Effiziente Entwicklung von J2EE-basierten Serveranwendungen durch Einsatz frei verfügbarer Komponenten

Diplomarbeit im Fach Informatik

an der
Eberhard-Karls-Universität Tübingen

vorgelegt von Tobias Frech

Tübingen im Mai 2003

1. Berichterstatter:	Prof. Dr.-Ing. Wilhelm G. Spruth
2. Berichterstatter:	Prof. Dr. rer. nat. Wolfgang Rosenstiel
Betreuer:	Dr. Dipl.-Inform. Klaus Beschorner

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe.

Tübingen, 1. Mai 2003 (Tobias Frech)

Inhaltsverzeichnis

Abbildungsverzeichnis **VII**

1 Einleitung **1**

1.1 Motivation 1

1.2 Aufbau der Arbeit 2

2 Problemstellung **5**

3 Grundlagen **9**

3.1 Java 2 Enterprise Edition 9

3.1.1 Enterprise JavaBeans 10

3.1.2 Webanwendungen 14

3.1.3 Dienste 15

3.2 Komponenten 17

4 Problemanalyse und Lösungskonzept **23**

4.1 Problematik der Komponentensuche und -auswahl 23

4.2 Alternativen der Strukturierung 25

4.3 Einzelfunktionalitäten 29

4.4 Spreading Activation 30

4.4.1 Verfahren 31

4.4.2 Anwendung 32

4.5 Reifebeurteilung mittels Integrationsgraph 33

4.6 Iterative und interaktive Suche 34

4.7 Lösungskonzept 36

4.7.1 Funktionalitäten 36
4.7.2 Suchbegriffe 37
4.7.3 Komponentenqualität 37
4.7.4 Interaktive Suche 37
4.7.5 Weitere Attribute 38
4.7.6 Lösungsanforderungen 38

5 Design und Umsetzung **41**
5.1 Arten von Prototypen 41
5.2 Grundsätzlicher Aufbau 43
5.3 Persistenz und Datenmodell 46
5.3.1 Entity Beans 46
5.3.2 Container Managed Persistence und Container Managed Relationships 50
5.3.3 Xdoclet 52
5.3.4 Entitäten und Datenmodell 54
5.4 Anwendungslogik 56
5.4.1 Alternativen 57
5.4.2 Umsetzung 61
5.5 Präsentation und Ablaufsteuerung 64
5.5.1 Alternativen 64
5.5.2 Tapestry 68
5.6 Packen und Installation 73
5.6.1 JAR 74
5.6.2 WAR 74
5.6.3 EAR 75
5.6.4 Installation 75
5.7 Beispiel eines Suchdurchlaufs 75
5.8 Technische Systemanforderungen 81

6 Zusammenfassung und Ausblick **83**

A Xdoclet **87**

B Server-Einrichtung **95**

B.1 JBoss mit Tapestry 95

B.2 PostgreSQL 98

B.3 Integration JBoss und PostgreSQL 101

C Entwicklungsumgebung **103**

C.1 Ant 103

C.2 AntSCP 108

C.3 Eclipse mit JBoss-IDE-Plugin 109

Literaturverzeichnis **113**

Zusammenfassung

Bei der Entwicklung von J2EE-basierten Serveranwendungen kann durch den Einsatz von bereits bestehenden und frei verfügbaren Komponenten der Entwicklungsaufwand reduziert werden. In dieser Diplomarbeit wird zunächst untersucht, welche Hindernisse dem Einsatz solcher Komponenten entgegenstehen. Basierend auf dieser Analyse werden dann Lösungsansätze zur effizienten Suche nach Komponenten sowie deren Vergleichbarkeit untereinander und die Einarbeitung in Komponenten vorgestellt. Sowohl diese Hindernisse als auch der Terminologieunterschied zwischen dem Entwickler und der Komponentendokumentation werden mit einem neu entwickelten System angegangen. Zur Speicherung der Komponenten in diesem System wird eine angemessene Repräsentation vorgestellt. Das System selbst ist auf J2EE basierend mit freien Komponenten implementiert und dient als Beispiel, wie sich Komponenten effizient in der Entwicklung einsetzen lassen. Ebenso werden der Aufbau einer mehrschichtigen J2EE-Server-Architektur, deren Konfiguration und Integration sowie die Zusammenstellung einer leistungsfähigen J2EE-Entwicklungsumgebung beschrieben.

Für meine Eltern,

die mir meinen Weg bis hierhin erst ermöglicht haben.

Danksagung

An dieser Stelle möchte ich allen danken, die mich während der Entstehung dieser Diplomarbeit auf vielfältige Art und Weise unterstützt haben.

Mein besonderer Dank gilt meinem Betreuer Klaus Beschorner, der sich bereit erklärt hat, dieses selbstgewählte Thema zu betreuen und mich mit Anregungen und konstruktiver Kritik bei der Durchdringung der komplexen Thematik unterstützt hat.

Für die Ermöglichung dieser Diplomarbeit möchte ich mich bei Herrn Prof. Spruth und bei Herrn Prof. Rosenstiel (Lehrstuhl Technische Informatik) bedanken.

Aus den vielen Personen, mit denen ich über die Thematik gesprochen habe, möchte ich Prof. Schweizer und Axel Braun herausheben, die mit mir diskutiert und mir neue Blickwinkel auf die Problematiken eröffnet haben. Meinen Dank möchte ich allen genannten und ungenannten Diskussionspartnern aussprechen.

Dankbar bin auch Carsten Schulz-Key für seine kompetente und rasche Unterstützung bei technischen Problemen und Anfragen bezüglich der Infrastruktur.

Für die freie Software, die in dieser Diplomarbeit Verwendung gefunden hat, und ohne welche sie nicht in diesem Umfang möglich gewesen wäre, gebührt den jeweiligen Autoren ebenfalls mein Dank.

Für die Durchsicht der Diplomarbeit und wertvollen Anregungen danke ich ganz herzlich Ulrich Hoffmann und Michael Bensch. Aus den gleichen Gründen und vor allem für die Geduld, die sie während der Erstellung der Diplomarbeit für mich aufgebracht hat, möchte ich meiner Freundin Gabi in Liebe danken.

Abbildungsverzeichnis

2.1 Das Model-View-Controller-Prinzip: Die Applikationsdaten werden im Model gehalten, der Controller nimmt die Benutzereingaben entgegen und steuert die View, welche wiederum die Daten aus dem Model visualisiert. 6

3.1 Die Auswahl einer Art von Enterprise JavaBeans anhand verschiedener Kriterien. 11

3.2 Jede EJB benötigt eine Implementierungsklasse und die beiden Schnittstellen *Remote Interface* und *Home Interface*. Die Implementierungsklasse läuft auf dem Applikationsserver, welcher auch die *Home Interface* Schnittstelle implementiert. Die beiden Schnittstellen werden auf dem Klienten zum Steuern der Serverimplementierungen verwendet. 13

3.3 Die Trennung nach dem Model-View-Controller-Konzept geschieht durch die Aufspaltung der Aufgaben auf Servlet, JSPs und JavaBeans (vgl. [SES99]). 15

4.1 Die Aufgabenstellung „Diskussionsforum“ wird vertikal in Schichten und horizontal in Einzelteile zerlegt, die jeweils abgegrenzte Aufgabengebiete haben. Die unter den Einzelteilen angegebenen Namen stellen eine Möglichkeit dar, die Einzelteile durch Komponenten oder J2EE-Bestandteile abzudecken. Mit „J2EE“ beginnende Namen kennzeichnen dabei eine J2EE-Technologie. 24

4.2 Die vier möglichen Grundmethoden zur Klassifikation. Das Quadrat repräsentiert jeweils das zu klassifizierende Objekt. Die Kreise stellen die verschiedenen Kategorien bzw. Attributsausprägungen dar. 26

4.3 In diesem Integrationsgraphen ist eine Auswahl an Projekten der Apache Software Foundation mit ihren Integrationsabhängigkeiten untereinander dargestellt. 35

5.1 Für die verschiedenen Schichten ist die Zuordnung zu einzelnen Teilen des Gesamtsystem dargestellt. Der J2EE-Applikationsserver deckt die mittleren Teile des Systems ab. 45

5.2 Das Klassendiagramm der Datenschicht. 56

5.3 Die Erzeugung der Session Beans für die kriterienbasierte Suche und Kooperation zwischen diesen bei Aufrufen durch den Klienten. 63

5.4 Das Aktivitätsdiagramm der Interaktion zwischen Browser und Tapestry-Komponenten bei der Anzeige einer Beschreibung (Description EJB). 71

5.5 Die Darstellung der verschiedenen Zustände der Benutzerschnittstelle in Abhängigkeit von den Eingaben des Benutzers. 73

5.6 Die Browser-Anzeige nach dem Aufruf der Startseite zeigt alle dem System bekannten Begriffe und Komponenten an. Der Benutzer kann nun ein Kriterium ein- oder ausschliessen oder einen Suchbegriff eingeben. 76

5.7 Die Browser-Anzeige nach der Eingabe des Suchbegriffs „Design“. Bei den Komponenten und Begriffen, in deren Beschreibung der Suchbegriff gefunden wurde, wird die Raute vor dem Begriff blau gefärbt und größer dargestellt. 77

5.8 Die Browser-Anzeige nach der Auswahl des Kriteriums „Dekorator/Template einschliessen“. Für die anderen Kriterien hat das System die Anzeige aktualisiert, welche Begriffe implizit ein- oder ausgeschlossen wurden und welche noch als relevant ausgewählt werden können. Die Auflistung der Komponenten wurde auf die Menge der kriterienerfüllenden eingeschränkt. 78

5.9 Die Browser-Anzeige nach der Auswahl des zweiten Kriteriums „JavaScript einschliessen“. Es verbleibt nur noch eine Komponente und es können keine weiteren Kriterien hinzugefügt werden. . . . 79

5.10 Die Browser-Anzeige mit der Detailansicht einer ausgewählten Komponente zeigt alle im System enthaltenen Informationen inklusive einer Beschreibung der Funktionsweise der Komponente an. 80

C.1 Durch das JBoss-Plugin wird die „Content Assist“-Funktionalität in Eclipse um die deklarativen Tags für Xdoclet erweitert. 110

C.2 Die Konfiguration der auszuführenden Ant-Targets in Eclipse. . . 111

Kapitel 1

Einleitung

1.1 Motivation

Die *Java 2 Enterprise Edition* (J2EE) Spezifikation von Sun Microsystems bildet die Grundlage für einen Markt von Applikationsservern, der 2002 laut der Giga Information Group eine Größe zwischen 2 und 2,8 Milliarden Dollar erreichte (vgl. [WEI03]). In den letzten Jahren wurden zunehmend unternehmenskritische Prozesse und zentrale Funktionalitäten in Programmen, welche auf J2EE-Applikationsservern ablaufen, umgesetzt. Die Unternehmen können von mehreren etablierten Herstellern Applikationsserver beziehen, welche die J2EE-Spezifikation implementieren und als J2EE-konform zertifiziert sind. Die Entwickler von Programmen, die auf einem J2EE-konformen Server laufen, können für ihre Programme eine Reihe von Diensten des Applikationsservers in Anspruch nehmen und müssen diese nicht selbst programmieren. Ebenso stehen Schnittstellen für ein inzwischen ausgereiftes Objektmodell und für die vereinfachte Handhabung von Anfragen über das HTTP-Protokoll zur Verfügung. All diese Funktionalitäten sind durch die J2EE-Spezifikation standardisiert und stehen damit auf jedem J2EE-konformen Applikationsserver zur Verfügung. Der Entwickler kann sich somit stärker auf die anwendungsspezifischen Anforderungen konzentrieren und muss nicht die in diesem Bereich häufig benötigten Funktionalitäten, wie zum Beispiel für die Infrastruktur, selbst implementieren.

In der J2EE-Umgebung sind nun mehr und mehr Softwarekomponenten entstanden, welche die J2EE-Spezifikation ergänzen oder mit Teilen dieser konkurrieren. Insbesondere auf der Ebene der HTTP-Verarbeitung und HTML-Benutzerschnittstellen stehen viele Frameworks, wie zum Beispiel „Struts“ (vgl. [HUS02]), zur Verfügung, die durch verschiedene Ansätze den Funktionsumfang von J2EE sehr praxisrelevant erweitern. Die Anzahl solcher Komponenten bewegt sich zum jetzigen Zeitpunkt bei ca. 50 und erhöht sich nahezu wöchentlich.

Ein Entwickler kann nun den Gesamtaufwand einer Anwendungsentwicklung um eine Menge an Konzeptions-, Entwicklungs- und Testaufwand reduzieren, wenn er die richtigen Komponenten aus den zur Verfügung stehenden Komponenten auswählt (vgl. [FRA94]). Die Effizienz des Entwicklungsprozesses kann um so weiter verbessert werden, je mehr Teile einer Anwendung mit Unterstützung bereits verfügbarer und ausgetesteter Komponenten umgesetzt werden können. Die Entwickler können sich dann noch stärker auf die anwendungsspezifischen Anforderungen konzentrieren. Durch die Möglichkeit, aus vielen verschiedenen Komponenten auswählen zu können, bleibt die Flexibilität erhalten, sich auf die spezifischen Anforderungen an Leistungsumfang und Ressourcenbedarf anzupassen.

Der Nutzung all dieser geschilderten Vorteile durch den Einsatz von bereits verfügbaren Komponenten stehen jedoch zwei Probleme entgegen: Zuerst muss ein Entwickler die für seine Aufgabenstellung nützlichen Komponenten suchen und identifizieren, um sich anschließend in die jeweilige Art und Weise, wie die Komponenten zu verwenden sind, einzuarbeiten. Dabei können diese beiden Schritte leider selten getrennt voneinander betrachtet werden, da es bisher oft notwendig ist, sich teilweise in eine Komponente einzuarbeiten, um die Leistungsfähigkeit und die Angemessenheit der Komponente für die gestellte Aufgabe ausreichend beurteilen zu können.

Der vorliegenden Arbeit lag die Aufgabenstellung zu Grunde, den Entwickler bei den beiden Hindernissen „Suche“ und „Einarbeitung“ zu unterstützen. Dadurch soll eine stärkere Nutzung von bereits verfügbaren Komponenten ermöglicht und damit auch die Effizienz für die Entwicklung von Programmen im J2EE-Bereich erhöht werden. Diese Unterstützung sollte durch ein geeignetes System umgesetzt werden und möglichst vielen Entwicklern zugänglich sein.

1.2 Aufbau der Arbeit

Die in der Einleitung beschriebene Problemstellung wird in Kapitel 2 detaillierter erläutert. Zudem werden fachliche und technische Kriterien für einen Lösungsansatz festgelegt. In Kapitel 3 wird als Grundlage ein Überblick über die J2EE-Technologie gegeben. Weiterhin wird der verwendete Komponentenbegriff definiert und die betrachteten Komponentenarten näher festgelegt. Die Problemstellung wird dann in Kapitel 4 ausführlich analysiert und die gefundenen Lösungsansätze werden vorgestellt. Das Kapitel 5 befasst sich mit der Umsetzung dieser Lösungsansätze in ein ablauffähiges System. Dabei werden die getroffenen Designentscheidungen vorgestellt und die konkrete Umsetzung anhand einzelner Beispiele illustriert. Das Kapitel schließt mit einem Beispiel einer Benutzersitzung ab. Abschließend werden die erreichten Ergebnisse in Kapitel 6 zusammengefasst und es wird ein Ausblick auf Weiterentwicklungsmöglichkeiten der erstellten Lösung gegeben. Der Anhang dokumentiert die in der Arbeit aufgebaute Infra-

struktur an Entwicklungswerkzeugen (A), Serverinstallationen und -integration (B) und Entwicklungsumgebung (C).

Kapitel 2

Problemstellung

Die Zielsetzung, den Aufwand für die Anwendungsentwicklung im J2EE-Bereich durch Komponentenverwendung zu reduzieren, soll durch ein Programmsystem erreicht werden, das es zu entwickeln und zu implementieren gilt. Dieses System soll bei der Suche nach hilfreichen Komponenten und bei der Einarbeitung in diese unterstützen. Damit ein Entwickler mit einem solchen System sinnvoll arbeiten kann, müssen die Suche nach einer passenden Komponente, die notwendige Einarbeitung und die Entwicklung mit der ausgewählten Komponente zusammen weniger Aufwand verursachen, als die Entwicklung der gewünschten Funktionalität ohne jegliche Komponente benötigt hätte. Somit reicht es nicht aus, dass lediglich die Entwicklung mit einer Komponente effizienter ist als ohne die selbe, sondern auch die Suche und die Einarbeitung müssen so unterstützt werden, dass sie mit geringem Aufwand durchgeführt werden können. Bisher musste für die Suche nach Komponenten eine nicht spezialisierte Suchmaschine oder eine Auflistung von Komponenten verwendet werden, um in Frage kommende Komponenten zu identifizieren. Für jede dieser Komponenten musste nun die entsprechende Webseite aufgesucht und die dort vorhandenen Informationen gesichtet werden. Das Herausfiltern der relevanten Information ist dabei sehr aufwendig, da von Kandidat zu Kandidat sowohl die Informationen unterschiedliche Strukturierungen und Navigation aufweisen, als auch in einem oder mehreren der vielen verschieden Informationsformate vorliegen können: Webseiten, Benutzerdokumentation, Tutorials, How-Tos, Mailinglisten, Foren oder Wikis[1]. Für einen Entwickler ist dieser Aufwand eher abschreckend, zumal vor Beginn der Suche nicht einmal klar ist, ob überhaupt eine geeignete Komponente gefunden werden kann. In der Industrie haben deshalb hauptsächlich nur diejenigen Komponenten breite Verwen-

[1] Ein Wiki ist ein System aus Webseiten deren Inhalt üblicherweise *jeder* Benutzer verändern darf. Durch eine Versionierung des Inhalts können die Folgen von Vandalismus rückgängig gemacht werden. Ein Wiki hat einen selbstorganisierenden Charakter und kann sehr rasch und unkompliziert Informationen strukturiert aufnehmen. Eine detaillierte Erklärung von Wikis ist als Wiki-Seite unter `http://c2.com/cgi/wiki?WhyWikiWorks` abrufbar.

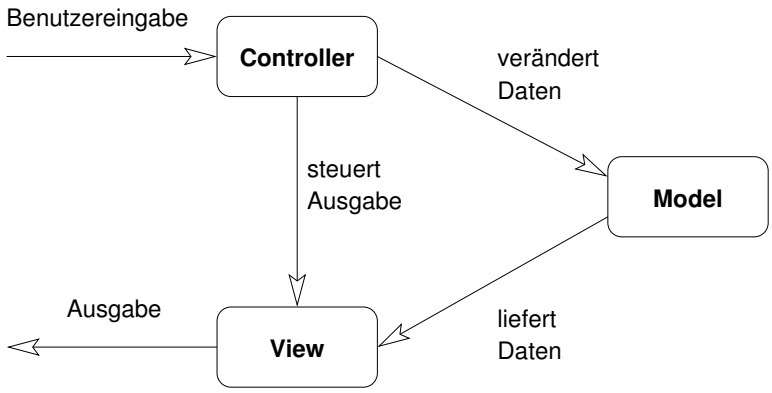

Abbildung 2.1: *Das Model-View-Controller-Prinzip: Die Applikationsdaten werden im Model gehalten, der Controller nimmt die Benutzereingaben entgegen und steuert die View, welche wiederum die Daten aus dem Model visualisiert.*

dung gefunden, die schon etwas Verbreitung in *Open Source* Projekten haben und unter Entwicklern weiterempfohlen wurden, oder die zum Beispiel über Artikel in Fachzeitschriften einem breiten Publikum vorgestellt wurden. Zwei prominente Beispiel für solche Komponenten sind zum einen „Log4J“, welches die Protokollierung mit dynamisch einstellbarer Genauigkeit erlaubt, und zum anderen „Struts“, welches für HTML-Benutzerschnittstellen eine saubere Trennung von Aufgaben nach dem *Model-View-Controller*-Prinzip (vgl. [KRA88] und Abbildung 2.1) innerhalb von J2EE-Anwendungen ermöglicht.

Um die oben beschriebene Suche effizienter gestalten zu können, muss zuerst für alle zu vergleichenden Komponenten eine einheitliche Repräsentation der verschiedenen Komponenteneigenschaften vorliegen. Der Entwickler benötigt für eine fundierte Entscheidung einen ausreichenden Überblick über alle möglichen Alternativen. Ohne eine einheitliche Repräsentation wäre allein dieser Überblick nur mit einem erheblich größeren Rechercheaufwand in den wie vorher dargestellt vielfältigen Informationsquellen möglich. Das zu entwickelnde System muss also eine adäquate Repräsentation aller in Frage kommenden Komponenten selbst bereitstellen. Dabei ist auf Ausgewogenheit der Repräsentation zwischen gebotener Kürze für eine möglichst hohe Effizienz und notwendiger Ausführlichkeit für eine ausreichende Detailinformation zu achten.

Das System muss weiterhin einen effizienten Suchmechanismus zur Suche in den Komponentenrepräsentationen anbieten. Dazu ist es notwendig, dem Benutzer des Systems die Möglichkeit zu geben, die Kriterien der Suche in Art, Ausprägung und Anzahl zu variieren (vgl. [WIL84]). Nur so kann der Benutzer abhängig von seinem Kenntnisstand vermeiden, dass er entweder mit Informationen aus dem System überflutet wird oder das System auf Grund zu restriktiver Kriterien keine Ergebnisse zur Suche liefert.

Um die Einarbeitung des Entwicklers in eine Komponente zu vereinfachen, muss die Repräsentation der Komponente so aufgebaut sein, dass sich der Entwickler schnell darüber orientieren kann, wie er die Komponente zur Problemlösung einsetzen kann. Zum einen muss er dazu einen Überblick über die gesamte Funktionsweise der Komponente und das interne Zusammenspiel der verschiedenen Komponententeile bekommen. Zum anderen muss er schnell identifizieren können, welcher Teil einer Komponente für welche Funktionalität eingesetzt werden kann.

Die Anforderungen an ein solches System lassen sich bis hier wie folgt zusammenfassen:

- adäquate Repräsentation der Komponenten im System
- schnelle Übersicht über die in Frage kommenden Alternativen
- effiziente Unterstützung der Suche mit flexiblen Kriterien
- Überblick über die Funktionsweise der Komponente geben
- Komponententeile verschiedenen Funktionalitäten zuordnen

Auf Grund der Neuheit eines solchen Systems ist damit zu rechnen, dass an der Funktionalität des Systems und an den aufgenommen Daten über Komponenten später Erweiterungen vorgenommen werden sollen. Um eine bessere Erweiterbarkeit zu gewährleisten, sollten die Funktionalitäten des Systems modularisiert umgesetzt werden. Ebenso sollten die zentralen Funktionen von der Präsentation getrennt realisiert werden, damit das System für andere als die realisierte Benutzerschnittstelle offen bleibt. Die Daten über Komponenten sollten in einem standardisierten Format gespeichert werden, damit eine nachträgliche Erweiterung der Struktur oder eine eventuelle Transformation der Daten möglich bleibt. Diese zusätzlichen technischen Anforderungen lassen sich wie folgt zusammenfassen:

- hohe Erweiterbarkeit durch modularisierte Implementierung der Funktionalitäten
- Flexibilität für verschiedene Benutzerschnittstellen durch Trennung von Präsentation und zentralen Funktionalitäten
- standardisierte Speicherung der Komponentendaten (zum Beispiel in einer relationalen Datenbank)

Kapitel 3

Grundlagen

Dieses Kapitel stellt den Inhalt der J2EE-Spezifikation von Sun Microsystems vor und beschreibt die Art der Komponenten, die mit dem System erfasst werden sollen. Die Beschreibung der J2EE-Spezifikation gliedert sich dabei in die drei folgenden Bereiche: Enterprise JavaBeans (EJB, Programmlogik und Datenspeicherung), Webanwendungen (HTTP- und HTML-Verarbeitung) und Querschnittsdienste. In der anschließenden Beschreibung des Begriffs „Komponente“ wird erläutert, welche Komponentendefinition in dieser Arbeit verwendet wird und welche weiteren Eigenschaft die betrachteten Komponenten aufweisen müssen.

3.1 Java 2 Enterprise Edition

Der Begriff „Java 2 Enterprise Edition“, kurz J2EE, steht für eine von der Firma Sun Microsystems erarbeitete und verabschiedete Spezifikation (vgl. [SUN01]). Die J2EE-Spezifikation soll sowohl die organisatorische Vorgehensweise als auch die Schnittstellen zur Verwendung von bestimmten Diensten standardisieren. Die beschriebenen Dienste werden vor allem für die Entwicklung von serverbasierten Anwendungen verwendet, die in unternehmenskritischen Bereichen angesiedelt sind und potentiell eine hohe Komplexität aufweisen können. Die J2EE-Anwendungen laufen auf einem Applikationsserver ab, der die spezifizierten Dienste für die Anwendung erbringt. Durch die J2EE-Spezifikation ist ein neuer Markt entstanden, auf dem es mehrere Anbieter von J2EE-konformen Applikationsservern gibt. Die Anzahl der Anbieter stieg zuerst rasch an, sank dann aber wieder auf eine überschaubare Anzahl an etablierten Anbietern, die sich mit ausgereiften Applikationsserverprodukten auf diesem Markt durchsetzen konnten. Auch Sun selbst bietet ein Applikationsserverprodukt gemäß der J2EE-Spezifikation an. Weiterhin stellt Sun eine Referenzimplementierung der J2EE-Spezifikation

bereit. Zum einen garantiert diese Implementierung die Umsetzbarkeit der Spezifikation und kann auch während der Entwicklung von J2EE-Anwendungen genutzt werden. Für den Produktionsbetrieb darf die Referenzimplementierung aus lizenzrechtlichen Gründen aber nicht verwendet werden und ist technisch dafür auch nicht geeignet. Zum anderen bietet Sun eine kostenpflichtige Lizensierung und Zertifizierung für J2EE-Applikationsserver an. Für die Zertifizierung steht ein Testwerkzeug zur Verfügung, das verschiedene Funktionstests durchführt. Dadurch soll garantiert werden, dass sich die Applikationsserver an ihre Seite des „Vertrages“ halten, der zwischen Benutzer und Applikationsserver in der J2EE-Spezifikation formuliert ist.

In einem speziellen Prozess – „Java Community Process“ (JCP) getauft – wird die J2EE-Spezifikation gemeinsam von Sun, Applikationsserver-Herstellerfirmen und Experten aus dem J2EE-Gebiet weiterentwickelt (vgl. [SUN02]). Dabei wird nicht nur die J2EE-Spezifikation selbst, sondern auch alle weiteren Spezifikationen, auf welche sich die J2EE-Spezifikation bezieht, weiterentwickelt. Diese Teilspezifikationen werden im Allgemeinen als einzelne Teile der J2EE-Spezifikation betrachtet. Die in dieser Arbeit verwendete J2EE-Spezifikation trägt die Versionsnummer 1.3 (vgl. [SUN01]). In dieser Spezifikation wird unter anderem auf die folgenden wichtigen Spezifikationen verwiesen, die ebenfalls in dieser Arbeit verwendet wurden: EJB-Spezifikation Version 2.0 (vgl. [MIC01]), Servlet-Spezifikation Version 2.3 (vgl. [COW01]), JSP-Spezifikation Version 1.2 (vgl. [PEL01]).

3.1.1 Enterprise JavaBeans

Der Bereich der Enterprise JavaBeans (EJB) wird durch die bereits erwähnte EJB-Spezifikation Version 2.0 abgedeckt. Als EJBs werden Objekte bezeichnet, die zusammen mit ihren zugehörigen Schnittstellen und Metadaten ein Objektmodell bilden, dessen Umgebung von einem Applikationsserver verwaltet wird. Dieses Objektmodell ist in der Spezifikation detailliert definiert. Es deckt für drei wichtige Kriterien alle sinnvollen Kombinationen der Kriterienausprägungen ab (siehe Abbildung 3.1). Als erstes Kriterium wird zwischen synchronen und asynchronen Methodenaufrufen unterschieden. Zweitens spielt es eine Rolle, ob das aufgerufene Objekt einen zustandsbehafteten oder zustandslosen Dienst erbringt. Drittens wird noch zwischen lokalen und entfernten Klienten unterschieden, die den Methodenaufruf durchführen.

Synchron oder asynchron: Bei synchronen Methodenaufrufen blockiert der aufrufende Klient während der Methodenausführung und wartet auf das Ergebnis des Aufrufs. Im Gegensatz dazu versendet der Klient bei einem asynchronen Methodenaufruf seine Nachricht an den Empfänger und läuft direkt weiter, ohne auf ein Ergebnis des Methodenaufrufs zu warten. Für die synchronen Aufrufe

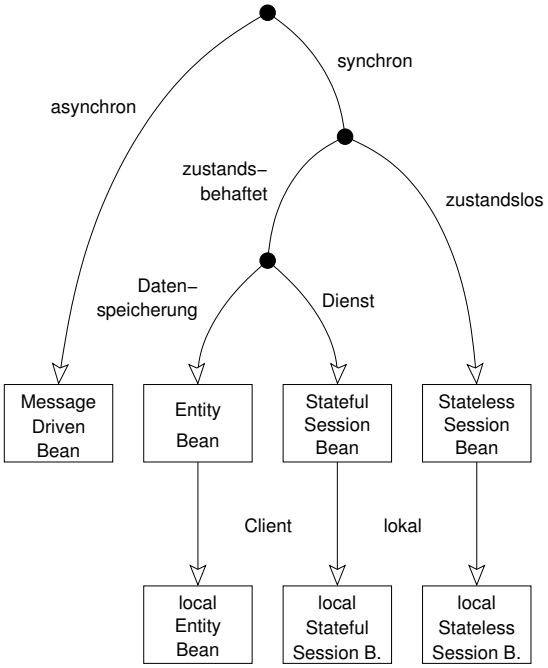

Abbildung 3.1: *Die Auswahl einer Art von Enterprise JavaBeans anhand verschiedener Kriterien.*

sind die EJB-Komponentenarten *Entity Beans* und *Session Beans* vorgesehen. Ab Version 2.0 der Spezifikation sind auch die *Message Driven Beans* (MDB) für asynchrone Aufrufe in der Spezifikation enthalten. Ein typischer Einsatzweck für MDBs ist zum Beispiel das Protokollieren von Ereignissen oder das Entgegennehmen von Arbeitsaufgaben, die sonst auch in einer Stapelverarbeitung erledigt werden könnten.

Zustandsbehaftet oder zustandslos: Ein Objekt wird als zustandslos angesehen, wenn das Ergebnis eines Methodenaufrufes nicht von den Daten oder Ergebnissen eines früheren Methodenaufrufes an das gleiche Objekt abhängt. Ist das Ergebnis eines Methodenaufrufes jedoch von einem früheren Methodenaufruf abhängig, so wird das Objekt als zustandsbehaftet bezeichnet. Die *Entity Beans* sind als objektorientierte Sicht auf eine zum Beispiel relationale Datenquelle entworfen worden. Sie sind somit immer zustandsbehaftet, da die in früheren Aufrufen in einer *Entity Bean* gespeicherten Daten bei späteren Aufrufen wieder abgerufen werden. Die permanente Datenspeicherung einer Auftragsposition ist zum Beispiel ein typischer Anwendungsfall für die Verwendung einer *Entity Bean.* Bei *Session Beans* wiederum können sowohl zustandsbehaftete oder zustandslose Dienste erbracht werden. Deswegen unterscheidet die EJB-Spezifikation zwischen *Stateful Session Beans* und *Stateless Session Beans*, die zustandsbehaftet

beziehungsweise zustandslos sind. Die *Stateful Session Beans* werden dem erzeugenden Klienten zugeordnet und können von anderen Klienten nicht entdeckt werden. Bei *Stateless Session Beans* kann der Applikationsserver eine oder mehrere Instanzen der Bean erzeugen und sie aufgrund der Zustandslosigkeit beliebig verschiedenen aufrufenden Klienten zuordnen. Als Beispiel für den Einsatz einer *Stateful Session Bean* lässt sich die Verwaltung eines Einkaufskorbs in einem Online-Handelssystem anführen, wogegen die Abfrage eines aktuellen Aktienwertes zumeist über ein *Stateless Session Bean* realisiert wird.

***Home Interface*, *Remote Interface* und die Implementierung:** Sowohl für *Session Beans* wie auch für *Entity Beans* sieht die Spezifikation drei Dateien pro Bean vor (siehe auch Abbildung 3.2):

1. Die wichtigste der drei Dateien ist die Implementierung der Bean in einer Java-Klasse. Bei einer Entity Bean enthält die Implementierungsklasse die Datenzugriffsmethoden und bei einer Session Bean die Methoden, welche bestimmte Dienste für den aufrufenden Klienten erbringen. Zusätzlich enthält die Implementierungsklasse noch Methoden, die der Applikationsserver beim Erzeugen einer neuen Instanz, vor dem Löschen einer Instanz oder beim Freigeben von Ressourcen aufruft.

2. In einer zweiten Datei, dem *Remote Interface*, werden alle Methoden, die dem Klienten zum Zugriff zur Verfügung stehen sollen, nochmals in einer Schnittstellenklasse deklariert.

3. Die dritte Datei, genannt *Home Interface*, enthält ebenfalls eine Schnittstellendeklaration, mit welcher die Lebenszyklussteuerung für alle Instanzen der EJB vom Klienten durchgeführt werden kann. Zur Steuerung des Objektlebenszyklus sind alle Methoden, die zur Erzeugung einer Instanz der EJB genutzt werden können, in dieser Schnittstellenklasse deklariert. So verwendet der Klient zum Beispiel zum Erzeugen einer neuen EJB die create()-Methode statt des `new`-Audrucks. In dieser Datei können für *Entity Beans* weiterhin alle Methoden deklariert werden, die zum Auffinden einer früher erzeugten Objektinstanz aus der Datenquelle benötigt werden. Dabei ist nicht nur die Suche über den obligatorischen primären Schlüssel, den jede *Entity Bean* besitzt, möglich, sondern es können auch Methoden deklariert werden, die eine sehr freie Festlegung der Kriterien über die Angabe des WHERE-Abschnitts einer SQL-Abfrage ermöglichen.

Entfernter oder lokaler Klient: Für Entity Beans und Session Beans wurde die EJB-Spezifikation in der Version 2.0 erweitert, um die Leistungsfähigkeit

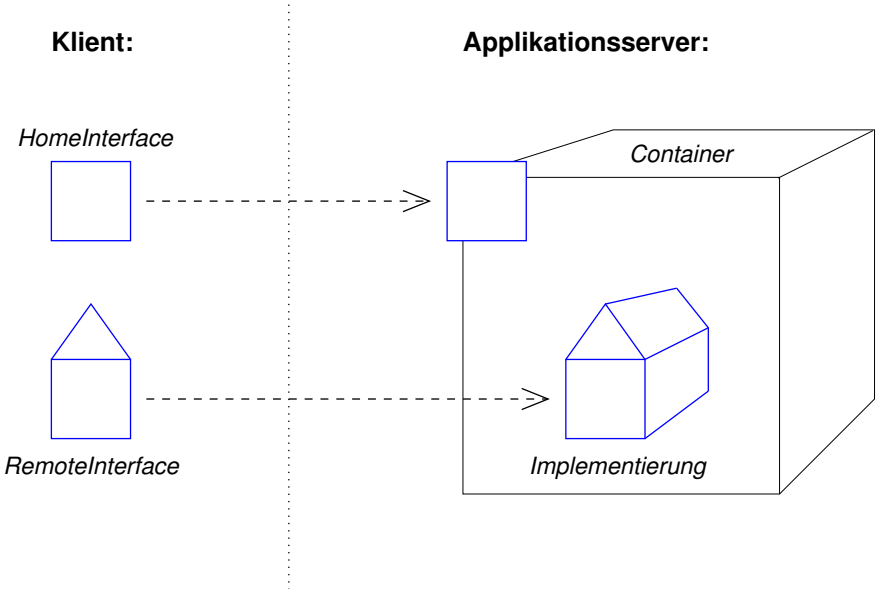

Abbildung 3.2: *Jede EJB benötigt eine Implementierungsklasse und die beiden Schnittstellen* Remote Interface *und* Home Interface*. Die Implementierungsklasse läuft auf dem Applikationsserver, welcher auch die* Home Interface *Schnittstelle implementiert. Die beiden Schnittstellen werden auf dem Klienten zum Steuern der Serverimplementierungen verwendet.*

von EJB-Objekten zu erhöhen: Die Spezifikation erlaubt eine Unterscheidung zwischen einem lokalen und einem entfernten Klienten, der eine Methode eines dieser EJB-Objekte aufruft. Die Definition von „lokal“ und „entfernt“ bezieht sich hierbei ausschließlich darauf, ob der aufrufende Klient in der gleichen Laufzeitumgebung wie der Applikationsserver abläuft. Bei der selben Laufzeitumgebung wird von einem *lokalen* Klienten und bei zwei getrennten Laufzeitumgebungen von einem *entfernten* Klienten gesprochen. Laufen Klient und Applikationsserver physikalisch auf dem gleichen Rechner, aber in zwei getrennten Laufzeitumgebungen ab, so handelt es sich nach dieser Definition trotzdem um einen entfernten Klienten.

Für lokale Klienten kann analog zu einem *Home Interface* und einem *Remote Interface* ein *Local Home Interface* und ein *Local Interface* erstellt werden. Die lokalen und entfernten Schnittstellen unterscheiden sich leicht in der Semantik der Parameterübergabe. Bei den entfernten Schnittstellen werden Parameter für Methoden und die Ergebnisse der Methoden mit einer *pass by value*-Semantik behandelt, d.h., es werden jeweils Kopien der übergebenen Objekte angelegt. Bei den lokalen Schnittstellen wird die für Java sonst übliche *pass by reference*-Semantik verwendet. Letztere ermöglicht eine wesentlich höhere Leistungsfähigkeit der EJB-Objekte. Diese Erweiterung war auch die Grundlage dafür, die au-

tomatische Datenspeicherung bei Entity Beans durch den Applikationsserver zu erweitern. Neben der Möglichkeit, bestimmte Attribute einer Entity Bean automatisch durch den Applikationsserver in einen Datenspeicher sichern zu lassen (*Container Managed Persistence*, CMP), können ab der Version 2.0 der Spezifikation einfache und komplexe Relationen zwischen verschiedenen Entity Beans durch den Applikationsserver verwaltet werden (*Container Managed Relationships*, CMR).

3.1.2 Webanwendungen

Die J2EE-Spezifikation bietet bei der Entwicklung von Benutzerschnittstellen zum einen Unterstützung für die Verarbeitung des HTTP-Protokolls an. Zum anderen wird auch die dynamische Erstellung von HTML-Seiten mit Inhalten, die von Java-Programmen geliefert werden, unterstützt. Dazu verweist die J2EE-Spezifikation auf die zwei Spezifikationen für Servlets (vgl. [COW01]) und JavaServer Pages (JSP, vgl. [PEL01]).

Im Laufe der Entwicklung zur heutigen J2EE-Spezifikation wurde zuerst die Servlet-Spezifikation eingeführt. Diese erlaubt das Ausführen von Java-Programmen auf der Webserverseite. Dabei übernimmt der Applikationsserver die Rolle eines Webservers und führt, je nach HTTP-Anfrage und Konfiguration des Applikationsservers, das entsprechende Servlet, also ein in Java erstelltes Programm, aus. Das Servlet kann nun über die durch den Applikationsserver zur Verfügung gestellten Schnittstellen auf eine Kontext- und Sitzungsverwaltung zurückgreifen und sehr einfach eventuell übergebene Parameter aus der HTTP-Anfrage auslesen. Abschließend erstellt das Servlet eine Antwort auf die HTTP-Anfrage, welche durch den Applikationsserver an den anfragenden Browser ausgeliefert wird. Die Antwort des Servlets wird in den häufigsten Fällen aus einer HTML-Seite bestehen, in die dynamische Inhalte eingefügt wurden.

Für HTML-Seiten, die sehr viele Designelemente aufweisen oder überwiegend statisch sind, ist die Zusammenstellung der HTML-Seite innerhalb eines Java-Programms (Servlet) nicht sehr übersichtlich und dadurch nicht wartungsfreundlich. Durch die Einführung von JavaServer Pages (JSP) wurde diese Problematik entschärft. Eine JSP-Seite ist eine normale HTML-Seite, in die an den Stellen mit dynamischen Inhalten jeweils Programmquelltext in Java eingefügt werden kann. Der Applikationsserver kompiliert eine solche JSP-Seite beim ersten Aufruf intern transparent in ein funktional äquivalentes Servlet und führt dieses aus.

Für komplexe Applikationen empfiehlt Sun inzwischen, sowohl Servlets als auch JSPs einzusetzen. In einer „Model 2“ genannten Architekturvorlage ist vorgesehen, dass alle HTTP-Anfragen von einem zentralen Servlet entgegengenommen werden (siehe Abbildung 3.3). Dieses Servlet übergibt dann die benötigten Parameter an normale Java-Klassen bzw. JavaBeans, welche die Datenverarbeitung

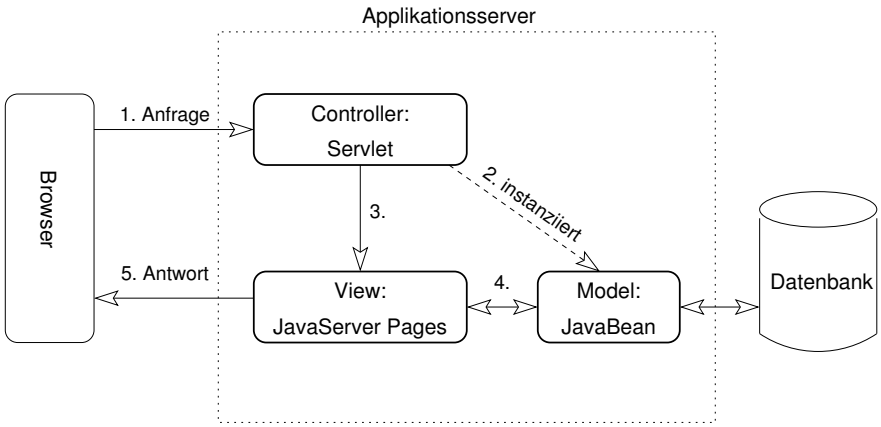

Abbildung 3.3: *Die Trennung nach dem Model-View-Controller-Konzept geschieht durch die Aufspaltung der Aufgaben auf Servlet, JSPs und JavaBeans (vgl. [SES99]).*

oder -aufbereitung vornehmen. Die Antwort an den Browser wird abschließend durch eine der verschiedenen JSP-Seiten erstellt, in welche auf Grund der bereits verarbeiteten Daten nur noch sehr wenig Java-Quelltext in den HTML-Text eingeflochten werden muss. Diese Architektur ermöglicht die Trennung von Steuerung und Präsentation und weist zudem wesentlich übersichtlichere HTML- und Java-Quelltexte auf. (vgl. [SES99])

3.1.3 Dienste

Sowohl für die in Abschnitt 3.1.1 beschriebenen EJBs, als auch für die vorhergehend dargestellten Servlets und JSPs sind in der J2EE-Spezifikation einige Querschnittsdienste spezifiziert, die häufig benötigte Dienstleistungen abdecken:

JDBC Durch die „Java Database Connectivity“-Schnittstelle (JDBC) steht eine standardisierte Schnittstelle für den Zugriff auf Datenbanken durch Java-Programme zur Verfügung. Die Hersteller von Datenbankmanagementsystemen (DBMS) müssen für ihr System eine Implementierung der JDBC-Schnittstelle zur Verfügung stellen. Damit ist das DBMS für jedes Java-Programm nutzbar, das bisher schon auf andere DBMS über JDBC zugegriffen hat. Der JDBC-Standard hat sich im Java-Bereich so gut etabliert, dass auf relationale DBMS fast ausschließlich über diese Schnittstelle zugegriffen wird und für alle bekannteren DBMS auch JDBC-Treiber zur Verfügung stehen. Die JDBC-Schnittstelle ist in [WHI99] spezifiziert.

JCA Die „J2EE Connector Architecture“ (JCA) erlaubt es, von einem Java-Programm aus auf beliebige andere Datenquellen zuzugreifen, für die keine JDBC-Implementierung, aber eine Implementierung der JCA-Schnittstelle zur Verfügung steht. Damit können auch Systeme eingebunden werden, die Daten nicht in relationalen Strukturen liefern. Die JCA-Spezifikation (vgl. [SHA01]) definiert dazu Ablaufregeln bezüglich der Verbindungen zwischen dem Applikationsserver und dem externen System, sowie für die Sicherheitsprotokolle und für die transaktionale Verarbeitung. Die JCA-Schnittstelle legt einige grundlegende Funktionsaufrufe fest, die von jeder Implementierung der JCA-Schnittstelle angeboten und erbracht werden müssen. Über diese Funktionen ist es möglich, ein JCA-System in einen Applikationsserver einzubinden, weitere angebotene Funktionalitäten der jeweiligen Implementierung zu identifizieren und diese dann benutzen zu können.

JNDI Die Abkürzung „JNDI“ steht für „Java Naming and Directory Interface“. Diese Erweiterung zur Java-Laufzeitumgebung bietet Java-Anwendungen eine einheitliche Schnittstelle für die verschiedensten Namens- und Verzeichnisdienste an. Diese Schnittstelle kann dazu verwendet werden, um einheitlich auf Verzeichnisdienste wie DNS, NDS, NIS (YP), X.500 oder LDAP zuzugreifen. Innerhalb eines J2EE-Servers werden die angebotenen Dienstleistungen, die installierten EJBs und alle verfügbaren Datenquellen (wie z.B. über JDBC eingebundene Datenbanken oder über JCA angebundene Unternehmensinformationssysteme) im JNDI-Dienst angemeldet. Die verschiedenen Klienten können dann über JNDI die benötigten Dienste, EJBs oder Datenquellen finden und benutzen. Die JNDI-Schnittstelle ist zum einen für die Seite der JNDI-Nutzer in [SUN99a] und zum anderen für die Seite der Anbieter von anderen Verzeichnisdiensten in [SUN99s] spezifiziert.

JTA Sowohl der Applikationsserver als auch Java-Programme selbst können über die „Java Transaction API“ (JTA) die Steuerung von Transaktionen übernehmen. Dabei werden auch Transaktionen nach dem X/Open XA-Transaktionsstandard (vgl. [XOP91]) unterstützt. Die JTA-Schnittstelle ist in [CHE02] spezifiziert.

JAAS Der „Java Authentication and Authorization Service“ (JAAS) erlaubt die Authentifizierung von Benutzern und darauf basierend die Einführung von benutzerbezogenen Zugriffsbeschränkungen in Java-Programmen. Die Java-Laufzeitumgebung bot bis zur Version 1.3 lediglich die Möglichkeit, Sicherheitsberechtigungen auf Basis der Programmherkunft zu vergeben. Mit der JAAS-Erweiterung für diese Laufzeitumgebung konnte die Berechtigungsverwaltung um benutzerbezogene Zugriffsberechtigungen erweitert werden. JAAS wird innerhalb der J2EE-Spezifikation ebenfalls für die Authentifizierung und Autorisierung von Benutzern

verwendet und wurde ab der Version 1.4 der Java-Laufzeitumgebung ein fester Bestandteil der Laufzeitumgebung.

JavaMail Die JavaMail-Schnittstelle bietet Java-Programmen die Möglichkeit, E-Mails über das SMTP-Protokoll zu versenden oder auf gespeicherte E-Mails auf Mailservern, welche das POP3- oder das IMAP-Protokoll unterstützen, zuzugreifen. Sun stellt zu der Spezifizierung der JavaMail-Schnittstelle in [SUN00] auch eine eigene Implementierung dieser Schnittstelle zur Verfügung.

3.2 Komponenten

Der Begriff „Komponente“ ist im Umfeld der Informationstechnik sehr häufig anzutreffen. Dabei wird er nicht einheitlich für eine allgemeingültige Definition verwendet, sondern in jedem Gebiet liegt diesem Begriff eine eigene Definition zu Grunde. Gemeinsam ist diesen unterschiedlichen Bedeutungen des Begriffs „Komponente“, dass eine Komponente ein Teil eines Systems ist oder sein kann und durch bestimmte Kriterien vom Rest des Systems abgegrenzt ist.

Eine Komponente wird in Anlehnung an [MUE97] im Rahmen dieser Arbeit wie folgt definiert:

> Eine Komponente ist eine Sammlung von Software, welche definierte Schnittstellen nach außen aufweist. Die Funktionsweise dieser Schnittstellen ist vertragsartig festgelegt, und alle kontextabhängigen Funktionsweisen sind explizit angegeben. Eine Komponente kann als ein Teil mit anderer Software integriert werden.

Diese Definition schließt bewusst sogenannte „Frameworks“ als eine Art von Komponenten ein. Als Unterscheidungskriterium zwischen dem, was üblicherweise als Softwarekomponente und dem, was als Softwareframework bezeichnet wird, kann man die Aufrufrichtung heranziehen. Bei einer Softwarekomponente wird diese durch das Programm aufgerufen, um eine festgelegte Funktionalität zu erbringen. Bei einem Softwareframework werden die entwickelten Programmteile durch das Framework aufgerufen, um zum Beispiel auf bestimmte Ereignisse zu reagieren.

In dieser Arbeit wird der Begriff „Komponente“ sowohl für Softwarekomponenten als auch für Softwareframeworks verwendet und die getroffenen Aussagen über Komponenten beziehen sich gleichermaßen auf Softwareframeworks und Softwarekomponenten.

Für die im Kapitel 2 dargestellte Problemstellung sollen nicht alle Arten von Komponenten betrachtet werden. Die in dem System aufzunehmenden und zu repräsentierenden Komponenten sollen die folgenden Eigenschaften aufweisen:

- Die betrachtete Komponente soll ein *integraler Bestandteil* des zu entwickelnden Softwareprodukts sein. Dies schließt Komponenten aus, die nur während der Entwicklung des Softwareprodukts eingesetzt werden, da diese als *Werkzeug* zur Entwicklung des Softwareprodukts, nicht aber als ein integraler Bestandteil des Softwareprodukts eingesetzt werden. Komponenten als Werkzeuge haben von der Funktionalität her eine andere Ausrichtung als Komponenten, die als Bestandteile in ein Softwareprodukt eingehen. Durch die Beschränkung auf die letztere Komponentenart soll die Bandbreite der Funktionalitäten, welche die im System erfassten Komponenten erbringen können, eingeschränkt werden. Damit lässt sich der Entwurf der ersten Version des Systems vereinfachen.

- Die Funktionalität, welche durch eine betrachtete Komponente erbracht wird, soll nicht auf ein spezielles Sachgebiet bezogen sein, sondern eine technisch allgemeine Funktionalität darstellen. Ein Beispiel für sachgebietsbezogene Komponenten sind Komponenten, die vollständig eine Bestellungsaufnahme oder eine Diskussionsplattform anbieten. Stattdessen sollen Komponenten betrachtet werden, die eine technisch allgemeine Funktionalität, wie zum Beispiel eine Benutzersitzungsverwaltung oder Eingabeformularerzeugung und -verwaltung, anbieten. Die Konzentration auf Komponenten mit technisch allgemeiner Funktionalität ist durch die Beobachtung begründet, dass sich rein sachgebietsbezogene Komponenten nur in Spezialfällen durchsetzen konnten.[1] Insbesondere im Bereich von J2EE haben sich in den letzten Jahren aber immer mehr Komponenten durchgesetzt, die eine technisch allgemeine Funktionalität anbieten. Diese Entwicklung könnte dadurch begründet sein, dass Komponenten im Allgemeinen nur bei der Entwicklung von Individualsoftware eingesetzt werden. Für Standardsoftware wie SAP oder Office-Produkte werden stattdessen üblicherweise fertige Zusatzpakete angeboten. Soll nun statt des Einsatzes von Standardsoftware Individualsoftware entwickelt werden, so ist dafür häufig die mangelnde Anpassbarkeit der Standardsoftware an individuelle Bedürfnisse als Grund zu sehen. Die sachgebietsbezogenen Komponenten bieten aber nicht die hohe Anpassbarkeit, die bei der Entwicklung von Individualsoftware gefordert ist. Die Komponenten mit technisch allgemeiner Funktionalität können aber auch bei Individualsoftware Funktionalitätsteile abdecken, die häufig wiederkehrend sind. Als weiterer Grund für den zunehmenden Einsatz von

[1] Nur in einigen Spezialfällen konnte sich die Entwicklung mit sachgebietsbezogenen Komponenten etablieren. Beispiele für solche Fälle sind Komponenten für Diskussionsforen oder die Transaktionsabwicklung bei Kreditkartenzahlung, für die jeweils mehrere Anbieter am Markt existieren (vgl. [C-SRC]). Für technisch allgemeine Funktionalitäten wie z.B. Objektpersistenz ist ein wesentlich größerer Markt entstanden (vgl. [C2W]). Existierende Marktplattformen zeigen durch die vorhandene Kategorisierung der angebotenen Komponenten eine sehr stark ausgeprägte Ausrichtung auf technisch allgemeine statt auf sachgebietsbezogene Funktionalitäten (vgl. [C-SRC] und [DEVD]).

technisch allgemeinen Komponenten kann der Grad der möglichen Modularisierung angeführt werden, der in den letzten Jahren durch den Einsatz von komponentenorientierten Technologien wie J2EE erheblich gesteigert werden konnte.

- Die betrachtete Komponente soll im Bereich der Entwicklung von J2EE-basierten Programmen relevant sein. Dazu wird die Komponente meistens eine Funktionalität anbieten, die über J2EE nicht abgedeckt ist und somit die J2EE-Funktionalitäten ergänzt. In Einzelfällen kann aber auch eine Funktionalität durch die Komponente erbracht werden, die mit Teilen der J2EE-Funktionalität konkurriert.

- Die verwendeten Komponenten sollen unter einer freien Lizenz vertrieben werden (vgl. [OSI03]) und im Quelltextformat vorliegen.

Für das letzte Kriterium der Komponenten mit einer freien Lizenz sind verschiedene Gründe zu nennen:

- Komponenten mit freier Lizenz (vgl. [OSI03]) sind sowohl für die Erstellung dieser Arbeit al s auch für den Entwickler während der Softwareentwicklung erheblich einfacher verfügbar, als Komponenten, die zuerst erworben oder über einen anderen Prozess beschafft werden müssen. Je weiter der in Kapitel 2 dargestellte Aufwand für den Entwickler bei der Verwendung von Komponenten erhöht wird, desto stärker sinkt die Wahrscheinlichkeit, dass Komponenten zur Entwicklung eingesetzt werden. Um den Aufwand für den Entwickler möglichst gering zu halten, sollen deshalb zu Beginn nur Komponenten mit freier Lizenz in das System aufgenommen werden.

- Durch Verwendung von Komponenten mit freier Lizenz kann das Risiko für ein Softwareprojekt dadurch reduziert werden, dass vor der Entscheidung für den Einsatz einer Komponente diese auf Grund der freien Lizenz in ausreichendem Umfang getestet werden kann.

- Das Risiko von Softwareprojekten kann durch den Einsatz von Komponenten mit freier Lizenz zudem dadurch verringert werden, dass bei eventuell auftretenden Fehlern in Komponenten, falscher oder lückenhafter Dokumentation der Schnittstellen, der Quelltext der Komponenten vorhanden ist. Eventuelle Fehler können im Notfall selbst behoben und Funktionsweisen nachvollzogen werden. Werden dagegen Komponenten verwendet, für die der Quelltext nicht verfügbar ist, ist das Risiko für die Entwicklung durch die Abhängigkeit vom Komponentenhersteller erhöht.

- Bei Komponenten mit freier Lizenz ist potentiell ein höherer Investionsschutz gegeben. Sollte die Weiterentwicklung einer verwendeten Komponente eingestellt werden, so ist man lizenzrechtlich nicht an einen Hersteller gebunden, kann die Komponente an kleinere technologische Änderungen auch selbst anpassen und damit wesentlich länger verwenden.

Abschließend sollen noch einige konkrete Beispiele für Komponenten genannt werden, die der obigen Komponentendefinition entsprechen und auch die vorhergehend genannten zusätzlichen Kriterien erfüllen:

Komponentenname	Bezugsquelle
Arch4J	`http://arch4j.sourceforge.net/`
Axis	`http://ws.apache.org/axis/`
Batik	`http://xml.apache.org/batik/`
Castor	`http://www.castor.org/`
Cayenne	`http://www.objectstyle.org/cayenne/`
Clickstream	`http://www.opensymphony.com/clickstream/`
Cocoon	`http://cocoon.apache.org/`
Commons	`http://jakarta.apache.org/commons/`
ECS	`http://jakarta.apache.org/ecs/`
EJBCA	`http://ejbca.sourceforge.net/`
Echo	`http://www.nextapp.com/products/echo/`
FOP	`http://xml.apache.org/fop/index.html`
Formproc	`http://formproc.sourceforge.net/`
Freemarker	`http://freemarker.sourceforge.net/`
Fulcrum	`http://jakarta.apache.org/turbine/fulcrum/`
JAX-RPC	`http://java.sun.com/xml/downloads/jaxrpc.html`
JByte	`http://javaby.sourceforge.net/`
JExcelAPI	`http://www.andykhan.com/jexcelapi/`
JSDT	`http://java.sun.com/products/java-media/jsdt/`
Jahia	`http://www.jahia.org/jahia/Jahia`
JasperReports	`http://jasperreports.sourceforge.net/`
Javassist	`http://jboss.org/developers/projects/javassist.html`
Jetspeed	`http://jakarta.apache.org/jetspeed/site/`
Log4J	`http://logging.apache.org/log4j/docs/`
Lucene	`http://jakarta.apache.org/lucene/docs/index.html`
Maverick	`http://mav.sourceforge.net/`
Millstone	`http://www.millstone.org/`
ORO	`http://jakarta.apache.org/oro/`
OScache	`http://www.opensymphony.com/oscache/`
OScore	`http://www.opensymphony.com/oscore/`
OSworkflow	`http://www.opensymphony.com/osworkflow/`

Komponentenname	Bezugsquelle
POI	http://jakarta.apache.org/poi/
Quartz	http://www.opensymphony.com/quartz/
Regexp	http://jakarta.apache.org/regexp/
Sitemesh	http://www.opensymphony.com/sitemesh/
Slide	http://jakarta.apache.org/slide/
Struts	http://jakarta.apache.org/struts/index.html
Taglibs	http://jakarta.apache.org/taglibs/index.html
Tapestry	http://jakarta.apache.org/tapestry/index.html
Tea	http://teatrove.sourceforge.net/tea.html
Torque	http://db.apache.org/torque/
TransformTags	http://wiki.opensymphony.com/space/TransformTags
Turbine	http://jakarta.apache.org/turbine/index.html
Velocity	http://jakarta.apache.org/velocity/index.html
Webmacro	http://www.webmacro.org/
Webwork	http://www.opensymphony.com/webwork/
Xalan	http://xml.apache.org/xalan-j/
Xerces	http://xml.apache.org/xerces2-j/index.html
Xwt	http://www.xwt.org/

Kapitel 4

Problemanalyse und Lösungskonzept

In diesem Kapitel wird zunächst dargestellt, welche Problematiken sich für den Entwickler, der sich auf die Suche nach hilfreichen Komponenten begibt, entstehen. Für die einzelnen identifizierten Problempunkte werden dann die im Rahmen dieser Arbeit gefundenen Lösungsansätze vorgestellt. Das Kapitel wird mit der Zusammenfassung der Lösungsansätze in einem Lösungskonzept abgeschlossen.

4.1 Problematik der Komponentensuche und -auswahl

Um die Umsetzung eines Systems zu vereinfachen, werden die funktionalen Anforderungen an dieses vom Entwickler, wie dies in Abbildung 4.1 illustriert ist, horizontal und vertikal verschiedenen Einzelteilen des Systems zugeordnet. Hierbei stammen die Anforderungen an das Gesamtsystem meist aus unterschiedlichen fachlichen Gebieten und werden nach diesen Gebieten den horizontal angeordneten Systemteilen zugeordnet. Die horizontalen Einzelteile bzw. Schichten des Systems liefern Dienste für die vertikal oberhalb liegenden Teile. Nach der Aufteilung des Systems versucht der Entwickler zuerst, die verschiedenen horizontalen und vertikalen Teile nach Möglichkeit schon bestehender Software oder bereits bekannten Technologien zuzuordnen. Hierbei dürften, im vertikalen Verlauf betrachtet, vor allem die unteren Schichten oft durch die J2EE-Technologien und damit durch einen Applikationsserver abgedeckt werden. Für alle nicht zuordenbaren Systemteile oder für Teile, die bewusst nicht direkt eine J2EE-Technologie verwenden sollen , müssen die funktionalen Anforderungen durch den Entwickler selbst programmiert werden. Alternativ kann für diese Teile die eigene zu

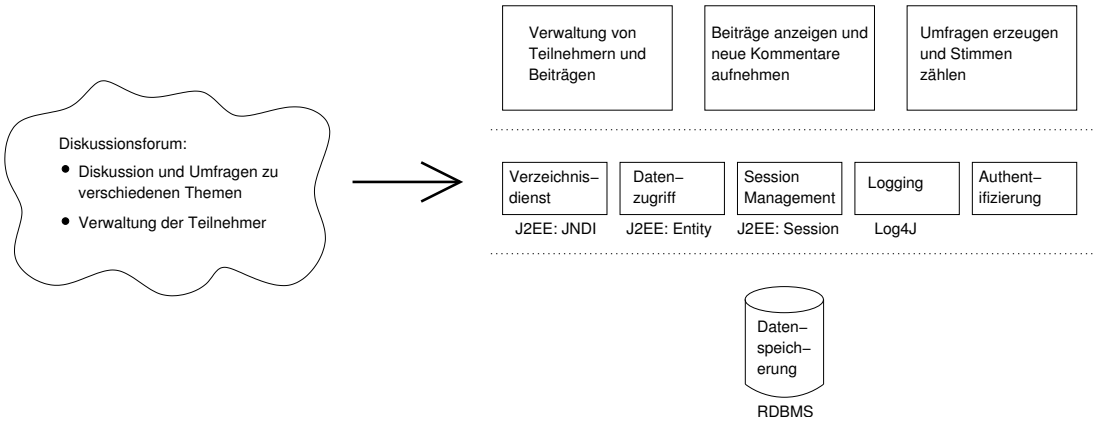

Abbildung 4.1: *Die Aufgabenstellung „Diskussionsforum“ wird vertikal in Schichten und horizontal in Einzelteile zerlegt, die jeweils abgegrenzte Aufgabengebiete haben. Die unter den Einzelteilen angegebenen Namen stellen eine Möglichkeit dar, die Einzelteile durch Komponenten oder J2EE-Bestandteile abzudecken. Mit „J2EE“ beginnende Namen kennzeichnen dabei eine J2EE-Technologie.*

erbringende Programmierleistung durch den Einsatz von Komponenten verringert werden. Erst hier begibt sich der Entwickler auf die Suche nach passenden Komponenten, welche die Anforderungen an die identifizierten Einzelteile des Gesamtsystems am besten erfüllen.

Der Entwurfsprozess von der Aufgabenstellung an das Gesamtsystem bis hin zur Identifizierung hilfreicher Komponenten lässt sich also in zwei große Schritte gliedern:

- Im ersten Schritt wird das Gesamtsystem durch den Entwickler in Einzelteile zerlegt, welche zusammen genommen die Anforderungen an das Gesamtsystem erfüllen sollen.
- Erst in einem zweiten Schritt werden für Einzelteile des Systems, an die wenige, definierte Anforderungen gestellt werden, eine oder mehrere Komponenten gesucht, die diese Anforderungen für ein Einzelteil ganz oder teilweise erfüllen können.

Die Gesamtheit aller Anforderungen an ein Gesamtsystem weist einen sehr hohen Grad an Komplexität auf. Aus diesem Grund und der zusätzlichen Schwierigkeit, dass diese Anforderungen aus beliebigen fachlichen Gebieten kommen können, liegt die Unterstützung des Entwicklers beim ersten Schritt des oben beschriebenen Entwicklungsprozesses außerhalb des Rahmens dieser Arbeit.

Hat der Entwickler jedoch das Gesamtsystem in einzelne Teilsysteme mit ihren spezifischen Anforderungen zerlegt, so soll ihn das zu entwickelnde System beim

zweiten Schritt des Findens von Komponenten für die einfachere Umsetzung von Teilsystemen unterstützen. Für dieses System wird im folgenden Text der Name „ICApps“ als Kurzform für „*Interactive Cookbook for Applications*“ verwendet. Um die Unterstützung für diesen zweiten Schritt möglichst effektiv und effizient zu gestalten, muss der Lösungsansatz für ein Programm zur Unterstützung der Suche und Auswahl (ICApps) an die folgenden Gegebenheiten angepasst sein:

1. Die in ICApps erfassten und dokumentierten Komponenten erfüllen meist mehr als eine Funktionalität. Oft werden verwandte oder sich gut ergänzende Funktionalitäten in einer Komponente gemeinsam angeboten. Durch die allgemeine Ausrichtung der Komponenten (siehe Abschnitt 3.2) lassen sich oft die in einer Komponente gegebenen technischen Möglichkeiten für das Erbringen von verschiedenen Funktionalitäten nutzen. Es kann also im Allgemeinen nicht davon ausgegangen werden, dass eine Komponente nur genau eine Funktionalität zur Verfügung stellt.

2. Sollen für die Unterstützung der Suche Schlüsselwörter oder Suchbegriffe eine Rolle spielen, so ist zu beachten, dass die Wahrscheinlichkeit, dass zwei Personen für eine Sache den selben Begriff wählen, durchschnittlich in der Größenordnung von nur 20 % anzusiedeln (vgl. [FUR87]). Innerhalb des ICApps-Systems ist die erste Person diejenige, welche die Komponente in ICApps dokumentiert hat bzw. die Strukturierung der Daten vorgenommen hat, und die zweite Person stellt der Entwickler, welcher nach eine Komponente sucht, selbst dar.

3. Der Entwickler wird Begriffe wählen, die relativ stark aus dem fachlichen Gebiet, für welches das Gesamtsystem erstellt werden soll, stammen. Die Person, die eine Komponente aber in ICApps dokumentiert, wird Begriffe wählen, die allgemeiner und technischer geprägt sind. Dadurch wird die im vorherigen Punkt beschriebene Problematik noch weiter verschärft.

Bevor im Folgenden beschrieben wird, wie auf diese Punkte eingegangen wird, sollen vorab die grundsätzlichen Möglichkeiten aufgezeigt werden, wie ein Themengebiet im Allgemeinen strukturiert werden kann. Die gefundenen Erkenntnisse aus der allgemeinen Betrachtung werden dann auf das spezielle Themengebiet „Komponenten für Serveranwendungen“ übertragen.

4.2 Alternativen der Strukturierung

Es gibt viele Methoden, eine Menge von Elementen aus einem Themengebiet zu strukturieren. Die Vielzahl der Methoden ergibt sich durch kleine Unterschiede

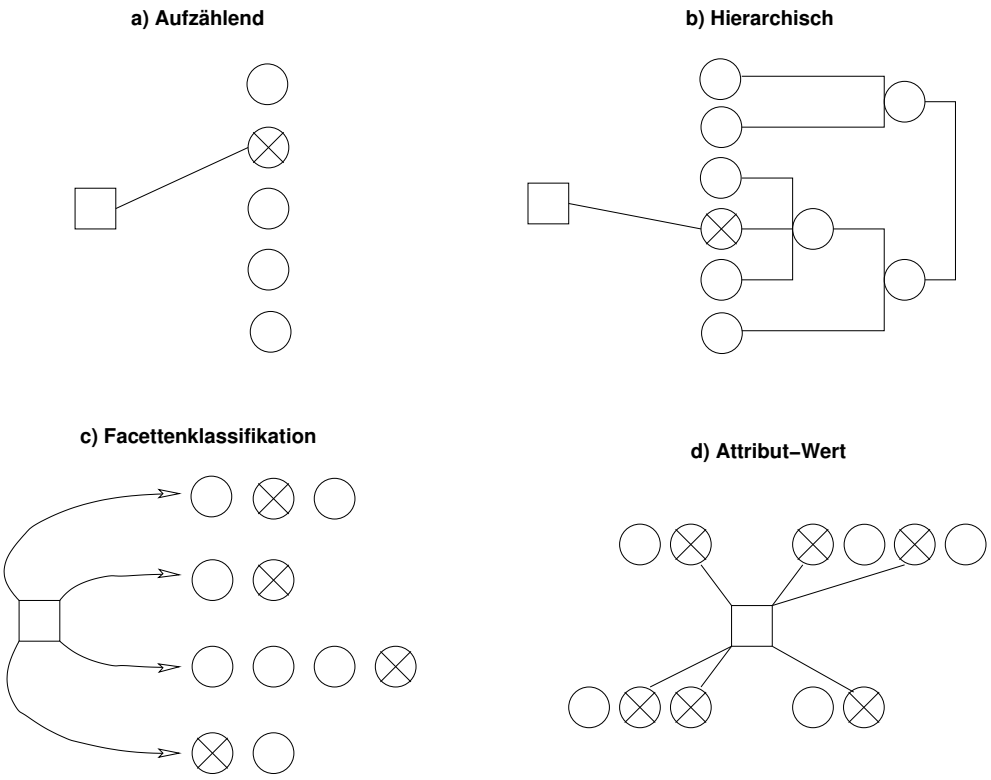

Abbildung 4.2: *Die vier möglichen Grundmethoden zur Klassifikation. Das Quadrat repräsentiert jeweils das zu klassifizierende Objekt. Die Kreise stellen die verschiedenen Kategorien bzw. Attributsausprägungen dar.*

im Detail einer Methode. Betrachtet man jedoch nur das grundlegende Vorgehen dieser Methoden, so lassen sich diese in wenige, übergeordneten Methoden zusammenfassen (vgl. [HEN97]):

Aufzählen (engl.: enumerated) Bei dieser Methode werden viele nebeneinanderstehende Kategorien geschaffen und ein Element wird genau *einer* Kategorie zugeordnet (siehe Abbildung 4.2a). Die Kategorien können durch verschiedenartige Überlegungen geschaffen werden, wodurch nicht zwingend ein sinnvoller Zusammenhang zwischen den Kategorien entstehen muss. Als Beispiel für diese Methode läßt sich die Kategorisierung von Berufen in die Kategorien „Landwirtschaft“, „Industrie“, „Büro“ und „Dienstleistung“ anführen. Ein Vorteil dieser Methode ist die extrem einfache Struktur. Daraus erwachsen aber auch eine ganze Reihe von Nachteilen, da sich viele reale Dinge nicht mit einer so einfachen Struktur erfassen lassen. Zum Beispiel ist durch die willkürliche Wahl der Kategorien nicht gesichert, dass disjunkte Kategorien für bereits existierende, aber auch zukünftige Elemente, die zugeordnet werden sollen, geschaffen wurden. Für viele reale Elemente

entsteht somit eine Zuordnungsproblematik, die sich verschärft, wenn die Kategorisierung über einen langen Zeitraum hinweg Bestand haben und verwendet werden soll. Im gewählten Beispiel der Berufskategorien ist es zum Beispiel schwierig, neu entstandene Berufe wie den „Fachberater für ökologische Landwirtschaft“ einzuordnen, der eine Dienstleistung innerhalb einer häufig bereits industrialisierten Landwirtschaft erbringt.

Hierarchisch (engl. hierarchical) Ähnlich der Aufzählen-Methode werden Kategorien geschaffen. Jedoch werden die Kategorien allgemeiner gehalten und können Unterkategorien enthalten. Die Unterkategorien können wiederum jeweils durch weitere Unterkategorien bis zu einer beliebigen Tiefe verfeinert werden. Von Vorteil bei dieser Methode ist die Verfeinerung, die sich mit weiteren Unterkategorien ergibt (siehe Abbildung 4.2b). Sowohl bei der Zuordnung als auch bei der Suche kann somit einfacher entschieden werden, in welcher „Fein“-Kategorie ein Element letztendlich enthalten ist. Im Beispiel der Berufskategorisierung kann mit dieser Methode zuerst nach „überwiegend körperlicher Arbeit“ und „überwiegend geistiger Arbeit“ unterschieden werden. Für die Überkategorie „überwiegend körperliche Arbeit“ können dann die Unterkategorien „Arbeitsort in der Industrie“ bzw. „Arbeitsort in natürlicher Umgebung“ eingeführt werden, die jeweils wiederum in Unterkategorien untergliedert werden und so weiter.
Der grundlegende Nachteil dieser Methode, dass ein Element nur einer Kategorie zuzuordnen ist und somit Probleme dabei entstehen können, bleibt bestehen. Jedoch kann für manche problematischen Fälle, in denen ein Element zwei Unterkategorien zuordenbar wäre, dieses einfach der gemeinsamen Oberkategorie zugeordnet werden, wenn man den damit einhergehenden Informationsverlust akzeptieren kann.
Das Problem der sehr verschiedenen Interpretation des gleichen Begriffs durch zwei Personen (siehe Kapitel 4.1, Punkt 2) führt auch bei dieser Methode zu erheblichen Schwierigkeiten. So muss der Suchende die vom Strukturierenden gemeinte Bedeutung der Unterkategorisierung richtig erfassen. Auf Grund des vorgenannten Problems ist zumindest mit einigen anfänglichen Fehlschlägen bei der Suche zu rechnen.

Facettenklassifikation (engl. faceted) Um das Problem der Zuordenbarkeit von Elementen zu mehreren Kategorien zu lösen, werden bei dieser Methode sog. Facetten eines Elements zur Zuordnung zu einer Kategorie verwendet. Eine Facette ist dabei ein bestimmter Aspekt bzw. eine Eigenschaft eines Elements. Das Element besitzt für jeden Aspekt (respektive Facette) genau eine bestimmte Ausprägung. Die Ausprägungen für alle Aspekte zusammen ergeben die Bezeichnung der Kategorie, zu der das Element zugeordnet wird (siehe Abbildung 4.2c). Damit ergibt sich für jedes Element eine eindeutige Kategorie. Zur kompakteren Schreibweise kann eine Reihenfolge der Fa-

cetten festgelegt werden, wodurch bei der Kategorieangabe der Name der entsprechenden Facette entfallen kann. Die Ausprägungen der Facette können zur leichteren Suche und Einordnung wiederum hierarchisch organisiert sein. Das Problem der mehrfachen Zuordenbarkeit innerhalb einer Facette ist hier deutlich entschärft, da für eine Facette nur ein Aspekt des Elements betrachtet wird und dadurch eine disjunkte Aufteilung der unteren Hierachiestufen einfacher möglich ist.
Für die Berufskategorisierung lassen sich zum Beispiel die Facetten „notwendiger Schulabschluss — Kundenkontakt — Arbeitsort — Verantwortungsumfang“ festlegen. Der Beruf „Kundenberater einer Bank“ läßt sich dann in die Kategotrie „Realschulabschluss — ja — Bankfiliale — eigene Arbeitsorganisation“ einordnen.
Ein Nachteil dieser Methode ist die komplexere Struktur, weshalb eine Unterstützung der Suche durch ein Werkzeug hier besonders wünschenswert ist.

Attribut-Wert (engl. attribut-value) Die Methode der Attribut-Wert Kombinationen ist eine Variation der Facettenklassifikation. Das Attribut gleicht der Facette bzw. dem Aspekt eines Elements. Jedoch wird zu jeder Ausprägung des Elements der jeweilige Attributsname zusätzlich zu den Ausprägungen genannt. Dadurch kann eine strikte Reihenfolge der Attribute entfallen, da diese jeweils mit Namen angegeben sind. Auch ist es nun möglich mehrere Werte je Attribut anzugeben (siehe Abbildung 4.2d). Als Resultat wird der Raum der möglichen Werte für ein Attribut anders aussehen, als bei den disjunkten Unterkategorien der Facettenklassifikation. Beispielsweise kann im vorhergehenden Beispiel die Facette „Kundenkontake“ durch das Attribut „Arbeitskontakte“ mit den möglichen Werten „Laufkundschaft“, „externe Kunden“, „interne Kunden / Arbeitskollegen“ und „übergeordnetes Management“ ersetzt werden. Mehrere oder auch keine Werte können dann dem Attribut „Arbeitskontakte“ für eine Kategorie zugeordnet werden. Für die Wertausprägungen eines Attributs ist es auch möglich, diese zu Oberkategorien innerhalb dieses Attributs zusammenzufassen. Einerseits muss bei der Auflistung der möglichen Werte für ein Attribut wieder stärker darauf geachtet werden, dass diese möglichst disjunkt sind. Andererseits ist der Fall, dass ein Attribut hier mehrere Werte annehmen kann, wesentlich einfacher zu erfassen als in der Facettenklassifikation. Die entstehende Organisationsstruktur ist einem Netzwerk nachempfunden. Auch hier sollte die Suche durch ein Werkzeug unterstützt werden.

4.3 Einzelfunktionalitäten

Wie in Abschnitt 4.1 beschrieben, erfüllen die meisten der betrachteten Komponenten mehr als eine Funktionalität. Eine Beschreibung einer Komponente, die versucht, alle Möglichkeiten der Komponente zu erfassen, wird somit eine komplexe Struktur aufweisen. Dadurch würde es dem Entwickler stark erschwert, schnell aus der Beschreibung zu entnehmen, ob eine Komponente potentiell geeignet ist, die von ihm gesuchte Funktionalität zu erfüllen. Identische Funktionalität würde mit einer solchen Beschreibung in verschiedenen Komponenten zudem leicht unterschiedlich beschrieben werden, was wiederum eine schnelle Orientierung für den Entwickler erschwert.

In ICApps soll deshalb der Ansatz verfolgt werden, allgemeinere Einzelfunktionalitäten zu definieren und diese unabhängig von einer konkreten Implementierung in einer Komponente zu beschreiben. Die Einzelfunktionalitäten können sich zwar ergänzen, sollen untereinander aber immer funktional disjunkt sein. Das System kennt dann eine Menge an Einzelfunktionalitäten, die sich aus der Zusammenfassung aller Einzelfunktionalitäten aller Komponenten ergibt. Eine Komponente kann somit dadurch beschrieben werden, dass dargestellt wird, welche Einzelfunktionalitäten eine Komponente anbietet und auf welche Art und Weise sie jede dieser Einzelfunktionalitäten erbringt. Ähnliche Funktionalitäten von mehreren Komponenten können weiterhin zunächst durch *eine* allgemeinere Einzelfunktionalität beschrieben werden und erst in der Beschreibung der Komponente selbst werden die komponentenspezifischen Feinheiten detailliert erläutert. Zusätzlich enthält die Beschreibung einer Komponente in ICApps einen kurzen Überblick über die grundlegende Funktionsweise der Komponente, sozusagen eine Beschreibung der Philosophie, mit der die Komponente entwickelt wurde. Daraus ergeben sich die folgenden Vorteile gegenüber einer „klassischen“ Beschreibung der Komponente:

- Durch die Unterteilung der Gesamtfunktionalitäten einer Komponente in die Beschreibung, welche Einzelfunktionalitäten von ihr erbracht werden und wie dies bewerkstelligt wird, ist die Beschreibung der Komponente wesentlich stärker strukturiert. Dies erleichtert es dem Entwickler, die gesuchte Einzelfunktionalität in der Beschreibung der Komponente zu finden. Außerdem ist für den Entwickler direkt ersichtlich, wie die Funktionalität umgesetzt wird.
- Da die Einzelfunktionalitäten disjunkt untereinander sind, kann sich der Entwickler erst in den Einzelfunktionalitäten orientieren und dann nach Komponenten zu suchen, welche die gewünschten Einzelfunktionalitäten aufweisen. Dies ist einfacher als in vollständigen, weniger strukturierten Beschreibungen von Komponenten nach einer bestimmten Funktionalität zu suchen.

- Ist klar, welche Einzelfunktionalität gesucht wird, so kann der Entwickler wesentlich einfacher alle Komponenten auffinden, die diese Einzelfunktionalität erbringen können. Ähnliche Komponenten werden dadurch auch deutlich leichter vergleichbar.

- Die Beschreibung einer Einzelfunktionalität hat eine geringere Terminologiediskrepanz zur Begriffswelt des Entwicklers als die Beschreibung von Komponentenfunktionalitäten. Dies wird dadurch erreicht, dass die Beschreibungen von Einzelfunktionalitäten allgemeiner gehalten sind, da sie nicht auf die technische Umsetzung eingehen müssen. Auch für den Fall, dass eine technische Möglichkeit einer Komponente für mehrere Funktionalitäten genutzt werden kann, muss nicht eine komplexe Beschreibung der technischen Details erstellt werden, die alle möglichen Funktionalitäten abdeckt. Stattdessen können einfach mehrere Einzelfunktionalitäten herangezogen und der Komponente zugeordnet werden.

Die Strukturierung der Suchproblematik durch Einzelfunktionalitäten in der beschriebenen Art und Weise ist der Attribut-Wert-Strukturierung sehr ähnlich. Dabei wird der Vorteil genutzt, dass keine Zuordnungskonflikte für eine Komponente zu lösen sind, wie dies zum Beispiel bei hierarchischer Strukturierung der Fall sein kann. Das Problem der potentiellen Unübersichtlichkeit der entstehenden Netzwerkstruktur durch die Attribut-Wert-Kombinationen wird dadurch vermieden, dass die Einzelfunktionalitäten untereinander disjunkt sind. Nur in diesen muss der Entwickler (mit Unterstützung durch das System) selbst suchen. Den Übergang von einer ausgewählten Einzelfunktionalität zu einer Komponente erledigt ICApps für ihn. Insgesamt nutzt diese Lösung also die Vorteile des Netzwerkansatzes und vermeidet gleichzeitig seine Nachteile.

4.4 Spreading Activation

Zwischen den Begriffen, die der Entwickler bei der Suche als Kriterien verwendet, und den Begriffen, die eine Komponente beschreiben, besteht eine Terminologiediskrepanz, die es durch die Suche zu überwinden gilt. Diese Diskrepanz wird durch die Verwendung von Einzelfunktionalitäten verringert. Die Begriffe aus der Beschreibung der Einzelfunktionalität weisen eine geringere Terminologiediskrepanz zu den Begriffen des Entwicklers auf, als die Begriffe aus der Beschreibung der Komponente, da Einzelfunktionalitäten allgemeiner beschrieben werden können als konkrete Komponenten. Weiterhin ist auch der gewählte Namen für die Einzelfunktionalität semantisch näher an den Begriffen des Entwicklers, da der Name der Einzelfunktionalität nur eine allgemeinere Funktionalität beschreibt.

Trotzdem bleibt die in Abschnitt 4.1 beschriebene Problematik bestehen und es ist eine verminderte Terminologiediskrepanz zwischen den Begriffen des Entwicklers und der Einzelfunktionalitäten zu überwinden. Eine Liste von Synonymen für die jeweiligen Begriffe kann helfen, diese Diskrepanz zu überbrücken. Allerdings ist der fachliche Bereich, aus dem der Entwickler seine Begriffe wählt, nicht festgelegt und somit müssten für die fachlichen Begriffe sehr viele Synonymlisten zur Verfügung stehen. Dies macht den Ansatz der Synonymlisten für die Seite der fachlichen Begriffe unpraktikabel.

Für die Seite der Funktionalitätenbeschreibungen ist es für eine einzelne Person, die eine Komponente in ICApps beschreibt, sehr schwierig, alle möglichen Synonyme zu finden. Zudem ist die Erstellung von Synonymlisten für jedes mögliche Schlüsselwort einer Funktionalitätsbeschreibung von Hand sehr aufwändig.

In [HEN97] beschreibt der Autor, wie er durch den Einsatz von sog. „Spreading Activation“ sehr gute Ergebnisse für automatische Erstellung von Synonymlisten erhält. Die Liste der Synonyme zu einem Begriff wurde hierbei auf Basis einer großen Menge von automatisch erfassten, kurzen Programmroutinen, die jeweils mit einer kurzen Funktionsbeschreibung versehen waren, erstellt. Dabei handelt es sich *nicht* um eine reine Liste von Synonymen, sondern um Begriffe, die häufig im Kontext des zu Beginn vorgegebenen Begriffs auftauchen. In den dargestellten Versuchen sind die gefundenen Begriffe jedoch zum größten Teil Synonyme zum gegebenen Begriff.

4.4.1 Verfahren

Das in CodeFinder benutzte Verfahren (CodeFinder und Verfahren werden in [HEN97] beschrieben) unterscheidet zwischen Elementen und Worten. Jedes Element besitzt einen Beschreibungstext, aus welchem die Worte, die dem Element zugeordnet sind, extrahiert werden. Wird ein Wort in einem Beschreibungstext ein oder mehrmals wiederholt, so bleibt es jedoch trotzdem bei nur *einer* Zuordnung zwischen dem Wort und dem Element des Beschreibungstextes. Kommt das gleiche Wort in mehreren Beschreibungstexten vor, so ist es auch mehreren Elementen zugeordnet. Die Zuordnungen zwischen den Elementen und Worten besitzen jeweils Gewichte. Dabei entspricht das Gewicht einer Zuordnung der inversen Häufigkeit des Auftretens eines Wortes in der Elementbeschreibung, d.h. tritt ein Wort häufig in einem Beschreibungstext auf, so erhält die Zuordnung ein kleines Gewicht. Tritt ein Wort hingegen selten oder nur einmal auf, so erhält die Zuordnung ein großes beziehungsweise das maximale Gewicht. Diese Wahl der Gewichtung wird vom CodeFinder-Autor durch die empirische Beobachtung begründet, dass selten vorkommende Worte das Element präziser beschreiben, als Worte, die häufig vorkommen.

Um für ein Wort die beschriebene Liste, die Synonyme enthält, zu erstellen, wird im ersten Schritt dieses Wort mit einem Wert von 1,0 aktiviert. Alle anderen Worte und die Elemente erhalten zu Beginn den Aktivierungswert 0,0. Die Aktivierung des Worts wird durch die Zuordnungen gemäß deren Gewicht an die Elemente weitergegeben. Im zweiten Schritt geben die Elemente ihre Aktivierung wieder durch die gewichteten Zuordnungen an alle verbundenen Wörter zurück und aktivieren diese ebenfalls. Es wird nun wieder bei Schritt eins begonnen, jedoch sind nun mehrere Wörter mit einer bestimmten Intensität aktiviert und können die ihnen zugeordneten Elemente aktivieren. Das zu Beginn ausgewählte Wort bleibt dabei auf dem Aktivierungswert 1,0 festgelegt. Alle anderen Aktivierungen werden durch eine Begrenzungsfunktion auf den maximalen Wert 1,0 limitiert. Mit ausreichend vielen Durchläufen stellt sich ein Aktivierungsgleichgewicht ein, wobei normalerweise 4 bis 5 Durchläufe schon für eine ausreichende Stabilisierung sorgen, so dass danach der Prozess abgebrochen werden kann. In die Liste werden dann in absteigender Reihenfolge die Worte mit der höchsten Aktivierung aufgenommen.

4.4.2 Anwendung

In ICApps sind die Beschreibungen von Komponenten komplexer als die reinen Texte zur Beschreibungen von Programmroutinen in CodeFinder. Um Spreading Activation in ICApps zur Erstellung von Listen mit Synonymen nutzen zu können, muss also festgelegt werden, wie die Elemente, auf denen der Algorithmus arbeitet, in ICApps definiert sind und woher die Worte genommen werden.

Die folgenden Beschreibungen und erläuternden Texte sind in ICApps vorhanden:

- Beschreibungstexte der Einzelfunktionalitäten
- Kurzbeschreibungen der Komponenten sowie Beschreibungstexte, welche die gesamte Funktionsweise der Komponente darstellen
- Beschreibungstexte in der Komponente, welche die Umsetzung von Einzelfunktionalitäten in der Komponente beschreiben
- Beschreibungstexte in der Komponente, welche erläutern, für welchen Zweck die unterstützten Technologien eingesetzt werden
- Beschreibungstexte der Lizenzen, Voraussetzungen und Technologien

Als Elemente für den Algorithmus sollten „Einzelfunktionalitäten“, „Komponenten“, „Voraussetzungen“ und „Technologien“ gewählt werden. Lizenzen sind als Elemente nicht sinnvoll, da nicht zu erwarten ist, dass in Lizenztexten hilfreiche

Informationen technischer Art enthalten sind. Stattdessen werden in den Lizenztexten vorwiegend juristische Sachverhalte erläutert. Für jede Elementart müssen auch die Textinformationen festgelegt werden, aus denen die Worte entnommen werden und die dem Element zugeordnet werden. Für Einzelfunktionalitäten, Voraussetzungen und Technologien stehen jeweils die Beschreibungstexte zur Verfügung. Für die Komponenten können die Teile „Kurzbeschreibung“, „Beschreibung der Komponente“ und „Beschreibungen der Umsetzung von Einzelfunktionalitäten“ jeweils für eine Komponente als Textquelle verwendet werden.

Um die Datenbasis, auf der der Algorithmus arbeitet, noch zu vergrößern und vor allem auch Formulierungen von möglichst vielen unterschiedlichen Autoren in die Datenbasis mit einzubeziehen, können in elektronischer Form verfügbare Artikel aus Fachzeitschriften über die einzelnen Komponenten als Elemente aufgenommen werden.

Um eine gute Funktionsweise des Algorithmus zu erreichen, muss die Datenbasis, auf der der Algorithmus arbeitet, ausreichend groß sein. Zur Ausarbeitung des Algorithmus und Feineinstellung der Parameter müssen also schon genügend Komponenten in ICApps aufgenommen worden sein.

4.5 Reifebeurteilung mittels Integrationsgraph

Verwendet man Komponenten in eigener Software oder programmiert gegen eine von einer Komponente zur Verfügung gestellten Schnittstelle, so besteht starkes Interesse daran, über den Reifegrad der verwendeten Komponente richtig informiert zu sein. Nur so kann das Risiko, das beim Einsatz von fremdem Code eingegangen wird, dem Risiko einer eigenen Umsetzung der Funktionalität gegenübergestellt werden.

In dem hier behandelten speziellen Fall von freien Komponenten, in dem auch der Quelltext zu den ausführbaren Komponenten vorhanden ist, kann man einige sich daraus ergebende Konsequenzen nützen, um den Reifegrad einer Komponente durch den Grad der Integration der Komponente in andere Komponenten abzuschätzen.

Eine Komponente mit guter Qualität wird mit steigender Reife in immer stärkerem Maße in andere Komponenten integriert werden. Mit steigender Verbreitung von Software steigt auch die Qualität der Software über die Versionen hinweg, insofern der Hersteller der Software ein Interesse an der Qualitätssteigerung hat. Im speziellen Fall von Open Source bestehen einige sehr begünstigende Faktoren: Die Anwender (die im Fall von Komponenten ja selbst Entwickler sind) können bei Problemen mit der Software selbst im Quelltext nachvollziehen, an welcher Stelle ein Fehler auftritt und was das Programm an der entsprechenden Stelle

ausführt. In manchen Fällen kann der Anwender sogar gleich einen Korrekturvorschlag an die Entwickler der Software zurückschicken. Diese Umstände und die üblicherweise wesentlich einfachere Kommunikation mit Autoren von Open Source Software führen dazu, dass gefundene Fehler in der Software viel schneller und umfangreicher beseitigt werden, als in anderen Entwicklungsmodellen. Zudem werden umso mehr Fehler gefunden, je verbreiteter die Software im Einsatz ist. Somit kann im Fall von Open Source Software und insbesondere, wenn als Anwender ein großer Anteil an Entwicklern zu erwarten ist, aus einer großen Verbreitung der Software auf die hohe Qualität der Software geschlossen werden. Bei den in ICApps betrachteten Komponenten handelt es sich um Komponenten mit allgemeiner Funktionalität (siehe 3.2). In diesem Fall besteht bei einer Komponente von hoher Qualität eine Korrelation zwischen dem Verbreitungsgrad der Komponente und dem Integrationsgrad in andere Komponenten. Somit lässt sich aus dem Integrationsgrad der in ICApps betrachteten Komponenten ein Rückschluss auf den Reifegrad der jeweiligen Komponenten ziehen.

Um dem Entwickler den Integrationsgrad einer Komponente aufzuzeigen, könnte die Integration der verschiedenen Komponenten in einem Graphen dargestellt werden. Komponenten werden als Rechteck um ihren Namen dargestellt und die Integration der Komponente B durch Komponente A wird durch einen Pfeil von A nach B dargestellt. Die betrachtete Komponente K wird nun in der Mitte des Graphen platziert, alle Komponenten, die K integrieren, werden links von K dargestellt. Alle Komponenten, die K integrieren und von K selbst integriert werden, werden ober- und unterhalb dargestellt. Rechts von K werden alle Komponenten dargestellt, die K integriert. Mit Pfeilen werden dann die Integrationsbeziehungen der Komponenten untereinander verdeutlicht.

Somit ist aus dem Graph für den Entwickler klar ersichtlich, durch wieviele und welche Komponenten die betrachtete Komponente integriert wird. Weiterhin hat ein solcher Graph den Vorteil, dass der Entwickler sehen kann, ob die Komponenten, die er für seine Gesamtanforderung benötigt, durch eine Komponente ganz oder teilweise integriert wird. So kann hier der Übergang von einer Komponente zu einer anderen, die mehr Funktionalität beinhaltet und eventuell den Anforderungen besser gerecht wird, vereinfacht werden. Ein solcher Integrationsgraph ist für eine kleine Auswahl aus den Projekten der Apache Software Foundation (vgl. [ASF]) beispielhaft in Abbildung 4.3 dargestellt.

4.6 Iterative und interaktive Suche

Da es sehr unwahrscheinlich ist, dass ein Entwickler mit der ersten Auswahl an Kriterien direkt das optimale Ergebnis für seine Suche erzielt, sollte der Suchmechanismus eine iterative Verfeinerung bzw. Abänderung der Suchkriterien durch den Entwickler zulassen. Eine iterative Suche ist auch deshalb zu unterstützen,

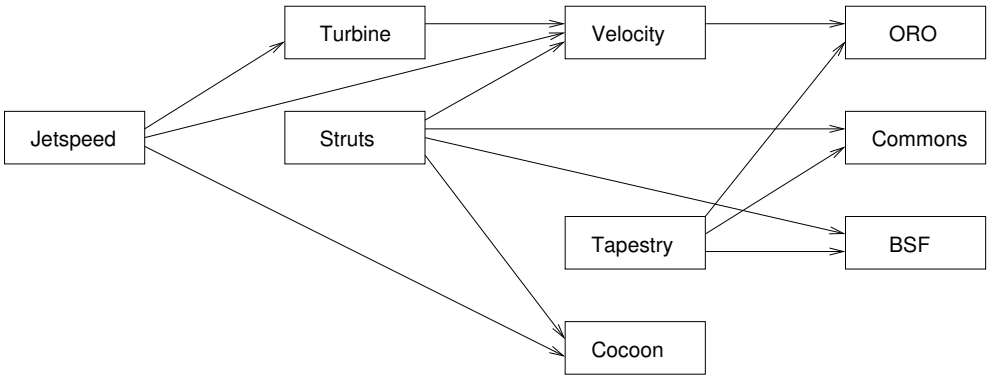

Abbildung 4.3: *In diesem Integrationsgraphen ist eine Auswahl an Projekten der Apache Software Foundation mit ihren Integrationsabhängigkeiten untereinander dargestellt.*

weil das menschliche Gehirn assoziativ auf Wissen zugreift (vgl. [WIL82]). Erst die Ergebnisse und Erkenntnisse aus einer vorhergehende Iteration der Suche lassen den Entwickler die Kriterien besser formulieren bzw. Korrekturen an den bisherigen Kriterien vornehmen. Zudem verändert sich der Informationsbedarf eines Entwicklers während der Suche, da durch neue Informationen aus den Ergebnissen der bisherigen Suche auch neue Fragestellungen entstehen (vgl. [HEN82]).

Das System sollte also „Suchen durch Neuformulierung“ (engl. „retrieval by reformulation“) unterstützen. Dazu sollte das System erlauben, Suchbegriffe hinzuzufügen, zu ändern oder zu löschen. Auch Kriterien wie zum Beispiel Lizenzen oder Voraussetzungen sollten als „notwendig“ (d.h. die gesuchte Komponente *muss* diese Ausprägung aufweisen) oder aber auch als „ausschließen“ (d.h. die gesuchte Komponente darf diese Ausprägung *nicht* aufweisen) kennzeichenbar sein. Dadurch wird eine iterative und interaktive Suche ermöglicht, die der Entwickler an seine sich veränderten Informationsbedürfnisse und sein momentanes Wissen anpassen kann. Zur Steigerung der Interaktivität sollte für alle verwendeten Begriffe (z.B. bei Lizenzen, Technologien und Voraussetzung) immer eine Beschreibung abrufbar sein.

Eine gute Möglichkeit, um den Entwickler weiter bei der Auswahl der Kriterien zu unterstützen, ist die Verwendung von „relevanten Begriffen“ (vgl. [LIN82]). Um eine Verwechslung zwischen Attributen und Attributwerten (also den konkreten Ausprägungen eines Attributs) zu vermeiden, wird im Folgenden anstatt „Attributwert“ die weniger verwechslungsträchtige Formulierung „Begriff“ für die konkreten Ausprägungen der Attribute verwendet. Bei der Suche mit „relevanten Begriffen“ werden für jedes Attribut, für das der Entwickler Begriffe als „notwendig“ oder „ausschließen“ auswählen kann, nur die Begriffe angeboten, welche die verbliebenen Komponenten unterscheiden. Als verbliebene Komponenten werden hierbei Komponenten bezeichnet, welche bei den bisherigen Iterationen der Su-

che alle Kriterien erfüllen und somit das bisherige Suchergebnis darstellen. Ist ein Begriff für ein Attribut aller verbleibenden Komponenten erfüllt oder für keine der verbleibenden Komponenten erfüllt, so ist dieser Begriff nicht mehr relevant und wird nicht mehr zur Auswahl angeboten. Stattdessen wird angezeigt, ob der Begriff enthalten ist oder nicht, also ob die bisherige Auswahl an Kriterien den Einschluss oder Ausschluss der Begriffs impliziert.

Eine Beschränkung auf die relevanten Begriffe bietet sowohl bei einer großen als auch bei einer kleinen Datenbasis Vorteile: Bei einer großen Datenbasis kann der Suchraum schneller eingeengt werden, da sichergestellt ist, dass mit jedem weiteren Kriterium ein Teil der Komponenten wegfällt. Bei einer kleinen Datenbasis führt dies sehr schnell zu einer einzigen Komponente, für die implizit die Ausprägung der Attribute festgelegt wird. Durch die implizite Festlegung der Attribute wird bei einer kleinen Datenbasis dem Entwickler somit auch sehr schnell klar, welche Kriterien die Anzahl der möglichen Komponenten sehr stark reduzieren und die Auswahl anderer, für ihn wichtigen Kriterien, nicht mehr möglich machen. Somit können Kriterien besser priorisiert werden, sollten sie für die vorhandene Datenbasis nicht gleichzeitig erfüllbar sein.

Ein weiterer Vorteil des Einsatzes der relevanten Begriffe ist, dass eine Auswahl an Kriterien in mehreren iterativen Schritten niemals zu einem leeren Ergebnis führen kann. Ein leeres Ergebnis würde weniger Informationen für den Entwickler bieten und ist demotivierender als das iterative Austesten, welche Kriterien für eine mögliche Lösung erfüllbar sind und welche nicht.

4.7 Lösungskonzept

In diesem Abschnitt soll nochmals kurz zusammengestellt werden, mit welchen Lösungsansätzen die Anforderungen aus Problemstellung und Analyse erfüllt werden sollen:

4.7.1 Funktionalitäten

Eine Komponente bietet häufiger eine *Vielzahl von Funktionalitäten* an. Die Repräsentation einer Komponente durch eine umfassende Beschreibung aller Funktionalitäten ist deshalb für eine schnelle Orientierung nicht klar genug strukturiert.

Als Lösungsansatz sollen die vorkommenden Funktionalitäten zu *Einzelfunktionalitäten* verallgemeinert werden. Die Repräsentation einer Komponente weist dann eine Aufzählung der erbrachten Einzelfunktionalitäten auf. Zu jeder Einzelfunktionalität wird zudem dargestellt, wie diese Funktionalität durch die Komponente erbracht wird.

4.7.2 Suchbegriffe

Mit steigender Anzahl an Komponenten und damit auch an Einzelfunktionalitäten ist es für den Entwickler nicht mehr möglich, alle im System verwendeten Einzelfunktionalitäten zu kennen. Es ist somit notwendig, den Entwickler bei der Suche durch einen *Suchmechanismus* zu unterstützen. Erschwerend kommen hierbei die Unterschiede zwischen der *anwendungsbezogenen Terminologie* des Entwicklers und der technischen neutralen Terminologie der Beschreibungstexte im System hinzu.

Als Lösungsansatz wird ein Suchsystem vorgeschlagen, das zum einen per *Volltextsuche* zutreffende Einzelfunktionalitäten oder Komponenten herausfindet. Zum anderen wird durch eine „*spreading activation*“ genannte Technologie eine Liste von Suchbegriffsynonymen oder anderen relevanten Begriffen automatisch erstellt. Dadurch lassen sich die oben beschriebenen Unterschiede in der Terminologie besser überbrücken.

4.7.3 Komponentenqualität

Damit Komponenten einsetzbar sind, müssen sie nicht nur die passende Funktionalität anbieten, sondern auch eine für die Anforderungen ausreichende *Qualität* aufweisen. Aus diesem Grund wird eine über die Qualitätsangaben des Komponentenentwicklers hinausgehende Qualitätsinformation benötigt.

Als Lösungsansatz wird der Aufbau eines *Integrationsgraphen* vorgeschlagen, der aufzeigt, welche Komponenten durch andere Komponenten integriert werden. Auf Grund der festgelegten Eigenschaften der Komponenten kann der Integrationsgrad zusätzliche Informationen zur Qualität einer Komponente liefern.

4.7.4 Interaktive Suche

Die Suche und Auswahl einer passenden Komponente ist eine komplexe Aufgabe: Der Entwickler muss aus einer Vielzahl von *Kriterien* die wichtigen identifizieren können. Erst während der Suche sammelt der Entwickler das notwendige Wissen, um dem System die notwendigen Kriterien vorgeben zu können.

Als Lösungsansatz wird ein Suchsystem vorgeschlagen, das eine *iterative Suche* zulässt. So kann der Entwickler bei jedem Iterationsschritt neue Erkenntnisse durch das Anpassen der Kriterien einfließen lassen. Um bei der Auswahl der Ausprägungen von Attributen diesen Prozess möglichst effizient zu gestalten, soll das System nur „*relevante Begriffe*“ für die Attribute anbieten.

4.7.5 Weitere Attribute

Für die Repräsentation der Komponenten im System ist es wichtig, dass der Entwickler sich schnell einen Überblick verschaffen kann, weshalb die Repräsentation gut *strukturiert* sein muss. Die Repräsentation muss aber ebenfalls ausreichend *Detailinformationen* enthalten, so dass der Entwickler sich einen umfassenden Überblick über die Funktionsweisen und Unterschiede verschiedener Komponenten gewinnen kann.

Als Lösungsansatz werden für die Repräsentation der Komponenten neben den Einzelfunktionalitäten noch die Attribute „Lizenzen“, „Technologien“ und „Voraussetzungen“ mit dem gleichen Aufbau, sowie eine Gesamtbeschreibung der Komponente hinzugefügt. Die Repräsentation einer Komponente entspricht dann folgender *Schablone*:

Attribut	**Inhalt und Zweck**
Name	zur Identifikation der Komponente
Einzelfunktionalitäten	die von der Komponente erbrachten Einzelfunktionalitäten, mit der Beschreibung wie die Komponente sie erbringt
Technologien	die Technologien, die bei der Benutzung der Komponente eingesetzt werden, mit der Beschreibung, für welchen Zweck welche Technologie eingesetzt wird
Lizenzen	eine oder mehrere Lizenzen, unter denen die Komponente vertrieben wird
Voraussetzungen	technische Voraussetzungen, die zum Einsatz der Komponente gegeben sein müssen
Beschreibung	vollständige Beschreibung der Funktionsweise der Komponente, hier kann auch das einer Komponente zugrundeliegende Konzept oder die „Philosophie“ erläutert werden.
Adresse der Webseite	für weiterführende Informationen und Dokumentation

4.7.6 Lösungsanforderungen

Das vorgestellte Lösungskonzept soll nun den in der Problemstellung (Kapitel 2) beschriebenen, nicht technischen Anforderungen gegenübergestellt werden. Die folgende Tabelle erläutert, wie alle nicht technischen Anforderungen durch das Lösungskonzept erfüllt werden:

Anforderung	**Anforderungserfüllung**
adäquate Repräsentation der Komponenten im System	Die nach verschiedenen Attributen strukturierte Komponentenrepräsentation ermöglicht eine schnelle Übersicht und gute Vergleichbarkeit.
schnelle Übersicht über die möglichen Alternativen ermöglichen	Die Strukturierung der Komponentenrepräsentation nach Attributen, insbesondere die Darstellung der Einzelfunktionalitäten, lässt eine Formulierung der Anforderungen als Attributsausprägungen zu. Damit können schnell Alternativen zu gegebenen Anforderungen gefunden werden.
effiziente Unterstützung der Suche mit flexiblen Kriterien	Die Volltextsuche ermöglicht einen schnellen Einstieg in mögliche Kriterien. Interaktive und iterative Suche lassen flexible Anpassung der Kriterien in mehreren Schritten zu. Die Suche mit „relevanten Begriffen“ gestaltet die Kriterienauswahl effizient.
Überblick über die Funktionsweise der Komponente geben	Die in der Komponentenrepräsentation enthaltene Beschreibung des Komponentenkonzepts kann diesen Überblick geben.
Komponententeile verschiedenen Funktionalitäten zuordnen	Bei den Einzelfunktionalitäten wird beschrieben, wie diese durch die Komponente erbracht werden. Ebenfalls werden die verwendbaren Technologien einem Einsatzzweck zugeordnet.

Kapitel 5

Design und Umsetzung

In den folgenden Abschnitten wird dargestellt, wie die Konzepte und Zielsetzungen des vorangegangenen Kapitels umgesetzt wurden. Auf Grund des engen Zusammenhangs zwischen Design und Umsetzung werden diese nicht getrennt dargestellt, sondern die relevanten Designüberlegungen erläutert und interessante Aspekte der Umsetzung dokumentiert. Die als Prototyp realisierte Umsetzung befindet sich im vollständigen Quelltext und kompiliert auf der als Anlage beigefügten CD.

Zuvor wird jedoch auf die Art des erstellten Prototyps eingegangen und anschließend der mehrschichtige Aufbau des Prototypen erläutert sowie jede Schicht einzeln vorgestellt. Nach der Beschreibung des Packens der Applikation für den Applikationsserver und dem Starten auf dem selbigen wird das Kapitel mit einer Darstellung eines exemplarischen Suchlaufes abgeschlossen.

Die Installations- und Konfigurationsbeschreibungen der für die Entwicklung und den Betrieb des Gesamtsystems notwendigen Software finden sich für die verwendeten Werkzeuge in Anhang A, für die eingerichtete, mehrschichtige Serverarchitektur in Anhang B und für die Entwicklungsumgebung Eclipse mit Erweiterungen in Anhang C.

5.1 Arten von Prototypen

Das ICApps-System stellt ein neuartiges System mit zahlreichen innovativen Ansätzen dar. Aus diesem Grund wurde ein Prototyp des Systems angefertigt. Für die Auswahl der Art des Prototypen standen folgende Alternativen zur Verfügung (vgl. [DIT02]):

Experimenteller Prototyp Der experimentelle Prototyp wird auch als Wegwerf-Prototyp bezeichnet. Oft tritt er in Verbindung mit Rapid Prototyping auf.

Bei dieser Art von Prototyp wird auf umfassende Architekturüberlegungen und Design soweit als möglich verzichtet. Ziel des Prototypen ist es, offene Fragestellungen zu klären bzw. Erfahrung mit neuen Technologien zu sammeln und diese zu testen. Der Prototyp wird üblicherweise mit der Intention erstellt, zwar die gewonnenen Kenntnisse aus dem Prozess für das Design des richtigen Programms zu nutzen, aber den Programmtext des Prototyps selbst vollständig zu verwerfen. Damit fallen auch sehr viele Anforderungen an Dokumentation und Erweiterbarkeit des Programmtextes weg. Sollten Teile des Prototypen doch in das richtige Programm übernommen werden, so sollte dies erst nach eingehender Prüfung dieser Programmteile und der Erweiterung um die zusätzlichen Anforderungen geschehen.

Explorativer Prototyp Ein explorativer Prototyp tritt oft als Benutzerschnittstellen-Prototyp auf. Er kann zum Beispiel nur aus Eingabemasken bestehen, die aber keine reale Funktionalität besitzen oder nur Testdaten liefern. Auch ist es möglich, das Resultat auf Benutzereingaben durch eine Person in Realzeit erstellen zu lassen. Diese Art von Prototyp dient dem Entwurf von Benutzerschnittstellen. Dabei können fachliche Anforderungen an das zu erstellende System wesentlich besser geklärt werden, als nur mit reinen Textspezifikationen. Oftmals werden viele Systemanforderungen durch den Anwender erst dann formuliert, wenn er sich eine graphische Vorstellung von einer Anwendung machen kann.

Evolutionärer Prototyp Bei einem evolutionären Prototyp wird nicht die gesamte Funktionalität, die ein System erbringen soll, in einem Schritt umgesetzt. Stattdessen werden mehrere Iterationen durchlaufen, in denen jeweils mehr Funktionalität in das Programm integriert wird. Jede Iteration liefert selbst aber schon ein für den Anwender anwendbares und lauffähiges Programm. Mit jeder Iteration soll durch die dazu gekommene Funktionalität der Nutzen des Programms für den Anwender erhöht werden. Durch kleine Iterationsschritte kann der Anwender so frühzeitig auf Missverständnisse hinweisen und neue Funktionalitäten besser priorisieren. Die iterative Entwicklungsweise fordert auf der Seite des Entwicklers durch die potentiell nötigen strukturellen Anpassungen im Programmtext zwischen den Iterationen ein höheres Maß an Flexibilität. Jedoch wird das Risiko des Gesamtprojektes durch die Iterationen und die frühe Rückmeldung durch den Anwender vermindert. Somit eignet sich diese Entwicklungsweise insbesondere für hoch dynamische und neuartige Aufgabenstellungen.

Für das ICApps-System ist ein explorativer Prototyp nicht ausreichend, da keine Funktionalität vollständig benutzbar umgesetzt wäre. Bei der Entscheidung zwischen einem experimentellen und evolutionären Prototypen musste zwischen einem vollen Funktionsumfang, der aber nicht wart- und erweiterbar sein würde

und einem partiellen Funktionsumfang, der aber später eingesetzt und weiterentwickelt werden kann, entschieden werden. Durch die Neuheit des Systemansatzes auf diesem Gebiet fiel die Entscheidung zugunsten des evolutionären Prototyps, der noch nicht alle geplante Funktionalität enthält. Stattdessen soll mit dem gewählten Lösungsansatz Erfahrung gesammelt werden und basierend auf dieser das System weiterentwickelt werden. Das ICApps-System zur Unterstützung der Suche nach Java-Komponenten sollte selbst auch auf Java-Komponenten basieren. Das System wurde deshalb in der Programmiersprache Java umgesetzt. Die Verwendung von Komponenten wiederum unterstützt die Entwicklung nach einem evolutionären Prinzip, da neue Funktionalitäten teilweise mit neuen Komponenten abgedeckt werden können. Somit ergänzen sich hier die Entscheidungen sowohl zur Minimierung des Risikos als auch zur Entwicklung mit Komponenten und sprechen für die Wahl der Entwicklung eines evolutionären Prototyps.

Mit diesem Prototyp können bereits Komponenten aufgenommen, verändert und angezeigt werden. Dazu wurde die erarbeitete Struktur der Komponentenrepräsentation modelliert und umgesetzt (vgl. Abschnitt 4.7.5). Weiterhin wurde es dem Entwickler ermöglicht, eine Suche auf der Datenbasis auszuführen. Dazu werden zum einen eine Volltextsuche und zum anderen eine Auswahl nach verschiedenen, strukturierten Kriterien angeboten. Hierbei kommt das Prinzip der „relevanten Begriffe“ zum Einsatz (vgl. Abschnitt 4.6).

Eine weitergehende Unterstützung der Suche durch Spreading Activation (vgl. Abschnitt 4.4) und einen Integrationsgraphen (vgl. Abschnitt 4.5) sind Funktionalitäten, die für weitere Iterationsschritten außerhalb dieser Arbeit vorgesehen sind.

5.2 Grundsätzlicher Aufbau

Für das ICApps-System wurde die erste Iteration der Entwicklung mit der Erstellung des Prototypen, der Dokumentation der ersten Komponenten und damit der Eingabe von ersten Daten in das System durchgeführt. Nach Abschluss dieser Arbeit soll das System in weiteren Iterationen erweitert werden und vor allem die Datenbasis durch die Aufnahme weiterer Daten vergrößert werden (vgl. [HOF03]). Aus diesem Grund sollten die Daten zentral auf einem Server gehalten werden. Dies erspart einen aufwändigen Datenabgleich, der bei einer verteilten Datenhaltung nötig wäre. Der Nachteil, dass zum Recherchezeitpunkt eine Verbindung zum Server bestehen muss, ist in diesem Fall nicht sehr ausgeprägt, da im bisherigen alternativen Fall bei der Recherche nach Komponenten auch eine Verbindung zum Internet bestehen musste, um auf die Beschreibungen der Komponenten zugreifen zu können. Für das Programm selbst ist die Entscheidung, dieses auf einem zentralen Server ablaufen zu lassen, ebenfalls von Vorteil, da

somit bei Erweiterungen des Programms nicht alle bereits dezentral installierten Programme auf den aktuellen Stand gebracht werden müssen.

Aus den zu erwartenden Änderungen am Programm selbst ergibt sich eine weitere Anforderung: Die Daten für ICApps sollten unabhängig von der programmtechnischen Umsetzung gespeichert werden. Weiterhin erweisen sich Daten im Allgemeinen als wesentlich langlebiger und oft auch wertvoller als die Programme, die darauf arbeiten. Deshalb wurde für die Speicherung der Daten eine relationalen Datenbank gewählt, welche von einem Datenbankmanagementsystem (DBMS) verwaltet wird. Diese Art der Speicherung der Daten ist langjährig in der Industrie erprobt, findet breite industrielle Unterstützung und es stehen umfangreiche Werkzeuge zur Ansicht, Restrukturierung und Datensicherung solcher Datenbanken zur Verfügung (vgl. [GIG01]).

Um das Programm möglichst einfach erweiterbar und wartbar zu halten, sollten die Programmteile, die Daten verarbeiten oder ändern, getrennt von den jeweiligen Programmteilen, die den Ablauf der Benutzerschnittstelle steuern und Daten anzeigen, erstellt werden. So wird es auch mit minimalem Aufwand ermöglicht, dass mehrere Benutzerschnittstellen auf die zentralen Funktionalitäten des Programms zugreifen können und diese nicht für jede Benutzerschnittstelle neu umgesetzt werden müssen. (vgl. [SIN02])

Aus den vorgestellten Anforderungen und Überlegungen ergibt sich für die grundlegende Struktur des Systems ein dreischichtiger Aufbau:

Schicht 1: Datenspeicherungs-Schicht, in der die Daten in einem DBMS verwaltet werden

Schicht 2: Geschäftslogik-Schicht, die alle Programmteile enthält, die Daten verarbeiten oder verändern

Schicht 3: Präsentations-Schicht, welche die Programmteile zur Präsentation von Daten und zur Ablaufsteuerung der Benutzerschnittstelle enthält

Die Schichten 1 und 2 sollen hierbei auf einem zentralen Server ausgeführt werden. Die Dienste der Schicht 3 können entweder ganz auf einem Klienten oder zum größten Teil auf dem Server und nur zu sehr geringem Teil auf dem Klienten ausgeführt werden. Auf diese Fragestellung geht Abschnitt 5.5.1 genauer ein.

Als DBMS wurde PostgreSQL ausgewählt, da dies zum einen unter einer freien Lizenz (BSD-Lizenz) vertrieben wird und zum anderen nicht nur einfache SQL-Befehle akzeptiert, sondern den SQL-Sprachstandard in allen wichtigen komplexeren Konstrukten unterstützt (vgl. [PGSQL]). Als komplexere SQL-Konstrukte wurden hier beispielsweise Transaktion, geschachtelte Abfragen, referentielle Integrität, Stored Procedures und Sichten angesehen. Die Installation und Konfiguration von PostgreSQL ist in Anhang B.2 dokumentiert.

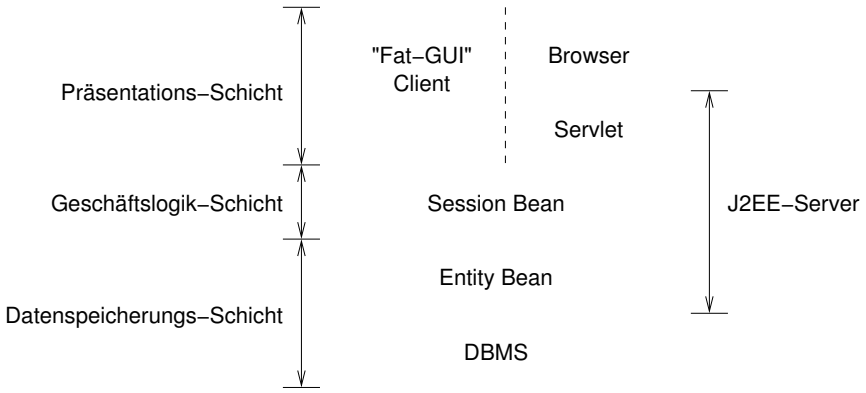

Abbildung 5.1: *Für die verschiedenen Schichten ist die Zuordnung zu einzelnen Teilen des Gesamtsystem dargestellt. Der J2EE-Applikationsserver deckt die mittleren Teile des Systems ab.*

Die Programmteile der Schicht 2 und bei serverseitiger Umsetzung auch der Schicht 3 bieten sich an, auf einem J2EE-Server umgesetzt zu werden (vgl. [SIN02]). Die Zuordnung der Schichten zu J2EE-Technologien sieht dabei wie folgt aus:

Datenspeicherungs-Schicht	Anbindung an eine Datenbank via JDBC und Datenkapselung in *Entity Beans*
Geschäftslogik-Schicht	Umsetzung der Funktionalitäten in *Stateful* oder *Stateless Session Beans*
Präsentations-Schicht	Entweder durch entfernten Zugriff von einem Klienten aus oder durch *Java ServerPages* oder *Servlets*

Die Zuordnung zwischen den Schichten und J2EE-Technologien ist zusammenfassend in Abbildung 5.1 dargestellt.

Als J2EE-Applikationsserver wurde JBoss[1] ausgewählt, da dieser unter einer freien Lizenz (LGPL) erhältlich ist, die J2EE-Spezifikation bis zur Version 1.3 voll unterstützt und gleichzeitig eine ähnlich hohe Verbreitung wie Applikationsserver kommerzieller Anbieter aufweist (vgl. [JBOSS]). Die Installation und Konfiguration von JBoss ist in Anhang B.1 dokumentiert.

[1]Der heutige Name „JBoss“ des Applikationsservers hat die folgende Entstehungsgeschichte: Die erste Version dieses Applikationsservers trug den Namen „EJBOSS“, welcher aus den Teilen EJB für Enterprise JavaBeans und OSS für Open-Source Software zusammengesetzt war. Auf Grund von Markenrechten verlangte Sun Microsystems von den Entwicklern, den geschützten Begriff „EJB“ nicht im Namen zu verwenden. Gemäß der schon in vielen anderen Projekten üblichen Namensgebung erschien ein mit „J“ (für Java) beginnender Name als angemessen, was einfach durch das Streichen des führenden Buchstaben „E“ aus „EJBOSS“ die Kombination „JBOSS“ entstehen ließ. Im Laufe der Zeit setzte sich dann die heutige Schreibweise „JBoss“ durch.

Die Möglichkeiten der verwendeten Programmiersprache Java wurden durch ein objektorientiertes Design und eine objektorientierte Umsetzung genutzt. Das Design der einzelnen Schichten des ICApps-Systems wird in den folgenden Abschnitten ausführlich beschrieben.

5.3 Persistenz und Datenmodell

Die Objekte, die in einem objektorientierten System verwendet werden, enthalten nicht nur die Daten einer aktuell ablaufenden Recherche, sondern auch alle Daten, die das System über die im System dokumentierten Komponenten kennt. Somit müssen die Daten über die Komponenten einen Neustart des System überdauern können. Dafür werden persistente Objekte benötigt, d.h. Objekte, deren Ausprägungen von Attributen in einem permanenten Datenspeicher gesichert werden und die diese Ausprägungen auch nach dem Neustart des Systems noch besitzen. Diese Daten sollen, wie bereits erwähnt, in einer relationalen Datenbank gespeichert werden, welche von einem relationalen DBMS verwaltet wird. Die Datenspeicherungsschicht wird hier erweitert um die Persistenzmechanismen für persistente Objekte gesehen.

5.3.1 Entity Beans

Im J2EE-Modell wird die definierte Datenspeicherungsschicht sehr gut durch *Entity Beans* abgebildet (vgl. Abschnitt 3.1.1). Durch die Verwendung von Container Managed Persistence (CMP) kann die Aufgabe der Persistenz vollständig dem Applikationsserver überlassen werden. Dieser übernimmt damit die Aufgabe, die Attributwerte in ein eingebundenes DBMS zu speichern. Dadurch ist das System auch vollständig von der darunterliegenden Datenbank abstrahiert. Nötige Anpassungen an verschiedene Datenbankmanagementsysteme wirken sich nicht auf die *Entity Beans* aus, sondern werden vollständig unabhängig durch die Konfiguration des DBMS-Zugriffs im Applikationsserver abgedeckt.

Auf die verwendeten Entity Beans soll ausschließlich über lokale Schnittstellen zugegriffen werden. Es werden also folgende drei Java Klassen pro Entity Bean benötigt (vgl. [MIC01]):

1. Die Implementierung der Entity Bean, die auf dem Applikationsserver ausgeführt wird:
Sie enthält Zugriffsmethoden auf die Attribute des Objekts, Methoden zur Erzeugung und Entfernung des Objekts sowie eventuelle Geschäftsmethoden.

```
import javax.ejb.EntityBean;
```

```
import javax.ejb.EntityContext;
public abstract class FunctionalityBean implements EntityBean
{
  EntityContext ctx = null;

  public String ejbCreate(String name) throws CreateException {
    setFunctionalityID(IdSequence.nextId());
    setName(name);
    return null;
  }
  public void ejbPostCreate(String name) {};
  // Properties
  public abstract Integer getFunctionalityID();
  public abstract void setFunctionalityID(Integer functionalityID);
  public abstract String getName();
  public abstract void setName(String name);
  public abstract String getDescription();
  public abstract setDescription(String description);
  // implementing EntityBean
  public void setEntityContext(EntityContext ctx) { this.ctx = ctx; }
  public void unsetEntityContext() { ctx = null; }
  public void ejbActivate() { }
  public void ejbPassivate() { }
  public void ejbLoad() { }
  public void ejbStore() { }
  public void ejbRemove() { }
}
```

Im Quelltext der *Entity Bean* wird die Schnittstelle `javax.ejb.EntityBean` implementiert. Die Methoden dieser Schnittstelle werden im Lebenszyklus des Objekts vom Applikationsserver aufgerufen, der dem Objekt damit mitteilt, in welchen Zustand es übergehen wird bzw. übergegangen ist. Damit über den Applikationsserver neue Instanzen der Entity Bean erzeugt werden können, ist auch eine ejbCreate-Methode nötig, die hier bereits ein Attribut (den Namen) der Entity Bean übernimmt. In der ejbCreate Methode muss der primäre Schlüssel für die Instanz festgelegt werden. Da dieser hier nicht von außen vorgegeben wird, wird er vom Hilfsobjekt `IdSequence` angefordert. Zu jeder ejbCreate-Methode muss ebenso eine ejbPostCreate-Methode mit gleicher Signatur vorhanden sein. Die Zugriffsmethoden für die Attribute der Entity Bean können abstrakt bleiben, wie in Abschnitt 5.3.2 näher beschrieben wird. Als Konsequenz der abstrakten Methoden ist auch die ganze Klasse als abstrakt zu deklarieren.

2. Die Definition der Schnittstelle für den lokalen Zugriff auf die Methoden der Implementierung:

```
public interface FunctionalityLocal
```

```
    extends javax.ejb.EJBLocalObject {
  public java.lang.String getDescription( ) ;
  public java.lang.Integer getFunctionalityID( ) ;
  public java.lang.String getName( ) ;
  public void setDescription( java.lang.String description ) ;
  public void setFunctionalityID( java.lang.Integer functionalityID ) ;
  public void setName( java.lang.String name ) ;
}
```

Da diese Schnittstelle nicht für entfernte (pass-by-value), sondern für lokale (pass-by-reference) Zugriffe genutzt werden soll, muss sie von der Schnittstelle `javax.ejb.EJBLocalObject` abgeleitet werden. Alle Methoden der Implementierungsklasse, auf die ein Klient zugreifen können soll, müssen in dieser Schnittstelle deklariert sein. Für den hier beschriebenen Fall einer Schnittstelle für lokale Zugriffe befindet sich das aufrufende Objekt immer auf dem Applikationsserver, also lokal zur Implementierungsklasse. Dadurch ist die Möglichkeit der pass-by-reference Semantik immer gegeben und Methodenaufrufe können sehr rasch ausgeführt werden, da im Gegensatz zu pass-by-value die Duplikation und Übertragung der Parameterobjekte wegfällt.

3. Die Definition der Schnittstelle, über die der Applikationsserver die Steuerung des Lebenszykluses dieses Entity Bean Typs anbietet:

```
public interface FunctionalityLocalHome
    extends javax.ejb.EJBLocalHome {
  public FunctionalityLocal create(String name)
    throws javax.ejb.CreateException;
  public java.util.Collection findAll()
    throws javax.ejb.FinderException;
  public FunctionalityLocal findByPrimaryKey(Integer pk)
    throws javax.ejb.FinderException;
}
```

Für eine lokale *Entity Bean* muss die obige Schnittstelle wiederum von der Klasse `javax.ejb.EJBLocalHome` abgeleitet werden. Für jede ejbCreate()-Methode aus der Implementierung muss hier eine create()-Methode mit der selben Signatur deklariert werden. Weiterhin sind alle Finder-Methoden zu deklarieren, über die einzelne oder alle früher erstellten Instanzen der Entity Bean wiedergefunden werden können. Die Implementierung zu den Finder-Methoden werden vom Applikationsserver automatisch erstellt und müssen nicht programmiert werden.

Deskriptor-Eintrag Weiterhin muss noch für jede *Entity Bean* ein Eintrag im Deployment-Deskriptor `ejb-jar.xml` im XML-Format vorgenommen werden:

```
<ejb-jar>
  <enterprise-beans>
    <entity>
      <description>Functionality</description>
      <ejb-name>Functionality</ejb-name>
      <local-home>FunctionalityLocalHome</local-home>
      <local>FunctionalityLocal</local>
      <ejb-class>FunctionalityCMP</ejb-class>
      <persistence-type>Container</persistence-type>
      <prim-key-class>java.lang.Integer</prim-key-class>
      <reentrant>False</reentrant>
      <cmp-version>2.x</cmp-version>
      <abstract-schema-name>Functionality</abstract-schema-name>
      <cmp-field>
        <field-name>functionalityID</field-name>
      </cmp-field>
      <cmp-field>
        <field-name>name</field-name>
      </cmp-field>
      <cmp-field>
        <field-name>description</field-name>
      </cmp-field>
      <primkey-field>functionalityID</primkey-field>
    </entity>
  </enterprise-beans>
</ejb-jar>
```

Der Deskriptor gibt an, welche drei Klassen oder Schnittstellen zu einer Entity Bean gehören und welcher Persistenz-Mechanismus (hier CMP) verwendet werden soll. Weiterhin müssen alle Attribute aufgeführt werden, die der Applikationsserver persistent machen soll.

Über diesen in der J2EE-Spezifikation vorgesehenen und standardisierten Deskriptor hinaus kann jeder Applikationsserver noch einen proprietären Deskriptor hinzufügen. Bei JBoss bestimmt dieser zum Beispiel die Konfiguration, mit der die Bean verwaltet und unter welchem Namen die *Entity Bean* in den JNDI-Dienst aufgenommen werden soll (vgl. [STARK]). Dem zusätzlichen Deskriptor namens `jboss.xml` werden folgende Inhalt hinzugefügt:

```
<jboss>
  <enterprise-beans>
    <entity>
```

```
        <ejb-name>Functionality</ejb-name>
        <local-jndi-name>FunctionalityLocal</local-jndi-name>
      </entity>
    </enterprise-beans>
  </jboss>
```

Hierbei werden zum Beispiel für das Caching oder für Poolgrößen keine von den Standardeinstellungen abweichenden Einstellungen vorgenommen. Für den in `ejb-jar.xml` festgelegten *Entity Bean*-Namen wird ein neuer JNDI-Name festgelegt.

5.3.2 Container Managed Persistence und Container Managed Relationships

Beim Schritt der J2EE-Spezifikation von 1.2 nach 1.3 wurde die enthaltene Spezifikation für Java Enterprise Beans (EJB) von 1.1 auf 2.0 umgestellt. Die Spezifikation EJB 2.0 hat auf dem Gebiet der Container Managed Persistence (CMP) einige Neuerungen mit sich gebracht: Im Vergleich zu EJB 1.1 müssen für die Attribute eines Objekts keine Variablen mehr im Objekt vorgesehen werden. Ebenso müssen die get- und set-Methoden nicht implementiert werden, sondern können als abstrakt deklariert werden. Der Applikationsserver ergänzt dann selbst die nötige Implementierung. Dies hat den großen Vorteil, dass der Applikationsserver schreibende Zugriffe auf Attribute (via set-Methode) wesentlich besser feststellen kann, als bei der in EJB 1.1 noch vorhandenen Variable. Somit kann der Applikationsserver auch wesentlich besser steuern, wann Änderungen an Attributen in die Datenbank geschrieben werden müssen und somit kann die Leistungsfähigkeit von *Entity Beans* erheblich gesteigert werden. *Entity Beans* lassen sich mit dieser Architektur als intelligente Pufferschicht zwischen anderen EJBs und der Datenbank nutzen.

Eine weitere Neuerung von EJB 2.0 ist die Einführung von Container Managed Relationships (CMR). Die Referenzen zwischen Entity Beans müssen nun nicht mehr, wie bei EJB 1.1, von Hand programmiert werden, sondern werden ebenfalls vom Applikationsserver verwaltet. Für jede vom Applikationsserver verwaltete Referenz ist hierbei anzugeben, ob die Referenz beidseitig oder nur von einer Seite verfolgt werden kann. Für die einseitige Variante muss die Referenzrichtung angegeben werden. Ebenso ist die Multiplizität der Referenz für beiden Seiten festzulegen. Die Schnittstellen zur Erzeugung und Abfrage der Referenzen werden als abstrakte Methoden deklariert und wiederum vom Applikationsserver implementiert:

```
public abstract FunctionalityLocal getFunctionality();
public abstract void setFunctionality(FunctionalityLocal t);
```

Auch CMRs müssen im `ejb-jar.xml` Deskriptor angegeben werden:

```
<ejb-jar>
  <relationships>
    <ejb-relation>
      <ejb-relation-name>funclink-functionality</ejb-relation-name>
      <ejb-relationship-role>
        <ejb-relationship-role-name>funclink points
          to functionality</ejb-relationship-role-name>
        <multiplicity>Many</multiplicity>
        <relationship-role-source >
          <ejb-name>FuncLink</ejb-name>
        </relationship-role-source>
        <cmr-field >
          <cmr-field-name>functionality</cmr-field-name>
        </cmr-field>
      </ejb-relationship-role>
      <ejb-relationship-role >
        <ejb-relationship-role-name>functionality is
          refered by funclink</ejb-relationship-role-name>
        <multiplicity>One</multiplicity>
        <relationship-role-source >
          <ejb-name>Functionality</ejb-name>
        </relationship-role-source>
      </ejb-relationship-role>
    </ejb-relation>
  <relationships>
<ejb-jar>
```

Soll auf einer Seite der Referenz mit vielfacher (statt einfacher) Multiplizität auf die Referenzen zu anderen Entity Beans zugegriffen werden, so wird als Objekttyp für die get- und set-Methoden nicht die Schnittstelle der referenzierten Entity Bean verwendet, sondern eine `java.util.Collection` (wahlweise auch ein `java.util.Set`). Dieses `Collection`-Objekt enthält dann die Schnittstellen der referenzierten Entity Beans. Die Deklarationen der Zugriffsmethoden müssen entsprechend angepasst werden.

Auch hier kann der Applikationsserver weitere Einstellungen über einen eigenen Deskriptor festlegen. Dieser ist im Fall von JBoss der Deskriptor `jbosscmp-jdbc.xml`, der größtenteils die gleichen Angaben enthält wie `ejb-jar.xml`. Darüber hinaus wird angegeben, ob die Datenbanktabellen von JBoss automatisch erzeugt und nach dem Beenden der Applikation auch wieder gelöscht werden sollen. Sowohl die automatische Erstellung als auch das automatische Löschen haben sich während der Entwicklung als sehr hilfreich herausgestellt. Eine weitere ergänzende Angabe bezieht sich auf die Modellierung der Referenzen zwischen den

Entity Beans in der Datenbank. Die Referenzen können als ein Feld mit dem Fremdschlüssel in der Tabelle oder als eigenständige Tabelle von Referenzen in der Datenbank abgelegt werden. N-zu-m-Relationen können ausschließlich durch die letztere Methode (Referenztabelle) abgebildet werden.

Bemerkenswert bei der Programmierung und Konfiguration von CMRs ist, dass der nachträgliche Übergang von einer unidirektionalen zu einer bidirektionalen Referenz oder die Umkehrung der Referenzrichtung zwar Änderungen in den Java Klassen benötigt, jedoch keine Änderungen in der Datenbankstruktur zur Folge hat. D.h., wird zu einem späteren Zeitpunkt für eine bisher unidirektionale Referenz auch die Rückrichtung benötigt, so kann man die entsprechenden Methoden einfach in die Java Klassen aufnehmen, muss jedoch keinerlei Änderungen an der Datenbankstruktur vornehmen und kann somit alle bereits bestehenden Datenbankdaten unverändert weiter verwenden.

5.3.3 Xdoclet

Xdoclet ist eine Doclet-Erweiterung für JavaDoc. Xdoclet ermöglicht die deklarative Programmierung vieler Bereiche, die durch die J2EE-Spezifikation abgedeckt werden (vgl. [XDOC]). Hier wurde Xdoclet eingesetzt, um die Erstellung von Entity Beans erheblich zu vereinfachen. Alle Klassen und Deskriptoren, die in den beiden Abschnitten 5.3.1 und 5.3.2 beschrieben wurden, werden durch eine einzige Java Klasse ersetzt, welche die nötigen Informationen als JavaDoc-Tags enthält. Die nötigen Java Klassen und Deskriptoren werden dann durch Xdoclet aus der Ausgangsklasse erzeugt. Im folgenden wird eine deklarative beschriebene Entity Bean dargestellt, die sowohl Attribute, als auch Referenzen zu anderen *Entity Beans* enthält. Die Eigenschaften, welche die *Entity Bean* als Ganzes betreffen, sowie der *Entity Bean*-Name, der Persistenz-Mechanismus oder der Typ des Primärschlüssels werden der Klassendefinition vorangestellt:

```
import javax.ejb.CreateException;
import javax.ejb.EntityBean;

/**
 * @ejb:bean name="FuncLink"
 *           type="CMP"
 *           jndi-name="ejb/FuncLink"
 *           primkey-field="funcLinkID"
 *           view-type="local"
 * @ejb:pk class="java.lang.Integer"
 * @ejb:finder signature="Collection findAll()"
 * @ejb:interface local-class="FuncLinkLocal"
 *
```

```
 * @jboss:table-name "funclink"
 * @jboss:create-table "true"
 * @jboss:remove-table "true"
 *
 */
```

Die eigentliche Klasse wird wie bisher definiert, jedoch um den auszeichnenden Xdoclet-Tag für eine create-Methode erweitert:

```
public abstract class FuncLinkBean implements EntityBean {
  /**
   * @ejb.create-method view-type="local"
   */
  public Integer ejbCreate() throws CreateException {
    setFuncLinkID(IdSequence.nextId());
    return null;
  }
  public void ejbPostCreate(String name) {};
```

Für die folgenden Attribute werden die abstrakten Methoden wie bekannt deklariert. Die deklarative Auszeichnung dieser Methoden für den Zugriff auf Attribute erfolgt auch hier über Xdoclet-Tags:

```
/**
  * @ejb.interface-method view-type="local"
  * @ejb.persistent-field
  */
 public abstract Integer getFuncLinkID();
 /**
  * @ejb.interface-method view-type="local"
  */
 public abstract void setFuncLinkID(Integer functionalityID);
 /**
  * @ejb.interface-method view-type="local"
  * @ejb.persistent-field
  */
 public abstract String getDescription();

/**
  * @ejb.interface-method view-type="local"
  */
 public abstract void setDescription(String description);
```

Abschliessend wird eine Referenz zu einer anderen Entity Bean, wie bekannt, mit den abstrakten Methoden und den deklarativen Xdoclet-Tags deklariert:

```
    /**
     * @ejb.interface-method view-type="local"
     * @ejb.relation name="funclink-functionality"
     *         role-name="funclink points to functionality"
     *         target-ejb="Functionality"
     *         target-role-name="functionality is refered by funclink"
     *         target-multiple="yes"
     *
     * @jboss.relation
     *         related-pk-field="functionalityID"
     *         fk-column="functionalityID"
     */
    public abstract FunctionalityLocal getFunctionality();
    /**
     * @ejb.interface-method view-type="local"
     */
    public abstract void setFunctionality(FunctionalityLocal t);
}
```

Alle benötigten Dateien werden aus einer einzigen Datei erstellt. Dies erleichtert die Entwicklung von EJBs erheblich und vermeidet das sonst erhebliche Problem der Inkonsistenz zwischen den verschiedenen Schnittstellen und Implementierungen. Trotzdem erhält man als Zwischenschritt alle für die J2EE-kompatible Entwicklung nötigen Dateien im Quelltext. Xdoclet kann durch das Verändern einer Konfigurationsvariablen auf alle weit verbreiteten J2EE-Applikationsserver eingestellt werden und erzeugt auch die, für den jeweiligen Applikationsserver spezifischen, Deskriptoren automatisch und korrekt. Welche Dateien Xdoclet genau aus der deklarativen Beschreibung erzeugt, ist in Anhang A dokumentiert.

5.3.4 Entitäten und Datenmodell

Zur strukturierten Repräsentation der dokumentierten Komponenten müssen verschiedene Entitäten definiert werden, welche die benötigten Information über die Bestandteile des Systems und die Beziehungen zwischen ihnen darstellen können (vgl. Abschnitt 4.7). Für die verschiedenen Bestandteile des Systems wurden die folgenden Entitäten identifiziert:

Komponente: enthält die Informationen über die im System dokumentierten Komponenten

Funktionalität: dient zur Repräsentation und Beschreibung der Einzelfunktionalitäten

Lizenz: repräsentiert die verschiedenen Lizenzen, die dem System bekannt sind und Komponenten zugeordnet werden können

Voraussetzung: stellt verschiedene technologische Voraussetzungen oder Technologien dar, die eine Komponente benötigt, um ablaufen zu können

Technologie: enthält verschiedene Technologien wie beispielsweise JavaServer Pages, Servlets, JDBC oder XML, die von einer Komponente unterstützt werden können

Die Entität „Komponente“ steht jeweils in Beziehung mit jeder der anderen, oben genannten Entitäten. Da für Einzelfunktionalitäten (Entität: Funktionalität) beschrieben werden soll, auf welche Art und Weise diese durch die Komponente erbracht wird, muss die Beziehung selbst noch Eigenschaften beinhalten. Dazu ist es auf der technischen Ebene nötig, eine weitere Klasse für die Komponente-Einzelfunktionalität-Beziehung einzuführen, die diese Eigenschaften der Beziehung enthalten kann. Ebenso soll für eine Technologie beschreibbar sein, wie diese mit der Komponente genutzt werden kann. Auch hier wird also eine Klasse für diese Beziehung benötigt:

FunktionalitätVerbindung enthält die Beschreibung, wie eine Einzelfunktionalität durch die Komponente umgesetzt wird. Die Komponente verweist nun nicht direkt auf eine Einzelfunktionalität, sondern auf die FunktionalitätVerbindung-Klasse, welche wiederum auf die Einzelfunktionalität verweist.

TechnologieVerbindung enthält die Beschreibung, wie die jeweilige Technologie in der Komponente genutzt werden kann. Auch hier verweist die Komponente auf diese Klasse, welche wiederum auf die Technologie verweist.

Alle bisher beschriebenen Entitäten und Klassen besitzen jeweils eine Beschreibung. Für die Beschreibung sollte es möglich sein, auch graphische Elemente einbinden zu können. Es liegt also nahe, eine Entität für Beschreibungen sowie eine Entität für graphische Elemente zu schaffen.

Beschreibung enthält einen Beschreibungstext und kann auf mehrere dazugehörende Grafiken verweisen.

Grafik speichert die Daten einer graphischen Darstellung zusammen mit einem Untertitel.

Anstatt also ein eigenes Beschreibungsattribut zu besitzen, referenzieren alle Entitäten auf eine nur ihnen zugeordnete Beschreibungs-Entität. Somit erhalten

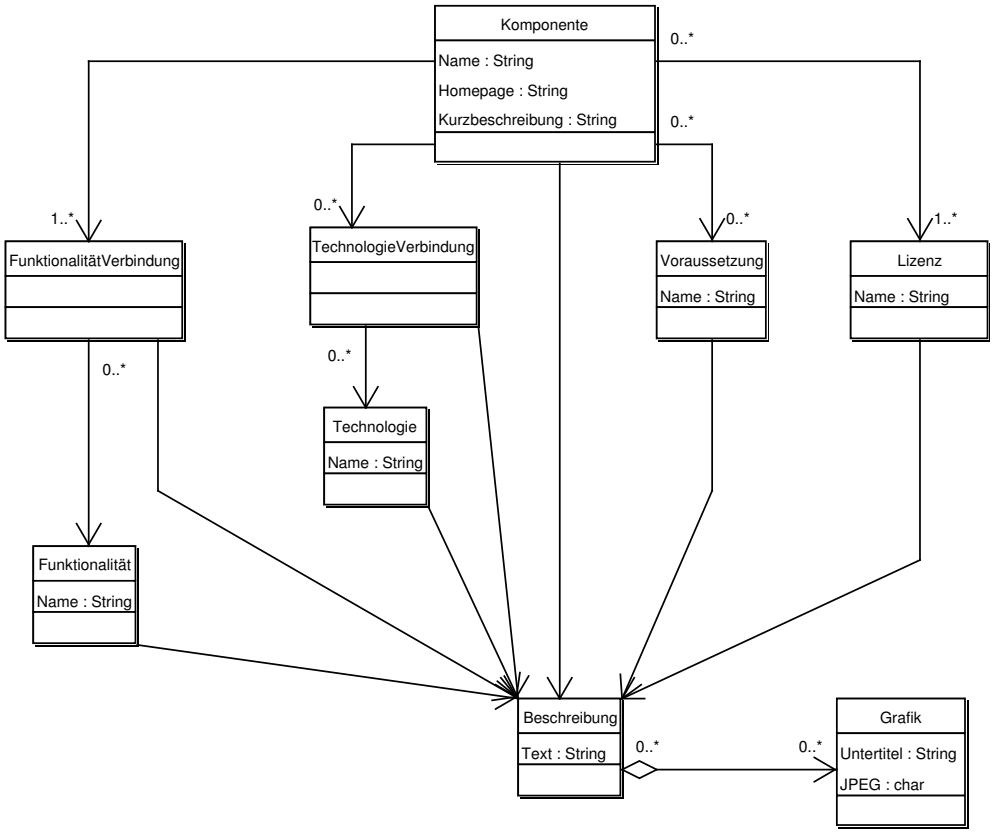

Abbildung 5.2: *Das Klassendiagramm der Datenschicht.*

alle Entitäten auch die Möglichkeit, graphische Elemente in der Beschreibung zu verwenden.

Das UML-Klassendiagramm in Abbildung 5.2 stellt nochmals alle Entitäten und die Beziehungen zwischen ihnen dar. Die Attribute, welche die Referenzen zu anderen Entitäten enthalten, sind in diesem Diagramm nicht aufgelistet. Für die Umsetzung der Entitäten in Klassen wurden die Attribute durch get- und set-Methoden ersetzt. Weiterhin wurden Methoden zum Erstellen und Abrufen der Referenzen in jeder Klasse eingeführt.

5.4 Anwendungslogik

Die in Abschnitt 4.6 vorgestellte Auswahl von Kriterien anhand relevanter Begriffe erfordert für die jeweilig durch den Entwickler ausgewählten Kriterien nicht nur eine Ausgabe an bisherigen Ergebnissen, sondern auch die Information, welches die verbleibenden relevanten Begriffe sind. Der Suchalgorithmus muss deswegen

intensiv auf der Datenbasis arbeiten. Dies und die Zielsetzung, alle Funktionalitäten, die unabhängig von einer konkreten Benutzerschnittstelle genutzt werden können, in eine eigene Schicht zu platzieren, sprechen für die Umsetzung des Suchalgorithmus in einer Schicht neben der Schicht der Entity Beans und auch außerhalb der Präsentationsschicht. Innerhalb von J2EE bieten sich dafür die sogenannten Session Beans an, die auf dem Applikationsserver ausgeführt werden. Session Beans können entweder zustandsbehaftet (*Stateful Session Beans*) oder zustandslos (*Stateless Session Beans*) sein (siehe Abschnitt 3.1.1). Im ersten Unterpunkt dieses Abschnittes werden die verschiedenen Möglichkeiten, den Suchalgorithmus zu entwerfen, vorgestellt, bevor im zweiten Unterpunkt die Umsetzung der gewählten Alternative als Stateful Session Beans beschrieben wird.

5.4.1 Alternativen

Um eine Liste aller verbliebenen Komponenten erstellen zu können, muss der Suchalgorithmus alle Komponenten als Liste liefern, für welche die bisher ausgewählten Kriterien zutreffend sind. Die Kriterien können dabei sowohl aus Regeln der Art „Attribut x muss Wert y enthalten“ als auch aus Regeln der Art „Attribut x darf den Wert y *nicht* enthalten“ bestehen. Ebenso sollen alle für die verbliebenen Komponenten noch relevanten Begriffe je Attribut ausgegeben werden. Um diese „relevanten Begriffe“ zu finden, müssen alle Begriffe aufgelistet werden, die zwei Bedingungen erfüllen:

1. Ein Begriff muss unter allen verbliebenen Komponenten mindestens einmal vorkommen.

2. Dieser Begriff darf aber nicht bei *allen* verbliebenen Komponenten vorkommen, d.h. es muss mindestens eine Komponente existieren, der dieser Begriff nicht zugeordnet ist.

Für jeden dieser Begriffe lässt sich die Menge der verbliebenen Komponenten jeweils in zwei Untermengen A und B aufteilen. Die Untermenge A enthält alle Komponenten, die den Begriff im betrachteten Attribut nicht aufweisen. Diese Untermenge A kann nicht leer sein, da nur Begriffe ausgewählt wurden, die nicht in *allen* verbliebenen Komponenten enthalten sind (s.o.). Die Untermenge B, die alle Komponenten enthält, die den Begriff beim betrachteten Attribut aufweisen, kann ebenfalls nicht leer sein, da nur Begriffe ausgewählt wurden, die in mindestens einer Komponente vorkommen. Durch das Hinzufügen genau eines weiteren Kriteriums („Begriff einschließen“ oder „Begriff ausschließen“) zu den bisherigen Kriterien wird also eine der beiden Untermengen an Komponenten als neue Menge an verbleibenden Komponenten ausgewählt und die andere ausgeschlossen.

Da beide Untermengen nicht leer sind, nimmt die Anzahl der verbliebenen Komponenten stetig ab, die Menge der verbliebenen Komponenten wird aber auch niemals leer, weil keine der beiden Untermengen leer ist. Stattdessen können bei der nachfolgenden Suche nach relevanten Begriffen irgendwann keine relevanten Begriffe zu der Menge der verbliebenen Komponenten mehr gefunden werden. Dies ist dann der Fall, wenn alle verbliebenen Komponenten die exakt gleichen Ausprägungen bei den Attributen aufweisen. Enthält die Menge der verbliebenen Komponenten nur noch eine Komponente, so ist dies sehr einfach einzusehen.

Für die Umsetzung des Suchalgorithmus wurden die im Folgenden angegebenen drei Alternativen untersucht. Um eine Aussage über die Komplexität des jeweiligen Ansatzes machen zu können, wurde angenommen, dass ein Aufruf an die Datenbank in konstanter Zeit unabhängig von der Komplexität der Abfrage abläuft. Dies ist in Realität sicherlich nicht der Fall. Jedoch ist anzunehmen, dass die Menge an Daten in der Datenbasis sehr groß sein muss, damit die Ausführung der Abfrage durch das DBMS signifikant mehr Zeit benötigt als die Übertragung der Abfrage zum und des Ergebnisses vom entfernten DBMS. Dieser Umfang der Datenbasis würde erst nach etlichen Iterationen der Entwicklung des ICApps-System eventuell erreicht. Sollte sich dann ein anderer als der gewählt Algorithmus als vorteilhafter und realisierbar erweisen, so ist dieser durch die Kapselung der Funktionalität in einer *Session Bean* einfach umsetzbar.

Alternative 1: Iterieren über die Komponenten in Java

1. Die Liste K wird mit allen dem System bekannten Komponenten gefüllt.
2. Die Liste R der Kriterien wird leer angelegt.
3. Eine Liste X wird leer angelegt. Für jedes Attribut a der Komponentenbeschreibung wird:
   (a) Eine Liste L_a angelegt.
   (b) Für jeden möglichen Begriff des Attributs a wird
      i. ein Zählerobjekt mit dem Namen des Begriffs und dem Zählerstand Null der Liste L_a hinzugefügt.
   (c) Die Liste L_a als Element der Liste X hinzugefügt.
4. Für jede Komponente aus der Liste K wird
   (a) für jedes ihrer Attribute a jeder Begriff, der unter dem Attribut a vorkommt, betrachtet und der Zähler in der Liste L_a für den jeweiligen Begriff um eins erhöht.

5. Alle Begriffe, deren Zählerstand größer als Null und kleiner als die Anzahl der Komponenten in K ist, werden dem Benutzer unter dem entsprechenden Attribut zur Auswahl angeboten.

6. Je nach Benutzerauswahl wird der gewählte Begriff der Liste der Kriterien R als „ausschließen“ oder „einschließen“ Kriterium hinzugefügt. Alternativ kann der Benutzer auch ein Kriterium aus der Liste R entfernt haben.

7. Die Liste K der Komponenten wird anhand der Kriterien aus R neu aus der Datenbank aufgebaut.

8. Die Zählerstände aller Zählerobjekte, die in den Listen L_a in der Liste X enthalten sind, werden auf Null gesetzt. Es wird zu Schritt 4 zurückgekehrt.

Für alle Schritte außer Schritt 7 können die benötigten Daten von den Entity Beans direkt geholt werden. Die durch CMP im Hintergrund erzeugten Abfragen an die Datenbank bestehen aus einfachen SELECT-Befehlen. Nur für Schritt 7 muss ein spezieller SQL-Befehl benutzt werden, der nicht in der grundsätzlichen Schnittstelle einer Entity Bean enthalten ist. Dieser SQL-Befehl besteht aus einem SELECT-Befehl, der als Ergebnis Komponenten liefert, die alle Kriterien in der WHERE Klausel erfüllen.

Der Vorteil dieses Algorithmus besteht darin, dass keine komplexen SQL-Befehle benötigt werden. Als Nachteil muss jedoch gesehen werden, dass der Schritt 4 eine hohe Komplexität aufweist. Für die obere Grenze der Komplexität S_{max} kann angegeben werden:

S_{max} = (Anzahl aller Komponenten) * (Anzahl aller Begriffe über alle Attribute)

Alternative 2: Iterieren über die Attribute Um eine bessere Leistung zu erreichen, unterscheidet sich dieser Algorithmus vom vorangegangenen Algorithmus in Schritt 4:

4. Für jede Liste L_a aus X wird

   (a) für alle Zählerobjekte aus L_a in der Datenbank gleichzeitig in einem Aufruf abgefragt, wie oft jeder Begriff in den Komponenten aus K vorkommt.

Für Schritt 4 wird nun auch ein SQL-Befehl benötigt, der für jeden Begriff, der bei einem Attribut vorkommen kann, sowohl den Begriff als auch die Anzahl seines Vorkommens in den angegebenen Komponenten als Ergebnis liefert. Für die drei verbliebenen Komponenten mit den Primärschlüsseln 22, 27 und 29 kann der Befehl wie folgt aussehen:

```
SELECT technologyid, COUNT(technologyid) FROM techlink WHERE
componentid IN ('22', '27', '29') GROUP BY technologyid
```

Mit dieser Datenbankabfrage reduziert sich die Komplexität S_{max} von Schritt 4 auf:

$S_{max} =$ (Anzahl der Attribute einer Komponentenbeschreibung)

Der oben dargestellte SQL-Befehl kann aber momentan nicht spezifikationskonform zur J2EE-Spezifikation mit Entity Beans umgesetzt werden. In der Spezifikation der EJBs besteht ab Version 2.0 die Möglichkeit, eine Abfragesprache namens EJB-QL zu benutzen, die sehr stark an SQL angelehnt ist. Allerdings ist EJB-QL im Vergleich zu SQL noch wesentlich weniger mächtig und kennt in der Version EJB 2.0 noch keine COUNT()-Funktion. Außerdem erlaubt EJB-QL als Ergebnistyp nur EJB-Objekte oder ein einzelnes Feld eines EJB-Objekts. Die Rückgabe eines EJB-Objekts zusammen mit einem weiteren Wert ist also bisher nicht vorgesehen. (vgl. [MIC01])

Momentan wird die EJB-Spezifikation innerhalb des „Java Specification Requests 153“ auf Version 2.1 erweitert (vgl. [JSR153]). Diese Version der Spezifikation wurde aber im Erstellungszeitraum dieser Arbeit noch nicht endgültig verabschiedet. Sie wird die Verwendung von Funktionen wie COUNT() für Ergebnisse erlauben, wird es jedoch noch nicht ermöglichen, EJB-Objekte gemeinsam mit den Werten einer COUNT()-Operation als Ergebnis zu liefern.

Der verwendete Applikationsserver JBoss bietet verschiedene Erweiterungen der Abfragesprache an, die nicht über die J2EE-Spezifikation abgedeckt sind (vgl. [SUNST]). Die Erweiterung JBossQL fügt EJB-QL die Konstrukte ORDER BY, IN und LIKE hinzu, die die selbe Semantik wie in SQL besitzen. Die zweite Erweiterung „DynamicQL“ erlaubt die Konstruktion einer Abfrage aus JBossQL zur Laufzeit. Beide Erweiterungen ermöglichen leider nicht die Verwendung von COUNT(). Eine dritte Erweiterung, „DeclaredSQL“, ermöglicht es, die verschiedenen Teile des SQL-Befehls relativ frei zu definieren. Jedoch erlaubt auch DeclaredSQL bisher nicht die Verwendung von COUNT() als Ergebnis einer Abfrage.

Auch die Erweiterungen der *Entity Beans* um „*home methods*“ und „*select methods*“ in der Version 2.0 der EJB-Spezifikation bieten keine weiteren Möglichkeiten, den obigen SQL-Befehl spezifikationskonform umzusetzen. *Select methods* sind zur Definition des ausgeführten SELECT-Befehls ebenfalls auf EJB-QL beschränkt und können deshalb nicht mehrere Ergebnisse zurückliefern. *Home methods* sind für Funktionalitäten gedacht, die unabhängig von einer konkreten Entität angeboten werden und können eigene Ergebnisklassen definieren. Durch diese Ergebnisklassen können mehrere Ergebnisse zurückgeliefert werden, jedoch besitzen auch *home methods* keine weiteren als die bereits vorgestellten Möglichkeiten, um auf Daten in der Datenbank zuzugreifen und können somit die benötigten Daten auch nicht innerhalb eines SQL-Befehls ermitteln.

Alternative 3: Iterieren über die Begriffe Um die Komplexität des SQL-Befehls in Schritt 4 zu reduzieren, wird dieser nochmals verändert:

4. Für jede Liste L_a aus X wird

   (a) für jedes Zählerobjekt aus L_a in der Datenbank abgefragt, wie oft der zugehörige Begriff in den Komponenten aus K vorkommt.

Der für Schritt 4 nötige SQL-Befehl muss nur noch ein Ergebnis für einen Begriff zurückliefern. Er vereinfacht sich damit beispielsweise auf:

```
SELECT COUNT(technologyid) FROM techlink WHERE componentid
IN ('22', '27', '29') AND technologyid = '15'
```

Durch den vereinfachten SQL-Befehl ergibt sich für Schritt 4 eine Komplexität S_{max} von:

$S_{max} =$ (Anzahl aller Begriffe über alle Attribute)

Diese Komplexität liegt zwischen den Alternativen 1 und 2. Jedoch ist der SQL-Befehl einfach genug, dass dieser in Entity Beans umgesetzt werden kann, sobald die COUNT() Funktion nach der Verabschiedung der EJB-Spezifikation 2.1 zur Verfügung steht.

5.4.2 Umsetzung

Auf Grund der bisherigen technischen Limitierungen von EJB-QL wurde die momentan einzig technisch mögliche Alternative 1 zur Umsetzung des beschriebenen Suchalgorithmus gewählt. Da in das ICApps-System die Daten über die Komponenten sukzessive eingegeben werden, ist die Datenbasis des Systems zu Beginn noch klein. Mit einer kleinen Datenbasis können alle Daten der Entity Beans nach der ersten Abfrage durch den Applikationsserver im Cache gehalten werden. Die den eigentlichen Aufwand verursachenden Anfragen an das Datenbankmanagementsystem können also mit einer kleinen Datenbasis fast vollständig vermieden werden. Es ist deshalb nicht zu befürchten, dass die Alternative 1 (trotz der höheren Komplexität) zu Beginn eine zu geringe Performance aufweist. Sollte die Datenbasis jedoch erheblich gewachsen und die COUNT()-Funktion EJB-QL offiziell hinzugefügt worden sein, so empfiehlt es sich, die entsprechenden Teile der Umsetzung an das Vorgehen der Alternative 3 oder sogar der Alternative 2 anzupassen.

Wie in Abschnitt 5.2 erläutert, soll die Suche in Session Beans umgesetzt werden, um zentrale Funktionalitäten des Systems nicht mit Systemteilen zur Präsentation von Daten zu vermengen und den Suchalgorithmus möglichst nah an den Daten ablaufen lassen zu können. Als Session Beans wurden *Stateful Session Beans* ausgewählt, da sowohl die bisher durch den Benutzer ausgewählten Kriterien als auch die verbleibenden Komponenten einen Zustand des Suchvorgangs darstellen, der die Suchergebnisse beeinflusst. Es handelt sich somit bei der Suche um einen zustandsbehafteten Dienst. Rein technisch könnte ein Teil des Suchzustandes weiter zum Klienten zum Beispiel in *Cookies* oder nicht sichtbaren Eingabefeldern verlagert werden. Jedoch basiert der iterative Suchalgorithmus stark auf der Integrität der Daten. Bei der Speicherung eines Teil des Suchzustandes beim Klienten könnten diese Daten absichtlich oder unabsichtlich durch den Klienten manipuliert werden und der Algorithmus als Folge mit einem inkonsistenten Zustand konfrontiert werden.

Um die Komplexität der Suche mit relevanten Attributen zu reduzieren, wurde die Funktionalität auf zwei Session Beans verteilt, die eng zusammenarbeiten:

Session Bean	Funktionalität	Methoden
SearchBean	berechnet auf Grund der verbliebenen Komponenten die relevanten, eingeschlossenen und ausgeschlossenen Attributwerte	setActualComponents getExcluded[Attribute] getIncluded[Attribute] getRelevant[Attribute]
CriteriaSearchBean	verwaltet die ausgewählten Kriterien und trennt eingeschlossene Attributwerte nach eingeschlossenen und ausgewählten Attributwerten auf, ebenso werden ausgeschlossene Attributwerte nach abgewählten und ausgeschlossenen Attributwerten getrennt	getActualComponents getExcluded[Attribute] getIncluded[Attribute] getRelevant[Attribute] getSelected[Attribute] getDeselected[Attribute] select[Attribute] deselect[Attribute] remove[Attribute]

Pro Sitzung wird vom Klienten (hier ist dies die Steuerungsschicht) ein CriteriaSearchBean erzeugt (vgl. Abb. 5.3, Nr. 1). Dieses wiederum erzeugt selbständig ein mit ihm kooperiendes SearchBean für diese Sitzung (vgl. Abb. 5.3, Nr. 2), welchem es initial alle dem System bekannten Komponenten als verbliebene Komponenten übergibt (vgl. Abb. 5.3, Nr. 3).

Der Klient übergibt CriteriaSearchBean die vom Benutzer ausgewählten Kriterien (vgl. Abb. 5.3, Nr. 4). CriteriaSearchBean berechnet darauf hin die Menge der verbliebenen Komponenten neu und teilt diese SearchBean mit (vgl. Abb. 5.3, Nr. 5).

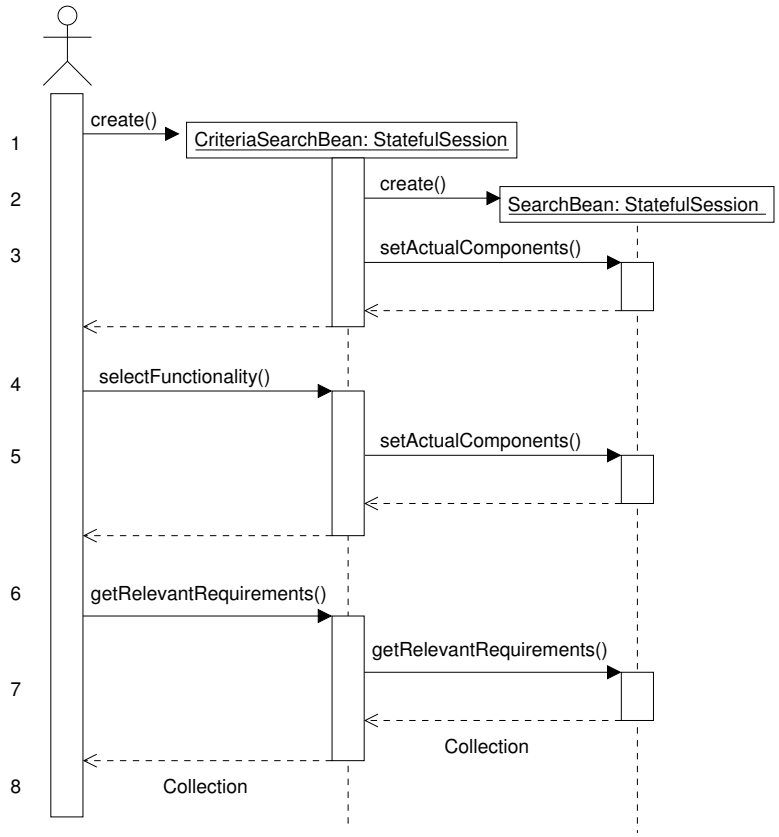

Abbildung 5.3: *Die Erzeugung der Session Beans für die kriterienbasierte Suche und Kooperation zwischen diesen bei Aufrufen durch den Klienten.*

Bei Bedarf fordert die Präsentationsschicht alle ausgewählten, abgewählten, eingeschlossenen, ausgeschlossenen und relevanten Begriffe von CriteriaSearchBean an (vgl. Abb. 5.3, Nr. 6).

Die angeforderten Mengen seien repräsentiert durch:

$C^a_{ausgewaehlt}$	für das Attribut a durch den Benutzer ausgewählte Begriffe
$C^a_{abgewaehlt}$	für das Attribut a durch den Benutzer abgewählte Begriffe
$C^a_{eingeschlossen}$	für das Attribut a implizit eingeschlossene Begriffe
$C^a_{ausgeschlossen}$	für das Attribut a implizit ausgeschlossene Begriffe
$C^a_{relevant}$	für das Attribut a relevante Begriffe für die Benutzerauswahl

CriteriaSearchBean speichert intern die folgenden Mengen:

$K^a_{ausgewaehlt}$	für das Attribut a alle durch den Benutzer ausgewählte Begriffe
$K^a_{abgewaehlt}$	für das Attribut a alle durch den Benutzer abgewählte Begriffe
R	die Menge der nach den Kriterien $K^a_{ausgewaehlt}$ und $K^a_{abgewaehlt}$ verbliebenen Komponenten

CriteriaSearchBean erhält von SearchBean die folgenden Mengen zur Berechnung (vgl. Abb. 5.3, Nr. 7):

$S^{a,R}_{eingeschlossen}$	für Attribut a bei verbliebenen Komponenten R implizit eingeschlossenen Begriffe
$S^{a,R}_{ausgeschlossen}$	für Attribut a bei verbliebenen Komponenten R implizit ausgeschlossenen Begriffe
$S^{a,R}_{relevant}$	für Attribut a bei verbliebenen Komponenten R relevanten Begriffe für die Benutzerauswahl

SearchBean unterscheidet bei den Mengen an ein- und ausgeschlossenen Begriffen also nicht, ob diese Begriffe direkt durch eine Benutzerauswahl oder implizit durch andere Kriterien ein- oder ausgeschlossen wurden. Die geforderten Ergebnismengen berechnet CriteriaSearchBean nun nach folgendem Schema:

$C^{a}_{relevant} = S^{a,R}_{relevant}$

$C^{a}_{ausgewaehlt} = K^{a}_{ausgewaehlt}$

$C^{a}_{abgewaehlt} = K^{a}_{abgewaehlt}$

$C^{a}_{eingeschlossen} = S^{a,R}_{eingeschlossen} \setminus K^{a}_{ausgewaehlt}$

$C^{a}_{ausgeschlossen} = S^{a,R}_{ausgeschlossen} \setminus K^{a}_{abgewaehlt}$

Diese Ergebnismengen werden nun von CriteriaSearchBean an den aufrufenden Klienten zurückgegeben (vgl. Abb. 5.3, Nr. 8).

5.5 Präsentation und Ablaufsteuerung

Die Präsentationsschicht hat sowohl die Aufgabe, die im System vorhandenen Daten und Funktionalitäten dem Benutzer zugänglich zu machen, als auch den Ablauf der interaktiven Dialoge mit dem Benutzer zu steuern. Sie stellt also die Benutzerschnittstelle des Systems dar und ist damit der Teil des ICApps-Systems, der die Wahrnehmung des Benutzer bezüglich der Leistungsfähigkeit und Einfachheit der Bedienung des ICApps-Systems am stärksten bestimmt.

5.5.1 Alternativen

Nach der Speicherung der Daten in einer zentralen Datenbank und der Umsetzung der zentralen Funktionalitäten auf einem Applikationsserver kann die Aufarbeitung der Daten für die Präsentation entweder zum größten Teil ebenfalls auf der Seite des Servers stattfinden, oder erst auf der Seite des Klienten erledigt werden. Mit dem heutigen Stand der Technik bestehen also die beiden folgenden Alternativen:

fat client: Die Benutzerschnittstelle wird durch ein Programm auf der Klientenseite erzeugt. Die Steuerung der interaktiven Vorgänge erfolgt ebenso durch dieses Programm. Entgegen der klassischen „fat client"-Variante werden aber die zentral zur Verfügung gestellten Funktionalitäten vom Applikationsserver genutzt und nicht lokal ausgeführt.

thin client: Der Applikationsserver erzeugt die Benutzerschnittstelle im HTML-Format, welche per HTTP an einen Browser auf Klientenseite übertragen und dargestellt wird. Die Steuerung von interaktiven Abläufen wird hauptsächlich von der Serverseite gesteuert.

Die folgende Tabelle stellt Vor- und Nachteile der beiden Alternativen anhand verschiedener Aspekte gegenüber:

	fat client	**thin client**
Installationsaufwand	Das Programm oder eine Ablaufumgebung muss passend für die Plattform installiert werden.	Der Browser ist bereits installiert oder muss installiert werden.
Anpassbarkeit	Bei einer Änderung der zugrundeliegenden Datenstrukturen oder einer Erweiterung der Programmfunktionalität kann die alte Version nur eingeschränkt oder überhaupt nicht mehr verwendet werden, bis eine aktualisierte Version installiert wurde.	Zukünftige Programmänderungen können zentral auf dem Server durchgeführt werden und sind somit sofort für alle Anwender verfügbar.
Wartbarkeit	Zur Identifikation und Korrektur von Programmfehlern müssen diese vom Benutzer des Systems zurückgemeldet werden und der Kontext des Auftretens sowie evtl. Fehlermeldungen müssen übermittelt werden.	Fehlermeldungen sind direkt auf dem Server ersichtlich und der Kontext, in dem ein Fehler aufgetreten ist, kann bei geeigneter Konfiguration direkt auf dem Server rekonstruiert werden.

	fat client	**thin client**
Ablaufsteuerung	Das Programm kann optimal und direkt auf die Benutzereingaben reagieren und bei interaktiven Abläufen den Benutzer klar auf vorgesehene Optionen einschränken.	Über JavaScript kann teilweise direkt auf Benutzereingaben reagiert werden. Da JavaScript jedoch nicht als gegeben vorausgesetzt werden kann, sollten darin auch keine wichtigen Benutzerschnittstellenfunktionen umgesetzt sein. Ansonsten kann der Ablauf des Programms immer nur zu den Zeitpunkten verändert werden, zu denen der Benutzer durch Auswahl eines Links oder eines Buttons eine neue HTTP-Anfrage des Browsers an den Server veranlasst.
Sitzungsverwaltung	Das Programm selbst stellt eine zustandsbehaftete Sitzung dar. Es muss deshalb kein weiterer Aufwand für die Sitzungsverwaltung vorgesehen werden.	Das HTTP-Protokoll selbst sieht keinen die Anfragen übergreifenden Zustand vor. Eine Sitzungsverwaltung muss deshalb getrennt realisiert werden. Bei der Verwendung der JSP-Technologie ist eine rudimentäre Speicherung von Sitzungsdaten und eine Sitzungszuordnung über Cookies enthalten.
Plattformabhängigkeit	Bei der Verwendung einer Laufzeitumgebung, wie dem Java 2 Runtime Environment, kann das Programm auf allen Plattformen betrieben werden, für welche die Umgebung portiert wurde. Dies sollte für die gebräuchlichsten Plattformen der Fall sein.	Da bisherige Recherchen auch mit einem Browser im Internet durchgeführt wurden, kann dieser als vorhanden vorausgesetzt werden. Es bestehen keine weiteren Einschränkungen.

Wie in der vorhergehenden Tabelle dargestellt, hat die „fat client“-Alternative gegenüber der „thin client“-Alternative Vorteile bei der Ablaufsteuerung und Sitzungsverwaltung. Der Vorteil bei der Sitzungsverwaltung wird aber dadurch vermindert, dass auf der Serverseite auf jeden Fall eine Sitzungsverwaltung stattfinden muss, da auch auf die in Stateful Session Beans realisierten zentralen Funktionalitäten innerhalb einer Sitzung zugegriffen wird. Weiterhin lässt sich die Sitzungsverwaltung für die Präsentationsschicht bei der „thin client“-Alternative

durch die Wahl der entsprechenden Technologie erheblich vereinfachen. Die Vorteile bei der Installation und der Anpassbarkeit des Programms sind Punkte, die stark für die Verwendung der „thin client"-Alternative sprechen. Das ICApps-System soll eine möglichst geringe Benutzungshemmschwelle für potentielle Anwender haben, d.h. die möglichen Hinderungsgründe, die einen potentiellen Anwender von der Benutzung des Systems abhalten könnten, sollen minimiert werden. Bei der „thin client"-Alternative kann die Installation als Hemmschwelle vollständig wegfallen. Da das ICApps-System zuerst als Prototyp umgesetzt und dann evolutionär weiterentwickelt werden soll, gewinnt der Punkt der deutlich einfacheren Anpassbarkeit des Programmes ein besonders starkes Gewicht. Auf Grund dieser beiden wichtigen Punkte wurde die Präsentationsschicht in Form der „thin client"-Alternative umgesetzt.

In der „thin client"-Alternative werden serverseitig die HTML-Seiten durch die Präsentationsschicht erzeugt und der Ablauf des Systems abhängig von den Benutzereingaben gesteuert. Zur Umsetzung dieser Funktionalitäten können verschiedene Technologien alternativ eingesetzt werden. Die folgenden Alternativen wurden dabei betrachtet:

Servlet und JSPs: Ein zentrales Servlet erhält die vom Browser gesandten Anfragen, reicht Daten weiter und steuert den Programmablauf. Das Ergebnis einer Anfrage wird über eine *JavaServer Pages*-Seite (JSP) an den Browser gesandt. Diese Architektur wurde durch die Sun-Spezifikationen auch unter dem Namen „Model 2" bekannt. (vgl. [SIN02])

MVC-Framework: Es existieren verschiedene Frameworks, welche die *Controller*-Funktionalität des *Model-View-Controller*-Prinzips (MVC) übernehmen können (vgl. [KRA88]). Ein Servlet des Frameworks nimmt dabei alle Anfragen des Browsers entgegen. Der Steuerungsablauf des Programms kann dann zum Beispiel in einer XML-Datei oder in Objekten, die eine spezielle Schnittstelle implementieren, festgelegt werden. Häufig wird zusätzlich eine Templating-Technologie verwendet, um die Gestaltung der HTML-Seiten (*View*) von der Erzeugung der dynamischen Inhalte (aus dem *Model*) zu trennen.

Tapestry: Mit dem Tapestry-Framework (vgl. [TAPES]) wird die Benutzerschnittstelle vollständig aus Präsentationsobjekten aufgebaut, wobei jedes Präsentationsobjekt selbst wiederum klar zwischen Präsentation und der nötigen Logik für die Datenverarbeitung und Ablaufsteuerung trennt. Entgegen einem MVC-Framework muss das Programm nicht Browseranfragen entgegennehmen und diese dekodieren, sondern das entsprechende Präsentationsobjekt wird über die ausgelöste Aktion direkt durch einen aktionsspezifischen Methodenaufruf informiert.

Die in dieser Reihenfolge vorgestellten Alternativen weisen jeweils Vorteile gegenüber der vorhergehenden Alternative auf. Die Alternative „Servlet und JSP“ greift den MVC-Ansatz schon grundsätzlich auf, wobei der Controller im Servlet und die Views in den JSPs realisiert werden. Hier muss der Entwickler aber noch selbst die vom Browser empfangenen Anfragen dekodieren. Ein MVC-Framework nimmt dem Entwickler diese Aufgabe ab und bietet zum Beispiel über eine XML-Datei eine wesentlich entwicklungs- und wartungsfreundlichere Möglichkeit, den Ablauf des Programms anzupassen. In ein MVC-Framework können Templating-Technologien im Allgemeinen einfach integriert werden, um die Gestaltung der HTML-Benutzerschnittstelle von der Logik, welche die dynamischen Daten holt und aufbereitet, zu trennen. Der Ansatz des Tapestry-Frameworks verbindet die beiden Zielsetzungen, den Entwickler von der Anfragedekodierung zu entbinden und die Gestaltung von der Datenaufbereitung zu trennen, in einem Framework. Dies vermindert den nötigen Einarbeitungsaufwand von zwei (MVC & Templating) auf ein Framework (Tapestry). Zusätzlich wird in Tapestry die Benutzerschnittstelle aus einzelnen Präsentationskomponenten zusammengestellt. Dazu können eine Reihe vordefinierter oder selbst erstellter Präsentationskomponenten verwendet werden, die jeweils selbst aus derartigen Komponenten zusammengestellt werden können. Dies erlaubt einen sehr hohen Grad an Wiederverwendung in der Entwicklung der Benutzerschnittstelle, wenn häufig die selben Daten an verschiedenen Stellen in der Benutzerschnittstelle dargestellt werden müssen. Dies ist zum Beispiel für das gemeinsame Beschreibungsobjekt aller Entitäten im ICApps-Systems der Fall. Weiterhin kann der Entwickler die Sitzungsverwaltung und die Verwaltung des Kontextes einer Sitzung vollständig dem Tapestry-Framework überlassen. Aus den dargestellten Gründen wurde die Präsentationsschicht mit dem Tapestry-Framework umgesetzt.

5.5.2 Tapestry

Wie im vorangegangenen Abschnitt dargestellt, stellt Tapestry eine Alternative zu JSPs und Servlets für die Realisierung von HTML-Benutzerschnittstellen dar. Beide Technologien schließen sich jedoch nicht gegenseitig aus, sondern können bei Bedarf kombiniert werden. Tapestry selbst wird in die J2EE-Umgebung durch ein mitgeliefertes Servlet installiert, welches alle Anfragen entgegennimmt und sogenannte „Engines“ für die einzelnen Sitzungen der Benutzer erzeugt. Eine Engine verwaltet den Status einer Benutzersitzung mit ihrem Kontext und liefert bei HTTP-Anfragen das Ergebnis eines Präsentationsobjektes als HTML-Seite an den Browser des Benutzers. (vgl. [SHI02])

Alle HTML-Seiten der Benutzerschnittstelle werden als Präsentationsobjekte in Tapestry angelegt. Jedes Präsentationsobjekt wird durch drei Dateien definiert:

Darstellung: In dieser Datei wird in reinem HTML-Code der graphische Aufbau des Objekts definiert. An den Stellen, die dynamische Inhalte oder interaktive Steuerungselemente wie Eingabe- und Bestätigungsfelder beinhalten sollen, wird den entsprechenden HTML-Tags ein weiteres Attribut namens „jwcid“ hinzugefügt. Der Name „jwcid“ steht hierbei für „Java Web Component ID“. Der Wert des Attributs ist ein eindeutiger Bezeichner (ID) innerhalb dieser Datei.
Beispiel einer Tabellenzeile in HTML, welche den Komponentennamen darstellt:

```
<tr><td>
<b>Name:</b>
</td><td>
<span jwcid="CompName">Name wird eingefügt</span>
</td></tr>
```

Datenverabeitung und Steuerung: In einer normalen Java-Klasse werden verschiedene Methoden implementiert, welche die Daten für die dynamischen Inhalte des Objekt liefern. Ebenso werden Methoden implementiert, die durch den Benutzer eingegebene Daten aufnehmen können, oder die bei durch den Benutzer ausgelösten Aktionen die Daten entsprechende verändern und den Verlauf des Programms steuern.
Beispiel einer Programmmethode in Java, die den Namen der aktuellen Komponente liefert:

```
public String getKompName(){
return komponente.getName();
}
```

Deskriptor: Diese dritte Datei im XML-Format dient der Verknüpfung der Bezeichner aus der Darstellungsdatei mit den Methoden der Java-Klasse. Für jeden Bezeichner lässt sich festlegen, aus welcher Methode der Java-Klasse er die dynamischen Daten erhält oder die Benutzereingaben ablegt. Alternativ kann ein Bezeichner auch auf eine weitere Präsentationskomponente verweisen, die wiederum wie diese Präsentationskomponente definiert ist. Hierbei kann auf eigene oder auf im Tapestry-Framework bereits vorhandene Präsentationskomponenten verwiesen werden. Für alle häufig wiederkehrenden interaktiven Elemente werden zum Teil recht komplexe Präsentationskomponenten durch das Tapestry-Framework schon bereit gestellt.
Beispiel eines XML-Deskriptorausschnitts, der die Verbindungen zwischen den beiden vorangegangen Beispielen herstellt:

```
<component id="CompName" type="Insert">
<binding name="value" expression="kompName"/>
</component>
```

Die Trennung der Darstellung von der Datenverarbeitung und Steuerung ermöglicht es, die HTML-Datei in einem beliebigen Designwerkzeug für Webseiten zu erstellen und auch später zu verändern, ohne dabei die Funktionsfähigkeit im Framework zu zerstören. Somit können Webdesigner und Programmierer sich jeweils vollständig in ihrer Domäne bewegen (HTML oder Java), ohne dabei Restriktionen der jeweils anderen Seite zu unterliegen.

Die Möglichkeit, im Deskriptor auf andere Präsentationskomponenten verweisen zu können, ermöglicht die Verwendung eines geschachtelten Designs, das aus einzelnen, einfachen Präsentationskomponenten aufgebaut wird. Dieser Ansatz erweist sich als sehr mächtig und erweiterungsfreundlich. Selbst sehr komplexe Benutzerschnittstellen lassen sich in kleine, überschaubare und gut strukturierte Einzelteile zerlegen, die einfach erweitert und insbesondere gut wiederverwendet werden können.

Durch die Erstellung von eigenen Präsentationskomponenten, welche in die Seiten der Benutzerschnittstelle eingebunden wurden, konnte ein hoher Grad von Wiederverwendung schon innerhalb der Arbeit selbst erreicht werden. Es wurden die folgenden Präsentationskomponenten erstellt:

displayDescription: Als Parameter wird eine Description Entity Bean erwartet. Der Beschreibungstext dieser *Description EJB* wird als Text dargestellt. Für die von der *Description EJB* referenzierten Bilder in Form von *Image EJBs* wird jeweils eine URL ausgegeben, die den Browser veranlassen wird, das jeweilige Bild nachzuladen und anzuzeigen (siehe Abb. 5.4).

editDescription: Für das als Parameter übergebene *Description Entity Bean* wird ein HTML-Formular erzeugt, das die Eingabe oder Pflege der Beschreibung erlaubt. Weiterhin können die bestehenden Grafiken verwaltet oder neue Grafiken hinzugefügt werden.

bullet: Diese Präsentationskomponente stellt einen Punkt dar, wie er in Auflistungen verwendet wird. Dieser Punkt kann aktiviert oder deaktiviert sein. Verwendung im ICApps-System findet dieses Objekt bei der Auflistung von Komponenten, Funktionalitäten, Technologien etc., um darzustellen, ob das entsprechenden Element den in der Volltextsuche eingegebenen Text enthält.

verboseURL: Hiermit kann eine URL als Text ausgeben werden und gleichzeitig ein HTML-Link auf diese Adresse sein, wie dies zum Beispiel für alle Homepage-Adressen der Komponenten im ICApps-System realisiert ist.

selector: Der selector ist eine komplexe Präsentationskomponente, die für eine bestimmte Attributart alle ausgewählten und ausgeschlossenen Attributwerte anzeigt und eine Benutzerschnittstelle aufbaut, mit welcher der Benutzer weitere Kriterien bezüglich dieser Attributart auswählen kann. Der

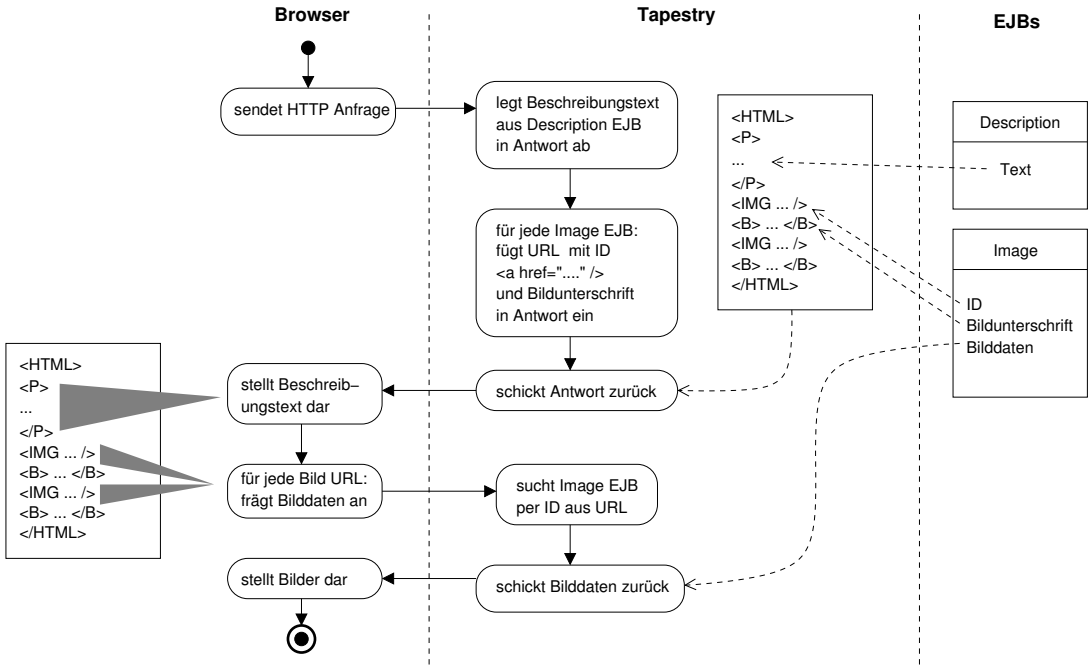

Abbildung 5.4: *Das Aktivitätsdiagramm der Interaktion zwischen Browser und Tapestry-Komponenten bei der Anzeige einer Beschreibung (Description EJB).*

generischen selector-Implementierung wird ein Delegationsobjekt übergeben. Dieses Delegationsobjekt implementiert eine definierte Schnittstelle, über die das selector-Objekt die jeweilig benötigten Daten erhält und die vom Benutzer getroffenen Aktionen weitergibt. Das Delegationsobjekt ist dann für die Weiterleitung der Datenanfragen an die für die Attributart richtigen Methoden der Session Bean für die Suche zuständig. Ebenso werden die Benutzeraktionen auf diesem Weg an die Such-Session Bean weitergeleitet.

Weiterhin wurden folgende Seiten als Präsentationsobjekte implementiert, die von den vorhergehenden Präsentationsobjekten regen Gebrauch machen:

Komponentenuebersicht: Die Seite enthält die zentrale Suchfunktionalität. Für alle wichtigen Attributarten werden per selector-Komponente verschiedene Auswahlmöglichkeiten für Suchkriterien angeboten. Ebenso werden alle Komponenten, welche die aktuellen Suchkriterien erfüllen, aufgelistet. Die Auswahl einer Komponente aus dieser Liste führt zur Detaildarstellung der Komponente auf der „Komponentendetail“-Seite. Weiterhin steht ein Eingabefeld für eine Volltextsuche zur Verfügung, wobei für jede Komponente und jeden Attributwert per bullet-Komponente angezeigt wird, ob

die Volltextsuche den Suchtext in der jeweiligen Beschreibung entdecken konnte.

Komponentendetail: Die ausgewählte Komponente wird mit allen ihren Attributen und deren Ausprägungen dargestellt. Für Funktionalitäten und Technologien wird jeweils noch dargestellt, wie diese unterstützt bzw. verwendet werden können. Die Beschreibung der Komponente selbst wird durch die displayDescription-Komponente dargestellt, wodurch auch das Einblenden der Grafiken abgedeckt ist. Über einen Link kann zur Pflege der Komponentendaten auf die „Komponentendetailedit"-Seite gewechselt werden.

Image: Diese Seite bricht aus dem üblichen Konzept der Tapestry-Präsentationskomponenten aus. Die Image-Seite wird in den URLs angegeben, die eine Grafik darstellen sollen. Die Image-Seite extrahiert die in die URL kodierte ID der Grafik und liefert durch Überladen der entsprechenden Methoden diese Grafikdaten direkt an den Browser zurück. Dabei wird der übliche Templating-Mechanismus für Präsentationskomponenten umgangen, damit die Grafikdaten direkt ohne störende HTML-Tags ausgeben und der Content-Type der Seite korrekt gesetzt werden können (vgl. [FIE99]).

Die folgenden Präsentationsobjekte ermöglichen dem Benutzer, die Datenbasis des ICApps-Systems direkt in seinem Browser formularbasiert zu erweitern oder zu verändern:

Komponentendetailedit: Für die Komponentenbeschreibung werden durch die Verwendung der editDescription-Komponente entsprechende Felder zur Pflege der Beschreibung und zur Verwaltung der Grafiken angezeigt. Die Ausprägungen der verschiedenen Attribute mit beschreibendem Text können hier ebenfalls gepflegt werden.

FuncLinkEdit: Wird eine Funktionalität als „unterstützt" einer Komponente hinzugefügt oder soll der Beschreibungstext, wie eine Funktionalität durch die Komponente erbracht wird, verändert werden, so zeigt diese Seite die dafür notwendigen Eingabefelder an.

TechLinkEdit: Diese Seite bietet die gleichen Möglichkeiten wie die FuncLinkEdit-Seite, jedoch hier für Technologien.

ImageEdit: Diese Seite findet Verwendung, wenn eine neue Grafik eingebunden oder der Bilduntertitel verändert werden soll. Durch die Verwendung der von Tapestry bereitgestellten Upload-Komponente kann die ImageEdit-Seite selbst trotz der komplexeren Aufgabenstellung des Hochladens einer Datei sehr einfach strukturiert gehalten werden.

Der gesamte Ablauf der Benutzerschnittstelle des Programms ist, abhängig von den Benutzereingaben, in Abbildung 5.5 zusammenfassend dargestellt.

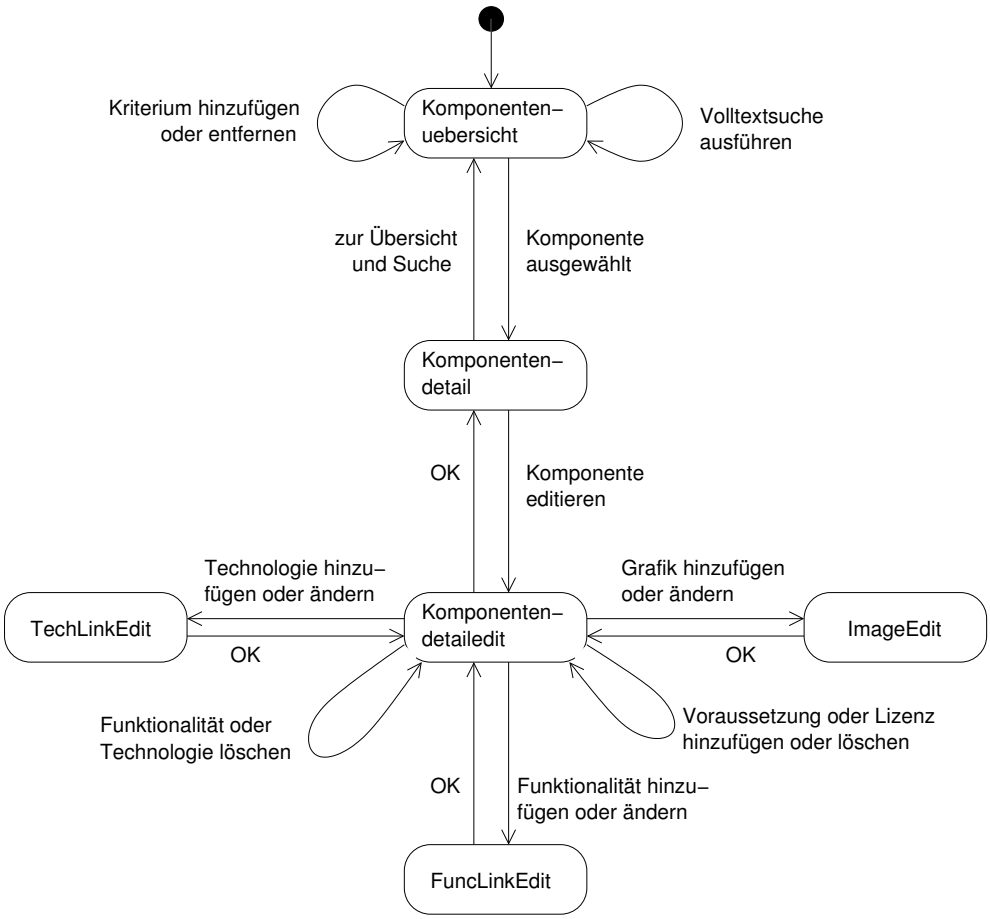

Abbildung 5.5: *Die Darstellung der verschiedenen Zustände der Benutzerschnittstelle in Abhängigkeit von den Eingaben des Benutzers.*

5.6 Packen und Installation

Das fertige System muss in einem bestimmten Format auf dem Applikationsserver installiert werden. Dieses Format ist durch die J2EE-Spezifikation fast vollständig festgelegt, um eine möglichst hohe Portierbarkeit zwischen den Applikationsservern verschiedener Hersteller zu erreichen (vgl. [SUN01]). Die nicht festgelegten Teile des Formates beziehen sich auf zusätzliche Deskriptordateien, die dazu verwendet werden können, herstellerspezifische Funktionen, die über die in der Spezifikation definierten Funktionen hinausgehen, nutzen und konfigurieren zu können. Jeder Hersteller verwendet auf Grund unterschiedlicher zusätzlicher Funktionen und unterschiedlicher Ansätze ein anderes Format für diese Deskriptordateien. Im Folgenden ist der Aufbau des nötigen Formates, in welches das ICApps-System für den Applikationsserver gebracht werden muss, dargestellt.

Dabei sind die für den Applikationsserver JBoss spezifischen Dateien durch einen Stern (*) gekennzeichnet.

5.6.1 JAR

Alle Enterprise JavaBeans, also die Entity Beans für die Datenspeicherung sowie die Session Beans für die Programmlogik, müssen in ein sogenanntes JAR-Archiv (*Java archiv*) gepackt werden. Das JAR-Archiv selbst ist eine Datei im Kompressionsformat ZIP (vgl. [PKW03]), welche folgende Verzeichnisstruktur und Dateien enthält:

```
Verzeichnis: META-INF
+ Datei: ejb-jar.xml
+ Datei: jboss.xml (*)
+ Datei: jbosscmp-jdbc.xml (*)
Verzeichnis: info
+ Verzeichnis: frech
  + Dateien: Alle .class-Datein der EJBs
```

Es ist notwendig, die binären Java-Klassen (.class-Dateien) in den Unterverzeichnissen „info“ und „frech“ abzulegen, da diese Klassen dem `package info.frech` angehören und sonst nicht vom Classloader des Applikationsserver gefunden werden. Das JAR-Archiv muss einen Dateinamen erhalten, der auf `.jar` endet, damit es richtig vom Applikationsserver erkannt wird. Als Name für das Archiv wurde `ejbs.jar` gewählt.

5.6.2 WAR

Für die HTML-Benutzerschnittstelle und das Tapestry-Servlet muss ein WAR-Archiv (*web application archiv*) erzeugt werden. Dieses ist ebenso wie das JAR-Archiv im ZIP-Format, besitzt jedoch die Endung `.war` und hat die folgende Struktur:

```
Verzeichnis: WEB-INF
+ Datei: web.xml
+ Verzeichnis: classes
  + Verzeichnis: app
  | + Dateien: die .html- und .page-Dateien
  | |          der Tapestry Komponenten
  | + Datei: Main.application
  + Verzeichnis: info
```

```
    + Verzeichnis: frech
      + Datei: MainServlet.class
      + Dateien: die .class-Dateien der Tapestry Komponenten
Verzeichnis: gfx
+ Dateien: alle Grafikdateien die statisch verwendet werden
+ Datei: index.html
```

Für das WAR Archiv wurde der Name `frontend.war` gewählt.

5.6.3 EAR

Ein EAR-Archiv (*enterprise application archiv*) ist ebenfalls eine Datei im ZIP-Format mit der Endung `.ear` . Das JAR- und das WAR- Archiv werden nun in einem EAR-Archiv zusammengefasst, welches dann auf dem Applikationsserver installiert werden kann. Dabei erhält das EAR-Archiv die folgende Struktur:

```
Datei: ejbs.jar
Datei: frontend.war
Verzeichnis: META-INF
+ Datei: application.xml
```

5.6.4 Installation

Für den JBoss-Applikationsserver gestaltet sich der Installationsvorgang für das EAR-Archiv denkbar einfach: Das Archiv muss lediglich in das Verzeichnis `jboss/server/default/deploy/` kopiert werden. JBoss stellt automatisch fest, dass diese Datei neu hinzu gekommen ist oder sich geändert hat. Eine eventuell schon vorhandene, alte Version der Applikation wird heruntergefahren und die neue Version gestartet. Wenige Sekunden nach dem Kopieren des EAR-Archivs kann dann mit einem Browser auf die Applikation zugegriffen werden.

5.7 Beispiel eines Suchdurchlaufs

Im Folgenden soll exemplarisch mit einigen Testdaten aufgezeigt werden, wie für einen Benutzer der Suchprozess im System ablaufen kann. Dazu wurden die verschiedenen Ansichten, die ein Browser dem Benutzer anzeigt, abgebildet. Die in Abbildung 5.6 dargestellte Seite ergibt sich nach dem Aufrufen der Startseite des Systems.

Auf dieser Seite werden für jedes der vier Komponentenattribute „Funktionalität“, „Technologie“, „Lizenz“ und „Voraussetzung“ alle Begriffe für das jeweilige

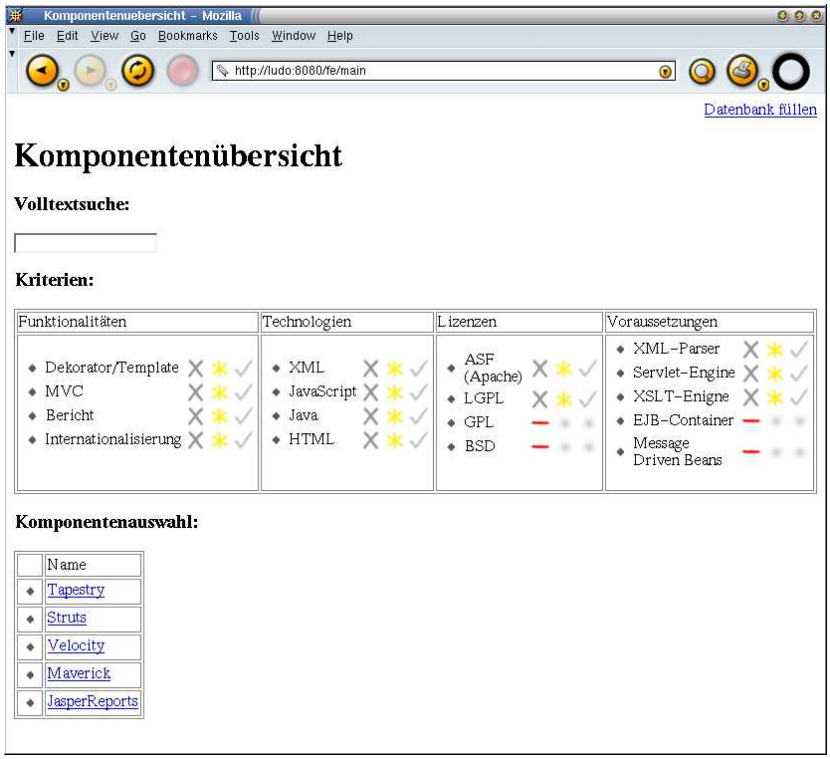

Abbildung 5.6: *Die Browser-Anzeige nach dem Aufruf der Startseite zeigt alle dem System bekannten Begriffe und Komponenten an. Der Benutzer kann nun ein Kriterium ein- oder ausschliessen oder einen Suchbegriff eingeben.*

Attribut aufgelistet. Hinter jedem Begriff werden die drei Symbole angezeigt. Normalerweise ist dies zu Beginn ein graues Kreuz, ein gelber Stern und ein grauer Haken. Wird die Maus über das graue Kreuz bewegt, so wird dieses rot gefärbt, analog dazu färbt sich der graue Haken grün, solange sich die Maus darüber befindet. Der Benutzer kann nun über die Auswahl des Kreuzes bzw. des Hakens einen Begriff als neues Kriterium ausschließen bzw. einschließen.

Da der Benutzer noch keine genauere Vorstellung davon hat, welche Kriterien für seine Suche zutreffend sind, gibt er in der Volltextsuche das Schlagwort „Design“ ein. Das System antwortet ihm mit der in Abbildung 5.7 dargestellten Seite, in der alle Begriffe oder Komponenten vom System markiert wurden, in denen der Suchbegriff gefunden wurde.

Der Benutzer liest die Beschreibungen der verschiedenen Kriterien (hier nicht dargestellt) und wählt zuerst das Kriterium für die Funktionalität „Dekorator/Template“ aus, indem er den Haken hinter dem Begriff auswählt. Das System aktualisiert darauf hin, wie in Abbildung 5.8 dargestellt, die Auswahl der Kriterien und schränkt die Anzahl der zutreffenden Komponenten ein.

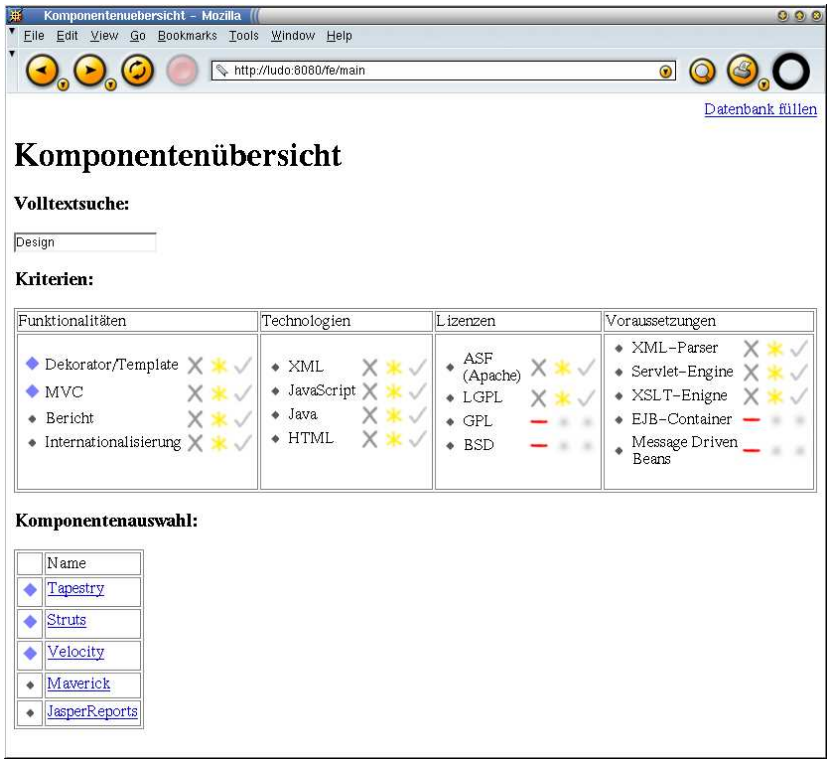

Abbildung 5.7: *Die Browser-Anzeige nach der Eingabe des Suchbegriffs „Design“. Bei den Komponenten und Begriffen, in deren Beschreibung der Suchbegriff gefunden wurde, wird die Raute vor dem Begriff blau gefärbt und größer dargestellt.*

Hinter dem ausgewählten Begriff wird nun der Haken dauerhaft in grün dargestellt. Der Stern ist grau geworden und wird gelb gefärbt, solange sich die Maus darüber befindet. Durch Auswahl des Sterns könnte dieses Kriterium wieder zurückgenommen werden. Hinter anderen Begriffen werden grüne Pluszeichen bzw. rote Minuszeichen anstatt der Kreuze und Haken angezeigt. Diese zeigen an, welche Begriffe durch die bisherige Auswahl implizit als ein- bzw. ausgeschlossen festgelegt wurden.

Der Benutzer möchte weiterhin JavaScript einsetzen und wählt dieses deshalb bei den unterstützen Technologien aus. Das System schränkt die Komponenten auf lediglich eine verbliebene Komponente ein. Wie in Abbildung 5.9 dargestellt, können in der Konsequenz keine weiteren Kriterien mehr ausgewählt werden, welche die Suche weiter verfeinern würden.

Abschliessend wählt der Benutzer die gefundene Komponente aus und bekommt vom System eine Detailansicht mit allen im System enthaltenen Beschreibungen angezeigt, die in Abbildung 5.10 dargestellt ist. Diese Detailansicht ermöglicht den schnellen Überblick über die Voraussetzungen, verwendbaren Technologien,

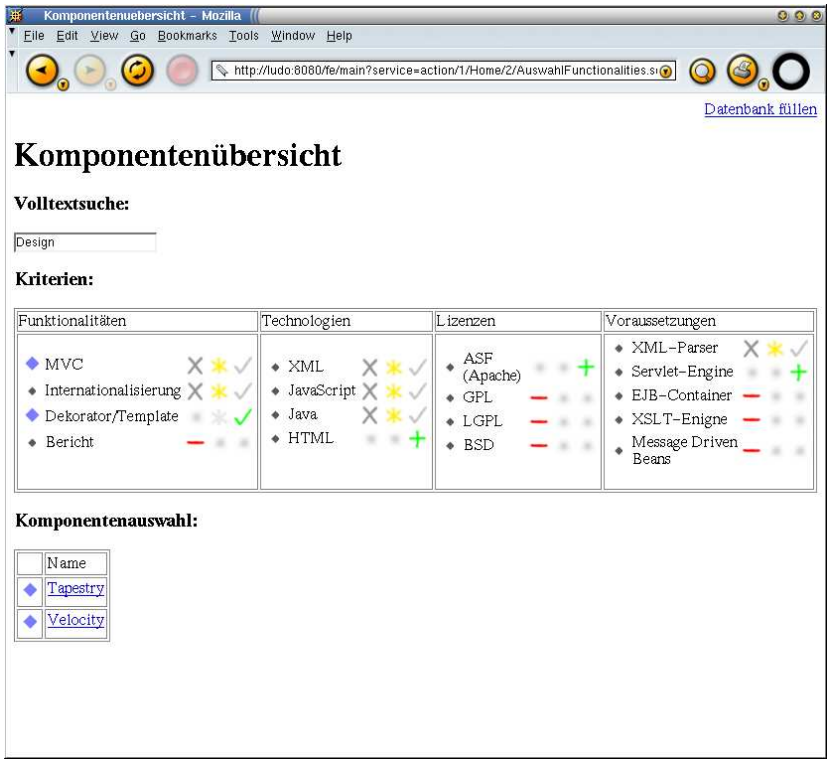

Abbildung 5.8: *Die Browser-Anzeige nach der Auswahl des Kriteriums „Dekorator/Template einschliessen“. Für die anderen Kriterien hat das System die Anzeige aktualisiert, welche Begriffe implizit ein- oder ausgeschlossen wurden und welche noch als relevant ausgewählt werden können. Die Auflistung der Komponenten wurde auf die Menge der kriterienerfüllenden eingeschränkt.*

erbrachten Funktionalitäten und ermöglicht zusätzlich über die Beschreibung der Funktionsweise der Komponente eine beschleunigte Einarbeitung in diese Komponente.

Sollte die gefundene Komponente den Anforderungen des Entwicklers nicht genügen, so kann er zur Suche zurückkehren und mindestens ein bisher gewähltes Kriterium wieder zurück nehmen. Daraufhin werden ihm wieder neue relevante Kriterien angeboten und die jeweils passenden Komponenten aufgelistet.

Abbildung 5.9: *Die Browser-Anzeige nach der Auswahl des zweiten Kriteriums „JavaScript einschliessen“. Es verbleibt nur noch eine Komponente und es können keine weiteren Kriterien hinzugefügt werden.*

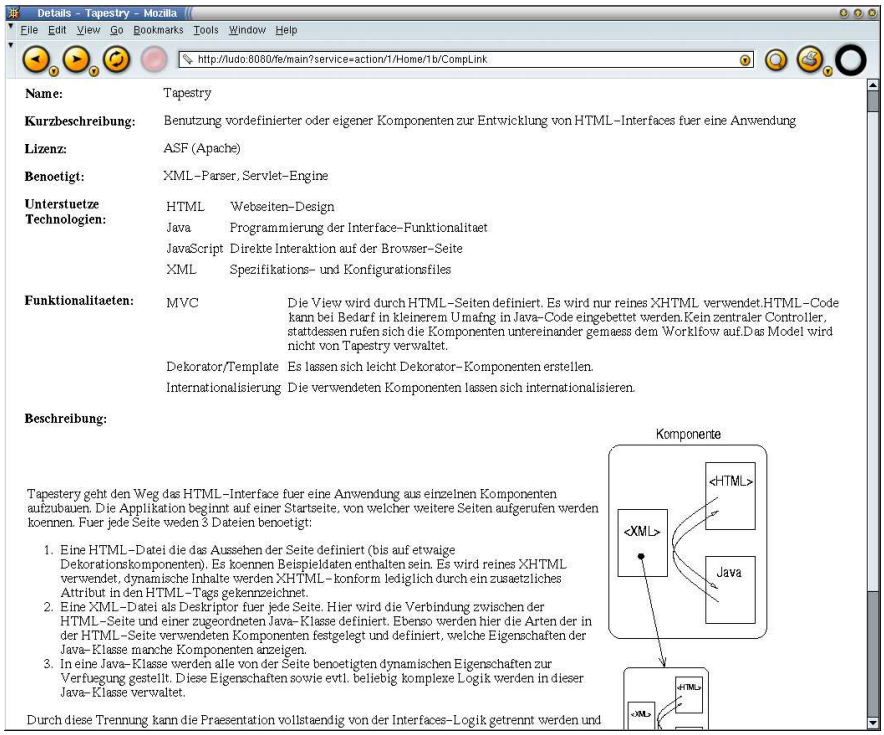

Abbildung 5.10: *Die Browser-Anzeige mit der Detailansicht einer ausgewählten Komponente zeigt alle im System enthaltenen Informationen inklusive einer Beschreibung der Funktionsweise der Komponente an.*

5.8 Technische Systemanforderungen

Für das implementierte System soll im folgenden dargestellt werden, durch welche Designentscheidungen die technischen Anforderungen aus der Problemstellung (Kapitel 2) erfüllt werden:

Anforderung	**Anforderungserfüllung**
hohe Erweiterbarkeit durch modularisierte Implementierung der Funktionalitäten	Durch die Verwendung von Session Beans konnten die Systemfunktionalitäten klar von anderen Systemteilen getrennt werden. Dadurch können sowohl neue Funktionalitäten hinzugenommen als auch die Implementierung von bestehenden Funktionalitäten abgeändert werden, ohne dabei Einfluss auf andere Systemteile zu haben.
Flexibilität für verschiedene Benutzerschnittstellen durch Trennung von Präsentation und zentralen Funktionalitäten	Die Benutzerschnittstelle wurde basierend auf dem Tapestry-Servlet umgesetzt. Damit ist die Benutzerschnittstelle vollkommen von Funktionalitäten (Session Beans) und Modell (Entity Beans) getrennt. Andere alternative Benutzerschnittstellen können sehr einfach auf die bestehenden Session Beans und Entity Beans über den Applikationsserver zugreifen.
standardisierte Speicherung der Komponentendaten	Durch die Verwendung von Entity Beans mit CMP 2.0 und CMR werden die Objektattribute und -relationen des Modells in einer relationalen Datenbank abgelegt. Das Datenbankschema erhält dabei ein Design, wie es auch bei einer direkten Modellierung des Schemas entstanden wäre. Auf die relationale Datenbank kann über die standardisierte SQL-Sprache zugegriffen werden.

Kapitel 6

Zusammenfassung und Ausblick

Um die in dieser Arbeit gestellte Aufgabe der „effizienten Entwicklung von J2EE-basierten Serveranwendungen durch Einsatz frei verfügbarer Komponenten“ erfüllen zu können, wurde zunächst der Entwicklungsprozess betrachtet, der durch einen Entwickler durchgeführt wird. Dabei wurden hauptsächlich zwei Hindernisse erkannt, die dem Einsatz von frei verfügbaren Komponenten entgegen stehen und die durch eine geeignete Unterstützung mit dieser Arbeit reduziert werden: Zum einen ist der Aufwand für die Suche nach Komponenten und die Informationsbeschaffung für eine Komponente sehr hoch, zum anderen besteht ein Terminologieunterschied zwischen den Anforderungen des Entwicklers und der Beschreibung der Komponenten.

Der Suchaufwand für den Entwickler wurde durch den Aufbau eines Systems, in dem die zur Verfügung stehenden Komponenten in einer strukturierten Art und Weise repräsentiert werden, erheblich reduziert. Es wurde eine Repräsentationsform für die Komponenten im System erarbeitet, die durch die Einführung der „Einzelfunktionalitäten“ eine Abstraktionsebene zwischen die Darstellung der Komponenten und die Formulierung der Entwickleranforderungen stellt. Durch diese Abstraktionsebene konnte der gegebene Terminologieunterschied reduziert werden. Für eine weitere Reduzierung dieses Unterschiedes wurde die „spreading activation“ Technologie vorgestellt und auf die hier gegebene, konkrete Anwendung übertragen.

Die erarbeitete Repräsentation der Komponenten im System durch einen Attribut-Wert-Ansatz in Kombination mit Fließtextteilen benötigt eine gute Suchunterstützung durch das System. Neben einer Volltextsuche wurde daher das Konzept der „relevanten Begriffe“ für die Kriterienauswahl durch den Entwickler im Suchsystem umgesetzt und damit eine sehr effiziente Methode zur Kriterienfindung, -eingabe und -priorisierung zur Verfügung gestellt. Das Suchsystem lässt sich iterativ in mehreren Suchschritten verwenden und ermöglicht damit die Anpassung der Kriterien an den Erkenntnisgewinn des Entwicklers während der Suche.

Als unterstützende Informationsquelle für die Qualitätsbeurteilung von Komponenten wurde der Aufbau eines Integrationsgraphen zwischen den Komponenten vorgeschlagen und vorgestellt.

Die vorhergehend erarbeiteten, theoretischen Ansätze für das Such- und Repräsentationssystem wurden für die Umsetzung des Systems in eine Benutzeroberfläche integriert, die einfach strukturiert gehalten werden konnte, aber trotzdem die volle Funktionalität anbietet. Durch die einfache Strukturierung der Benutzeroberfläche entstehen beim Entwickler keine neuen Barrieren, die einen Einsatz des Systems behindern könnten.

Die zur Entwicklung des Systems notwendige Arbeits- und Betriebsumgebung wurde vollständig aus Werkzeugen und Programmen aufgebaut, die unter einer freien Lizenz verfügbar sind. Als Entwicklungsumgebung wurde Eclipse ausgewählt und zusammen mit dem JBoss-IDE-Plugin erfolgreich eingesetzt. Diese Kombination unterstützt die Entwicklungstätigkeiten in vielfacher Weise positiv: Durch den Einsatz der Entwicklungswerkzeuge Ant und Xdoclet konnte zum einen die Durchlaufzeit für einen Kompilierungs- und Installationslauf, der über mehrere Systeme hinweg abläuft, auf weniger als eine Minute reduziert werden. Zum anderen konnten redundante Programmquelltextteile durch die automatische Erzeugung von Programmquelltext vermieden und die notwendige Quelltextgröße dabei um rund die Hälfte verringert werden. Als Datenbank- und J2EE-Applikationsserver wurden erfolgreich PostgreSQL und JBoss konfiguriert und eingesetzt. Bemerkenswert erscheint bei dieser Arbeits- und Betriebsumgebung, dass sie trotz der ausschließlichen Verwendung von freier Software bei einem Vergleich mit kommerziell vertriebenen Lösungen, wie zum Beispiel dem WebSphere Application Developer Studio von IBM, für die Entwicklung von J2EE-Anwendungen die Qualität und den notwendigen Funktionsumfang ebenso erbingen konnte (vgl. [IBM02]).

Das System selbst, das zur Verwaltung und Suche der Komponenten entwickelt wurde, ist mit einer web-basierten Benutzerschnittstelle ausgerüstet worden. Diese Benutzerschnittstelle erlaubt die strukturierte Eingabe und die Verwaltung von Komponentenbeschreibungen in Text- und Bildform. Weiterhin wurde ein iteratives Suchsystem mit Volltextsuche und Kriterieneingabe mit dem Ansatz der „relevanten Begriffe“ umgesetzt. Die web-basierte Benutzerschnittstelle verhindert auch hier das Entstehen von Barrieren, die den Entwickler vom Einsatz des Systems abhalten könnten. Der Entwickler kann über das Internet direkt auf die strukturierten Beschreibungen der Komponenten zugreifen. Das System ist selbst unter Verwendung einiger, in ihm dokumentierter Komponenten entwickelt worden und stellt somit ein Beispiel für die heutige Realisierbarkeit der Entwicklung J2EE-basierter Serveranwendungen mit frei verfügbaren Komponenten dar.

Sollte sich in zukünftigen Versionen des Systems die Repräsentation der Komponenten nur noch wenig ändern, so kann über eine Implementierung anderer

Benutzerschnittstellen für das System nachgedacht werden. Ein als Klient eigenständiges Java-Programm könnte direktere Reaktionen auf Benutzereingaben ermöglichen, sogar eine Integration in eine Entwicklungsumgebung wie Eclipse als Plugin wäre denkbar. Das System ist für derartige Erweiterungen durch die schon jetzt bestehende Trennung der Präsentation von der Logik vorbereitet. In der Weiterentwicklung des als evolutionären Prototyps realisierten Systems soll auch die im Konzept geplante „spreading activation"-Technologie in die Suchfunktionalität mit aufgenommen sowie die Repräsentation der Komponenten um den vorgestellten Integrationsgraphen erweitert werden.

Um den Aufwand für die Einarbeitung in die Komponenten und das Verstehen der internen Funktionsweise der Komponenten für den Entwickler weiter zu reduzieren, könnten die Komponentenrepräsentationen noch um Beispielquelltext erweitert werden. Hierbei ist zu klären, ob ein einheitliches Beispiel für alle Komponenten oder ein einheitliches Beispiel je Einzelfunktionalität sinnvoller wäre, als individuelle Beispiele je Komponente.

Über eine Erweiterung des Systems um eine Rechteverwaltung könnte der Zugriff auf Änderungs- und Managementfunktionen für einzelne Personen geöffnet werden. Somit könnte es Komponentenautoren ermöglicht werden, ihre Komponenten selbst im System zu dokumentieren und die Repräsentation zu verwalten.

Mit einer ausreichenden Datenbasis im System wäre es auch interessant, zu untersuchen, inwieweit sich ein Entwurfsmuster-basierter Ansatz in das System integrieren lässt. Eventuell lassen sich Komponenten oder Einzelfunktionalitäten einzelnen Entwurfsmustern zuordnen.

Sicherlich wird in zukünftigen Versionen des Systems die Komponentenrepräsentation durch die Rückmeldungen der Entwickler über ihre Bedürfnisse weiterentwickelt werden. Denkbar wäre hier zum Beispiel die Ergänzung der Komponentenrepräsentation um ein Bewertungssystem, das Entwickler je Komponente verschiedene Aspekte der Komponente bewerten lässt.

Anhang A

Xdoclet

Xdoclet ermöglicht die deklarative Beschreibung aller für EJBs relevanten Zusatzangaben über den Implementierungsquelltext hinaus. Somit lassen sich aus einer Datei die für eine EJB normalerweise benötigten vier Dateien erzeugen. Dieser Vorgang wird im Folgenden erläutert. Der Ausgangspunkt ist die Datei FunctionalityBean.java:

```
Datei: FunctionalityBean.java
-----------------------------
package info.frech;
import javax.ejb.CreateException;
import javax.ejb.EntityBean;
/**
 * Functionality
 *
 * @ejb:bean name="Functionality"
 *           type="CMP"
 *           jndi-name="ejb/Functionality"
 *           primkey-field="functionalityID"
 *           view-type="both"
 * @ejb:pk class="java.lang.Integer"
 * @ejb:finder signature="Collection findAll()"
 * @ejb:interface remote-class="info.frech.interfaces.Functionality"
 *                local-class="info.frech.interfaces.FunctionalityLocal"
 *
 * @jboss:table-name "functionality"
 * @jboss:create-table "true"
 * @jboss:remove-table "true"
 *
 */

public abstract class FunctionalityBean implements EntityBean
{
   /**
```

```
     * @ejb.create-method view-type="both"
     */
     public Integer ejbCreate(String name) throws CreateException {
             setFunctionalityID(IdSequence.nextId());
             setName(name);
             return null;
     }

     public void ejbPostCreate(String name) {};

    /**
     * @ejb.interface-method view-type="both"
     * @ejb.persistent-field
     */
     public abstract Integer getFunctionalityID();

    /**
     * @ejb.interface-method view-type="both"
     */
     public abstract void setFunctionalityID(Integer functionalityID);

    /**
     * @ejb.interface-method view-type="both"
     * @ejb.persistent-field
     */
     public abstract String getName();

    /**
     * @ejb.interface-method view-type="both"
     */
     public abstract void setName(String name);

    /**
     * @ejb.interface-method view-type="both"
     * @ejb.persistent-field
     */
     public abstract String getDescription();

    /**
     * @ejb.interface-method view-type="both"
     */
     public abstract void setDescription(String description);

}
```

Xdoclet generiert aus dieser Datei die folgenden weiteren Dateien:

```
Datei: FunctionalityCMP.java
----------------------------
package info.frech;
```

```
/**
  * CMP layer for Functionality.
  */
public abstract class FunctionalityCMP
  extends info.frech.FunctionalityBean implements javax.ejb.EntityBean {
    public void ejbLoad() {}
    public void ejbStore() {}
    public void ejbActivate() {}
    public void ejbPassivate() {}
    public void setEntityContext(javax.ejb.EntityContext ctx) {}
    public void unsetEntityContext() { }
    public void ejbRemove() throws javax.ejb.RemoveException {}
    public abstract java.lang.Integer getFunctionalityID() ;
    public abstract void setFunctionalityID(
                          java.lang.Integer functionalityID ) ;
    public abstract java.lang.String getName() ;
    public abstract void setName( java.lang.String name ) ;
    public abstract java.lang.String getDescription() ;
    public abstract void setDescription( java.lang.String description ) ;
}
```

Diese Datei enthält die Klasse, welche die Implementierung des Objektes auf dem Server darstellt. Die abstrakten Zugriffsmethoden werden durch den Applikationsserver selbst implementiert. Weiterhin werden die beiden Schnittstellendeklarationen für entfernte Klienten benötigt:

Datei: FunctionalityHome.java

```
------------------------------
package info.frech;
/**
  * Home interface for Functionality.
  */
public interface FunctionalityHome extends javax.ejb.EJBHome {
  public static final String COMP_NAME="java:comp/env/ejb/Functionality";
  public static final String JNDI_NAME="ejb/Functionality";
  public info.frech.interfaces.Functionality create(java.lang.String name)
    throws javax.ejb.CreateException,java.rmi.RemoteException;
  public java.util.Collection findAll()
    throws javax.ejb.FinderException,java.rmi.RemoteException;
  public info.frech.interfaces.Functionality
    findByPrimaryKey(java.lang.Integer pk)
    throws javax.ejb.FinderException,java.rmi.RemoteException;
}
```

Datei: Functionality.java

```
-------------------------
package info.frech.interfaces;
/**
 * Remote interface for Functionality.
 */
```

```
public interface Functionality extends javax.ejb.EJBObject {
  public java.lang.String getDescription( )
    throws java.rmi.RemoteException;
  public java.lang.Integer getFunctionalityID( )
    throws java.rmi.RemoteException;
  public java.lang.String getName( )
    throws java.rmi.RemoteException;
  public void setDescription( java.lang.String name )
    throws java.rmi.RemoteException;
  public void setFunctionalityID( java.lang.Integer functionalityID )
    throws java.rmi.RemoteException;
  public void setName( java.lang.String name )
    throws java.rmi.RemoteException;
}
```

Für lokale Klienten werden zusätzlich die optimierten Versionen der Schnittstellen erstellt:

Datei: FunctionalityLocalHome.java

```
----------------------------------
package info.frech;
/**
  * Local home interface for Functionality.
  */
public interface FunctionalityLocalHome extends javax.ejb.EJBLocalHome {
  public static final String COMP_NAME="java:comp/env/ejb/FunctionalityLocal";
  public static final String JNDI_NAME="FunctionalityLocal";
  public info.frech.interfaces.FunctionalityLocal create(java.lang.String name)
    throws javax.ejb.CreateException;
  public java.util.Collection findAll()
    throws javax.ejb.FinderException;
  public info.frech.interfaces.FunctionalityLocal
    findByPrimaryKey(java.lang.Integer pk)
    throws javax.ejb.FinderException;
}
```

Datei: FunctionalityLocal.java

```
------------------------------
package info.frech.interfaces;
/**
 * Local interface for Functionality.
 */
public interface FunctionalityLocal extends javax.ejb.EJBLocalObject {
  public java.lang.String getDescription( ) ;
  public java.lang.Integer getFunctionalityID( ) ;
  public java.lang.String getName( ) ;
  public void setDescription( java.lang.String name ) ;
  public void setFunctionalityID( java.lang.Integer functionalityID ) ;
  public void setName( java.lang.String name ) ;
}
```

Sowohl die Implementierungsklasse als auch die Schnittstellen müssen mit ihren Metadaten in die Deskriptoren eingetragen werden. Der standardisierte ejb-jar.xml Deskriptor enthält die folgenden Fragmente, die bezüglich des Functionality-Objekts von Xdoclet erzeugt wurden:

```
<entity >
   <description><![CDATA[Functionality]]></description>
   <ejb-name>Functionality</ejb-name>
   <home>info.frech.FunctionalityHome</home>
   <remote>info.frech.interfaces.Functionality</remote>
   <local-home>info.frech.FunctionalityLocalHome</local-home>
   <local>info.frech.interfaces.FunctionalityLocal</local>
   <ejb-class>info.frech.FunctionalityCMP</ejb-class>
   <persistence-type>Container</persistence-type>
   <prim-key-class>java.lang.Integer</prim-key-class>
   <reentrant>False</reentrant>
   <cmp-version>2.x</cmp-version>
   <abstract-schema-name>Functionality</abstract-schema-name>
   <cmp-field >
      <description><![CDATA[]]></description>
      <field-name>functionalityID</field-name>
   </cmp-field>
   <cmp-field >
      <description><![CDATA[]]></description>
      <field-name>name</field-name>
   </cmp-field>
   <cmp-field >
      <description><![CDATA[]]></description>
      <field-name>description</field-name>
   </cmp-field>
   <primkey-field>functionalityID</primkey-field>
    <!-- Write a file named ejb-finders-FunctionalityBean.xml
         if you want to define extra finders. -->
</entity>
<ejb-relation >
   <ejb-relation-name>funclink-functionality</ejb-relation-name>
   <ejb-relationship-role >
      <ejb-relationship-role-name>funclink-points-to-functionality
  </ejb-relationship-role-name>
      <multiplicity>Many</multiplicity>
      <relationship-role-source >
         <ejb-name>FuncLink</ejb-name>
      </relationship-role-source>
      <cmr-field >
         <cmr-field-name>functionality</cmr-field-name>
```

```
      </cmr-field>
    </ejb-relationship-role>
    <ejb-relationship-role >
      <ejb-relationship-role-name>functionality-is-refered-by-funclink
   </ejb-relationship-role-name>
      <multiplicity>One</multiplicity>
      <relationship-role-source >
        <ejb-name>Functionality</ejb-name>
      </relationship-role-source>
    </ejb-relationship-role>
</ejb-relation>
```

Im ersten entity-Fragment werden für die EJB die benötigten Java-Klassen spezifiziert sowie die Persistenzmethode festgelegt. Das zweite ejb-relation-Fragment listet die Container-Managed-Relationships für diese EJB auf. Der JBoss-spezifische Deskriptor jboss.xml enthält folgendes Fragment:

```
<entity>
   <ejb-name>Functionality</ejb-name>
   <jndi-name>ejb/Functionality</jndi-name>
   <local-jndi-name>FunctionalityLocal</local-jndi-name>
</entity>
```

In diesem Deskriptor könnten noch spezielle Container-Konfigurationen ausgewählt werden. In diesem Beispiel wird jedoch lediglich der JNDI-Name der EJB festgelegt. Für die Konfiguration der Container-Managed-Persistence (CMP 2.0) sind noch die folgenden Fragmente in dem Deskriptor jbosscmp-jdbc.xml notwendig:

```
<entity>
  <ejb-name>Functionality</ejb-name>
  <create-table>false</create-table>
  <remove-table>false</remove-table>
  <table-name>functionality</table-name>
   <cmp-field>
      <field-name>functionalityID</field-name>
   </cmp-field>
   <cmp-field>
      <field-name>name</field-name>
   </cmp-field>
   <cmp-field>
      <field-name>description</field-name>
   </cmp-field>
    <!-- merge point: jbosscmp-jdbc-load-{0}.xml -->
```

```
</entity>
<ejb-relation>
  <ejb-relation-name>funclink-functionality</ejb-relation-name>
  <foreign-key-mapping/>
  <ejb-relationship-role>
      <ejb-relationship-role-name>funclink-points-to-functionality
        </ejb-relationship-role-name>
            <key-fields/>
  </ejb-relationship-role>
  <ejb-relationship-role>
      <ejb-relationship-role-name>functionality-is-refered-by-funclink
        </ejb-relationship-role-name>
      <key-fields>
         <key-field>
           <field-name>functionalityID</field-name>
           <column-name>functionalityID</column-name>
         </key-field>
      </key-fields>
  </ejb-relationship-role>
</ejb-relation>
```

All diese Dateien und Einträge in den Deskriptordateien werden durch Xdoclet aus der einzelnen, zu Beginn vorgestellten Datei FunctionalityBean.java erstellt. Xdoclet hat sich aus diesem Grund als ein wichtiges Werkzeug zur Aufwandsreduktion in der EJB-Entwicklung erwiesen. Zudem werden durch den Einsatz von Xdoclet Duplikationen vermieden und die sonst daraus resultierenden Inkonsistenzen können nicht auftreten.

Anhang B

Server-Einrichtung

B.1 JBoss mit Tapestry

Verwendete Versionen:

	Versionsnummer	Distributionspaket
JBoss:	3.2.1	jboss-3.2.1.zip
Tapestry:	2.4 (Alpha 5)	Tapestry-2.4-alpha-5-bin.tar.gz

JBoss entpacken Das Distributionspaket von JBoss muss in der Verzeichnisstruktur an der Stelle entpackt werden, an welcher der Applikationsserver installiert sein soll.

```
unzip jboss-3.2.1.zip
```

Dadurch wird die folgende Verzeichnisstruktur entpackt (vgl. [STARK]):

jboss-3.2.1 Alle Dateien aus dem Distributionspaket werden in ein Verzeichnis entpackt, das als Namen die Versionsnummer trägt.

bin In diesem Verzeichnis finden sich die Skripte, um JBoss auf den verschiedenen Architekturen zu starten. Ebenso sind hier die Java-Archive (JAR), welche die nötigen Java-Startklassen enthalten, vorzufinden.

client Alle hier enthaltenen Java-Archive (JAR) sind für die Verwendung auf der Klientenseite gedacht, falls der Klient nicht selbst im Applikationsserver abläuft. Je nach verwendeter Funktionalität (EJB, JNDI, JMS, JAAS, Logging etc.) müssen nur wenige oder alle hier enthaltenen JARs in den Klassenpfad (Umgebungsvariable CLASSPATH) des Klienten aufgenommen werden.

docs Neben Document-Type-Definition-Dateien (DTD, [BRA00]) für alle verwendeten Deskriptoren sind hier einige Beispielkonfigurationen für die meisten Datenbankmanagementsystem enthalten. Weiterhin sind die vollständigen Ergebnisse der projekteigenen Testsuite für diese Distribution enthalten.

...

lib Die hier enthaltenen Java-Archive werden von JBoss beim Start benötigt. Normalerweise werden in dieses Verzeichnis keine Dateien vom Benutzer hinzugefügt.

server JBoss selbst ist auf Grund seines durchgehend modularen Aufbaus sehr stark konfigurierbar. Nicht verwendete Funktionalitäten können in der Konfiguration ausgeschlossen werden und werden beim Start dann auch nicht geladen. In diesem Verzeichnis werden alle zur Verfügung stehenden Konfigurationen des Server angelegt. Es sind drei verschiedene Konfigurationen vorhanden, von denen eine JBoss beim Start übergeben wird. Die Standardauswahl ist „default“. Es können auch eigene Unterverzeichnisse mit eigenen Server-Konfigurationen neben den drei mitgelieferten Konfigurationen angelegt werden.

all In dieser Konfiguration sind alle für JBoss verfügbaren Funktionalitäten aktiviert. Der Server braucht in dieser Konfiguration die längste Zeit zum Starten und den größten Speicherbedarf.

...

default In der „default“-Konfiguration sind alle Optionen, die normalerweise nicht für einen einfachen Entwicklungs- oder Produktionsserver benötigt werden, nicht enthalten. Es wurden zum Beispiel solche Dinge wir Clustering, Farming, IIOP und Web-Services ausgeschlossen.

conf In einem Unterverzeichnis mit diesem Namen sind für alle Konfigurationen die Konfigurations-Deskriptoren enthalten.

deploy In dieses Verzeichnis werden alle Applikationsarchive (JAR, WAR, EAR) kopiert, die der Applikationsserver starten soll. Aber auch Dienste oder Ressourcenkonfigurationen (SAR) können hier im laufenden Applikationsserver gestartet werden. Dieses Verzeichnis enthält in der Distribution schon einige Dateien, die verschiedene Dienste als Erweiterungen zum Serverkern starten.

...

lib Der Inhalt aller Java-Archive in diesem Verzeichnis wird für die entsprechende Konfiguration automatisch für alle Applikationen im Klassenpfad zur Verfügung gestellt.

minimal In der hier abgelegten Serverkonfiguration wird lediglich der Serverkern gestartet. Diese Konfiguration bietet die geringste Startzeit und den minimalsten Speicherbedarf.

...

Tapestry integrieren Zur Integration von Tapestry in JBoss ist es lediglich notwendig, den Applikationen die entsprechenden Klassen auf dem Applikationsserver im Klassenpfad verfügbar zu machen. Dazu wird zuerst das Tapestry-Distributionspaket entpackt:

```
tar xzf Tapestry-2.4-alpha-5-bin.tar.gz
```

Das Archiv mit den zentralen Tapestry-Klassen befindet sich in einem Unterverzeichnis:

```
cd Tapestry-2.4-alpha-5/lib/
```

Von dort wird die Datei `tapestry-2.4-alpha-5.jar` in das Verzeichnis `jboss-3.2.1/server/default/deploy/` kopiert, welches unterhalb des Installationspunkts des Applikationsservers liegt. Außer diesem Archiv werden noch weitere Hilfsklassen aus anderen freien Komponenten gebraucht, die sich in anderen Archiven in einem weiteren Unterverzeichnis befinden:

```
cd ext/
```

Von dort sollten abschließend die folgenden Klassen in das gleiche Zielverzeichnis wie das vorherige Archiv kopiert werden:

- `jakarta-oro-2.0.6.jar`
- `ognl-2.3.0-opt.jar`
- `commons-logging-1.0.2.jar`
- `commons-digester-1.4.jar`
- `commons-fileupload-1.0-beta-1.jar`
- `commons-lang-1.0.jar`
- `commons-collections-2.1.jar`
- `bsf-2.3.0.jar`

JBoss installiert alle kopierten Archive im Betrieb selbständig und macht den Applikationen die Klassen auf dem Server zugängig. Die Archive können auch im laufenden Betrieb hinzugefügt oder ausgetauscht werden.

JBoss starten Es ist keine weitere Konfiguration des Servers notwendig. Er kann nun direkt gestartet werden. Dazu wird das Arbeitsverzeichnis auf das Unterverzeichnis `bin` eingestellt, die Umgebungsvariable JAVA_HOME auf das verwendete *Java Runtime Environment* gesetzt und das entsprechende Startskript ausgeführt. Für das verwendete Solaris Betriebssystem wurde die mit folgenden Befehlen erreicht:

```
setenv JAVA_HOME /afs/informatik.uni-tuebingen.de/sun4x_58/j2sdk-1.4.1/
cd jboss-3.2.1/bin/
./run.sh
```

Der Applikationsserver beginnt dann mit dem Startvorgang:

```
=========================================================================
JBoss Bootstrap Environment
JBOSS_HOME: /tmp/frech/jboss-3.2.1
JAVA: /afs/informatik.uni-tuebingen.de/sun4x_58/j2sdk-1.4.1//bin/java
JAVA_OPTS: -server -Dprogram.name=run.sh
CLASSPATH: /tmp/frech/jboss-3.2.1/bin/run.jar:
/afs/informatik.uni-tuebingen.de/sun4x_58/j2sdk-1.4.1//lib/tools.jar
=========================================================================
18:29:02,556 INFO [Server] Starting JBoss (MX MicroKernel)...
18:29:02,559 INFO [Server] Release ID: JBoss [WonderLand] 3.2.1
(build: CVSTag=JBoss_3_2_1 date=200305011533)
18:29:02,560 INFO [Server] Home Dir: /tmp/frech/jboss-3.2.1
18:29:02,561 INFO [Server] Home URL: file:/tmp/frech/jboss-3.2.1/
18:29:02,561 INFO [Server] Library URL: file:/tmp/frech/jboss-3.2.1/lib/
...
```

Nach kurzer Zeit ist der Start abgeschlossen:

```
...
18:32:07,070 INFO [jbossweb] successfully deployed file:/tmp/frech/jboss-3.2.1/server/default/
tmp/deploy/server/default/deploy/management/web-console.war/28.web-console.war to /web-console
18:32:07,106 INFO [MainDeployer] Deployed package: file:/tmp/frech/jboss-3.2.1/server/default/
deploy/management/
18:32:07,115 INFO [URLDeploymentScanner] Started
18:32:07,315 INFO [MainDeployer] Deployed package: file:/tmp/frech/jboss-3.2.1/server/default/
conf/jboss-service.xml
18:32:07,319 INFO [Server] JBoss (MX MicroKernel)
[3.2.1 (build: CVSTag=JBoss_3_2_1 date=200305011533)] Started in 33s:715ms
```

B.2 PostgreSQL

Verwendete Version:

	Versionsnummer	Distributionspaket
PostgreSQL:	7.3.2	postgresql-7.3.2.tar.gz

Das Distributionsarchive muss zuerst entpackt werden:

```
tar xzf postgresql-7.3.2.tar.gz
```

Danach wird in das erstellte Verzeichnis gewechselt und das Kompilierungsystem auf die lokalen Gegebenheiten eingestellt:

```
cd postgresql-7.3.2/
./configure --prefix=/afs/wsi/sun4x_58/postgresql-7.3.2/\
--with-includes=/afs/wsi/sun4x_58/readline-4.2a/include:\
/afs/wsi/sun4x_58/zlib-1.1.4/include\
--with-libraries=/afs/wsi/sun4x_58/readline-4.2a/lib:\
/afs/wsi/sun4x_58/zlib-1.1.4/lib --with-rpath
```

Damit werden die nötigen Dateien für die Kompilierung erstellt. Die binären Dateien sollen im AFS abgelegt werden und für die Bibliothek „readline“ soll die Version aus dem AFS benutzt werden. Der Parameter `--with-rpath` bewirkt, dass die zur Kompilierung verwendeten dynamischen Bibliotheken auch zum Ausführungszeitpunkt verwendet werden, auch wenn sich die Konfiguration des Systems geändert hat.

Die binären Dateien müssen nun erstellt werden:

```
make
```

Nach der erfolgreichen Kompilierung der Programmdateien müssen diese in das AFS installiert werden:

```
make install
```

PostgreSQL soll auf dem Rechner „linus“ laufen. Die Daten der Datenbanken werden dort lokal in das Verzeichnis /local/postgres/pgsql-data gespeichert. Dieses Verzeichnis muss zunächst für die Aufnahme einer Datenbank vorbereitet werden:

```
mkdir /local/postgres/pgsql-data
/afs/wsi/sun4x_58/postgresql-7.3.2/bin/initdb \
-D /local/postgres/pgsql-data/
```

Das DBMS wird dann durch Aufruf des folgenden Befehls gestartet:

```
/afs/wsi/sun4x_58/postgresql-7.3.2/bin/pg_ctl \
-D /local/postgres/pgsql-data/ -l /local/postgres/logfile start
```

Soll das DBMS gestoppt werden (z.B. beim Herunterfahren des Systems), so ist der folgende Befehl notwendig:

```
/afs/wsi/sun4x_58/postgresql-7.3.2/bin/pg_ctl \
-D /local/postgres/pgsql-data/ stop
```

Die Datenbank namens „dafrech“ wird auf dem Rechner „linus“ angelegt durch:

```
/afs/wsi/sun4x_58/postgresql-7.3.2/bin/createdb \
-D /local/postgres/pgsql-data/ dafrech
```

Die folgenden Änderungen an der Konfiguration des DBMS wurden in der Datei `/local/postgres/pgsql-data/postgresql.conf` vorgenommen (vgl. [POS02]):

Änderung	Erläuterung
`fsync = true`	Mit dieser Einstellung führt PostgreSQL häufig ein fsync aus, was ein Schreiben eventuell geänderter Daten auf die Festplatte bewirkt. Die Leistung des DBMS wird dadurch etwas reduziert, jedoch wird sichergestellt, dass keine Information aus der internen Logdatei bei einem Absturz verlorengehen und somit die Transaktionseigenschaften garantiert bleiben.
`tcpip_socket = true`	In der Grundeinstellung akzeptiert PostgreSQL nur Verbindungen über *Unix domain sockets* und nicht über TCP/IP. Für den entfernten Zugriff auf das DBMS wird jedoch TCP/IP-Unterstützung benötigt.

PostgreSQL verwaltet eigene Benutzerkonten. Diese können im mitgelieferten SQL-Werkzeug angelegt und die Rechte dieser Benutzer eingestellt werden. Dazu wird das Werkzeug „psql“ gestartet:

```
/afs/wsi/sun4x_58/postgresql-7.3.2/bin/psql dafrech
```

Der bestehende Benutzer „frech“ bekommt ein neues Passwort zugewiesen, ein neuer Benutzer namens „postgres“ wird angelegt und erhält Super-User Rechte:

```
alter user frech password 'geheim';
create user postgres with password 'geheim';
alter user postgres createuser createdb;
```

PostgreSQL führt eine *host based authentication* durch, weshalb alle entfernt zugreifenden Rechner den Zugriff erlaubt bekommen müssen. In der Datei `/local/`

`postgres/pgsql-data/pg_hba.conf` werden alle Methoden aufgelistet, mit denen eine Verbindung zu PostgreSQL aufgebaut werden darf. Für jede Methode wird ebenfalls angegeben, von welchen Rechnern dies geschehen darf und mit welcher Methode sich ein Benutzer beim System authentifizieren muss. Der Inhalt der Datei ph_hba.conf wurde wie folgt festgelegt:

```
# TYPE DATABASE USER IP-ADDRESS IP-MASK          METHOD
local  all      all                             password
host   all      all  127.0.0.1  255.255.255.255  password
host   all      all  134.2.8.0  255.255.248.0    md5
```

Die erste Zeile ist eine Kommentarzeile, die den Inhalt der Spalten beschreibt. Die zweite Zeile erlaubt allen Benutzern die Verbindung zu allen Datenbanken über sogenannte *unix domain sockets*, also nur von „linus“ selbst aus. Die dritte Zeile erlaubt den soeben beschrieben Zugriffspfad auch über eine TCP/IP-Verbindung von „linus“ selbst aus. Bei den beiden Möglichkeiten aus Zeile zwei und drei muss der Benutzer seinen Namen und ein in PostgreSQL hinterlegtes Passwort zur Authentifizierung angeben. Dieses wird nicht verschlüsselt, was aber auch nicht notwendig ist, da es nicht über das Netzwerk übertragen wird. In der vierten und letzten Zeile werden für alle PostgreSQL-Benutzer Verbindungen auf alle Datenbanken von allen Rechnern des Informatik-Netzwerks zugelassen. Die Verbindungen werden per TCP/IP aufgebaut. Die Benutzer müssen sich auch hier mit Benutzernamen und Passwort ausweisen, wobei das Passwort jedoch durch einen MD5-Hash nicht direkt im Klartext übertragen wird.

Der Datenbankserver muss neu gestartet werden oder ein entsprechendes Signal erhalten, damit die neuen Konfigurationsdaten eingelesen und verwendet werden.

Wichtig erscheint, darauf hinzuweisen, dass die oben beschriebene Datei lediglich den grundsätzlichen Verbindungsaufbau zum PostgreSQL-System regelt. Die konkreten Zugriffs- und Manipulationsrechte auf die einzelnen Datenbanken und Relationen werden innerhalb von PostgreSQL für jeden Benutzer einzeln verwaltet.

B.3 Integration JBoss und PostgreSQL

PostgreSQL soll als Datenquelle (engl. Datasource) für JBoss dienen. Dazu muss ein JDBC-Treiber für PostgreSQL verwendet werden. Das verwendete Java-Archiv mit den Treiberdateien ist `pg73jdbc3.jar` von der Adresse `http://jdbc.postgresql.org/download.html`. Dieses Archiv wird zur Installation in das Verzeichnis `jboss-3.2.1/server/default/deploy/` kopiert. Das PostgreSQL-System wird als Datenquelle in JNDI-Dienst unter dem Namen java:/PostgresDS eingebunden, indem eine Datei namens postgres-ds.xml mit dem folgenden Inhalt ebenfalls nach `jboss-3.2.1/server/default/deploy/` kopiert wird:

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
  <local-tx-datasource>
    <jndi-name>PostgresDS</jndi-name>
    <connection-url>jdbc:postgresql://linus:5432/dafrech</connection-url>
    <driver-class>org.postgresql.Driver</driver-class>
    <user-name>frech</user-name>
    <password>geheim</password>
  </local-tx-datasource>
</datasources>
```

Anhang C

Entwicklungsumgebung

C.1 Ant

Ant ist ein Werkzeug, welches aus der Entwicklung von Tomcat herausgelöst und innerhalb des Apache Jakarta Projektes weiterentwickelt wurde. Inzwischen ist Ant ein eigenständiges Apache Projekt und wird als Werkzeug zur Automatisierung komplexer Kompilierungs- oder anderer Arbeitsabläufe eingesetzt. Häufig wird Ant als das Java-Pendant zu dem aus Unix bekannten Programm „make“ bezeichnet, weist jedoch inzwischen wesentlich mehr Fähigkeiten als „make“ auf.

Für die Ausführung von Ant wird eine zentrale XML-Datei benötigt, in der sogenannte „Targets“ aufgelistet sind. Innerhalb eines Targets können dann mehrere sogenannte „Tasks“ aufgelistet werden. Ein Task erbringt eine mehr oder weniger komplexe Funktionalität wie zum Beispiel das Kopieren von Dateien oder das Kompilieren von Quelltext in Binärdateien. Es stehen eine Vielzahl von Tasks für alle möglichen Gebiete aus der Entwicklungstätigkeit zur Verfügung. Es können auch durch Java-Klassen neue, selbst entwickelte Tasks dem Ant-System hinzugefügt werden.

Verschiedene Targets, welche die Tasks enthalten, können untereinander in Abhängigkeiten stehen. So kann sichergestellt werden, dass vor der Ausführung der Tasks eines Targets zuerst die Tasks eines anderen Targets ausgeführt werden. Xdoclet ist als ein solcher Task verfügbar. So kann zum Beispiel sichergestellt werden, dass Xdoclet zuerst aus den Quelltexten weitere Quelltexte erzeugt, bevor letztere durch den Java-Compiler in Bytecode-Dateien übersetzt werden.

Für die Entwicklung des ICApps-Systems wurden die folgenden Targets erstellt:

Target-Name	Funktion der enthaltenen Tasks
xdoclet	Erzeugt aus den Programmquelltexten weitere Quelltextdateien und Deskriptordateien mit Metadaten.

compile	Kompiliert alle EJB-Quelltextdateien zu Bytecode-Dateien, die von der Java Virtual Machine ausgeführt werden können.
jar	Packt die Bytecode-Dateien für die EJBs mit den entsprechenden Deskriptoren in ein JAR-Archiv.
deploy	Kopiert das JAR-Archiv in einen lokal installierten JBoss-Server und startet damit die Installation der enthaltenen Applikation.
undeploy	Löscht das JAR-Archiv aus einem lokal installierten JBoss-Server und deinstalliert damit die enthaltene Applikation.
war-compile	Kompiliert alle zu Servlets gehörigen Quelltextdateien zu Bytecode-Dateien.
war	Packt die Bytecode-Dateien des Servlet, das Tapestry-Servlet und die notwendigen Deskriptoren in ein WAR-Archiv.
war-deploy	Kopiert das WAR-Archiv in einen lokal installierten JBoss-Server.
ear	Packt das JAR-Archiv, das WAR-Archiv und die notwendigen Deskriptoren zusammen in ein EAR-Archiv.
remote-jar-deploy	Kopiert via ssh-Protokoll das JAR-Archiv auf einen entfernt installierten JBoss-Server.
remote-war-deploy	Kopiert via ssh-Protokoll das WAR-Archiv auf einen entfernt installierten JBoss-Server.
remote-ear-deploy	Kopiert via ssh-Protokoll das EAR-Archiv auf einen entfernt installierten JBoss-Server.
clean	Löscht alle automatisch erzeugten Quelltextdateien, alle erzeugten Deskriptoren, Archive und alle Bytecode-Dateien.

Die dazu notwendige build.xml Datei hat den folgenden Inhalt:

```
<project name="DA" default="remote-ear-deploy" basedir=".">

<property name="jboss.home" value="/local/frech/jboss-3.2.1" />
<property name="xdoclet.home" value="/local/frech/xdoclet-1.2" />
<property name="ant.home" value="/local/frech/jakarta-ant-1.5.1" />
<property name="log4j.jar"
  value="/local/frech/jakarta-log4j-1.2.7/dist/lib/log4j-1.2.7.jar" />
<property name="tapestry.home" value="/local/frech/Tapestry-3.0-beta-1a" />
<property name="lucene.jar" value="/local/frech/lucene-1.2/lucene-1.2.jar" />

<taskdef name="antscp" classname="net.ericalexander.antscp.AntSCP"
```

```
  classpath="/local/frech/AntSCP/AntSCP.jar:
  /local/frech/AntSCP/newestMindtermSCP.jar"/>

<taskdef name="ejbdoclet" classname="xdoclet.modules.ejb.EjbDocletTask">
        <classpath>
                <fileset dir="${xdoclet.home}/lib/">
                        <include name="*.jar"/>
                </fileset>
                <filelist dir="${jboss.home}/client/"
                          files="jboss-j2ee.jar"/>
        </classpath>
 </taskdef>


<target name="xdoclet">
  <ejbdoclet destdir="generated/"
                       ejbspec="2.0">
    <fileset dir="jar/" includes="**/*Bean.java"/>
    <homeinterface/>
    <remoteinterface/>
    <localinterface/>
    <localhomeinterface/>
    <entitypk/>
    <entitycmp/>
    <session/>
    <deploymentdescriptor destdir="output/jar/META-INF/" />
    <jboss version="3.0"
                destdir="output/jar/META-INF/"
                datasource = "java:/PostgresDS"
                datasourcemapping="PostgreSQL" />
   </ejbdoclet>
</target>

<target name="compile" depends="xdoclet">
  <javac destdir="output/jar/"
         classpath="${jboss.home}/client/jboss-j2ee.jar;${lucene.jar}"
         listfiles="yes">
        <src path="."/>
        <include name="jar/**/*.java"/>
        <include name="generated/**/*.java"/>
  </javac>
</target>

<target name="jar" depends="compile">
  <jar destfile="output/ejbs.jar"
       basedir="output/jar/" />
</target>

<target name="deploy" depends="jar">
  <copy file="output/ejbs.jar" todir="${jboss.home}/server/default/deploy/"/>
</target>
```

```
<target name="undeploy">
  <delete file="${jboss.home}/server/default/deploy/ejbs.jar"
          failonerror="false"/>
</target>

<target name="war-compile">
  <mkdir dir="output/war/WEB-INF/classes/"/>
  <javac destdir="output/war/WEB-INF/classes/"
     classpath="${jboss.home}/server/default/lib/javax.servlet.jar:
       ${tapestry.home}/lib/tapestry-3.0-beta-1a.jar:
       ${jboss.home}/client/jboss-j2ee.jar:output/jar/:
       ${lucene.jar}" listfiles="yes">
         <src path="war/src/"/>
         <include name="**/*.java"/>
  </javac>
</target>

<target name="war" depends="war-compile">
  <copy todir="output/war/">
    <fileset dir="war/resources/"/>
  </copy>
  <copy todir="output/war/WEB-INF/classes/app/">
    <fileset dir="war/tapestry/" />
  </copy>
  <jar destfile="output/frontend.war"
       basedir="output/war/" />
</target>

<target name="war-deploy" depends="war">
  <copy file="output/frontend.war"
        todir="${jboss.home}/server/default/deploy/"/>
</target>

<target name="remote-war-deploy" depends="war">
  <antscp
        remoteHost="ludo"
        remoteUserName="frech"
        remotePassword="geheim"
        localFile="/home/frech/DA/storage/src/output/frontend.war"
        remoteFile="/tmp/frech/jboss/server/default/deploy/frontend.war"
  />
</target>

<target name="remote-jar-deploy" depends="jar">
  <antscp
        remoteHost="ludo"
        remoteUserName="frech"
        remotePassword="geheim"
        localFile="/home/frech/DA/storage/src/output/ejbs.jar"
        remoteFile="/tmp/frech/jboss/server/default/deploy/ejbs.jar"
```

```
    />
  </target>

  <target name="ear" depends="jar,war">
    <mkdir dir="output/ear/META-INF/"/>
    <copy todir="output/ear/META-INF/">
      <fileset dir="ear/" />
    </copy>
    <copy file="output/ejbs.jar" todir="output/ear/"/>
    <copy file="output/frontend.war" todir="output/ear/"/>
    <jar destfile="output/figure.ear"
         basedir="output/ear/" />
  </target>

  <target name="remote-ear-deploy" depends="ear">
    <antscp
          remoteHost="ludo"
          remoteUserName="frech"
          remotePassword="geheim"
          localFile="/home/frech/DA/storage/src/output/figure.ear"
          remoteFile="/tmp/frech/jboss/server/default/deploy/figure.ear"
    />
  </target>

  <target name="clean">
    <delete includeEmptyDirs="true" failonerror="false">
      <fileset dir="output"/>
      <fileset dir="generated"/>
    </delete>
  </target>

  </project>
```

Die Verwendung von Ant für das automatische Erzeugen von Quelltext, für das Kompilieren, Packen und Installieren des ICApps-Systems hat die folgenden Vorteile:

- Ant ist unabhängig von einer Entwicklungsumgebung. Für die bekanntesten Entwicklungsumgebungen besteht aber eine Ant-Integration. Somit bleibt das Build-System bei einem Wechsel der Entwicklungsumgebung erhalten.

- Die aufgelisteten Vorgänge können auch ohne eine Entwicklungsumgebung aufgerufen werden. Bei komplexeren Systemen gewinnt das Build-System in zunehmender Weise gegenüber dem Quelltext an Wichtigkeit, bei der Verwendung von Ant kann es mit dem Quelltext verteilt werden.

C.2 AntSCP

AntSCP ist ein Task für Ant, mit dem (analog zu scp) Dateien über das ssh-Protokoll auf einen entfernten Rechner kopiert werden können. Der Einsatz dieses Tasks ermöglicht es, das Deployment nicht nur lokal, sondern auch auf einem entfernten JBoss-Applikationsserver automatisiert auszuführen. Dies wird auch durch die in JBoss vorhandene „Hot Deployment“-Funktionalität ermöglicht.

Bei „Hot Deployment“ muss das entsprechende Applikationsarchiv (JAR, WAR oder EAR) nur in das deploy/ Verzeichnis kopiert werden. JBoss erkennt dann diese Datei als neu hinzugefügt und stößt den restlichen Deployment-Prozess selbständig an. Ist die Datei bereits vorhanden und damit eine ältere Version der Applikation bereits installiert gewesen, so bemerkt JBoss die Aktualisierung des Archivs und fährt zuerst die alte Version der Applikation herunter, bevor die neue Version gestartet wird.

Damit der AntSCP-Task innerhalb der Ant-Steuerdatei verwendet werden kann, muss dieser am Anfang eingebunden werden. D.h., mit dem folgenden Fragment wird Ant mitgeteilt, für welchen Tasknamen welche Klasse aufgerufen werden soll:

```
<taskdef name="antscp" classname="net.ericalexander.antscp.AntSCP"
classpath="/local/frech/AntSCP/AntSCP.jar:
/local/frech/AntSCP/newestMindtermSCP.jar"/>
```

In diesem Fragment wird auch das im AntSCP-Paket vorhandene JAR-Archiv in den Klassenpfad aufgenommen, damit Ant die für AntSCP notwendigen Klassen aus diesem Archiv laden kann.

Eine weitere Integration des Ant-Systems mit AntSCP in die Entwicklungsumgebung namens Eclipse zeigte einen noch in AntSCP vorhandenen Fehler auf. Im Gegensatz zur Kommandozeilenausführung besteht in Eclipse nach der Ausführung eines Ant-Targets der aufrufende Prozess weiterhin. Damit werden nicht automatisch alle noch verbliebenen Threads beendet, sondern diese Threads können weiterlaufen. AntSCP startet einen Thread, der in regelmäßigen Abständen einen neuen Schlüssel für die Verschlüsselung erzeugt. Dieser Thread sollte nach dem Beenden aller ssh-Verbindungen angehalten werden. Durch einen Fehler in AntSCP wurde dies aber bisher nicht getan. In der Kommandozeilenumgebung fällt dieser Fehler nicht auf, da bei Beendigung des Ant-Prozesses auch alle noch bestehenden Threads angehalten werden. Durch den fortbestehenden Prozess bei der Integration in Eclipse fiel dieser Fehler jedoch auf. Bei jeder Ausführung von AntSCP wurde ein solcher zusätzlicher Thread gestartet und nicht mehr angehalten. Diese Threads werden in kurzen Zeitintervallen aktiv und verbrauchen dann einen kurzen Moment lang Rechenzeit. Diese Rechenzeit stand dem restlichen

System nicht mehr zur Verfügung, wodurch es durch mehrfaches Ausführen von Ant mit AntSCP fast bis zum Stillstand gebracht werden konnte.

Der folgende Patch auf den Sourcecode in der Version „1.2.1 SCP 3“ des AntSCP korrigiert diesen Fehler:

```
diff -r -u origsrc/mindbright/ssh/SSHClient.java src/mindbright/ssh/SSHClient.java
--- origsrc/mindbright/ssh/SSHClient.java 2003-03-07 13:38:50.253871000 +0100
+++ src/mindbright/ssh/SSHClient.java 2003-03-07 11:59:04.269999000 +0100
@@ -53,6 +53,8 @@
  * @see SSHSocketFactory
  * @see SSHSocketImpl */
  public class SSHClient extends SSH {
+
+ SecureRandom rand = null;
  static public class AuthFailException extends IOException {
  public AuthFailException(String msg) {
@@ -324,6 +326,8 @@
  }
  public void forcedDisconnect() {
+ if(rand != null && rand.updater != null)
+ rand.updater.stop();
  if(controller != null)
  controller.sendDisconnect("exit");
  else if(interactor != null)
@@ -584,7 +588,7 @@
  }
  void generateSessionKey() {
- SecureRandom rand = secureRandom();
+ rand = secureRandom();
  sessionKey = new byte[SESSION_KEY_LENGTH / 8];
  rand.nextBytes(sessionKey);
  rand.startUpdater();
```

C.3 Eclipse mit JBoss-IDE-Plugin

Eclipse ist eine integrierte Entwicklungsumgebung (engl. *Integrated Development Environment*, IDE) die in der Programmiersprache Java im Auftrag von IBM entwickelt wurde und unter einer freien Lizenz (vgl. [OSI03]) vertrieben wird. Im Kern von Eclipse befindet sich eine minimale Plattform, die Grundfunktionalitäten zur Verfügung stellt und das Management von Plugins übernimmt. Jegliche für die IDE notwendige Funktionalität, die über die Grundplattform hinaus geht, wird durch Plugins erbracht. So ist es möglich, die Eclipse-Plattform für verschiedene Programmiersprachen und Einsatzzwecke durch das Hinzunehmen oder Weglassen von Plugins zu konfigurieren. Die Eclipse Java-Entwicklungsumgebung

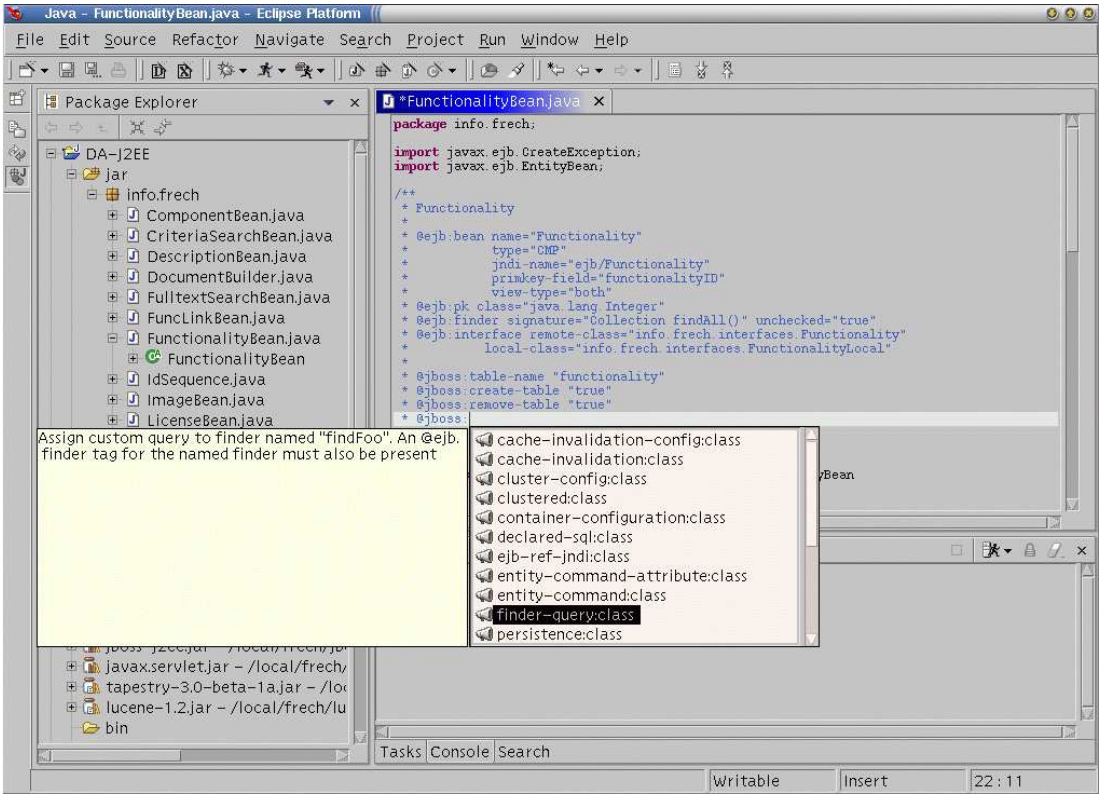

Abbildung C.1: *Durch das JBoss-Plugin wird die „Content Assist“-Funktionalität in Eclipse um die deklarativen Tags für Xdoclet erweitert.*

besteht in der Standardkonfiguration aus der Grundplattform und ca. 60 weiteren Plugins. Darin enthalten sind schon Plugins zu Integration von Ant in Eclipse.

Neben Erleichterungen wie einem Plugin für XML-Editierunterstützung und Verifikation anhand der DTD wurde insbesondere das JBoss-IDE Plugin verwendet. Dieses erweitert die „Content Assist“ genannte Funktionalität von Eclipse um die für Xdoclet verwendbaren Tags. Dabei werden sowohl die allgemeinen Xdoclet-Tags als auch die speziell für JBoss in Xdoclet enthaltenen Tags unterstützt (siehe Abbildung C.1).

Durch die enthaltene Ant-Integration (siehe Abbildung C.2) können die Ant-Targets vorausgewählt und durch die Auswahl eines Befehls innerhalb von Eclipse ausgeführt werden. Somit lässt sich in Eclipse der gesamte Prozess mit Quelltexterzeugung, Kompilation, Packen und Installation komfortabel starten.

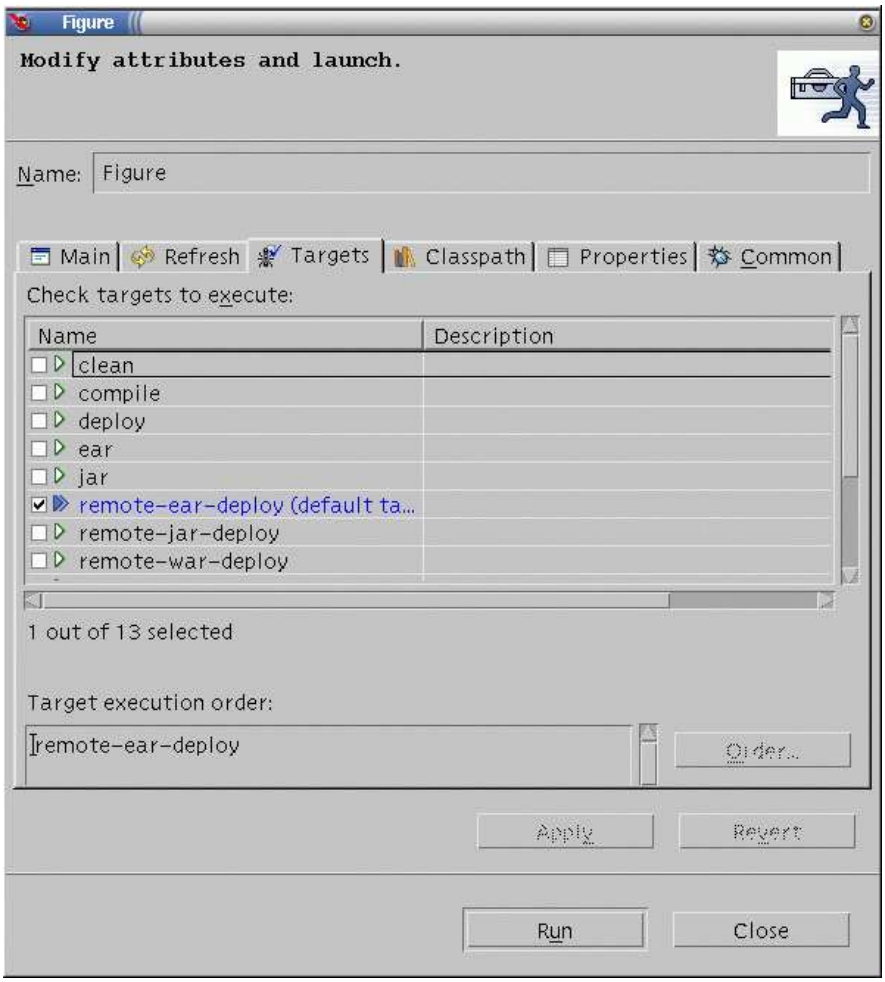

Abbildung C.2: *Die Konfiguration der auszuführenden Ant-Targets in Eclipse.*

Literaturverzeichnis

[ASF] Internet-Seite der „Apache Software Foundation“: http://www.apache.org/foundation/

[BRA00] Bray, Tim/Paoli, Jean/Sperberg-McQueen, C. M./Maler, Eve: *Extensible Markup Language (XML) 1.0*. 2. Auflage, verabschiedet 6. Oktober 2000, http://www.w3.org/TR/2000/REC-xml-20001006

[C2W] Cunningham & Cunningham, Inc.:*„Object Relational Tool Comparison“*-Seite des WikiWebs, http://c2.com/cgi/wiki?ObjectRelationalToolComparison

[CHE02] Cheung, Susan/Matena, Vlada: *Java Transaction API (JTA) Specification*. Version 1.0.1B, Sun Microsystems, 5. November 2002, http://java.sun.com/products/jta/

[COW01] Coward, Danny: *Java Servlet API Specification*. Version 2.3, Final Release, Sun Microsystems, 17. September 2001, http://www.jcp.org/aboutJava/communityprocess/final/jsr053/

[C-SRC] Internet-Seite des Unternehmens ComponentSource: http://www.componentsource.com/

[DEVD] Internet-Seite des Unternehmens Dev Direct Ltd.: http://www.devdirect.com/

[DIT02] Dittert, Kerstin: *Agile Prototypen: Evolution ohne Dogma*. In: *OBJEKTspektrum*, Heft 6, 2002, http://www.sigs.de/publications/os/2002/06/dittert_OS_06_02.pdf

[FIE99] Fielding, R./UC Irvine/Gettys J. et al.: *RFC 2616 (Hypertext Transfer Protocol - HTTP/1.1)*. Juni 1999, ftp://ftp.isi.edu/in-notes/rfc2616.txt

[FRA94] Frakes, William B./Pole, Thomas P.: *An empirical study of representation methods for reusable software components*. In: *IEEE Transactions on Software Engineering*, 20. Jg. Heft 8 (1994), Seite 617-630

[FUR87] Furnas, George W./Landauer, Thomas K./Gomez, Louis M./Dumais, Susan T.: *The Vocabulary Problem in Human-System Communication.* In: *Communications of the ACM*, 30. Jg. Heft 11 (1987), Seite 964-971

[GIG01] Carl Zetie: *Market Overview: Data Modeling Tools and Trends, 2001 Update.* Giga Information Group, 30. April 2001

[GOE00] Goetz, Brian: *The Lucene search engine: Powerful, flexible, and free.* In: *Java World*, September 2000, `http://www.javaworld.com/javaworld/jw-09-2000/jw-0915-lucene.html`

[HEN82] Henninger, Scott: *Using Iterative Refinement to Find Reusable Software.* In: *IEEE Software*, 11. Jg. Heft 5 (September 1994), Seite 48-59

[HEN97] Henninger, Scott: *An Evolutionary Approach to Constructing Effective Software Reuse Repositories.* In: *ACM Transactions on Software Engineering and Methodology*, 6. Jg. Heft 2 (April 1997), ACM, New York, Seite 111-140, `http://pooh.unl.edu/~scotth/papers/TOSEM-henninger97.ps`

[HOF03] Hoffmann, Karsten: *IT-Projektmanagement in der modernen Softwareentwicklung.* In: *Projektmanagement aktuell*, 14. Jg. Heft 1 (1/2003), GPM Deutsche Gesellschaft für Projektmanagement e.V., ISSN 0942-1017, Seite 18-28

[HUS02] Husted, Ted N./Dumoulin, Cedric/Franciscus, George/Winterfeldt, David: *Struts in Action.* Manning, Greenwich November 2002, ISBN 1-930110-50-2

[IBM02] IBM Corporation: *IBM WebSphere Studio Application Developer for Linux and Windows, V5.0*, 3. Dezember 2002, `http://www-306.ibm.com/common/ssi/rep_ca/0/897/ENUS202-330/ENUS202-330.PDF`

[JBOSS] Internet-Seite des JBoss-Projektes: `http://www.jboss.org/`

[JSR153] Internet-Seite des „Java Specification Requests 153: Enterprise JavaBeans 2.1“: `http://www.jcp.org/en/jsr/detail?id=153`

[KRA88] Krasner, Glenn E./Pope, Stephen T.: *A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 system.* In: *Journal of object oriented programming*, 1. Jg. Heft 3 (August/September 1988), Seite 26-49, `http://www.ccmrc.ucsb.edu/~stp/PostScript/mvc.pdf`

[LIN82] Lindig, Christian: *Concept-Based Component Retrieval.* In: *Working Notes of the IJCAI-95 Workshop on Formal Approaches to the Reuse*

of Plans, Proofs, and Programs, August 1995, Seite 21-25, http://www.st.cs.uni-sb.de/~lindig/papers/ijcai-95/ijcai.pdf

[MIC01] DeMichiel, Linda G./Yalçinalp, L. Ümit /Krishnan, Sanjeev: *Enterprise JavaBeans Specification*. Version 2.0, Sun Microsystems, 22. August 2001

[MUE97] Mühlhäuser, Max: *Workshop Reader of the 10th European Conference on Object-Oriented Programming ECOOP '96 Linz*. dpunkt, Heidelberg 1997, ISBN 3-920993-67-5

[OSI03] http://www.opensource.org/licenses/index.php, 2003

[PEL01] Pelegrí-Llopart, Eduardo: *JavaServer Pages (JSP) Specification*, Version 1.2, Sun Microsystems, 17. September 2001, http://www.jcp.org/aboutJava/communityprocess/final/jsr053/

[PKW03] PKWARE Inc.: *White Paper „.ZIP File Format Specification"*. 2003, http://www.pkware.com/products/enterprise/white_papers/appnote.html

[POS02] The PostgreSQL Global Development Group: *PostgreSQL 7.3.2 Administrator's Guide*. In: *PostgreSQL 7.3.2 Documentation*, 2002, http://www.postgresql.org/docs/

[PGSQL] Internet-Seite des PostgreSQL-Projektes: http://www.postgresql.org/

[SES99] Seshadri, Govind: *Understanding JavaServer Pages Model 2 architecture*. JavaWorld, 29. Dezember 1999, http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc.html

[SHA01] Sharma, Rahul: *J2EE Connector Architecture Specification*. Version 1.0, Sun Microsystems, 22. August 2001, http://java.sun.com/j2ee/connector/download.html

[SHI02] Ship, Howard Lewis: *Tapestry Developer's Guide*. 2002, http://jakarta.apache.org/tapestry/doc/DevelopersGuide/DevelopersGuide.pdf

[SIN02] Singh, Inderjeet/Stearns, Beth/Johnson, Mark: *Designing Enterprise Applications with the J2EE Platform*. 2. Auflage, Addison-Wesley, 15. Juni 2002, http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/

[STARK] Stark, Scott/The JBoss Group: *JBoss Administration and Development*. 2. Auflage, http://jboss.org/index.html?module=html&op=userdisplay&id=docs/index

[SUN99a] Sun Microsystems: *Java Naming and Directory Interface Application Programming Interface (JNDI API)*. Version 1.2, 14. Juli 1999, ftp://ftp.javasoft.com/docs/j2se1.3/jndi.pdf

[SUN99s] Sun Microsystems: *Java Naming and Directory Interface Service Provider Interface (JNDI SPI)*. Version 1.2, 14. Juli 1999, ftp://ftp.javasoft.com/docs/j2se1.3/jndispi.pdf

[SUN00] Sun Microsystems: *JavaMail API Design Specification*. Version 1.2, September 2000, http://java.sun.com/products/javamail/JavaMail-1.2.pdf

[SUN01] Sun Microsystems: *Java 2 Platform, Enterprise Edition (J2EE) Specification*. Final Release. 22. August 2001, http://java.sun.com/j2ee/j2ee-1_3-fr-spec.pdf

[SUN02] Sun Microsystems: *JCP 2: Process Document*. Version 2.5, 29. Oktober 2002, http://www.jcp.org/en/procedures/jcp2

[SUNST] Sundstrom, Dain/The JBoss Group: *JBossCMP*. 2. Auflage, http://jboss.org/index.html?module=html&op=userdisplay&id=docs/index

[TAPES] Internet-Seite des Tapestry-Projekts: http://jakarta.apache.org/tapestry/index.html

[WEI03] Weinschenk, Carl: *The Application Server Market Is Dead; Long Live the Application Platform Market*. ServerWatch, 11. Juli 2003, http://www.serverwatch.com/tutorials/article.php/10825_2234311_1

[WHI99] White, Seth/Hapner, Mark: *JDBC 2.0 Standard Extension API*. Sun Microsystems, 7. Dezember 1998, http://java.sun.com/products/jdbc/jdbc20.stdext.pdf

[WIL82] Williams, Michael D./Tou, Frederich N./Fikes, Richard E./Henderson, Austin/Malone, Thomas: *RABBIT: Cognitive Science in Interface Design*. In: *Fourth Annual Conference of the Cognitive Science Society*, 1982, Seite 82-85

[WIL84] Williams, Michael David: *What Makes Rabbit Run?*. In: *International Journal of Man-Machine Studies*, 21. Jg. (1984), Seite 333-352

[XDOC] Xdoclet Internet-Seite: http://xdoclet.sourceforge.net/

[XOP91] X-Open Company: *Distributed Transaction Processing: The XA Specification*. X/Open Company Ltd., England, 1991, ISBN 1-872630-24-3