bibliothek zu finden. Dieser Ansatz funktioniert auch dann, wenn die Bibliothek über die derzeit enthaltenen Komponenten hinaus erweitert wird.

Es macht keinen Sinn, wenn eine Factory auf einem Rechner mehr als einmal gestartet wird. Deshalb wird die Implementation des Singleton-Patterns [GHJV94] vorgeschrieben. Dieses Pattern bedingt, daß es nur eine Instanz eines Objektes gibt und alle Klienten sich mit dieser Instanz verbinden. Hier wird die Forderung auf einen Rechner beschränkt, es darf also nur eine Instanz pro Rechner geben.

Die Implementation des Patterns verwendet den VisiBroker OSAgent und den VisiBroker LocationService, um die Objekte gleichen Typs zu ermitteln. Befindet sich beim Start einer Factory bereits eine Instanz auf dem Rechner, dann bricht die neue Instanz mit einer Fehlermeldung ab.

Klasse	BaseFactory
Stereotyp	Interface
Beschreibung	BaseFactory ist die abstrakte Basis-Klasse. Alle Komponenten Factories müssen von diesem Datentyp abgeleitet werden.

Klasse	DOMFactory	
Stereotyp	Interface	
Beschreibung	Die DOMFactory wird eingesetzt, um Instanzen von XML-Dokumenten zu erzeugen. Eine DOMFactoryException wird ausgeworfen, falls ein Fehler in den Methoden auftritt.	
Methoden	`createDOMImplementation`	Erzeugt ein Objekt vom Typ DOMImplementation.
	`createDocument`	Erzeugt ein Objekt vom Typ Document.
	`deleteDocument`	Löscht ein Objekt vom Typ Document.
	`getDocument`	Liest ein Dokument von einer URL und erzeugt den Baum mit einer Wurzel vom Typ Document.
	`putDocument`	Speichert das Dokument unter einer URL.
	`getTextByName`	Liefert die CDATA Bereiche eines durch `tagname` spezifizierten Teilbaums.
	`getTextByNode`	Liefert die CDATA Bereiche eines durch `Node` spezifizierten Teilbaums.

6.4.1 DOM

Die DOMFactory nimmt eine Sonderstellung unter den Factories ein, da mit ihr nur Objekte mit transienter Objektreferenz erzeugt werden können. Dies entspricht der Session-Kategorie des CCM. Die Objekte werden also nur innerhalb einer Session erzeugt und können nicht mit anderen Sessions ausgetauscht werden. In Verbindung mit dem VisiBroker ORB bedeutet das, daß transiente Objekte nicht mit einem OSAgent registriert werden.

Mit der DOMFactory können Instanzen des DOM-APIs und damit von XML-Dokumenten erstellt und gelöscht werden. In einer ersten Stufe unterstützt das DOM nur HTTP URLs. In einer weiteren Stufe sollen aber auch FTP und FILE ULRs unterstützt werden.

Document Parser	**REC** (156kB, 34% MD)	**chrmed** (873kB, 6% MD)	**med** (1235kB, 33% MD)	**chrbig** (3337kB, 2% MD)	**big** (4934kB, 33% MD)
C-Expat	0,050	0,110	0,380	0,340	1,480
C-Rxp	0,100	0,320	0,740	1,060	2,937
Java-xp	2,400	2,693	4,770	4,010	12,587
Java-xml4j	3,033	3,470	6,770	5,280	19,230
Perl	1,413	3,420	8,410	10,750	32,357
Python	1,650	4,797	12,893	15,893	48,473

Tabelle 6.2: Benchmarkergebnisse aktueller XML-Parser (Ergebnisse in Sekunden, MD = Markup Density)

6.4.1.1 Expat

Die XML-Fähigkeiten der DOM-Komponente werden vom Expat-Parser bereitgestellt.

Der Quelltext des vollständig in C geschriebenen Expat Parsers unterliegt der GNU General Public License (GPL) und ist damit frei verfügbar. Der Parser benötigt jeweils einen Event-Handler für Start-Tags und einen Handler für Stop-Tags.

Der Benchmark (s. Tab. 6.2) zeigt eindeutig, daß die beiden C-Implementation die höchste Performance aufweisen. Die Java-Parser belegen das Mittelfeld und die Skript-Parser, die beide den Expat selbst verwenden, bilden das Performance-Schlußlicht. Die Beschreibung des Benchmarks und die vollständige Auswertung kann man in [Coo99] nachlesen.

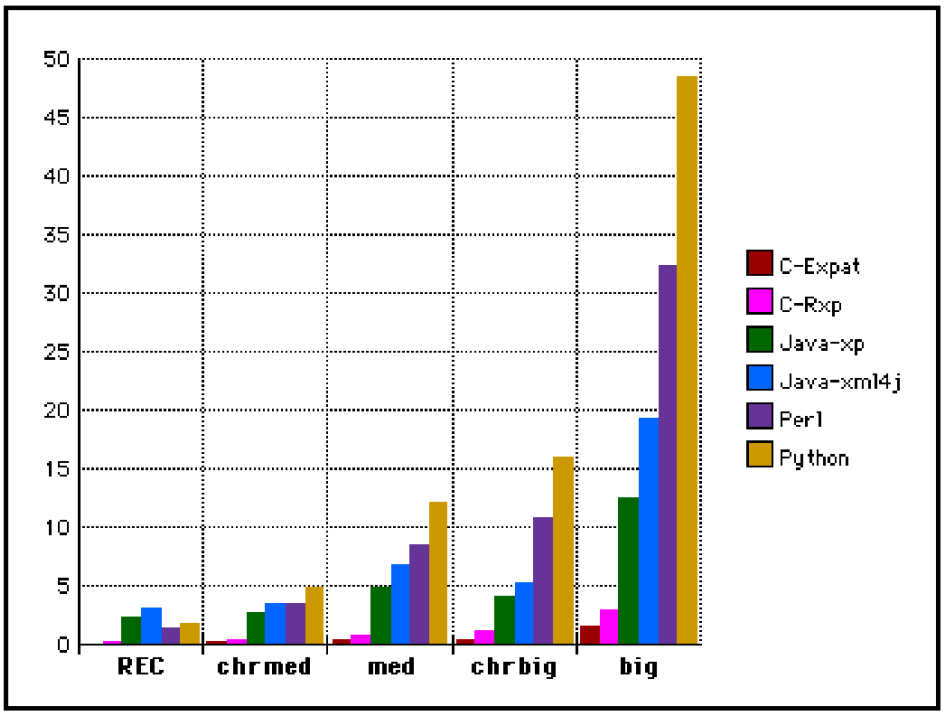

Abbildung 6.3: Benchmarkergebnisse aktueller XML-Parser (Ergebnisse in Sekunden)

Klasse	SNFactory	
Stereotyp	Interface	
Beschreibung	Die SearchNetwork-Factory wird zum Erzeugen und Löschen von Proxy- und Manager-Instanzen eingesetzt. Die create- bzw. delete-Funktionen werfen eine CREATION_FAILED bzw. eine DELETION_FAILED Exception aus, wenn ein Fehler auftritt.	
Attribute	`proxies`	Liste der instantiierten Proxies.
	`managers`	Liste der instantiierten Manager.
Methoden	`createProxy`	Instantiiert einen Proxy.
	`deleteProxy`	Löscht eine Proxy-Instanz. Beim Versuch eine Proxy-Instanz zu löschen, die noch mit einem Manager verbunden ist wird eine PROXY_CONNECTED Exception ausgeworfen.
	`createManager`	Instantiiert einen Manager.
	`deleteManager`	Löscht eine Manager-Instanz.

Klasse	ManagerList
Stereotyp	Typedef
Beschreibung	Die ManagerList ist eine CORBA sequence vom Typ Manager.

6.4.2 SearchNetwork

Im Gegensatz zur DOM-Komponente sind die Objekte des SearchNetworks persistent im Sinne der Objekt-Referenz, d.h. sie werden im Verzeichnisdienst registriert und können damit von mehreren Klienten verwendet werden. Diese Art von Komponenten entspricht der Entity-Kategorie im CCM.

Die SNFactory ermöglicht das Erzeugen und Löschen von Manager- und Proxy-Instanzen.

6.4.2.1 ACE

Die *Adaptive Communication Environment (ACE)* Bibliothek wird verwendet, um die SearchNetwork Komponenten Plattform-unabhängig zu implementieren. Es handelt sich dabei um eine Open-Source C++-Bibliothek, die von Prof. Douglas C. Schmidt, Direktor des Fachbereichs für verteilte Objek-

te an der Universität Washington entwickelt wurde. Sie stellt eine umfangreiche C++-Klassenbibliothek zur Verfügung [ACE]. Diese Klassen kapseln die Plattform-spezifischen Unterschiede in den Kommunikationsmechanismen der Betriebssysteme. Die Bibliothek wird von großen Unternehmen, wie z.B. Motorola gefördert, die damit ihre Software, wie z.B. die Sattelitenkommunikation, implementieren. Aufbauend auf der ACE gibt es sogar eine freie CORBA Implementation, *The ACE ORB (TAO)* [TAO], die zum Beispiel von Boeing, einer der Sponsoren, testweise zur Steuerung eines Flugzeugs eingesetzt wird. Die ACE Bibliothek und TAO unterliegen der *GNU General Public License (GPL)*, wodurch sich die Portierung auf fast jedes Betriebsystem und die Unterstützung vieler C++Compiler erklären läßt. Sogar Microsoft tritt als Sponsor auf und hat die Portierung auf Windows CE finanziert.

Klasse	FFactory	
Stereotyp	Interface	
Beschreibung	Eine FFactory dient der Erzeugung und Terminierung von Fulcrum Repository-Instanzen.	
Attribute	`repositories`	Liste der von dieser Factory verwalteten Instanzen.
Methoden	`createCustomRepository`	Erzeugt ein Repository in dem alle Dokument-Formate gespeichert werden können.
	`createXMLRepository`	Erzeugt ein Repository in dem nur XML-Dateien gespeichert werden können.
	`deleteRepository`	Löscht eine Repository-Instanz. Wird der Parameter destroy auf TRUE gesetzt, dann werden alle Dateien physikalisch gelöscht.

6.4.3 Fulcrum Repository

Das Fulcrum Repository besitzt eine persistente Objekt-Referenz. Damit entspricht es, wie die Objekte des SearchNetworks, der Entity-Kategorie des CCMs.

Die FFactory kann Repository-Instanzen vom Typ Custom und XML erzeugen bzw. löschen. In einem Custom-Repository können beliebige Dokumente abgelegt werden, die Indizierung übernimmt ein Custom-TextReader. In einem XML-Repository können Dokumente im XML-Format indiziert werden. Für diese Aufgabe wird der XML-TextReader verwendet.

6.5 Anwendungsbeispiel

Als Beispiel einer Anwendung auf Basis von KN2 wird eine WWW-basierte Fehlerdatenbank „KnowledgeBase“ implementiert. An diesem Beispiel sind die Laufzeitumgebung, sowie alle Komponenten beteiligt. Der Aufbau orientiert sich an der „Sun JDC Bug Database“[5]. Diese Datenbank speichert Fehlerbeschreibungen zu den Java-Produkten von Sun.

Der Benutzer hat die Auswahl zwischen der Suche in der Datenbank und dem Hinzufügens von neuen Einträgen. Die Benutzerschnittstelle ist in JSP implementiert. Abbildung 6.7 zeigt das Formular mit dem ein neuer Eintrag zum Repository hinzugefügt werden kann. Die JavaServer Page, die den POST-Request bearbeitet erzeugt aus den Formulardaten das folgende XML-Dokument:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="kbase.xsl" type="text/xsl"?>
<PAPER DATE="01.12.1999" ID="944066066172" TIME="18:47:00">
<SYNOPSIS>OAD version problem</SYNOPSIS>
<DESCRIPTION>We found different versions of the OAD.
1. VisiBroker C++ 3.3 OAD.EXE
2. VisiBroker Java 3.4 OADJ.EXE
3. VisiBroker Java 3.4 (JBuilder3) OADJ.EXE
The registration and deregistration process
was corrupt. The implementations were not
started correctly.
</DESCRIPTION>
<REPRODUCE></REPRODUCE>
<AUTHOR>MRU</AUTHOR>
<STATUS>resolved</STATUS>
<URGENCY>low</URGENCY>
<TYPE>Product</TYPE>
<SEVERITY>fatal</SEVERITY>
<APPLIESTO>VisiBroker OAD</APPLIESTO>
<SOLUTION>Use the JBuilder 3 Enterprise version of the
VisiBroker for Java OADJ.EXE.</SOLUTION>
<REFERENCES></REFERENCES>
<PLATFORM>
        <ITEM>Windows NT</ITEM>
        <ITEM>Windows 98</ITEM>
        <ITEM>Windows 95</ITEM>
        <REM>The Solaris version works fine.</REM>
</PLATFORM>
<LANGUAGE>
```

[5] java.sun.com/cgi-bin/bugreport.cgi

```
        <ITEM>Java</ITEM>
        <REM>none</REM>
</LANGUAGE>
</PAPER>
```

In Abbildung 6.4 ist die Such-Schnittstelle dargestellt. Die Suche nach dem Schlüsselwort „OAD“ resultiert in der Ergbnisliste aus Abbildung 6.5. Nach dem Klick auf den Link des gefundenen Ergebnisses erfolgt die Ausgabe wie in Abbildung 6.6. Diese Ausgabe wird mit Hilfe des folgenden XSL-Stylesheets erzeugt:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
  <xsl:template match="/">
<HTML>
  <TITLE>
    <xsl:value-of select="PAPER/SYNOPSIS"/>
  </TITLE>
  <BODY>
    <TABLE BORDER="1">
      <TR><TD VALIGN="top"><B>ID:</B></TD>
      <TD><xsl:value-of select="PAPER/@ID"/></TD></TR>
      <TR><TD VALIGN="top"><B>Date:</B></TD>
      <TD><xsl:value-of select="PAPER/@DATE"/></TD></TR>
      <TR><TD VALIGN="top"><B>Time:</B></TD>
      <TD><xsl:value-of select="PAPER/@TIME"/></TD></TR>
      <TR><TD VALIGN="top"><B>Author:</B></TD>
      <TD><xsl:value-of select="PAPER/AUTHOR"/></TD></TR>
      <TR><TD VALIGN="top"><B>Synopsis:</B></TD>
      <TD><xsl:value-of select="PAPER/SYNOPSIS"/></TD></TR>
      <TR><TD VALIGN="top"><B>Description:</B></TD>
      <TD><xsl:value-of select="PAPER/DESCRIPTION"/></TD></TR>
      <TR><TD VALIGN="top"><B>How to reproduce:</B></TD>
      <TD><xsl:value-of select="PAPER/REPRODUCE"/></TD></TR>
      <TR><TD VALIGN="top"><B>Status:</B></TD>
      <TD><xsl:value-of select="PAPER/STATUS"/></TD></TR>
      <TR><TD VALIGN="top"><B>Urgency:</B></TD>
      <TD><xsl:value-of select="PAPER/URGENCY"/></TD></TR>
      <TR><TD VALIGN="top"><B>Type:</B></TD>
      <TD><xsl:value-of select="PAPER/TYPE"/></TD></TR>
      <TR><TD VALIGN="top"><B>Severity:</B></TD>
      <TD><xsl:value-of select="PAPER/SEVERITY"/></TD></TR>
      <TR><TD VALIGN="top"><B>Applies to:</B></TD>
      <TD><xsl:value-of select="PAPER/APPLIESTO"/></TD></TR>
```

```
      <TR><TD VALIGN="top"><B>Solution:</B></TD>
      <TD><xsl:value-of select="PAPER/SOLUTION"/></TD></TR>
      <TR><TD VALIGN="top"><B>References:</B></TD>
      <TD><xsl:value-of select="PAPER/REFERENCES"/></TD></TR>
      <TR><TD VALIGN="top"><B>Platforms:</B></TD>
      <TD><TABLE>
        <xsl:for-each select="PAPER/PLATFORM/ITEM">
          <TR><TD><xsl:value-of/></TD></TR>
        </xsl:for-each>
          </TABLE><BR/>
      (Comment: <xsl:value-of select="PAPER/PLATFORM/REM"/>)
      </TD></TR>
      <TR><TD VALIGN="top"><B>Language:</B></TD>
      <TD><TABLE>
        <xsl:for-each select="PAPER/LANGUAGE/ITEM">
          <TR><TD><xsl:value-of/></TD></TR>
        </xsl:for-each>
          </TABLE><BR/>
      (Comment: <xsl:value-of select="PAPER/LANGUAGE/REM"/>)
      </TD></TR>
    </TABLE>
  </BODY>
</HTML>
  </xsl:template>
</xsl:stylesheet>
```

Abbildung 6.4: Formular für die Volltext-Suche in der KnowledgeBase

KnowledgeBase Example

Add an article...
Find an article...

Results:

Number of results: 1

Your query was: OAD (number of keywords = 1)

article	weight
OAD version problem	100

Abbildung 6.5: Suchergebnisse einer Volltext-Suche in der KnowledgeBase

KnowledgeBase Example

Add an article...
Find an article...

ID:	944066066172
Date:	01.12.1999
Time:	18:47:00
Author:	MRU
Synopsis:	OAD version problem
Description:	We found different versions of the OAD. 1. VisiBroker C++ 3.3 OAD.EXE 2. VisiBroker Java 3.4 OADJ.EXE 3. VisiBroker Java 3.4 (JBuilder3) OADJ.EXE The registration and deregistration process was corrupt. The implementations were not started correctly.
How to reproduce:	
Status:	resolved
Urgency:	low
Type:	Product
Severity:	fatal
Applies to:	VisiBroker OAD
Solution:	Use the JBuilder 3 Enterprise version of the VisiBroker for Java OADJ.EXE.
References:	
Platforms:	Windows NT Windows 98 Windows 95 (Comment: The Solaris version works fine.)
Language:	Java (Comment: none)

Abbildung 6.6: XSL-Anzeige eines Such-Ergebnisses

KnowledgeBase Example

Add an article...
Find an article...

To add an article, please fill out the form and submit it to the KnowledgeBase.

Bug ID	944066066172
Date	01.12.1999
Time	17:34:26
Synopsis	OAD version problem
Description	We found three different versions of the OAD. 1. VisiBroker C++ 3.3 OAD.EXE 2. VisiBroker Java 3.4 OADJ.EXE 3. VisiBroker Java 3.4 (JBuilder3) OADJ.EXE The registration and deregistration process was corrupt. The implementations were not started correctly.
Reproduce	
Author	MRU
Status	(•) resolved ( ) in progress ( ) outstanding
Urgency	(•) low ( ) medium ( ) high
Type	Configuration Programming Operating system Library Product
Severity	( ) warning ( ) information (•) fatal
Solution/Workaround	Use the JBuilder 3 Enterprise version of the VisiBroker for Java OADJ.EXE.
References	
Platform	[x] Windows NT 4.0 [x] Windows 98 [x] Windows 95 [ ] Solaris Comments : The Solaris version 3.4 works fine.
Language	( ) C++ ( ) C (•) Java ( ) Visual Basic Comments :
Applies To (Product/Project)	VisiBroker OAD

submit your input...

Abbildung 6.7: Formular zum Hinzufügen eines Eintrags in die Knowledge-Base

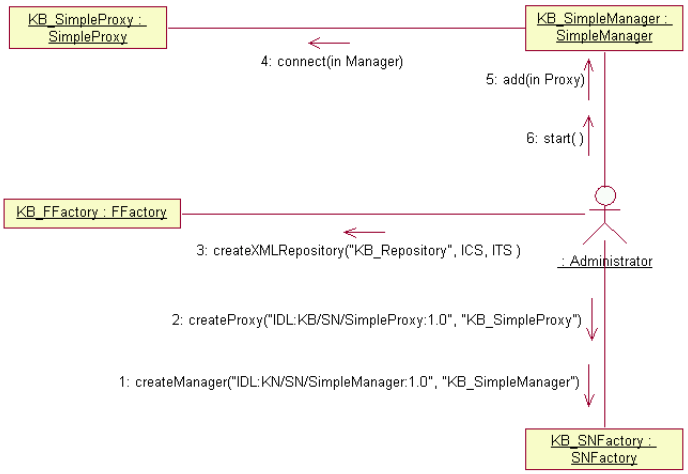

Abbildung 6.8: UML-Kollaborationsdiagramm: Erzeugung der Instanzen des Anwendungsbeispiels

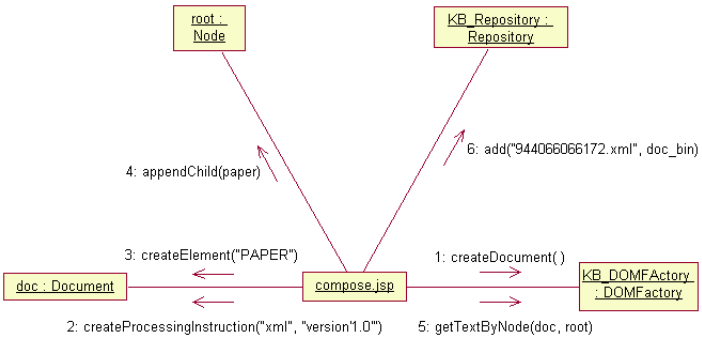

Abbildung 6.9: UML-Kollaborationsdiagramm: Überführung der Formulardaten in ein XML-Dokument und Speicherung des Dokuments im Repository (Aus Gründen der Übersichtlichkeit wird nur der PAPER-Tag eingefügt)

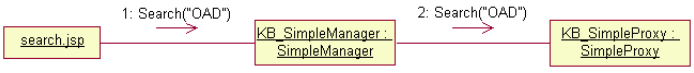

Abbildung 6.10: UML-Kollaborationsdiagramm: Volltext-Suche auf dem Index

6.6 Produkt-Struktur

Die Produkt-Struktur ist eine Empfehlung, wie die hier vorgestellten Systeme dem Benutzer gegenüber in Form logischer Pakete präsentiert werden können. Dabei schließt sich der Bogen zu den am Anfang definierten Use-Cases. Als Benutzer werden hier nur der Anwendungs-Entwickler und der Administrator betrachtet. Die restlichen Use-Cases sind entweder zu stark von der Anwendung abhängig oder es existiert bereits eine entsprechende Unterstützung. Der Kern-Entwickler und der Komponenten-Entwickler würden idealerweise mit einer aktuellen IDE, wie z.B. JBuilder3, arbeiten. Diese IDE bietet eine gute Unterstützung für die Erstellung von CORBA-Objekten an.

Betrachtet man das KN2 Application Programming Model, dann wird aber deutlich, daß es keine Entwicklungsumgebungen gibt, die ein solches Modell unterstützen. Applikations-Server, wie z.B. SilverStream, liefern zwar solche IDEs mit, allerdings lassen sich diese nicht vom Server losgelöst verwenden und sind aus diesem Grund für KN2 ungeeignet. Deshalb wird in diesem Projekt ein Paket aus einem kommerziellen HTML-Entwicklungswerkzeug mit JSP-Unterstützung (Homesite, www.allaire.com), der Sun JSP Referenzimplementation und der KN2-Laufzeitumgebung gebildet. Das Paket erhält in diesem Projekt den Namen *KN2 Developer Studio* und dient als Überbrückung, bis professionellere IDEs verfügbar sind, die ein entsprechendes APM unterstützen.

Als Produktionssystem wird ein Paket aus Apache, JServ, GNUJSP, Cocoon und der Laufzeitumgebung gebildet, das den Namen *KN2 Application Server* trägt.

Der Administrator wird in seinen Tätigkeiten einerseits von Kommandozeilen-Programmen, andererseits von einem Programm mit grafischer Oberfläche unterstützt. Mit allen Programmen ist es möglich Factory-Operationen durchzuführen, um Instanzen zu erzeugen. Diese Hilfsprogramme sind in das Paket der Laufzeitumgebung integriert und stehen damit sowohl im KN2 Application Server, als auch im KN2 Developer Studio zur Verwaltung der Laufzeitumgebung zur Verfügung.

6.7 Fazit

Als besondere Probleme während der Umsetzung stellten sich die vielen verschiedenen Module heraus. Bei neuen Versionen gab es regelmäßig Schwierigkeiten, diese in den Gesamtkomplex zu integrieren. Dabei stellt sich ein Nachteil und ein Vorteil der Freien Software heraus. Einerseits sind die Entwickler von freien Projekten nicht durch einen Kundenstamm zur Abwärtskompatibilität verpflichtet und können so schneller Altlasten über Bord werfen, als das in kommerziellen Systemen der Fall ist. Andererseits bedingt eine

solche Vorgehensweise eine häufigere Anpassung eines Software-Systems an neue Releases, da mit diesen häufig wichtige Fehlerkorrekturen verbunden sind.

Diese Problematik ist verwandt mit dem Problem der Lizenzen bei Komponenten-basierten Systemen. In einem Szenario mit vielen Komponenten verschiedener Hersteller wird die Verwaltung der vielen Lizenzen zum Problem. Aber dieses Problem beschränkt sich nicht nur auf kommerzielle Lizenzen. Auch im Open-Source Bereich bilden sich immer mehr Lizenzmodelle, die der GNU General Public License (GPL)[6] zwar ähnlich sind, allerdings dem geistigen Eigentümer, gegenüber der GPL, mehr Rechte belassen. Da Firmen gerade erst anfangen Teile ihrer Software unter GPL-ähnlichen Lizenzen zu veröffentlichen und der Komponenten-basierte Ansatz sich noch nicht auf breiter Front durchgesetzt hat, gibt es zur Zeit noch keine Anstrengungen eine Vereinheitlichung oder ein automatisiertes Management einzuführen.

Die beiden Problem würden sich durch einen globalen Rahmen für Komponenten lösen lassen. Generell sind EJB und das CCM Ansätze in die richtige Richtung. Die Tauglichkeit beider Systeme in der Praxis muß sich aber erst noch unter Beweis stellen.

Aufgrund der steigenden Komplexität der IT-Technologien ist es wichtig Standards, wo immer möglich einzusetzen. Jede proprietäre Erweiterung muß ein Benutzer erst erlernen. Entweder bringt ihm diese Erweiterung einen entscheidenden Produktivitätsgewinn oder sie wird einfach ignoriert. Proprietäre Systeme werden es in Zukunft immer schwieriger haben sich durchzusetzen, da ein zusätzlicher Schulungsaufwand damit verbunden ist.

Mit der steigenden Popularität und damit der Stabilität der freien ORBs ist die Entscheidung für den VisiBroker ORB nachträglich zu revidieren. Diese Entscheidung wurde sehr früh getroffen und aufgrund von nicht unerheblichen Ausgaben ist diese Entscheidung nicht so einfach rückgängig zu machen. Spätestens dann, wenn ein Upgrade auf eine neue Version des VisiBrokers und damit neue Ausgaben notwendig wären, läge die Überlegung nahe einen freien ORB wie z.B. ORBit einzusetzen. Durch den Portable Object Adapter (POA) ist es sogar möglich Unabhängigkeit auf Source-Code Ebene zu erreichen, so daß ein Wechsel „problemlos“ möglich ist. Außerdem läßt die Update-Häufigkeit von Inprise, dem Hersteller des VisiBroker ORB, zu wünschen übrig. Im Laufe des Projektes wurden drei Versionen entdeckt, die aufgrund verschiedener Fehler nicht korrekt funktionierten. Erst die letzte Version schien diese Probleme nicht mehr aufzuweisen. Das Problem der dritten Version war allerdings, daß sie nicht getrennt von Borland's JBuilder3 verwendet werden durfte.

[6] www.gnu.org/copyleft/licenses.html

Kapitel 7

Middleware

Der Begriff Middleware ist einer der Begriffe im IT-Bereich, die zwar häufig verwendet werden, aber für den keine präzise Semantik definiert werden kann. In diesem Kapitel werden Merkmale einer konzeptionellen Definition diskutiert und die Erkenntnisse aus diesem Diplomarbeitsprojekt zu diesen in Beziehung gesetzt.

7.1 Definitionen

Um die Bedeutungsproblematik zu verdeutlichen sind nachfolgend einige Definitionen zusammengetragen:

1. "Middleware is software that resides in the middle between an application program and the base-level OS and networking capabilities of a computing system. ... Middleware supports standards-based interfaces that enhance distributability of applications, interoperability between applications and portability of applications between systems." (The Middleware Source Book, John Colonna-Romano and Patricia Srite, Digital Press, August 1994)

2. „Middleware is software that allows elements of applications to interoperate accross network links, despite differences in underlying communications protocols, system architectures, OSes, databases, and other application services." (The Muddle in the Middle, John R. Rymer, BYTE, April 1996)

3. „Middleware - das Bindeglied zwischen Clients und Servern - übernimmt selbst umfassende Aufgaben in verteilten Informationssystemen." (Markus Tresch, Informatik Spektrum 19: 249-256 (1996))

4. „Middleware is a generic term for technology which corrects a mismatch between the services requested by a client application and the

capability of the server to deliver those services. This mismatch might be in terms of the number of users attempting to access a limited resource (as in a TP Monitor or Application Server), or it might be in terms of providing integration between applications in a fragmented environment.“ (Bloor Research, http://www.bloor-research.com/research_areas/middleware.html)

5. „Middleware is a very misunderstood but important positioning move for banks. Middleware will allow banks to separate rules from access technologies that are particular to a delivery channel and permit banks to efficiently keep pace with rapidly changing programming languages.“ (Andrew De Meo, President of Electronic Commerce Marketing Systems, Bank Technology News, December 1998)

6. „Communication middleware resides between client and server applications in distributed systems. It simplifies the application development by providing a uniform view for heterogeneous networks, protocols, and OS layers.“ (The Design and Performance of a Pluggable Protocols Framework for Object Request Brokers, Fred Kuhns, Carlos O'Ryan, Douglas C. Schmidt, Ossama Othmann, and Jeff Parsons, August 1998)

Alle Definitionen haben gemeinsam, daß sie versuchen, das Problem auf technischer Basis zu beschreiben. Die technischen Systeme sind aber oft nicht nur auf eine einzige Funktionalität beschränkt, sondern sie vereinen viele verschiedene Funktionen. Diese Tatsache, sowie das Fehlen von eindeutigen Kriterien, machen es so schwierig solche Systeme einzuordnen.

Als Konsequenz daraus kann nur eine konzeptionelle Definition hilfreich sein. Mit Hilfe dieser Definition könnten die einzelnen Subsysteme eines Systems auf Erfüllung der Kriterien untersucht werden und so eine eindeutige Zuordnung erreicht werden.

Ausgangspunkt ist eine allgemeine, aber anerkannte Definition: Middleware ist das verbindende Element zwischen Klient und Server.

Der Startpunkt aller Überlegungen liegt also im Zusammenhang zwischen Klient und Server. Ausgehend davon werden die bisher identifizierten Eigenschaften von Middleware aufgezeigt, um sie anschließend mit den Komponenten des Diplomarbeitsprojektes in Beziehung zu setzen.

7.2 Klient-Server

Der Begriff der *Rolle* bezeichnet die Beziehung zwischen zwei Parteien in einer bestimmten Situation.

Das „Klient-Server-Prinzip“ bezeichnet ein Anwendungsmodell, daß die Trennung von Aufgaben zwischen Klient und Server fordert. „Klient-Server“

Abbildung 7.1: Klient-Server Rolle

Adressierung	direkt, indirekt
Zuverlässigkeit	gegeben, nicht gegeben
Blockierung	synchron, asynchron
Pufferung	ungepuffert, gepuffert (Briefkasten)
Kommunikationsform	Meldung, Auftrag

Tabelle 7.1: Merkmale von Kommunikationsmodellen

hat jedoch noch eine weitere Bedeutung. Es bezeichnet auch eine Rolle zwischen zwei Parteien. Dabei offeriert der Server bestimmte Dienstleistungen und der Klient kann diese Dienste in Anspruch nehmen. Das allgemeine Verständnis impliziert häufig eine physikalische Trennung zwischen Klient und Server. Das Prinzip läßt sich aber ganz allgemein auch auf die funktionalen Ebenen einer Systemarchitektur anwenden.

Um die Dienste des Servers zu verwenden, muß der Klient mit dem Server kommunizieren. In dieser Kommunikationsbeziehung werden den Teilnehmern bestimmte Rollen zugewiesen. Sendet der Klient eine Nachricht zum Server, so ist in dieser Situation der Klient in der Rolle des Senders und der Server in der des Empfängers.

7.3 Eigenschaften

Der Begriff der Kooperation bezeichnet die aufgabenbezogene Zusammenarbeit zwischen Systeminstanzen. Das Verhältnis dieser Systeminstanzen zueinander wird durch eine Rollenbeziehung zum Ausdruck gebracht. Diese Rollenbeziehung wird durch ein bestimmtes Kommunikationsmodell realisiert. Die Merkmale eines Kommunikationsmodells sind in Tabelle 7.1 aufgelistet [Sch99].

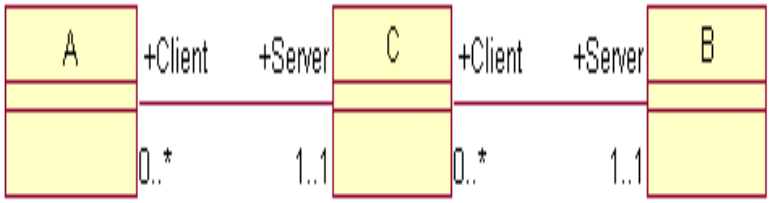

Abbildung 7.2: Middleware Rolle

Zusätzlich zu diesen Kriterien muß der Kanal, über den die Kommunikation abgewickelt wird, betrachtet werden. Hier sind zwei Phasen zu unterscheiden: die Einrichtung von Kanälen und deren Benutzung. Unter Einrichtung ist nicht nur die Herstellung einer Verbindung zu sehen, sondern auch die Koordinationsaspekte, wenn mehrere Klienten auf diese Ressource zugreifen möchten [Sch99].

Middleware führt eine Adaption zwischen verschiedenen Rollenbeziehungen durch. Da eine Rollenbeziehung durch die oben eingeführten Merkmale charakterisiert wird, erfolgt also durch die Middleware eine Anpassung dieser Merkmale zwischen den verschiedenen Systeminstanzen.

Eine wichtige Eigenschaft von Middleware ist, daß deren Einsatz in der Kommunikation zwischen Klient und Server für beide Parteien transparent erfolgt. Deshalb verhält sich die Midleware aus der Sicht des Klienten wie ein Server und aus der Sicht des Servers wie ein Klient (s. Abb. 7.2).

Ein weiteres Merkmal von Middleware ist die Etablierung von Abstraktionsebenen, um die Komplexität der Server-Systeme vor dem Anwender zu verbergen. Repräsentiert die Middleware eine Menge von Servern, so hilft sie im Idealfall die komplexen Details der Koordination zu verbergen.

Die Konsequenz aus all diesen Merkmalen ist, daß Middleware eine Schnittstelle zu dem oder den Diensten einführt. Diese Schnittstelle stellt die Middleware als den Stellvertreter der dahinterliegenden Server dar.

7.4 Bewertung

In den folgenden Abschnitten werden die KN2-Komponenten bzgl. der oben eingeführten Merkmale hin untersucht. Abschließend wird gesagt, ob es sich dabei um Middleware handelt.

7.4.1 DOM

Das DOM ist ein sehr gutes Beispiel für die Anpassung von Rollen. Der verwendete Parser ist Meldungs-orientiert. Er sendet Meldungen, sobald er einen Start- oder einen Stop-Tag im Datenstrom entdeckt. Der Klient des DOM-Objektes ist allerdings Auftrags-orientiert. Sobald er Zugriff auf einen Teil des Dokument-Baums benötigt, gibt er dem Objekt den Auftrag, dieses Element zurückzuliefern. Somit führt das DOM-Objekt eine Anpassung zwischen den Meldungs-orientierten und dem Auftrags-orientierten Merkmal durch. Außerdem erhält der XML-Parser mit dem DOM-API eine standardisierte Schnittstelle, die das proprietäre Callback-Interface vor dem Anwender versteckt.

7.4.2 SearchNetwork

Für die SearchNetwork-Komponente wird mit Hilfe einer CORBA-Schnittstelle eine Adaption verschiedener Schnittstellen durchgeführt. Jede WWW-Suchmaschine hat ihr eigenes API. Dies ist darauf zurückzuführen, daß dieses API nicht direkt von außen verwendet werden soll, da sich die Suchdienste teilweise über Werbung finanzieren, die bei einem direkten Zugriff auf die Schnittstelle vom Benutzer nicht wahrgenommen werden kann. Das SimpleSearch-Interface bildet die Abstraktion, also das kleinste gemeinsame Vielfache bezüglich der gebotenen Funktionalität von Suchmaschinen.

Zusätzlich führt das Manager-Objekt die Koordination der mit ihm verbundenen Proxies durch. Hinter ihm können sich mehrere Proxies verbergen, die an einer Suche beteiligt sind. Der Manager sorgt für die Zusammenführung der Suchergebnisse und die Elimination von Duplikaten.

Aufgrund dieser beiden Merkmale kann die SearchNetwork-Komponente als Middleware angesehen werden.

7.4.3 Fulcrum

Außer einer Kapselung der Fulcrum-Schnittstellen hinter einem CORBA-API lassen sich keine weiteren Middleware typischen Merkmale erkennen. Auch unterliegt dieses API keinem Standard, sondern ein proprietäres API wurde durch ein anderes proprietäres API ersetzt. Aus diesen Gründen stufe ich das Fulcrum Wrapper-Objekt nicht als Middleware ein.

7.5 Fazit

Der Begriff Middleware und damit zusammenhängende Begriffe wie Applikationslogik sind nicht präzise definiert und befinden sich im Fluß.

Jede technische Definition hat ihren eigenen Kontext, in dem sie erstellt wurde. Dieser Kontext entscheidet darüber welche Kriterien als wichtig erkannt wurden und dementsprechend die Definition beeinflußt haben.

In [Rym96] zum Beispiel findet sich eine Applikations-bezogene Sicht auf das Problem. Die Merkmale der dort vorgeschlagenen Definition sind: Applikationselemente, Interoperabilität und Integration unterschiedlicher Systeme. Der Artikel [Rit98] wiederum sieht Middleware eher in die Richtung von Datenbank-Middleware tendieren. Dementsprechend findet man andere Merkmale in [Ber96], [Tre96], [Pou98], [Fin95], [Lin97] und [Keu96].

Klar scheint aber zu sein, daß ein einzelnes Middleware-System niemals für alle Problemlösungen geeignet ist. Es wird immer spezialisierte Systeme geben, die sich auf bestimmte Teil-Probleme konzentrieren.

Kapitel 8

Zusammenfassung

Nachdem in Kapitel 2 die Technologien, die in diesem Projekt zum Einsatz kamen, kurz angerissen wurden, ist in Kapitel die Analyse des alten Systems beschrieben. Das Ergebnis dieser Analyse ist, daß es sich bei KN1 um ein System mit einer veralteten Architektur handelt, welche auf veralteten Technologien basiert. Diese Probleme sind struktureller Natur und können nur durch eine radikale Umstrukturierung gelöst werden. Die Skriptsprache ist nur eine Ansammlung von Funktionen, so daß der Begriff Sprache dafür nicht zutreffend ist. Erschwerend kommt hinzu, daß die Funktionen keine einheitliche Syntax besitzen. Eine gedruckte Dokumentation zu KN1 ist nicht vorhanden und der ursprüngliche Entwickler hat mittlerweile die Firma verlasen. Das größte Problem ist aber die Entscheidung des Managements die aktive Weiterentwicklung des Kernsystems einzustellen und die Software lediglich als Ausführungsplattform einzusetzen.

Die Anforderungen, die im Kapitel 4 definiert werden, adressieren natürlich diese Nachteile. Es wird eine erweiterbare Architektur gefordert, die von den ausgewählten Technologien optimal unterstützt wird. Die Management-Anforderungen haben natürlich keinen Einfluß auf die Architekturbeschreibung im nächsten und die Implementationsbeschreibung im übernächsten Kapitel. Eine Umsetzung dieser Anforderungen wird sich erst in Nachfolgeprojekten zeigen.

Die Architektur aus Kapitel 5 fordert eine klare Trennung zwischen Präsentationslogik und Anwendungslogik. Das J2EE Application Programming Model (J2EE APM) bildet das Modell zu dieser Partitionierung und wird mit einigen wenigen Adaptionen als strukturelle Basis von KN2 übernommen. Die Technologien, die dieses Modell integriert, unterstützen den Entwickler darin, die Trennung von Präsentations- und Anwendungslogik auch tatsächlich zu vollziehen. Als Container für die Applikationslogik bietet sich das CORBA Component Model (CCM) an, da die Forderung nach Offenheit auch die Unterstützung beliebiger Implementationsprachen beinhaltet. Als System zur Implementation der Präsentationslogik bietet sich die

JSP-Spezifikation an. Diese unterstützt in erster Linie Java als Implementationssprache, ist allerdings darauf vorbereitet in Zukunft andere Sprachen zuzulassen. XML und XSL sind Technologien die ebenfalls einem Separationsparadigma folgen, in diesem Fall allerdings auf Datenebene und nicht auf Ebene der Implementations-Logik.

Das Kapitel 6 beschäftigt sich mit der tatsächlichen Umsetzung der im vorherigen Kapitel vorgeschlagenen Architektur. Ein Umsetzung des CCM war mit den Ressourcen in diesem Projekt nicht zu realisieren. Deshalb mußte ein anderer Ansatz verfolgt werden, welcher nur die relevanten Teile umfaßt. Als besondere Herausforderung gestaltete sich die Koordination der vielen verschiedenen Technologie-Implementationen. Die Entwicklung für mehrere Plattformen schränkte die verwendeten Subsysteme stark ein. Bei Verwendung von Java ergeben sich im Allgemeinen keine Probleme, aber bei C++ als Implementationssprache treten Probleme bzgl. der STL und den vom Betriebssystem abhängigen Thread-Modellen auf. Zuerst müssen entsprechende Plattform-unabhängige Bibliotheken gefunden werden. In diesem Fall STLport als Implementation der Standard Template Library und ACE als Abstraktion der vielen verschiedenen Thread-Modelle, die eine Portierung z.B. zwischen Windows und Solaris erschweren.

Die Erfahrungen aus dieser Umsetzung sind, daß mit der Integration einer wachsende Anzahl von Technologien eine Steigerung der Komplexität verbunden ist. Diese Integration birgt einen immensen Aufwand, da die Implementationen dieser Technologien zumeist unabhängig voneinander sind, wodurch sich häufig Inkompatibilitäten zwischen den Subsystemen ergeben. Bei der Auswahl ist außerdem auf kompatible Lizenzmodelle zu achten, da nicht alle Projekte z.B. unter der GNU General Public License veröffentlicht werden. Erst eine allgemein anerkannte Komponenten-Laufzeitumgebung und ein damit verknüpfter Markt für Komponenten könnte hier Abhilfe schaffen, denn dann würden sich diese Probleme alltäglich zeigen und deren Lösung dringlicher werden.

Das Kapitel über Middleware identifiziert die allgemeinen Merkmale von Middleware und ordnet die SearchNetwork- und die DOM-Komponente aufgrund dieser Eigenschaften als Middleware ein.

Kapitel 9

Ausblick

Diese Kapitel eröffnet einen Ausblick über die zu erwartenden Entwicklungen in den zuvor angesprochenen Bereichen. Außerdem werden Verbesserungsvorschläge für dieses Projekt gemacht und Empfehlungen gegeben, in welche Richtung weiterentwickelt werden sollte.

9.1 Trends

Die Software-Entwicklung wird immer komplexer. Der Entwickler sieht sich mit einer ständig wachsenden Zahl an Technologien und Alternativen konfrontiert.

Vor etwa 10 Jahren waren Anwendungen in Pascal, die ihre eigene Textbasierte Oberfläche besaßen, gängige Praxis. Das Dateiformat war proprietär und die einzigen Technologien, die man zur Entwicklung solcher Anwendungen benötigte, waren Kenntnis des Betriebssystems, der Laufzeitbibliothek, der IDE und natürlich der Programmiersprache selbst.

Wenn man dies mit dem J2EE Application Programming Model vergleicht, dann sollte man einen Überblick über das Betriebssystem, TCP/IP, HTTP, HTML, XML, XSL, Java, EJB, SQL-Datenbanken, Transaktionsdienste, Messaging-Dienste, JDBC, CORBA, Java-Servlets, Java-Applets und die IDE haben. Handelt es sich um Multi-Plattform-Projekte, dann muß man zusätzlich Kenntnisse über die verschiedenen Betriebssysteme mitbringen und bei einer Integration heterogener Systeme ist häufig noch ein Adaptionsmodul in anderen Programmiersprachen zu erstellen, z.B. COBOL oder C++.

Diese steigende Komplexität wird auf den ersten Eindruck durch Komponentenmodelle wie CORBA, DCOM und EJB sicherlich noch verstärkt. Auf einer funktionalen Ebene sind sich die Modelle zwar sehr ähnlich, aber auf technischer Ebene zeigen sich gravierende Unterschiede.

Betrachtet man die Applikations-Server, so werden sich Systeme entwickeln, die einerseits offen und Standard-konform sind und andererseits Me-

chanismen mitbringen, um vor dem Anwendungsentwickler die komplexen Details verstecken. So bietet z.B. der Silverstream Application Server eine vollständige Entwicklungsumgebung, in der man mit Hilfe von Wizards die Grundgerüste der verschiedenen Applikationen automatisch erzeugen kann.

Das Problem solcher Integrationslösungen ist es, einen Mittelweg zwischen Vorgaben und Freiheitsgrad zu finden. Ein Einsteiger mag durch solche Hilfen schnell produktiv arbeiten können. Wenn einem fortgeschrittenen Entwickler aber der Blick hinter die Kulissen verwehrt bleibt, oder er keinen Einfluß auf den Erzeugungsprozeß nehmen kann, wird dies die Akzeptanz einer solchen Lösung erschweren.

Ein Trend in Bezug auf Datenformate ist in Verbindung mit XML zu erkennen. XML etabliert sich zur universellen Datenaustauschsprache und könnte so zu einer Vereinheitlichung der vielen proprietären Formate beitragen. Dieser Trend wird durch die Vielzahl von XML-Anwendungen belegt. XML gibt dem klassischen Electronic Data Interchange (EDI) einen neuen Schub und könnte auch die sog. Business-To-Business (B2B) Kommunikation einfacher gestalten. Dazu einigen sich immer mehr Anbieter von einstmals proprietären Lösungen auf standardisierte, XML-basierte Formate. Das Internet und die globale Vernetzung tragen dazu bei, daß gerade die Anbieter von Integrationslösungen immer mehr Schnittstellen zu anderen Systemen anbieten müssen. An Stelle der aufwendigen Entwicklung von Adaptern für die vielen verschiedenen Systeme, trägt die Einigung auf ein gemeinsames XML-Vokabular dazu bei, daß diese Ressourcen eingespart werden können.

Sollte die Entscheidung für den VisiBroker revidiert werden, so hätte das zur Folge, daß das KN2-System fast vollständig auf freien Implementation beruhen würde. Könnte man jetzt auch noch auf freie IDEs und Werkzeuge für die Unterstützung des Entwurfs zugreifen, dann wäre das Projekt von sämtlichen kommerziellen Lizenzen losgelöst. Im Bereich der IDEs ist eine solche Entwicklung schon im Gange. Für die Klasse der Entwurfswerkzeuge ist aber kaum Aktivität in der freien Software Bewegung vorhanden. Damit dies geschieht, muß sich die strukturierte Software-Entwicklung aber erst bei den „Hobby-Programmierern“ etablieren, damit sich private Entwickler finden, die ein solches Projekt umsetzen. Diese Entwicklung scheint aber, zumindest in den nächsten Jahren, mehr als fraglich. Die strukturierte Software-Entwicklung wird hauptsächlich aus dem Grund gefordert, um die Kosten eines Projektes besser abschätzen zu können. Auch wenn diese strukturierte Entwicklung sich in Software-Firmen auf breiter Basis durchsetzen sollten und die Entwickler sich an diese Prozesse gewöhnen, bleibt es trotzdem fraglich, ob sich diese Methoden auch beim Heimanwender durchsetzen, da die freie Software-Bewegung viel von Studenten und Jugendlichen getragen wird, die noch nicht im Berufsleben stehen und deshalb keinen externen Zwang haben solche Methoden anzuwenden. Die Konsequenz daraus ist, daß Firmen, die solche Werkzeuge bisher unter kommerziellen Lizenzen vertrieben haben, sich aufgrund der mangelnden Konkurrenz aus dem

Open-Source Lager nicht dazu gezwungen sehen, ihre Werkzeuge kostenlos zur Verfügung zu stellen.

Allgemein kann zu Entwurfsmethoden sicherlich festgestellt werden, daß die UML mittlerweile die Standardsprache auf diesem Gebiet darstellt. Die Adaption der UML als ISO-Standard wird dazu sicherlich noch weiter beitragen. Diese Entwicklung ist auch in den Werkzeugen zu erkennen. Vor einigen Jahren wurden noch mehrere Entwurfsmethoden unterstützt. Heute schwenken alle Hersteller auf UML um. Dadurch wird es, auf lange Sicht gesehen, natürlich zu einer Konsolidierung auf diesem Markt kommen.

9.2 Verbesserungsvorschläge

Die für KN2 vorgeschlagene IDE ist nur provisorischer Natur. Sie müßte dringend durch eine bessere Lösung ersetzt werden. In diesem Bereich gibt es zur Zeit viele verschiedene Entwicklungen. Erfreulicherweise auch in der Open-Source Bewegung. Als Beispiele können da die Projekte Freebuilder (www.freebuilder.org), Netbeans (www.netbeans.com) und JBuilder Foundation (www.inprise.com/jbuilder) aufgeführt werden. Das Problem ist allerdings, daß das J2EE APM noch sehr neu ist und es zur Zeit kaum Entwicklungsumgebungen gibt, die eine optimale Unterstützung für dieses Programmiermodell und das dahinterstehende Paradigma bieten. Lediglich die NetBeans IDE, welche kürzlich von Sun aufgekauft wurde, bietet in der speziellen Internet Edition Unterstützung für CORBA, EJB und JSP. Es ist zu erwarten, daß diese IDE, natürlich unter dem Einfluß von Sun, eine entsprechende Unterstützung anbieten wird.

Ein Test unter höherer Last ist sicherlich sinnvoll, um die Qualität der Komponenten zu erhöhen.

9.3 Empfehlung

Der kritischste Bereich im KN2 System ist die Komponentenbibliothek. Diese beeinflußt maßgeblich die Akzeptanz des Systems. Eine Unterstützung von JSP alleine, wird in Zukunft nicht mehr ausreichen, da in einem großen Teil der Technologie-Implementationen, mit Server-seitiger Applikationslogik, die Unterstützung von JSP ansteht. Es muß außerdem darauf geachtet werden, daß die angebotenen Komponenten einen bestimmten Qualitätsstandard erreichen. Es nutzt nichts, wenn eine umfangreiche Bibliothek an Komponenten vorhanden ist, die Komponenten aber nur umständlich zu handhaben sind bzw. zur Laufzeit fehlerhaft reagieren.

Damit wird deutlich, daß der wichtigste Schritt die Etablierung eines Komponenten-basierten Ansatzes in der Software-Entwicklung ist. Das Problem, das sich dadurch für das Projektmanagement stellt, ist vor allem der zusätzliche Aufwand, um bei neuen Projekten die universellen Komponenten

zu identifizieren und modellieren zu können. Der Nutzen einer solchen Vorgehensweise wird sich erst dann einstellen, wenn eine ausreichende Menge an Komponenten vorhanden ist, um die wichtigsten Bereiche abzudecken. Dies ist natürlich relativ zu sehen, da es von der Art der erstellten Anwendungen abhängt. In einem Auskunftssystem sind offensichtlich andere Komponenten von Bedeutung, als in einer Transaktions-gesteuerten E-Commerce Anwendung.

Das Versprechen von der Wiederverwendbarkeit wurde auch schon beim Übergang vom prozeduralen zum Objekt-orientierten Paradigma angegeben. Der Komponenten-basierte Ansatz ist in diesem Sinne also nur die logische Fortsetzung dieses Paradigmas von der Quelltext-Ebene auf die binäre Ebene. Es reicht aber nicht aus, daß das Projektmanagement dieses Paradigma nur zur Grundlage „deklariert“, es muß auch aktiv vom Management gefördert werden. Der gesamte Entwicklungsprozeß muß darauf ausgerichtet werden.

Anhang A

IDL

Aus Platzgründen enthält der folgende IDL-Quelltext keine Kommentare.

```
module KN {

exception DELETION_FAILED { };
exception CREATION_FAILED { };

exception BAD_PARAM {
        string name;
};

interface BaseFactory { };

struct Parameter {
        string name;
        string value;
        string description;
};
typedef sequence<Parameter> ParameterList;

module SN {

exception MANAGER_RUNNING { };
exception ALREADY_CONNECTED { };
exception STOPPED { };
exception WRONG_TYPE { };
exception TIME_OUT { };
exception NO_PROXIES { };
exception NO_CONNECTION { };
exception PROXY_CONNECTED { };
```

```
struct Result {
        string url;
        unsigned long rank;
};
typedef sequence<Result> ResultList;

interface SimpleSearch {

        ResultList Search (
                in StringList keywords )
                  raises (NO_CONNECTION, MAX_RESULTS, TIME_OUT);

        ResultList SearchEx (
                in StringList keywords,
                in long timeout,
                in long maxresults )
                  raises (NO_CONNECTION, MAX_RESULTS, TIME_OUT);

        string SearchXML (
                in StringList keywords )
                  raises (NO_CONNECTION, MAX_RESULTS, TIME_OUT);

        string SearchXMLEx (
                in StringList keywords,
                in long timeout,
                in long maxresults)
                  raises (NO_CONNECTION, MAX_RESULTS, TIME_OUT);

};

interface Manager {
        attribute boolean running;
        attribute ProxyList proxies;

        void add (
                in Proxy proxy )
                  raises (ALREADY_CONNECTED, WRONG_TYPE);

        void remove (
                in Proxy proxy )
                  raises (NO_PROXIES, MANAGER_RUNNING);

        void start ()
                  raises (NO_PROXIES);
```

```
        void stop ();

};
typedef sequence<Manager> ManagerList;

interface SimpleManager : Manager, SimpleSearch { };

interface Proxy {
        attribute long timeout;
        attribute long maxresults;
        attribute string type;
        attribute ParameterList parameters;
        attribute Manager connected;

        void connect (
                in Manager manager )
                  raises (ALREADY_CONNECTED, WRONG_TYPE);

        Manager disconnect ()
                  raises (MANAGER_RUNNING);

        StringList listTypes ();

};
typedef sequence<Proxy> ProxyList;

interface SimpleProxy : SimpleSearch, Proxy { };

struct ResultColumn {
        string name;
        StringList rows;
};

typedef sequence<ResultColumn> ResultColumnList;

struct ResultTable {
        string name;
        ResultColumnList cols;
};

interface SQLSearch {
        ResultTable Search (
                in string SQL )
```

```
                  raises (NO_CONNECTION, MAX_RESULTS, TIME_OUT);

        ResultTable SearchEx (
                in string SQL,
                in long timeout,
                in long maxresults )
                  raises (NO_CONNECTION, MAX_RESULTS, TIME_OUT);

        string SearchXML (
                in string SQL );

        string SearchXMLEx (
                in string SQL,
                in long timeout,
                in long maxresults )
                  raises (NO_CONNECTION, MAX_RESULTS, TIME_OUT);

};

interface SQLManager : Manager, SQLSearch { };

interface SQLProxy : Proxy, SQLSearch { };

interface SNFactory : BaseFactory {
        attribute ProxyList proxies;
        attribute ManagerList managers;

        Proxy createProxy (
                in string typeID,
                in string name )
                  raises (WRONG_TYPE, CREATION_FAILED);

        void deleteProxy (
                in Proxy proxy )
                  raises (PROXY_CONNECTED);

        Manager createManager (
                in string typeID,
                in string name )
                  raises (WRONG_TYPE, CREATION_FAILED);

        void deleteManager (
                in Manager manager )
                  raises (MANAGER_RUNNING, PROXY_CONNECTED);
```

```
};

};

module Fulcrum {

exception NOT_FOUND { };
exception ALREADY_EXISTS { };

typedef sequence< octet > doc_bin;

interface Repository {
        attribute string name;
        attribute long count;
        doc_bin get_config (
                in string file )
                  raises (NOT_FOUND);

        void put_config (
                in string file,
                in doc_bin doc );

        StringList list_config ();

        string add (
                in string file,
                in doc_bin doc )
                  raises (ALREADY_EXISTS);

        doc_bin get (
                in string file )
                  raises (NOT_FOUND);

        string rem (
                in string file )
                  raises (NOT_FOUND);

        string info (
                in string file )
                  raises (NOT_FOUND);

        StringList list (
                in string subdir,
                in boolean dirs,
```

```
                in boolean recursive )
                  raises (NOT_FOUND);
};

typedef sequence<Repository> RepositoryList;

interface FFactory : BaseFactory {
        attribute RepositoryList repositories;

        Repository createCustomRepository (
                in string name )
                  raises (CREATION_FAILED);

        Repository createXMLRepository (
                in string name,
                in string ICS,
                in string ITS )
                  raises (CREATION_FAILED);

        void deleteRepository (
                in Repository repository,
                in boolean destroy )
                  raises (DELETION_FAILED);
};

};

module DOM {

interface NodeList;

exception DOMException {
        unsigned short code;
};

exception DOMFactoryException {
        unsigned short code;
        string message;
};

enum DocType {
        HTML,
        XML
};
```

```
interface NameNodeMap {
        attribute long length;

        Node getNamedItem (
                in DOMString name );

        Node setNamedItem (
                in Node arg )
                  raises (DOMException);

        Node removeNamedItem (
                in DOMString name )
                  raises (DOMException);

        Node item (
                in long index );
};

interface DOMImplementation {
        boolean hasFeature (
                in DOMString feature,
                in DOMString version );
};

interface Node {
        attribute unsigned short ELEMENT_NODE;
        attribute unsigned short ATTRIBUTE_NODE;
        attribute unsigned short TEXT_NODE;
        attribute unsigned short CDATA_SECTION_NODE;
        attribute unsigned short ENTITY_REFERENCE_NODE;
        attribute unsigned short ENTITY_NODE;
        attribute unsigned short PROCESSING_INSTRUCTION_NODE;
        attribute unsigned short COMMENT_NODE;
        attribute unsigned short DOCUMENT_NODE;
        attribute unsigned short DOCUMENT_TYPE_NODE;
        attribute unsigned short DOCUMENT_FRAGMENT_NODE;
        attribute unsigned short NOTATION_NODE;
        attribute DOMString nodeName;
        attribute DOMString nodeValue;
        attribute unsigned short nodeType;
        attribute Node parentNode;
        attribute NodeList childNodes;
        attribute Node firstChild;
```

```
        attribute Node lastChild;
        attribute Node previousSibling;
        attribute Node nextSibling;
        attribute NameNodeMap attributes;
        attribute Document ownerDocument;

        Node insertBefore (
                in Node newChild,
                in Node refChild )
                  raises (DOMException);

        Node replaceChild (
                in Node newChild,
                in Node oldChild )
                  raises (DOMException);

        Node removeChild (
                in Node oldChild )
                  raises (DOMException);

        Node appendChild (
                in Node newChild )
                  raises (DOMException);

        boolean hasChildNodes ();

        Node cloneNode (
                in boolean deep );

        void setPreviousSibling__ (
                in Node newPrevSib );

        void setParentNode__ (
                in Node newParent );

        void setNextSibling__ (
                in Node newNextSib );
};

interface Entity : Node {
        attribute DOMString publicId;
        attribute DOMString systemId;
        attribute DOMString notationName;
};
```

```
interface NodeList {
        attribute unsigned long length;

        Node item (
                in long index );
};

interface EntityReference : Node { };

interface Element : Node {
        attribute DOMString tagName;

        DOMString getAttribute (
                DOMString name );

        void setAttribute (
                DOMString name,
                DOMString value )
                  raises (DOMException);

        void removeAttribute (
                DOMString name )
                  raises (DOMException);

        Attr getAttributeNode (
                DOMString name );

        Attr setAttributeNode (
                Attr newAttr )
                  raises (DOMException);

        Attr removeAttributeNode (
                Attr oldAttr )
                  raises (DOMException);

        NodeList getElementsByTagName (
                DOMString name );

        void normalize ();
};

interface DocumentType : Node {
        attribute DOMString name;
```

```
        attribute NameNodeMap entities;
        attribute NameNodeMap notations;
};

interface CharacterData : Node {
        attribute DOMString data;
        attribute long length;

        DOMString substringData (
                long offset,
                long count )
                  raises (DOMException);

        void appendData (
                DOMString arg )
                  raises (DOMException);

        void insertData (
                long offset,
                DOMString arg )
                  raises (DOMException);

        void deleteData (
                long offset,
                long count )
                  raises (DOMException);

        void replaceData (
                long offset,
                long count,
                DOMString arg )
                  raises (DOMException);
};

interface Comment : CharacterData { };

interface ProcessingInstruction : Node {
        attribute DOMString target;
        attribute DOMString data;
};

interface Notation : Node {
        attribute DOMString publicId;
        attribute DOMString systemId;
```

```
};

interface DocumentFragment : Node { };

interface Text : CharacterData {
        Text splitText (
                long offset )
                  raises (DOMException);
};

interface CDATASection : Text { };

interface Document : Node {
        attribute DocumentType doctype;
        attribute DOMImplementation implementation;
        attribute Element documentElement;

        Element createElement (
                DOMString tagName )
                  raises (DOMException);

        Text createTextNode (
                DOMString data );

        Comment createComment (
                DOMString data );

        CDATASection createCDATASection (
                DOMString data );

        ProcessingInstruction createProcessingInstruction (
                DOMString target,
                DOMString data )
                  raises (DOMException);

        Attr createAttribute (
                DOMString name )
                  raises (DOMException);

        EntityReference createEntityReference (
                DOMString name )
                  raises (DOMException);

        NodeList getElementsByTagName (
```

```
                DOMString tagname );
};

interface DOMFactory : BaseFactory {
        DOMImplementation createDOMImplementation ();

        Document createDocument ()
                raises (DOMFactoryException);

        void deleteDocument (
                in Document doc );

        Document getDocument (
                DOMString docurl,
                DocType doctype )
                  raises (DOMFactoryException);

        void putDocument (
                DOMString docurl,
                Document doc )
                  raises (DOMFactoryException);

        string getTextByName (
                in Document doc,
                in string tagname,
                in long tagnumber )
                  raises (DOMFactoryException);

        string getTextByNode (
                in Document doc,
                in Node node )
                  raises (DOMFactoryException);

};

interface Attr : Node {
        attribute DOMString name;
        attribute boolean specified;
        attribute DOMString value;
};

};

};
```

Anhang B

Soft- und Hardware

Die im folgenden aufgeführte Hard- und Software wurde zur Durchführung des Projekts verwendet. Für die Arbeit an diesem Projekt sind ähnliche Systemkonfigurationen notwendig.

B.1 Software

B.1.1 Windows

- WinCVS1.1.6b + TCL/TK 8.1, GUI für das Concurrent Version System (CVS) (www.wincvs.org + www.scriptics.com)
- Word97, Dokumentationsprogramm (www.microsoft.com/office)
- Powerpoint 97, Präsentationsprogramm (www.microsoft.com/office)
- Cygwin 20b1, Windows Portierung von UNIX Entwicklungs-Tools (sourceware.cygnus.com)
- VisualStudio 6 (SP3), VC++ und VB IDEs (www.microsoft.com/vstudio)
- RationalRose 98i Enterprise Edition, UML-Werkzeug (www.rational.com/rose)
- JBuilder3 Enterprise Edition, Java IDE (www.borland.com/jbuilder)
- Homesite 4.0.1, JSP und HTML Entwicklungswerkzeug (www.allaire.com/homesite)
- VisiBroker for C++ 3.3 und Java 3.4, C++ und Java ORB von Borland (www.borland.com/visibroker)
- Netscape 4.7 und InternetExplorer 5, Standard Web-Browser (home.netscape.com + www.microsoft.com/ie)

- Powerterm 5.2.2, Telnet-Programm (www.powerterm.com)
- XMLSpy 2.5, XML-Editor (www.xmlspy.com)
- Fulcrum SearchServer 3.7, Index-Server (www.pcdocs.com)
- PaintShopPro 6, Bildbearbeitung (http://www.jasc.com/)
- JDK 1.2.2 (incl. Documentation), Java2 DevelopmentKit (java.sun.com/products)
- Ghostscript+Ghostview 5.5, PS-Interpreter und Viewer (www.cs.wisc.edu/ ghost/index.html)
- Acrobat Reader 4.0, PDF-Viewer (www.adobe.com)
- UltraEdit32 6.10a, Texteditor (www.ultraedit.com)
- JWSDK1.0, Sun's JSP Referenzimplementation (java.sun.com/products)

B.1.2 Solaris

- Concurrent Version System (CVS), Versionskontrollsystem für Quelltexte (www.cyclic.com)
- Samba 2.0.5a, NT-Fileserver Dienst für UNIX (www.samba.org)
- VisiBroker for C++ 3.3 und Java 3.4, ORBs von Broland (www.borland.com/visibroker)
- Apache 1.3.9, HTTP-Server (www.apache.org)
- Apache JServ 1.0, Servlet-Engine (java.apache.org)
- JDK 1.2.2 (incl. Documentation), Java2 DevelopmentKit (java.sun.com/products)
- SPARCompiler 4.01, Sun's C++-Compiler (www.sun.com)

B.1.3 General

- Adaptive Computing Environment (ACE), Plattformunabhängige Bibliothek für Systemdienste (www.wustl.cs.edu/~schmidt/ACE)
- Expat 1.1, XML-Parser (www.jclark.com)
- STLport 3.2, Platform- und Compiler-unabhängige STL-Portierung (www.stlport.org)

B.2 Hardware

B.2.1 Windows

- Pentium II 350MHz
- 160 MB Ram
- 9 GB HD
- WinNT 4.0 SP5

B.2.2 Solaris

- Sun Enterprise 250
- 512MB Ram
- 36 GB HD
- SunOS 5.6

Literaturverzeichnis

[ACE] Adaptive Communication Environment. http://www.cs.wustl.edu/~schmidt/ACE.

[Bal96] Helmut Balzert. *Lehrbuch der Software-Technik - Software Entwicklung*. Spektrum Akademischer Verlag, 1996.

[Ben99] Steve Benfield. The Application Server Marketplace - Which Model is Right For You? *Web Techniques*, (2), February 1999.

[Ber96] Philip A. Bernstein. Middleware - A Model for Distributed System Services. *Communications of the ACM*, 39(2):86–98, February 1996.

[BL+99] Tim Berners-Lee et al. Hypertext Transfer Protocol – HTTP/1.1, June 1999. http://src.doc.ic.ac.uk/computing/internet/rfc/rfc2616.txt.

[BLC95] Tim Berners-Lee and Dan Connolly. Hypertext Markup Language - 2.0, November 1995. http://src.doc.ic.ac.uk/computing/internet/rfc/rfc1866.txt.

[Boo93] Grady Booch. *Object Oriented Analysis And Design With Applications*. Addison Wesley Longman, 2nd edition, 1993.

[Box98] Don Box. *Essential COM*. Addison Wesley Longman, Inc., December 1998.

[BRJ99] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison Wesley Longman, 1999.

[CCM98] CORBA Components - Volume I, July 1998. ftp://ftp.omg.org/pub/docs/formal/99-07-01.pdf.

[CCM99a] Aktualisierung, Anhang A: IDL Summary, August 1999. ftp://ftp.omg.org/pub/docs/orbos/99-08-12.pdf.

[CCM99b] Aktualisierung, Anhang B: XML DTDs, August 1999. ftp://ftp.omg.org/pub/docs/orbos/99-08-05.pdf.

[CCM99c] CORBA Components - Volume II, MOF-based Metamodels, August 1999. ftp://ftp.omg.org/pub/docs/orbos/99-07-02.pdf.

[CCM99d] CORBA Components - Volume III, Interface Repository, August 1999. ftp://ftp.omg.org/pub/docs/orbos/99-07-03.pdf.

[CGI] Common Gateway Interface Version 1.1 (CGI/1.1). http://hoohoo.ncsa.uiuc.edu/cgi/interface.html.

[Cla97] James Clark. Comparison of SGML and XML, December 1997. http://www.w3.org/TR/NOTE-sgml-xml-971215.

[Cla99] James Clark. XSL Transformation (XSLT) Version 1.0, November 1999. http://www.w3.org/TR/1999/REC-xslt-19991116.

[COM95] The Component Object Model Specification, 1995. http://www.microsoft.com/COM/resources/COM1598C.ZIP.

[Coo99] Clark Cooper. Benchmarking XML Parsers, A performance comparison of six stream-oriented parsers, May 1999. http://www.xml.com/pub/Benchmark/article.html.

[COR98a] CORBA Comparison Project, Final Project Report, June 1998. ftp://ftp.omg.org/pub/docs/formal/98-07-12.pdf.

[COR98b] Common Facilities Architecture, July 1998. ftp://ftp.omg.org/pub/docs/formal/98-07-01.pdf.

[COR98c] CORBAFinance: Financial Domain Specification, December 1998. ftp://ftp.omg.org/pub/docs/formal/98-12-02.pdf.

[COR98d] CORBAservices: Common Object Services Specification, December 1998. ftp://ftp.omg.org/pub/docs/formal/98-12-09.pdf.gz.

[COR98e] CORBAtelecoms: Telecommunications Domain Specifications, July 1998. ftp://ftp.omg.org/pub/docs/formal/98-07-12.pdf.

[COR99a] Common Object Request Broker Architecture, October 1999. ftp://ftp.omg.org/pub/docs/formal/99-10-07.pdf.gz.

[COR99b] CORBAmanufacturing: Manufacturing Domain Specifications, October 1999. ftp://ftp.omg.org/pub/docs/formal/99-10-01.pdf.

[COR99c] CORBAMed: Healthcare Domain Specifications, March 1999. ftp://ftp.omg.org/pub/docs/formal/99-03-01.pdf.

[CSS] CSS3 Working-Drafts Description. http://www.w3.org/Style/css/current-work#Descriptio.

[DCE] AES/Distributed Computing - Remote Procedure Call, Revision B. http://www.osf.org/mall/dce/free_dce.htm.

[Dea99] Stephen Deach. Extensible Stylesheet Language (XSL) Specification, April 1999. http://www.w3.org/TR/1999/WD-xsl-19990421.

[Fel] Boris Feldman. AppServer Zone - FAQ. http://www.appserverzone.com/appserverfaq.asp.

[Fin95] Rich Finkelstein. The New Middleware. *SIGMOD Record*, 24(4):102–106, December 1995.

[Ful96] Fulcrum. *Introduction to Search Server*. Fulcrum Technologies Inc., Ottawa, Canada, 1996.

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns - Elements of Reusable Object-Oriented Software*. Professional Computing Series. Addison Wesley Longman, October 1994.

[Gro98] Distributed Systems Research Group. CORBA Comparison Project, Final Project Report. Technical report, Department of Software Engineering, Faculty of Mathmatics and Physics, Charles University, Malostranske namesti 25, Prague, Czech Republic, June 1998. http://henya.ms.mff.cuni.cz/thegroup/COMP.

[H+98] Arnaud Le Hors et al. Document Object Model (DOM) Level 1 Specification, October 1998. http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001.

[H+99] Arnaud Le Hors et al. Document Object Model (DOM) Level 2 Specification, September 1999. http://www.w3.org/TR/1999/WD-DOM-Level-2-19990923.

[IET] The Tao of IETF – A Guide for New Attendees of the Internet Engineering Task Force. http://www.ietf.org/tao.html.

[Inp98a] Inprise. *VisiBroker for C++ 3.3, Installation and Administration*. Inprise Corporation, Los Angeles, United States of America, 1998.

[Inp98b] Inprise. *VisiBroker for C++ 3.3, Programmer's Guide*. Inprise Corporation, 100 Enterprise Way, Scotts Valley, CA 95066-3249, United States of America, 1998.

[Inp98c] Inprise. *VisiBroker for C++ 3.3, Reference*. Inprise Corporation, 100 Enterprise Way, Scotts Valley, CA 95066-3249, United States of America, 1998.

[Inp98d] Inprise. *VisiBroker for Java 3.3, Installation and Administration.* Inprise Corporation, 100 Enterprise Way, Scotts Valley, CA 95066-3249, United States of America, 1998.

[Inp98e] Inprise. *VisiBroker for Java 3.3, Programmer's Guide.* Inprise Corporation, 100 Enterprise Way, Scotts Valley, CA 95066-3249, United States of America, 1998.

[Inp98f] Inprise. *VisiBroker for Java 3.3, Reference.* Inprise Corporation, 100 Enterprise Way, Scotts Valley, CA 95066-3249, United States of America, 1998.

[ISO] ISO FAQ. http://www.iso.ch.

[ISO86] ISO. ISO 8879:1986, Information Processing – Text and office systems – Standard Generalized Markup Language (SGML), 1986. http://www.iso.ch.

[ISO92] ISO. ISO/IEC 9075:1992, Information Technology – Database Languages – SQL, 1992. http://www.iso.ch.

[ISO96] ISO. ISO/IEC 10179:1996, Information Technology – Processing Languages – Document Style Semantics and Specification Language (DSSSL), 1996. http://www.iso.ch.

[ISO98] ISO. ISO/IEC 14882-1998, Information Technology - Programming Languages - C++ (new standard), 1998. http://www.iso.ch.

[J2E99] Java 2 Platform, Enterprise Edition Application Programming Model, September 1999. http://java.sun.com/j2ee/apm.

[Jac94] Ivar Jacobson. *Object-oriented Software Engineering: A Use-Case Driven Approach.* Addison Wesley Longman, 2nd edition, 1994.

[Jav98] The Java Community Process Manual, December 1998. http://java.sun.com/aboutJava/communityprocess/ java_community_process.html.

[K+] Charly Kindel et al. Distributed Component Object Model Protocol – DCOM/1.0. http://msdn.microsoft.com/library/specs/ distributedcomponentobjectmodelprotocoldcom10.htm.

[Keu96] Warren Keuffel. Middleware on the Web. *Internet Systems*, (10), October 1996.

[Kir97] Mary Kirtland. The com+ programming model makes it easy to write components in any language. *Microsoft System Journal*, December 1997. http://www.msj.com.

[Kob99] Cris Kobryn. UML 2001: A Standardization Odyssey. *Communications of the ACM*, 42(10):29–37, October 1999.

[Kop96] Helmut Kopka. *LaTex Einführung*, volume Band 1. Addison-Wesley, 1996.

[Kru99] Philippe Kruchten. *The Rational Unified Process*. Addison Wesley Longman, 1999.

[LB98] Håkon Lie and Bert Bos. Using XSL and CSS together, September 1998. http://www.w3.org/TR/NOTE-XSL-and-CSS-19980911.

[LB99] Håkon Wium Lie and Bert Bos. Cascading Style Sheets, Level 1, CSS1 Specification, January 1999. http://www.w3.org/TR/1998/REC-CSS1-19990111.

[LBLJ98] Håkon Wium Lie, Bert Bos, Chris Lilley, and Ian Jacobs. Cascading Style Sheets, Level 2, CSS2 Specification, Mai 1998. http://www.w3.org/TR/1998/REC-CSS2-19980512.

[Lin97] David S. Linthicum. Next Generation Middleware. *DBMS Magazine*, (9), September 1997.

[Mat99] Thomas Mattern. Lösungsansätze für eine unternehmensweite Integration auf Basis von Applikationsservern. *OBJEKTspektrum*, (6):28–34, November/December 1999.

[MH99] Vlada Matena and Mark Hapner. Enterprise JavaBeans Specification, v1.1. Technical report, Sun Microsystems, Inc., October 1999. Preliminary, Public Release 2.

[MTS] Comparing Microsoft Transaction Server to Enterprise JavaBeans. http://www.microsoft.com/com/wpapers/mts-ejb.asp.

[OMA97] A Discussion of the Object Management Architecture, January 1997. http://www.omg.org/library/oma/oma-all.pdf.

[OMG97] Policies and Procedures of the OMG Technical Process, June 1997. http://www.omg.org/docs/pp/97-06-01.txt.

[OMG99] Amended and Restated By Laws of Object Management Group, Inc., August 1999. http://www.omg.org/docs/pp/99-08-01.txt.

[PC-] PC-Docs/Fulcrum Homepage. http://www.pcdocs.com.

[Pou98] Dick Pountain. Get The Message. *BYTE*, (7):7–11, July 1998.

[PR-97] HTML 3.2 Reference Specification, January 1997. http://www.w3.org/TR/1997/REC-html32-19970114.

[R+99] Dave Ragett et al. XHTML 1.0: The Extensible HyperText Markup Language, August 1999. http://www.w3.org/TR/1999/PR-xhtml1-19980824.

[Rag95] Dave Ragett. HyperText Markup Language Specification Version 3.0, September 1995. http://www.w3.org/MarkUp/html3/CoverPage.

[RBP+93] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Objektorientiertes Modellieren und Entwerfen.* Hanser und Prentice-Hall Internaltional, 1993.

[RHJ98] Dave Ragett, Arnaud Le Hors, and Ian Jacobs. HTML 4.0 Specification, April 1998. http://www.w3.org/TR/1998/REC-html40-19980424.

[RHJ99] Dave Ragett, Arnaud Le Hors, and Ian Jacobs. HTML 4.01 Specification, August 1999. http://www.w3.org/TR/1999/PR-html40-19980824.

[Rit98] David Ritter. The Middleware Muddle. *DBMS Magazine*, May 1998.

[RW99] Chris Vander Rhodes and Carol Wolicki. The OMG Outlines Application Server Standard - Enterprise JavaBeans and CORBA Form a Common, Compatible Platform for Application Servers, April 1999. http://www.omg.org/news/pr99/4_6.html.

[Rym96] John R. Rymer. The Muddle in the Middle. *BYTE*, April 1996.

[Sch99] Sebastian Schöning, editor. *Traditionelle und Verteilte Betriebssysteme.* FB14, Lehrstuhl für verteilte Systeme, 1998-1999. Vorlesungsskript.

[Skr99] Meinhard Skrzypek. Komponentenmodellierung und der Rational Unified Process. *OBJEKTspektrum*, (4), July/August 1999.

[SL95] Alexander Stepanov and Meng Lee. The Standard Template Library. Technical report, Silicon Graphics Inc. and Hewlett-Packard Laboratories, 1995.

[Som96] Ian Sommerville. *Software Engineering.* Addison-Wesley, 1996.

[STL] STLport. http://www.stlport.org.

[Str99] Bjarne Stroustrup. *The C++ Prgramming Language.* Addison Wesley Longman, 3 edition, 1999.

[Sun99a] Sun. JavaServer Pages Specification. Technical report, Sun Microsystems, Inc., October 1999. Version 1.1 - Public Release 2.

[Sun99b] Sun. Simplified Guide to the Java 2 Platform, Enterprise Edition. Technical report, Sun Microsystems, 1999.

[TAO] The ACE ORB. http://www.cs.wustl.edu/~schmidt/TAO.

[Tho99] Anne Thomas. Java 2 platform, enterprise edition - ensuring consistency, portability and interoperability. Technical report, Patricia Seybold Group, June 1999. Prepared for Sun Microsystems.

[Tre96] Markus Tresch. Middleware: Schlüsseltechnologie zur Entwicklung verteilter Informationssysteme. *Informatik Spektrum*, 19:249–256, August 1996.

[W+97] Yi-Min Wang et al. Dcom and corba side by side, step by step, and layer by layer. Technical report, Microsoft Research, Redmond Washington, 1997.

[W3C] W3 Consortium. http://www.w3.org.

[Wri99] Guy Wright. What is an Application Server? *Web Review*, (26), February 1999. http://www.webreview.com/wr/pub/sections/appserver/index2.html.

Tabellenverzeichnis

5.1 Vergleich der Rollen der verschiedenen Objektmodelle mit den KN2 Use-Cases 73

6.1 Vergleich der KN2 Implementation mit dem CCM 90
6.2 Benchmarkergebnisse aktueller XML-Parser (Ergebnisse in Sekunden, MD = Markup Density) 96

7.1 Merkmale von Kommunikationsmodellen 111

Abbildungsverzeichnis

1.1 Beispielarchitektur einer WWW-Applikation 9
1.2 CountryNet Homepage 14
1.3 CountryNet Länderauswahl 15
1.4 CountryNet Navigationsseite (Relocation/Key Facts) 15
1.5 CountryNet Anfrageseite 16
1.6 CountryNet Suchergebnisse 16
1.7 CountryNet Administration 17

2.1 Teildisziplinen der Software Technik 19
2.2 Der Rational Unified Process (RUP) 21
2.3 Anwendungen von XML 26
2.4 Baumdarstellung der XML E-Mail 28
2.5 Kombinationsmöglichkeiten von CSS und XSL 29
2.6 HTML-Ausgabe der mit XSL konvertierten E-Mail. 31
2.7 Referenzmodell der Object Management Architecture 35
2.8 Zeitliche Entwicklung der UML 39

3.1 CountryNet Projektstruktur 46
3.2 KN1 - Architektur 48
3.3 Ablaufdiagramm der NetDiff-Hauptschleife (Teil 1) 49
3.4 Ablaufdiagramm der NetDiff-Hauptschleife (Teil 2) 50
3.5 Ablaufdiagramm der NetDiff-Hauptschleife (Teil 3) 51
3.6 HTML-Template (Anweisungen markiert) 52

4.1 UML Use-Case: Administration 63
4.2 UML Use-Case: Software-Entwicklung 63

5.1 J2EE Application Programming Model (J2EE APM) 67
5.2 KN2 Application Programming Model (KN2 APM) 68
5.3 CCM Container Model 71
5.4 CCM Component Server 72
5.5 UML-Package-Diagramm: KN 73
5.6 UML-Klassendiagramm: Global 74
5.7 UML-Klassendiagramm: DOM 76
5.8 UML-Klassendiagramm: SearchNetwork 78

5.9 UML-Klassendiagramm: Fulcrum 85

6.1 Apache Architektur-Diagramm 89
6.2 UML Komponenten-Diagramm der Laufzeitumgebung 92
6.3 Benchmarkergebnisse aktueller XML-Parser (Ergebnisse in Sekunden) 97
6.4 Formular für die Volltext-Suche in der KnowledgeBase 103
6.5 Suchergebnisse einer Volltext-Suche in der KnowledgeBase . . 104
6.6 XSL-Anzeige eines Such-Ergebnisses 104
6.7 Formular zum Hinzufügen eines Eintrags in die KnowledgeBase 105
6.8 UML-Kollaborationsdiagramm: Erzeugung der Instanzen des Anwendungsbeispiels 106
6.9 UML-Kollaborationsdiagramm: Überführung der Formulardaten in ein XML-Dokument und Speicherung des Dokuments im Repository (Aus Gründen der Übersichtlichkeit wird nur der PAPER-Tag eingefügt) 106
6.10 UML-Kollaborationsdiagramm: Volltext-Suche auf dem Index 106

7.1 Klient-Server Rolle 111
7.2 Middleware Rolle 112