Techniken und Aspekte zur Realisierung proaktiver Informationssysteme

Dissertation
der Fakultät für Informations- und Kognitionswissenschaften
der Eberhard-Karls-Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.)

vorgelegt von
Dipl. -Inform. Ralf-Dieter Schimkat
aus Zürich, Schweiz

Tübingen
2003

Tag der mündlichen Qualifikation: 12. Februar 2003
Dekan: Prof. Dr. Ulrich Güntzer
1. Berichterstatter: Prof. Dr. Wolfgang Küchlin
2. Berichterstatter: Prof. Dr. Dietmar Seipel
Universität Würzburg

Vorwort

Die vorliegende Arbeit fertigte ich während meiner Tätigkeit als wissenschaftlicher Mitarbeiter im Arbeitsbereich *Symbolisches Rechnen* des Wilhelm-Schickard-Instituts für Informatik an der Universität Tübingen an.
Sehr viele Menschen haben durch ihre vielfältigen Impulse und wertvollen Hinweise zum Gelingen dieser Arbeit beigetragen. Sie mögen mir verzeihen, daß ihre explizite Nennung und eine Auflistung ihrer Verdienste an dieser Stelle nicht möglich ist – ihre Bedeutung für meine Arbeit wird dadurch nicht im geringsten geschmälert. Namentlich seien nachfolgend insbesondere all jene Personen genannt, ohne deren Zutun diese Arbeit in ihrer jetzigen Form keinesfalls zustande gekommen wäre.
Mein besonderer Dank gilt Herrn Prof. Dr. W. Küchlin, dessen Vertrauen in meine Arbeit nicht nur die Bearbeitung der gewählten Thematik ermöglichte, sondern mich auch über die gesamte Zeit hinweg in meinem Vorhaben bestärkte. Sein Verständnis, seine fachlichen Ratschläge, aber auch seine konstruktive und motivierende Unterstützung bildeten eine optimale Grundlage bei der Durchführung und Fertigstellung dieser Arbeit.
Zudem hatte ich das große Glück, in Herrn Prof. Dr. D. Seipel einen überaus konstruktiven Zweitgutachter zu gewinnen. Mein Dank bezieht sich daher nicht allein auf seine Bereitschaft, als Zweitkorrektor zu fungieren, sondern in gleichem Maße auf seine fortwährende Unterstützung, die wiederholt zur Lösung wissenschaftlicher Probleme im Rahmen der vorliegenden Arbeit beitrug.
Herrn Prof. Dr. U. Güntzer danke ich sehr herzlich für sein unbürokratisches Vorgehen, als es darum ging, mich während meiner Arbeit am Lehrstuhl für Datenbanken aufzunehmen. Dank seiner Unterstützung war somit nicht nur der Zugriff auf eine adäquate Infrastruktur gewährleistet, sondern auch die enge Kommunikation mit zahlreichen Wissenschaftlern aus verwandten Feldern sichergestellt.
Eben diese Kommunikation erwies sich in einer Vielzahl von Gesprächen und Diskussionen als äußerst gewinnbringend. Ein besonderer Dank gebührt daher meinen Kollegen am Wilhelm-Schickard-Institut, insbesondere im Arbeitsbereich Symbolisches Rechnen und im Lehrstuhl für Datenbanken, die mir sowohl mit konstruktiver Kritik als auch mit beflügelnder Inspiration zur Seite standen. Bei meinen Kollegen B. Heumesser und A. Ludwig möchte ich mich für ihre großzügig entgegengebrachte Gastfreundschaft am Lehrstuhl für Datenbanken bedanken.
Der Fa. T-Systems DMS – insbesondere den Herren A. Bittner und Herrn R. Krautter – danke ich für die mehrjährige, finanzielle Unterstützung dieser Arbeit im Rahmen einer Industriekooperation. Auf der Basis ihrer jederzeit kollegialen und vertrauensvollen Zusammenarbeit erhielt ich wiederholt die Möglichkeit, anhand interessanter Aufgaben im Geschäftsfeld Dokumentenmanagement die betriebswirtschaftliche Relevanz universitärer Forschung zu verifizieren.
Mein Dank gilt zudem den Herren M. Häusser, S. Schroeder, M. Schmidt-Dannert und B. Cassar, die als Diplomanden die Entwicklung einzelner Komponenten bereichert und ihre Implementierung forciert haben. Die Vergabe plakativer Namen

für die erarbeiteten Komponenten erfolgte im leidenschaftlichen Disput mit Herrn M. Geiger und seinen Lateinkenntnissen.
Angesichts ihrer umfassenden und nur schwer zu kategorisierenden Unterstützung möchte ich insbesondere Michael Friedrich, Stefan Müller und Gerd Nusser erwähnen, die mir sowohl fachlich als auch freundschaftlich während dieser Arbeit stets zur Seite standen. Die mit ihnen geführten Diskussionen waren und sind stets ein Quell neuer und interessanter Ideen, ohne die für mich eine erfolgreiche, wissenschaftliche Forschungstätigkeit undenkbar wäre.
Mit großer Freude und aus ganzem Herzen bedanke ich mich schließlich bei meinen Eltern und bei Susan für ihre langjährige Unterstützung, ihre große Geduld und ihr uneingeschränktes Vertrauen. Ohne sie hätte ich diese Arbeit weder beginnen noch zu Ende bringen können.

Ralf-Dieter Schimkat

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listing-Verzeichnis

Kapitel 1

Überblick und Aufbau der Dissertation

In heutigen Informationssystemen werden Dokumente und andere Informationsträger typischerweise in Datenbanken oder Dateisystemen verwaltet. Die Applikationslogik wird in Softwarekomponenten – beispielsweise in Applikationsservern – hinterlegt, welche für die korrekte Ablaufsteuerung innerhalb eines Informationssystems verantwortlich sind. Die Abläufe sind oft geprägt von simplen Anfrage/Antwort-Interaktionen, in denen der Benutzer häufig als einziger Initiator von Informations- und Kontrollflüssen in Erscheinung tritt. Dokumente übernehmen in solchen Informationssystemen lediglich die Rolle eines passiven Containers zur Verwaltung digitaler Dateninhalte.

Im Rahmen dieser Arbeit werden verschiedene Techniken und Aspekte zur Realisierung proaktiver Informationssysteme untersucht, die sich durch einen proaktiven Dokumentenbegriff auszeichnen. Im Mittelpunkt steht eine Objektivierung von Dokumenten, wodurch Dokumente sowohl daten- als auch verhaltensbezogene Eigenschaften erhalten. Dies führt zur konzeptionellen Annäherung von Dokumenten an Objekte im Sinne der objektorientierten Programmierung.

Die in dieser Arbeit vorgenommene Fokussierung auf eine dokumentenbezogene Sichtweise auf Informationssysteme entspricht der empirischen Bedeutung dokumentenzentrierter Informationssysteme. Insbesondere die selbstverständliche Verwendung des World-Wide-Web im Alltag illustriert den hohen Durchdringungsgrad von derartigen Informationssystemen in der gesamten Gesellschaft. So spielen Dokumente zunehmend eine maßgebliche Rolle sowohl für die Spezifikation von digitalen Inhalten als auch für den flexiblen Datenaustausch in Informationssystemen.

In der vorliegenden Dissertation wird eine Taxonomie für Informationssysteme erörtert, die im wesentlichen aus den drei Dimensionen der Benutzersicht, des Dokumentenbegriffs und der verwendeten Technologie besteht. Dabei werden die Eigenschaften der jeweiligen Dimension ausführlich erörtert. Zudem erfolgt eine inhaltliche Abgrenzung von mehrstufigen Client/Server- und netzwerkbasierten

Informationssystemen. Der Schwerpunkt der entwickelten Taxonomie liegt jedoch auf passiven, aktiven und proaktiven Dokumenten.
Die konzeptionellen und technischen Grundlagen zur Realisierung proaktiver Informationssysteme in dieser Arbeit bilden sogenannte *Living Documents*, die agentenbasierte Komponenten zur Verwaltung der Applikationslogik und semistrukturierte Datenbasen zur Organisation von Kontextinformationen vereinen. Um den vorteilhaften Einsatz von *Living Documents* zu unterstreichen, wird die Erstellung proaktiver Informationssysteme exemplarisch anhand der Transformation eines datenbankzentrierten Web-Informationssystems demonstriert. Zudem werden spezifische Eigenschaften proaktiver Informationssysteme – wie beispielsweise automatisierte Interaktionsformen – diskutiert.
Im einzelnen stellt sich der Aufbau der Dissertation folgendermaßen dar:

Kapitel 2 gibt eine Einführung in Informationssysteme. Neben den grundsätzlichen Systemkomponenten werden sogenannte Mediatoren als weitere wichtige Bestandteile von Informationssystemen vorgestellt und diskutiert. Schließlich werden zwei typische Architekturen von Informationssystemen – Client/Server- und netzwerkbasierte Informationssysteme – ausführlich erörtert.

Kapitel 3 stellt eine Taxonomie für Informationssysteme vor. Die in der vorliegenden Arbeit vorgenommene Klassifizierung orientiert sich an den in Kapitel 2 beschriebenen konzeptionellen Säulen von Informationssystemen. Die drei wesentlichen Merkmale der erarbeiteten Taxonomie – die *Benutzersicht*, der *Dokumentenbegriff* und die eingesetzte *Technologie* – werden im Detail erläutert. Die vorgestellte Taxonomie ermöglicht eine klare Einordnung komplexer Sachverhalte in Informationssystemen. Außerdem erfolgt eine Abgrenzung zu bestehenden Forschungsarbeiten und eine inhaltliche Einordnung der verschiedenen Aspekte zur Realisierung von Informationssystemen.

Kapitel 4 erörtert den Entwurf und die Realisierung des leichtgewichtigen Applikationsservers *Respondeo*. Es werden architektonische Eigenschaften und implementierungsrelevante Details vorgestellt. Dabei steht neben dem spezifischen Applikationsmodell von *Respondeo* vor allem der Entwurf des Nachrichtenbusses im Mittelpunkt der Betrachtung. Schließlich wird eine Reihe verschiedener Informationssysteme und Dienste beschrieben, die in *Respondeo* integriert wurden.

In **Kapitel 5** wird das Agentenframework *Okeanos* vorgestellt. *Okeanos* ist ein Framework zur Erstellung von mobilen Agenten in Java. Nach einer inhaltlichen Einordnung und Abgrenzung von mobilen Agenten werden die spezifischen Eigenschaften des Frameworks und der *Okeanos*-Agenten erläutert. Das Verhalten von *Okeanos*-Agenten wird entweder in objektorientierter

oder deklarativer Form über die Bestimmung von Regeln festgelegt. Generell bilden *Okeanos*-Agenten einen wesentlichen Bestandteil der Realisierung von proaktiven Informationssystemen, wie ausführlich in Kapitel 9 über *Living Documents* erläutert wird.

Kapitel 6 stellt eine spezielle Auszeichnungssprache (*Markup-Language*) für die Repräsentierung von Applikations- und Systemzuständen vor. In *SpectoML* werden Applikationszustände in allgemeiner und interoperabler Form dargestellt. Die speziell entwickelte Anfragesprache für *SpectoML*-Dokumente zeichnet sich durch performante Ausführungen aus. Der dazugehörige XML-Prozessor ist von geringer Codegröße. Ein weiteres Merkmal der *SpectoML* ist die Trennung zwischen der inhaltlichen Bedeutung und der syntaktischen Repräsentation der Zustände. Schließlich werden einige der Einsatzgebiete von *SpectoML* beschrieben, die anhand von Fallstudien evaluiert werden.

In **Kapitel 7** wird die Monitoring-Infrastruktur *Specto* vorgestellt. In dieser Fallstudie geht es um den Entwurf eines verteilten, XML-basierten Monitoringsystems. *Specto* zeichnet sich durch ein flexibles Monitoring-Modell aus, das im wesentlichen aus drei Phasen – der Generierung, Verwaltung und Präsentation von Zuständen – besteht. Im Mittelpunkt steht die systematische und einheitliche Bestimmung von System- und Applikationszuständen auf Grundlage der in Kapitel 6 diskutierten *SpectoML*. Zudem wird veranschaulicht, wie Zustandsinformationen in interaktiver oder automatisierter Form in einem netzwerkbasierten System verteilt werden. Abschließend wird *Specto* in die in Kapitel 3 entwickelte Taxonomie für Informationssysteme eingeordnet.

In **Kapitel 8** werden weitere Fallstudien, die verschiedene Techniken und Aspekte der *SpectoML* und des Agentenframeworks *Okeanos* näher untersuchen, überblicksartig erläutert.

In Abschnitt 8.1 wird das Testframework *Tempto* erörtert, in dessen Zentrum XML-basierte Testdokumente stehen. Ein Testdokument beinhaltet Informationen über den aktuellen Zustand eines Testfalls. Generell werden zur Generierung, Verwaltung und Präsentation der XML-basierten Testdokumente die gleichen Infrastrukturen benutzt, die bei der Monitoring-Infrastruktur von *Specto* zum Einsatz kommen. Obwohl *Tempto* und *Specto* unterschiedliche Anwendungsdomänen adressieren, ermöglicht *SpectoML* als allgemeine Auszeichnungssprache für Applikations- und Systemzustände eine uniforme Verwaltung der Dokumente.

Im Mittelpunkt der Fallstudie in Abschnitt 8.2 steht eine verteilte Berechnung aus dem Bereich des Symbolischen Rechnens. Das Ziel dieser Fallstudie besteht in der Demonstration der Plausibilität eines Einsatzes autonomer und dezentraler Systemkomponenten innerhalb eines verteilten Systems. Zu-

dem wird untersucht, inwiefern die Verwendung von Zustands- und Kontextinformationen basierend auf *SpectoML*-Dokumenten die Informationsflüsse in einem dynamischen, verteilten System positiv beeinflussen können.

Abschnitt 8.4 schildert eine Fallstudie, um bestimmte Inferenztechniken auf der Basis von XML-Dokumenten zu untersuchen. Konkret wird in diesem Abschnitt auf der Grundlage von *SpectoML*-Dokumenten untersucht, wie neue Informationen durch die Anwendung von logischen Inferenzmechanismen aus bestehenden *SpectoML*-Dokumenten gewonnen werden. Zu diesem Zweck wurde ein auf Prolog basierendes Web-Informationssystem entwickelt.

Kapitel 9 erörtert das Konzept der *Living Documents*, welche die konzeptionelle Grundlage zur Realisierung von proaktiven Informationssystemen in der vorliegenden Arbeit bilden. Basierend auf einem proaktiven Dokumentenbegriff ermöglichen sie eine Objektivierung von Dokumenten, die sich durch daten- *und* verhaltensbezogene Eigenschaften auszeichnen. Mobile *Okeanos*-Agenten fungieren als fein-granulare Mediatoren, welche sich für die Verwaltung der Applikations- bzw. Dokumentenlogik eines *Living Documents* verantwortlich zeigen. Zudem besitzt jedes *Living Document* eine semi-strukturierte Datenbasis, in der vielfältige Kontextinformationen strukturiert abgelegt sind und systematisch verwaltet werden.

In **Kapitel 10** werden zwei Fallstudien hinsichtlich des Einsatzes und der Anwendbarkeit von *Living Documents* zur Realisierung von Informationssystemen erörtert.

In Abschnitt 10.1 wird anhand von *PaperBase* – einem Informationssystem zur Verwaltung individuell anpaßbarer Informationsräume im Web – die graduelle Transformation eines typischen mehrstufigen Client/Server-Informationssystems zu einem netzwerkbasierten Informationssystem bestehend aus *Living Documents* diskutiert. In *PaperBaseLD* übernehmen aktivierte Dokumente die Funktionen, welche in traditionellen mehrstufigen Client/Server-Informationssystemen von Applikationsservern oder Datenbanken ausgeübt werden.

In Abschnitt 10.2 werden die Eigenschaften des netzwerkbasierten Informationssystems *Living Hypertext* erörtert. Ein *Living Hypertext* erweitert das im vorherigen Abschnitt vorgestellte netzwerkbasierte Informationssystem *PaperBaseLD* und veranschaulicht die Realisierung von navigationsbasierten Suchmöglichkeiten, wie sie typischerweise innerhalb des Web erfolgen. Zudem werden automatisierte Interaktionsformen zwischen den *Living Documents* diskutiert. Diese dienen zur Erstellung von Themenkomplexen in einem netzwerkartig organisierten Verbund von *Living Documents*.

Abschließend erfolgt eine Einordnung der Informationssysteme *PaperBaseLD* und *Living Hypertext* in die Taxonomie für Informationssysteme.

In **Kapitel 11** wird die Arbeit zusammengefaßt, und die wesentlichen Ergebnisse werden präsentiert. Zudem erfolgt eine Auflistung der im Zusammenhang mit dieser Arbeit veröffentlichten Publikationen.

In **Kapitel 12** wird ein Ausblick auf zukünftige Arbeiten und mögliche Erweiterungen gegeben.

Teil I

Allgemeine Grundlagen und Einführung in Informationssysteme

Kapitel 2

Einleitung

In diesem Kapitel werden die konzeptionellen und technischen Grundlagen zur die Gestaltung proaktiver Informationssysteme erläutert. Ausgehend von dem grundsätzlichen Aufbau von Informationssystemen in Abschnitt 2.1 werden die drei wesentlichen konzeptionellen Säulen von Informationssystemen vorgestellt: Die *Dokumente* bzw. *Informationen,* die eingesetzten *Technologien* und die benutzten *Arbeitsformen*.
Die in Abschnitt 2.2 beschriebenen *Mediatoren* stellen die konzeptionelle Basis für zahlreiche Systemkomponenten in heutigen Informationssystemen dar. Aufgrund ihrer generellen Bedeutung für den Entwurf und die Realisierung von Informationssystemen werden ihre Eigenschaften ausführlich dargelegt.
Schließlich werden in Abschnitt 2.3 *Client/Server-Informationssysteme* und *netzwerkbasierte Informationssysteme* vorgestellt. Neben dem grundsätzlichen Aufbau beider Systeme werden einige der jeweiligen konzeptionellen Eigenheiten beschrieben und erläutert.

2.1 Informationssysteme

Im folgenden Abschnitt wird zunächst der grundsätzliche und schematische Aufbau von Informationssystemen dargelegt. Nachdem die allgemeinen Bestandteile eines Informationssystems – wie beispielsweise die *Datenbasis*, der *Benutzer* und der *Datenlieferant* – beschrieben worden sind, werden einzelne Aspekte des Entwurfs und der Realisierung von Informationssystemen anhand von drei wichtigen konzeptionellen Säulen erläutert. Es sind dies:

- die *Dokumente* bzw. *Informationen,*
- die eingesetzten *Technologien* und
- die benutzten *Arbeitsformen*.

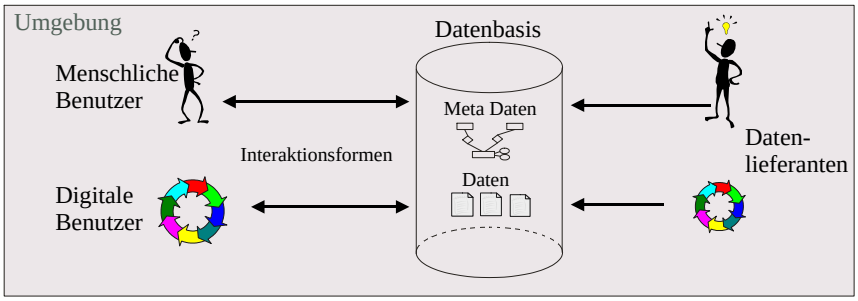


Abbildung 2.1: Schematischer Aufbau von Informationssystemen.

Diese Unterteilung von Informationssystemen orientiert sich an die von Levy und Marshall [LM95] vorgeschlagene grundsätzliche Klassifizierung von digitalen Bibliotheken. Dabei wird der Begriff einer digitalen Bibliothek in der vorliegenden Arbeit zunächst synonym für digitale Informationssysteme bzw. Informationssysteme im allgemeinen benutzt.

2.1.1 Schematischer Aufbau

In Abbildung 2.1 ist ein allgemeiner schematischer Aufbau von Informationssystemen illustriert. Wichtiger Bestandteil eines jeden Informationssystems ist die Existenz einer *Datenbasis*. In digitalen Informationssystemen liegt diese Datenbasis in digitaler Form vor. Der Begriff Datenbasis ist sehr weit gefaßt und beinhaltet jegliche Form von digitalisierten Daten. Grundsätzlich besteht eine Datenbasis aus zwei Teilen, den Metadaten und den eigentlichen Daten. Metadaten sind Daten, die andere Daten beschreiben, um die Nutzbarkeit letzterer zu verbessern [Mar98]. Agosti [AC93] betrachtet Metadaten als Zusatzinformationen, die es ermöglichen, Datenbasen zu beschreiben, zu beobachten und auf sie zuzugreifen. In der vorliegenden Arbeit wird von folgender Begriffsdefinition ausgegangen: Eine Datenbasis besteht gemäß Abbildung 2.1 aus Daten und Metadaten. Der Begriff der Information wird im Zusammenhang mit Datenbasen synonym mit Daten verwendet, d. h. Information erfordert das Vorhandensein von einer bestimmten Form von Metadaten. Demnach beinhalten Daten, die außerhalb des Anwendungskontextes von Datenbasen auftreten, keine Form von Metadaten, was den Zugriff und die Verwaltung dieser Daten wesentlich erschwert.
Ein typisches Beispiel einer Datenbasis ist eine relationale Datenbank, in der die Metadaten als relationales Datenbankschema abgelegt sind. Ein relationales Datenbankschema beruht zumeist auf einem Entity-Relationship-Modell (ER-Modell) [Che76], in dem Eigenschaften und Beziehungen zwischen den Daten beschrieben und bestimmt werden. Das Schema sagt nichts über die individuellen Datenausprägungen aus, sondern bestimmt primär die Metaeigenschaften der Daten.
Der *Benutzer* interagiert mit dem Informationssystem, d.h. er versucht in der Re-

gel ein Informationsdefizit zu beseitigen, in dem er entsprechende Anfragen an das Informationssystem stellt. Primäre Informationsquelle für die Beantwortung der Anfrage ist zunächst die Datenbasis. Es können sowohl der Mensch als auch Softwarekomponenten bzw. -prozesse (z. B. andere Informationssysteme) als Benutzer auftreten. Typisch dafür ist beispielsweise ein sogenannter Agenten-Crawler [CvdBD99] einer Web-Suchmaschine, die andere Web-Informationssysteme kontaktiert und die dabei ermittelten Informationen in einer eigenen Datenbasis ablegt.

Die *Interaktionsform* charakterisiert den Vorgang der Kommunikation eines Benutzers mit der Datenbasis. Dabei werden Kommunikationsschnittstellen in syntaktischer und semantischer Form festgelegt. Um mit einer Datenbasis zu kommunizieren, wird also neben den strukturellen Voraussetzungen auch eine inhaltliche Festlegung der Bedeutung der übertragenen Daten (Informationen) bei der Bestimmung der Interaktionsform vorgenommen. Dabei können semantische Bedeutungen in impliziter oder expliziter Form gegeben sein. Entscheidend für eine erfolgreiche Interaktion mit der Datenbasis ist letztendlich, daß Benutzer die strukturellen und semantischen Gegebenheiten kennen, bevor sie eine Interaktion mit der Datenbasis initiieren.

Ein *Datenlieferant* ist der Autor und Verantwortliche von Daten bzw. Informationen, die in die Datenbasis eines Informationssystems eingespeist werden. Im Fall digitaler Bibliotheken entspricht dies meist dem Bibliothekar, der digitalisierte Dokumente nach dem Vorgang der Kategorisierung in den Informationsbestand der digitalen Bibliothek hinzufügt. Im Falle des Bibliothekars gilt es darauf hinzuweisen, daß dieser keine originäre Autorenfunktion übernimmt, d. h. er selbst ist nicht Autor des betreffenden Dokuments, sondern ist für die Integration dieses Dokuments in die Datenbasis der digitalen Bibliothek verantwortlich. Generell fungieren im Zuge der software-zentrierten Automatisierung von Abläufen auch andere Softwareprozesse als Datenlieferanten für Datenbasen von Informationssystemen. Der zuvor angesprochene Agenten-Crawler ist nicht nur, wie im obigen Beispiel erläutert wird, ein digitaler Benutzer, sondern auch ein digitaler Datenlieferant, der die gesammelten Daten und Informationen der Datenbasis seiner Web-Suchmaschine zur Verfügung stellt.

Ein weiterer Bestandteil eines Informationssystem bildet die *Umgebung*, in welche das Informationssystem eingebettet ist. Dabei gilt es zu hinterfragen, ob es klar abgesteckte Grenzen gibt, in denen das Informationssystem und seine Systemkomponenten agieren. Ferner gilt es zu klären, ob offene und flexible Schnittstellen existieren, die es erlauben, daß sich bidirektionale Informationsflüsse zwischen Informationssystemen und der jeweiligen Umgebung etablieren können. Tennenhouse [Ten00] bezeichnet die Bindeglieder zwischen Informationssystemen und ihrer Umgebung als Sensoren und Aktoren, die letztlich die „virtuelle mit der realen Welt“ verbinden. Über die Sensoren werden Nachrichten und Ereignisse der realen an die virtuelle Welt mitgeteilt, wobei dann die Aktoren in der virtuellen Welt in die Lage versetzt werden, auf diese Ereignisse zu reagieren.

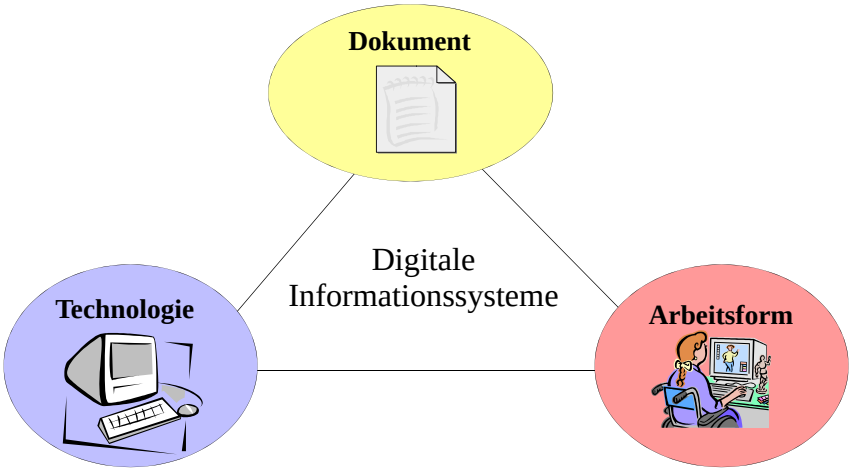


Abbildung 2.2: Die drei wesentlichen Sichtweisen von digitalen Informationssystemen gemäß Levy und Marshall [LM95].

2.1.2 Konzeptionelle Säulen von Informationssystemen

In Abbildung 2.2 sind die drei wesentlichen Sichtweisen auf digitale Informationssysteme illustriert:

- Der zentrale Informationsträger in digitalen Informationsssystemen ist ein *Dokument* [LM95]. Ein Dokument umfaßt eine sehr breite Palette von digitalen Dokumenteninhalten wie z. B. Bilder, Textdokumente (Officedokumente) oder E-Mails .
- Neben den Dokumenten spielt die verwendete *Technologie* eine wesentliche Rolle innerhalb digitaler Informationssysteme. Dokumente sind davon abhängig, daß sie zunächst erzeugt und dann entsprechend verwaltet werden.
- Die *arbeitsorientierte Sichtweise* auf digitale Informationssysteme fokussiert auf die Art und Weise, wie Informationssysteme benutzt werden.

Die dokumenten- und technikorientierten Sichten und die arbeitsorientierte Sicht ergänzen sich bzw. stehen orthogonal zueinander, d. h. die ausschließliche Konzentration auf eine Sichtweise würde eine nicht adäquate Betrachtung von Informationssystemen nach sich ziehen.
Wiederhold [Wie95] diskutiert die vorgestellten Säulen (Dokument, Technologie, Arbeitsweise) im Zusammenhang mit digitalen Bibliotheken, welche in dieser Arbeit als typische Vertreter von Informationssystemen betrachtet werden. Er kommt zu dem Schluß, daß eine Umorientierung und eventuelle Neudefinition der klassischen Rollenbegriffe innerhalb digitaler Bibliotheken erforderlich ist, da die zugrunde liegenden Annahmen nicht notwendigerweise haltbar sind. So müsse die

klassische Rolle des Datenlieferanten generalisiert werden: Es obliegt nicht mehr nur Verlagen, Daten bzw. Informationen für digitale Bibliotheken zu liefern, sondern der Typus des Datenlieferanten kann nach Wiederhold von unterschiedlichen Personen, Institutionen oder weiteren Informationssystemen – als digitale Datenlieferanten – ausgefüllt werden. Wiederholds Betrachtungen unterstützen den gewählten schematischen Aufbau von Informationssystemen in Abbildung 2.2, der zusätzlich zu den eher traditionell besetzten Rollen – wie z. B. menschliche Benutzer und Datenlieferanten – neue Rollenträger innerhalb von Informationssystemen einführt. Derartige Rollenträger sind beispielsweise digitale Benutzer und Datenlieferanten.
In den folgenden Abschnitten werden die oben genannten Säulen von Informationssystemen (Dokument, Technologie, Arbeitsweise) näher beschrieben und erläutert.

2.1.2.1 Dokumente

Traditionell werden digitale Dokumente als fixierte und statisch definierte Ausprägungen von digitalen Artefakten betrachtet [LM95]. Dieser sehr allgemeine Dokumentenbegriff zeichnet sich durch folgende Eigenschaften aus:

- *Zeitliche Dimension*. Ein digitales Artefakt konstituiert sich immer zu einem bestimmten Zeitpunkt. Diese Dokumentenerzeugung ist somit der Ausgangspunkt eines Vorgangs, der temporale Aktivitäten in Form von spatialen Objekten – nämlich Dokumenten – widerspiegelt.

- *Metadaten*. Der Begriff der Ausprägung in der obigen Definition impliziert, daß es sich nicht nur um eine lose Ansammlung digitaler Artefakte, sondern vielmehr um eine strukturierte und inhaltlich wertvolle Instanz bestimmter Informationen handelt. Die zugrunde liegende Struktur und inhaltliche Bedeutung der Dokumente werden als Meta Daten interpretiert, die entweder explizit - beispielsweise in Form eines Datenbankschemas - oder implizit - beispielsweise nur in den Köpfen der Datenlieferanten - gegeben sind.

- *Fixiertheit*. Dokumente bzw. deren digitale Artefakte werden gemäß der obigen Definition statisch definiert, d. h. es wird zu einem bestimmten Zeitpunkt, der Dokumentenerzeugung, die Bedeutung der Dokumenteninhalte festgelegt. Ferner wird die Ausprägung der Dokumenteninhalte hinsichtlich der zeitlichen Dimension fixiert und bleibt so starr und unverändert.

- *Allgemeiner Dokumentenbegriff*. Wenn man digitale Artefakte sehr allgemein interpretiert, stehen diese für nahezu jegliche digitale Inhalte. In dieser Hinsicht entsprechen auch Datenobjekte und -strukturen digitalen Artefakten. Allerdings wird die Allgemeinheit dieses Dokumentenbegriffs durch die oben genannten Einschränkungen – wie beispielsweise die Fixiertheit der Dokumente – relativiert. So ist eine starre Festlegung von Ausprägungen und Bedeutungen digitaler Artefakte nicht geeignet, um dynamische und

sich häufig ändernde reale Gegebenheiten adäquat in Informationssystemen abzubilden. Vielmehr eignet sich eine derartige Definition für Informationssysteme, die sich durch konstante Dokumente auszeichnen. Typische Vertreter derartiger Informationssysteme sind digitale Bibliotheken, in denen die enthaltenen Dokumente (Inhalt und Bedeutung) – wie wissenschaftliche Artikel oder andere digitalisierte Veröffentlichungen – sich über die Zeit hinweg nicht oder nur selten ändern.

Eine erste sinnvolle Erweiterung des obigen Dokumentenbegriffs wird von Levy [Lev94] vorgeschlagen. Er hebt zwei wesentliche Eigenschaften von Dokumenten als Informationsträger in zukünftigen Informationssystemen hervor.

- *Änderungsrate von Dokumenteninhalten*. Innerhalb bestimmter Zeitspannen können sich Dokumenteninhalte ändern. Durch diese konzeptionelle Erweiterung öffnet sich der Dokumentenbegriff einer Vielzahl von möglichen Anwendungsfeldern, die sich vor allem durch dynamische Gegebenheiten (Bedingungen) auszeichnen[1]. Diese Erweiterung ist für einen universellen Dokumentenbegriff von großer Bedeutung, da sich somit programmiersprachennahe Datenstrukturen und -objekte in einen einheitlichen Dokumentenbegriff einordnen lassen.

- *Dauerhaftigkeit*. Die Dauerhaftigkeit bezieht sich auf die Dauer der Gültigkeit von Dokumenten. Sie gibt an, wie lange eine Benutzung von Dokumenten sinnvoll ist. So haben beispielsweise digitalisierte, wissenschaftliche Aufsätze eine sehr große Dauerhaftigkeit, während digitalisierte Gesprächsnotizen eher von kürzerer Benutzungsdauer sind. Der Autor weist darauf hin, daß die Dauerhaftigkeit zunächst nichts über die Persistenz von Dokumenten aussagt und folglich nicht mit ihr zu verwechseln ist.

Die oben genannten Eigenschaften – Änderungsrate und Dauerhaftigkeit – sind voneinander unabhängige Dimensionen. Während Gesprächsnotizen unverändert während ihrer eher kurzen Benutzungsdauer bleiben, weisen beispielsweise wissenschaftliche Aufsätze einen sehr langen Benutzungshorizont auf.
Traditionelle Informationssysteme sind oft zu starr und restriktiv hinsichtlich eines flexiblen und variablen Dokumentenbegriffs. Levy und Marshall [LM95] führen für diese prinzipielle Einengung vorwiegend historische Gründe an. Ein wesentliches Ziel der vorliegenden Arbeit besteht darin, den Dokumentenbegriff für Informationssysteme der Zukunft so zu erweitern, daß konzeptionelle Einschränkungen hinsichtlich des Dokumentenbegriffs beseitigt werden. In Abschnitt 3.2 auf Seite 48ff wird detailliert auf den Dokumentenbegriff eingegangen.

[1] So werden beispielsweise Adreßbuch-Dokumente, die sich während ihrer Benutzung häufig ändern, in diesen erweiterten Dokumentenbegriff aufgenommen.

2.1.2.2 Technologien

In diesem Abschnitt werden technologische Aspekte erläutert, die eine maßgebliche Rolle beim Entwurf und der Realisierung von Informationssystemen spielen. Dabei wird vor allem auf den Entwicklungsprozeß - das Software-Engineering - sowie auf die verwendeten System- und Software-Architekturen eingegangen.

Frameworks Die Entwicklung moderner, komplexer Informationssysteme ist oft teuer und häufig sehr fehleranfällig. Die Ursache liegt im ständigen „Neuentdecken“ bereits existierender Kernkonzepte bei der Softwarentwicklung. Die Technologie objektorientierter Frameworks [FS97, BGK+97] ist ein vielversprechender Ansatz, der die Implementierung und den Entwurf bewährter Softwaresysteme zur Wiederbenutzung propagiert. Die damit verbundene Verkürzung der Softwareentwicklungszyklen und die Erhöhung des Qualitätsstandards der realisierten Informationssysteme führen zu einer Kostensenkung des gesamten Entwicklungsprozesses.
Ein *Framework* ist nach Johnson und Foote [JF88]

> ... eine Menge von Klassen, die einen abstrakten Entwurf [Architektur] zur Lösung einer ganzen Familie von ähnlichen Problemen beinhalten.

Hinter dem Begriff Entwurf verbergen sich nicht nur die einzelnen Bestandteile der Architektur, sondern auch die Beziehungen der Bestandteile zueinander. Der Aspekt der Kollaboration der Klassen ist elementarer Bestandteil eines jeden Frameworks, wie Beck et al. [BJ94] und Johnson [Joh97] besonders betonen. In dieser Hinsicht ist es angemessen, den Begriff Entwurf (Design) in der deutschen Übersetzung durch Architektur zu ersetzen, da dadurch das Zusammenspiel der einzelnen Klassen klarer zum Ausdruck kommt. Nowack [Now98] spricht in diesem Zusammenhang von Frameworks als „architektonische Abstraktionen, die ein wiederbenutzbares, abstraktes Design für spezifische Anwendungsbereiche zur Verfügung stellen“. Die Beschreibung der Beziehungen der einzelnen Klassen verdeutlicht die Gründe für eine bestimmte Architektur, d. h. „der Weg zur Architektur“ wird ersichtlich.
Die oben angeführte Definition kennzeichnet drei wesentliche Merkmale von Frameworks:

1. Ein Framework ist bestimmt durch einen *abstrakten Entwurf*, d. h. das Abstraktionsniveau der Frameworkarchitektur ist so hoch, daß es keine speziellen, anwendungsabhängigen Komponenten beinhaltet.

2. Ein Framework trägt *zur Lösung einer ganzen Familie von ähnlichen Problemen* bei, d. h. die Anpassungsfähigkeit ist so groß, daß nicht nur zahlreiche Anwendungen in das Framework integriert werden können, sondern darüber hinaus ein hoher Spezialisierungsgrad erreicht werden kann.

3. Der Domänenbezug von Frameworks wird klar zum Ausdruck gebracht, d. h. anstelle einzelner Ausprägungen steht die ganze Klasse von Applikationen im Mittelpunkt.

Einige Merkmale – wie beispielsweise Abstraktion und Spezialisierungsgrad – stehen zum Teil konträr zueinander. Solche Ambivalenzen erschweren den Entwurf und die Gestaltung von Frameworks in hohem Maße[2].
Generell stellen Frameworks einen vielversprechenden Ansatz zur effizienten Wiederbenutzung von Software dar. In dieser Hinsicht teilen sich Frameworks Techniken und Vorgehensweisen sowohl mit traditionellen (modularen) als auch mit objektorientierten Strategien. Der wesentliche Unterschied liegt in der Granularität der wiederbenutzten Software. Während bei traditionellen und objektorientierten Ansätzen die Wiederbenutzung von Programmcode in Gestalt von Bibliotheken und einzelnen Klassen im Vordergrund steht, fokussiert der Frameworkansatz auf die Wiederbenutzung von ganzen Architekturkomponenten, d. h. nicht nur auf die einzelnen Strukturbestandteile, sondern auch auf deren Beziehungen untereinander. Die Einbeziehung von Designentscheidungen in den Wiederbenutzungsprozeß stellt den tatsächlichen Mehrwert von Frameworks dar. Allgemein ist die Wiederverwendbarkeit softwarebezogener Prozesse [BP89] ein wichtiger technischer und ökonomischer Aspekt beim Entwurf, bei der Gestaltung und der Implementierung von Informationssystemen, da in erster Linie vorgefertigte und folglich mit geringeren Entwicklungskosten verbundene Softwarekomponenten zum Einsatz kommen.
In späteren Kapiteln wird ausführlich auf den Entwurf und die Realisierung von Frameworks eingegangen. So werden beispielsweise in Kapitel 4 anhand eines leichtgewichtigen Applikationsservers einige Vor- und Nachteile von frameworkorientierten Softwarekomponenten in Informationssystemen diskutiert.

Dienstbasierte Technologien Nach Schanz und Schmidt [SS01] ist der netzwerkzentrierte Entwurf ein wesentlicher Trend bei der Realisierung künftiger Informationssysteme. Im Vordergrund steht dabei der Entwurf neuer Informationssysteme durch die Zusammenführung und Integration separater, verteilter und oft autonomer Softwarekomponenten. An die Stelle von kompletten Neuentwicklungen von Informationssystemen treten so vielmehr Bestrebungen, bereits existente Softwarekomponenten bzw. Teile von bestehenden Informationssystemen sinnvoll und auf eine flexible Art und Weise über ein Netzwerk miteinander zu verknüpfen.
Voraussetzung für eine solche integrative und netzwerkzentrierte Vorgehensweise bei der Entwicklung von künftigen Informationssystemen ist eine dienstleistungsorientierte Sichtweise auf Informationssysteme und deren Softwarekomponenten.
Ein Dienst wird bestimmt durch

[2] In [DMNS97] wird sogar von der „Kunst“ des Entwurfs und der Gestaltung eines Frameworks gesprochen: „*The design of frameworks remains an art rather than a science because of the inherent conflict between reuse and tailorability*.“

> ... ein definiertes Verhalten, welches prinzipiell von beliebigen Softwarekomponenten implementiert und angeboten werden kann. Ein Dienst ist dabei ausschließlich durch die vertragsähnliche Beschreibung seines Verhaltens definiert.

Traditionell werden diejenigen Softwarekomponenten als Dienste bezeichnet, welche eine genau definierte Schnittstelle besitzen. Dabei ist die Granularität dieser Schnittstelle oft sehr grob. Sie bezeichnet die Zugangs- bzw. Kommunikationsmöglichkeiten der gesamten Softwarekomponente. Eine Softwarekomponente in diesem Zusammenhang steht für eine komplette Applikation. So wird beispielsweise ein *Dateisystem* innerhalb der Domäne Betriebssysteme als Dienst bezeichnet, der den Zugang und die Organisation aller Dateien verwaltet. Folglich sind Dateien an diesen Dienst gebunden und können nicht als eigenständige Dienste auftreten.
Dieser grob-granularen Betrachtung von Diensten steht ein vereinfachter Ansatz hinsichtlich von Diensten gegenüber. Kommerziell verfügbare Softwareframeworks wie SUN Microsystems' *Jini* [Wal99, Sun], Microsofts *.NET* [TL01, Mic02] und Hewlett Packards *e-speak* [Kar00, Hew02] beschränken aus konzeptioneller Sicht einen Dienst nicht auf eine (grob-granulare) Applikation. Vielmehr wird der oben angeführten Definition eines Dienstes folgend alles als Dienst bezeichnet, was sich durch eine eindeutige Menge von Eigenschaften oder Attributen beschreiben läßt. Die Funktionalitäten derartiger Dienste werden direkt – gegebenenfalls über ein Netzwerk – aufgerufen.. Wenn man dieser fein-granularen Konzeption von Diensten folgt, wäre im oben erwähnten Beispiel über Dateisysteme jede Datei selbst ein Dienst, der anhand der Spezifikation seiner Attribute gefunden und benutzt werden kann.
Eine solche dienstleistungsorientierte Sichtweise trägt zur Optimierung verteilter Informationssysteme in dynamischen Umgebungen bei [Wei91]. Die Abstraktion eines fein-granularen Dienstes ermöglicht die Realisierung von verteilten Informationssystemen, die sich durch eine dynamische Integration von Dienstleistungen in natürlicher Weise ergibt.
Die Funktionalitäten des entsprechenden Informationssystems

- können von beliebigen Softwarekomponenten erbracht werden, sofern diese der Dienstspezifikation entsprechen,
- können zur Laufzeit in das Informationssystem eingebunden werden.

Weisers Vision des Ubiquitous Computing [Wei91] baut ebenfalls auf dieser dienstleistungsorientierten Sichtweise auf, in der Dienste repräsentativ für Software- und Hardwareprozesse jeglicher Art stehen. Sowohl normale als auch kleine, eingebettete Computersysteme bilden letztlich ein netzwerkorientiertes Informationssystem, welches aus vielen autonomen bzw. selbständigen Entitäten besteht.

Um derartige verteilte Informationssysteme realisieren zu können, ist es von Bedeutung, die partizipierenden Softwarekomponenten als autonome Bestandteile eines großen und integrativen Informationssystems zu betrachten, da ansonsten die Handhabbarkeit hinsichtlich der Entwicklung und Wartung des gesamten Informationssystems unmöglich wird. Nach Zambonelli [Zam01] impliziert die Autonomie dieser partizipierenden Softwarekomponenten bzw. Dienste deren selbständigen und lokalen Kontrollfluß (flow of control). Demnach besteht ein aus autonomen Teilen dynamisch integriertes Informationssystem aus mehreren unabhängigen Kontrollflüssen. Klassisch nebenläufige und parallele Forschungsansätze haben hierfür entsprechende Programmier- und Softwaremodelle entworfen, die primär auf eine global ausgerichtete, effiziente und performante Ausführung des gesamten Systems fokussiert sind. Dies steht im Widerspruch zu den autonomen und selbständigen Kontrollflüssen und -aktivitäten in den jeweiligen Softwarekomponenten. Nach Parunak [Par97] ist der Bedarf an der Delegation von Kontrollflüssen an autonome Entitäten (Softwarekomponenten) keine Frage der Effizienz und Performanz, sondern vielmehr der konzeptionellen Einfachheit. Die Koordinierung eines globalen Kontrollflusses für eine extrem große Anzahl an partizipierenden Komponenten wird möglicherweise unpraktisch oder gar unmöglich. Die Autonomie wird dann eine zusätzliche Dimension der Modularität, welche auf natürliche Art und Weise Konzepte der Datenkapselung und Datenunabhängigkeit innerhalb des Datenmanagements bzw. der Datenbanksysteme erweitert, die auch von klassischen, objektorientierten Programmierkonzepten unterstützt werden.
Im Kapitel 9 wird ein sehr fein-granularer Dokumentenbegriff eingeführt, der auf autonomen und selbständigen Softwarekomponenten basiert. In Kapitel 10.2 wird dann anhand einer Fallstudie ausführlich auf die Eigenschaften solcher autonomer Softwarekomponenten innerhalb eines großen verteilten Informationssystems eingegangen. Zudem wird in Kapitel 8.4 eine Fallstudie [SFK01a, SBS+00] des Symbolischen Rechnens vorgestellt, in der die Berechnung innerhalb einer verteilten Systemumgebung mit autonomen Softwarekomponenten durchgeführt wird. Im Mittelpunkt der Betrachtung steht dabei die Einfachheit der Anwendbarkeit von autonomen Softwarekomponenten in verteilten Systemumgebungen. Die Aspekte der global effizienten und performanten Ausführung rücken dabei in den Hintergrund.

2.1.2.3 Arbeitsformen

Eine Arbeitsform beschreibt den Umgang und die Art der Interaktion eines Benutzers mit einem Informationssystem. Traditionelle Sichtweisen unterstellen, daß primär einzelne Individuen in quasi abgeschlossenen realen oder virtuellen Räumen die Benutzerschaft von Informationssystemen ausmachen. Dabei wird angenommen, daß sich Benutzer vor einem Computersystem befinden und entsprechende Interaktionen – wie eine Informationssuche – initiieren. Die Terminologie eines abgeschlossenen Raums soll darauf hinweisen, daß der betreffende Benutzer sich ausschließlich in seiner eigenen sowohl realen als auch virtuellen Informati-

onswelt bewegt und somit wenig oder gar keine Interaktion mit anderen Benutzern und deren Arbeitsformen zustandekommt [LM95].

Kooperative Arbeitsformen Ehrlich et al. [EC94] zeigen – im Gegensatz zu traditionellen Sichtweisen – anhand mehrerer Beispiele, daß eine digitale Bibliothek sehr wohl ein Ort kollaborierender Aktivitäten und Zusammenkünfte ist. Der kooperative Aspekt läßt sich vor allem am Beispiel der Informations- und Dokumentensuche verdeutlichen: Der kommunikative und letztlich kooperative Aspekt einer Informationssuche steht immer mehr im Vordergrund. Dabei sind folgende beispielhafte Fragestellungen von Bedeutung:

- *Wer* hat ein bestimmtes Dokument im Web ebenfalls angeschaut?
- *Was* für Reaktionen rief der Inhalt dieses Dokuments beim Leser hervor?
- *Wann* und *wo* hat der Leser das Dokument betrachtet?
- *Welche* weiteren Kontext- bzw. Zusatzinformationen zu dem betreffenden Dokument sind für den Leser wichtig ? Wurden Empfehlungen hinsichtlich zusätzlicher Literatur und Dokumente ausgesprochen ?

In [Mar90] wird auf der Basis von empirischen Studien über Informationsanalysten untersucht, wie sich deren Arbeitsweise hinsichtlich der Benutzung digitaler Bibliotheken im alltäglichen Leben gestaltet:

- Informationsintensive Arbeiten erfordern heterogene Dokumentenquellen, d. h. es werden unterschiedlichste Medien bzw. Dokumentenformate benutzt.
- Die benutzten und erstellten Dokumente sind hochgradig „flüchtig“ und ändern sich häufig. Zudem werden oft lokale Dokumentensammlungen erzeugt, auf denen später kooperativ von anderen Benutzern zugegriffen wird.
- Generell kann ein hohes Maß an informeller Kollaboration und Kommunikation der Informationsanalysten konstatiert werden.

Im allgemeinen läßt sich festhalten, daß heutige Arbeitsformen teilweise von einem hohen Maß an kooperativen Aspekten geprägt sind. Die Benutzer führen ihre Arbeit und Interaktionen nicht mehr nur in ihren eigenen virtuellen Arbeitsräumen aus, sondern es kommt zu einem losen Zusammenschluß verschiedener Benutzerschaften. Die Schwierigkeit liegt dabei in der kooperativen Zusammenführung der unterschiedlichen und personalisierten Arbeitsräume der Benutzer.

Automatisierte Arbeitsformen Neben den kooperativen Aspekten in der Arbeitsform heutiger und künftiger Informationssysteme ist die Art und Weise der Interaktionsform von Benutzern hervorzuheben. Interaktive Forschungsansätze von

1960 [Lic60] fokussierten auf menschlichen, benutzerzentrierten Interaktionsformen. Beim Entwurf und ebenso bei der Realisierung von Computersystemen stand dabei ausschließlich der Mensch im Mittelpunkt. Tennenhouse [Ten00] hebt hervor, daß diese exklusive Fokussierung auf den Mensch den Entwurf besserer und adäquaterer Informationssysteme in der Zukunft behindert.
Tennenhouse unterstreicht die Entwicklung, die bereits Weiser [Wei91] im Bereich des Ubiquituous Computing im Jahre 1991 vorhergesagt hat: Die Anzahl der Computer und eingebetteter Systeme bzw. Prozessoren wird derartig drastisch steigen, daß man bestehende Vorgehensweisen innerhalb der Informatik kritisch auf ihre Gültigkeit hin überprüfen sollte:

> ...[it] has come the creation of an IT industry that is hurtling toward the human/machine/network breakpoint - the point at which the number of networked interactive computers will surpass the number of people on the planet.
>
> [...] It is time for a change. The computer science research community a rare and exciting opportunity to redefine its agenda and establish the new goals that will propel society beyond interactive computing and the human/machine breakpoint. In lifting our sights toward a world in which networked computers outnumber human beings by a [...] thousand to one, we should consider what these „excess“ computers will be doing and craft a research agenda that can lead to increased human productivity. [Ten00]

Darüber hinaus fordert Tennenhouse u. a. anstelle menschlicher benutzerzentrierter Konzepte neue und adäquate Vorgehensweisen, in denen der Mensch primär als Administrator (Verwalter) einer ganzen Reihe von Softwareprozessen agiert[3]. Die damit verbundene Automatisierung von Interaktionen und Abläufen sehen Tennenhouse [Ten00] und Norman [Nor99] als wesentliche Charakteristika einer Richtung innerhalb der Informatik, die sie als *Proactive* bzw. *Invisible Computing* bezeichnen.
McCray und Gallagher [MG01] beziehen sich ebenfalls auf die Automatisierung von Abläufen in zukünftigen Informationssystemen. Wegen des überwältigenden Informationsangebots und der Informationsflut sollten Abläufe innerhalb von Informationssystemen möglichst automatisiert werden, damit der (menschliche) Benutzer sich auf die Kernaufgaben konzentrieren kann. Dies ermögliche ein effektives und wirschaftliches Betreiben von Informationssystemen.
Im folgenden Abschnitt soll anhand der automatischen Zusammenstellung von Informationsinhalten im Web ein aktuelles Beispiel für den existenten Trend der automatisierten Abläufe und Interaktionen gegeben werden: *Topic Distillation*

[3] Es sei darauf hingewiesen, daß der Mensch unverändert im Mittelpunkt beim Entwurf und bei der Realisierung von Informationssystemen stehen sollte. Allerdings sollte die Art und Weise seiner Interaktions- und Einflußmöglichkeiten auf eine Vielzahl von Softwareprozessen überdacht und neu definiert werden.

[Kle99,BH98,Nes01] bezeichnet den Vorgang, in dem im Web hierarchische Strukturen – sogenannte Webverzeichnisse – hinsichtlich bestimmter Themenkomplexe automatisch erstellt werden. Der menschliche Benutzer erhält mit Hilfe dieser Verzeichnisse die Möglichkeit, qualitativ bessere Suchanfragen zu stellen bzw. Suchergebnisse zu erhalten. Das Problem der Informationsflut im Web wird so zum Teil umgangen, in dem spezielle Softwareprozesse (z. B. Agenten-Crawler) existierende Web-Datenbasen (HTML-Dokumente) kontaktieren, darin enthaltende Informationen analysieren, entsprechende Informationen herausfiltern und für den menschlichen Benutzer sinnvoll aufbereiten. Diese Prozesse finden aus der Sicht des Menschen im Hintergrund statt und sind zeitlich ungebunden, d. h. kontinuierlich und fortwährend werden Aufgaben, die von Menschen initiiert und verwaltet werden, ausgeführt. Diese Art der Informationsaufbereitung ist schon aus praktischen Gesichtspunkten nicht manuell – mit traditionellen benutzerzentrierten Ansätzen – effizient zu bewältigen. Der Mensch tritt so primär als Verwalter zahlreicher im Hintergrund agierender Softwareprozesse auf. In Kapitel 10.2 wird ausführlicher auf die automatisierte Generierung und Zusammenstellung von Themenkomplexen aus unterschiedlichen Informationsquellen eingegangen.
Hinsichtlich der technischen Realisierbarkeit von automatisierten Interaktionen zwischen Softwareprozessen sind insbesondere folgende zwei Gesichtspunkte zu nennen:

1. Schnittstellen zwischen den betreffenden Akteuren (Softwareprozessen) sollten einerseits bekannt sein und andererseits sollte die Möglichkeit bestehen, diese auf einfache Art und Weise zu benutzen. Allgemein definiert eine Schnittstelle die Zugangsmöglichkeiten zu einem Akteur, d. h. die *Syntax* der ausgetauschten Daten wird festgelegt. Dabei ist sowohl der jeweilige Datentyp als auch die Reihenfolge der ausgetauschten Daten von Bedeutung.

2. Neben der Syntax ist die inhaltliche Bedeutung der Daten entscheidend. Für automatisierte Interaktionen in großen, verteilten Informationssystemen sollte die *Semantik* der Daten in einer expliziten bzw. externalisierten Form vorliegen. Die semantischen Bedeutungen und Interpretationen der Daten werden häufig implizit festgelegt. Eine implizite Bestimmung der inhaltlichen Bedeutungen wird von Menschen vorgenommen und ist somit mehr oder weniger nur in deren „Köpfen“ bekannt[4]. Eine explizit interpretierbare Semantik befähigt und unterstützt Softwareprozesse bei der automatisierten inhaltlichen Deutung von Dateninhalten[5].

In Kapitel 3 wird der Automatisierungsaspekt in die vorgestellte und in dieser

[4] Zudem findet man häufig nützliche Beschreibungen und Erläuterungen der Daten in der sogenannten *Systemdokumentation*. Generell dient diese Dokumentation allerdings nur als manuelles Nachschlagewerk für die Benutzer des Systems.

[5] Aktuelle Forschungsrichtungen im Bereich des *Semantic Web* [BLHL01, BL99, SEM01] fokussieren auf der Spezifikation und Entwicklung von software-zentrierten Automatismen zur Interpretation und Verarbeitung von inhaltlichen Bedeutungen im Web (z. B. OIL [FvHH$^+$01, FHvH$^+$00])

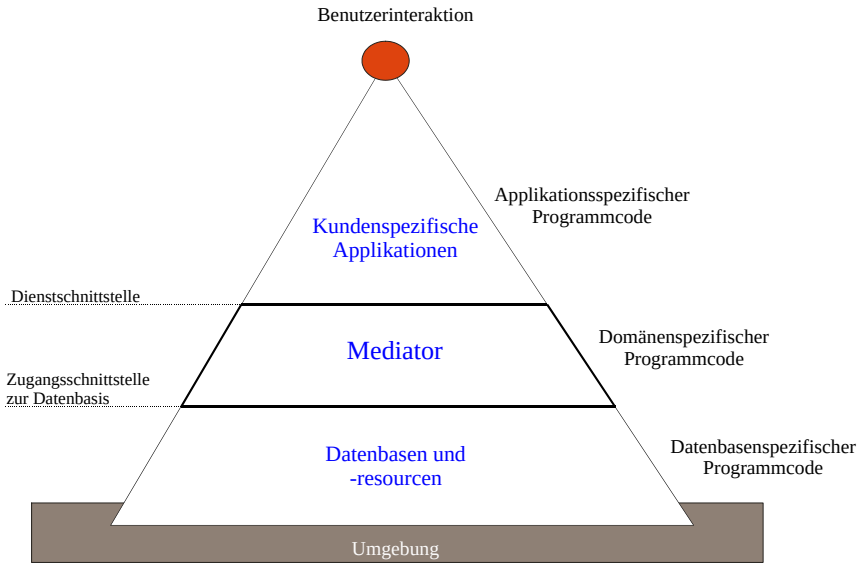


Abbildung 2.3: Mediationsschichten in Informationssystemen basierend auf den Ausführungen von Wiederhold und Genesereth [WG97].

vorliegenden Arbeit entwickelte Taxonomie von Informationssystemen aufgenommen.

2.2 Mediatoren

Die in diesem Abschnitt beschriebenen Mediatoren stellen die konzeptionelle Basis für zahlreiche Systemkomponenten in heutigen Informationssystemen dar. Nach einer allgemeinen Einführung in das Mediatorkonzept werden in Abschnitt 2.2.1 die Eigenschaften beschrieben, wobei in Abschnitt 2.2.2 mehrere Aufgaben von Mediatoren erörtert werden. In Abschnitt 2.2.3 wird die grundlegende Bedeutung des Mediatorkonzepts hervorgehoben, in dem Mediatoren mit Konzepten in objektorientierten Systemen verglichen werden.
Ein *Mediator* ist eine Systemkomponente in Informationssystemen, die sich gemäß Abbildung 2.3 zwischen den Datenbasen und den Endkunden-Applikationen befindet. Der Mediator ist für die Integration der Daten und Informationsflüsse aus den umgebenden zwei Schichten verantwortlich. Generell ermöglicht der Einsatz eines Mediators eine Aufteilung eines Informationssystems in mehrere, kleinere Subsysteme oder -komponenten, welches die Komplexität des Gesamtsystems hinsichtlich des Entwurfs und der Realisierung reduziert.
Gemäß Abbildung 2.3 werden Informationsssysteme in mehrere Mediationsschichten – in applikations-, domänen- und datenbasen-spezifische Schichten – unterteilt. Ein Mediator stellt das Bindeglied zwischen den Schichten dar. Der Zugriff von kundenspezifischen Applikationen auf einen Mediator erfolgt über dessen

Dienstschnittstelle. Der Mediator steht mit den Datenbasen über deren *Zugangsschnittstellen* direkt in Verbindung.
Das Prinzip der *Mediation*[6] ist charakteristisch für die Tätigkeiten und Funktionen, die ein Mediator ausübt. Das Ziel der Mediation besteht darin, die originären Daten derart aufzubereiten und zu transformieren, daß die Daten auf unterschiedliche Abstraktionsstufen flexibel und dynamisch abgebildet werden können. So werden beispielsweise gemäß Abbildung 2.3 diejenigen Daten, welche direkt von den Datenbasen kommen und eine niedrige Abstraktionsstufe besitzen, über die Mediatoren (*domänenspezifische Transformationen*) auf die Abstraktionsstufe von kundenspezifischen Anwendungen transformiert.
Unter der Abstraktionsstufe der Daten wird die generelle Anwendbarkeit der Dateninhalte verstanden. So haben Daten auf der unteren Ebene – in den Datenbasen – eine niedrige Abstraktionsstufe, da die Dateninhalte an ganz konkrete Attribute und Eigenschaften der jeweiligen Datenbasen gebunden sind. Die Abstraktionsstufe auf der Ebene der kundenspezifischen Applikationen ist dabei höher einzuordnen, da die Dateninhalte nicht an bestimmte Datenbasen und -ressourcen gebunden sind. Vielmehr spezifizieren die Dateninhalte der höheren Abstraktionsstufe einen allgemeineren Datenzustand auf der Ebene der kundenspezifischen Applikationen.

2.2.1 Eigenschaften von Mediatoren

Aus historischer Sicht stellte Wiederhold [Wie92a] 1992 erstmals das Konzept der Mediatoren in Informationssystemen vor:

> Mediation simplifies, abstracts, reduces, merges, and explains data. [...] Mediation covers a wide variety of functions that enhance stored data prior to their use in an application. Mediation makes an interface intelligent by dealing with representation and abstraction problems that you must face when trying to [...] use data and knowledge resources [Wie92a]

Demnach haben Mediatoren vielfältige Aufgaben und spielen eine wichtige Rolle für den Entwurf und die Realisierung von Informationssystemen. Generell dienen sie als *Zugangspunkte* für Applikationen auf heterogenen Datenbasen, wie es in Abbildung 2.3 illustriert ist. Darüber hinaus übernehmen Mediatoren eine wichtige *Kapselungsfunktion*. Ein Mediator ist immer einer Gruppe von Datenbasen zugeordnet und verwaltet alle Zugriffe der Applikationen auf die betreffenden Datenbasen[7]. Die Auswahl der Zugehörigkeit der Datenbasen orientiert sich dabei an den

[6] Mediation: vermittelndes Dazwischentreten, Duden [EHWM01].

[7] Der Begriff der Datenbasis ist sehr weit gefaßt, da er jegliche Form von digitalen Daten beinhaltet (siehe Abschnitt 2.1.1 auf Seite 10ff). In dieser Hinsicht können Mediatoren nicht nur Datenbanken, sondern auch verschiedenen anderen Systemkomponenten zugeordnet werden. Dies ist ein wichtiger Aspekt für die erweiterte Anwendung des Mediatorkonzepts. Beispielsweise werden Mediatoren in Abschnitt 3.3.3 auf Seite 71ff im Zusammenhang mit Software-Agenten diskutiert, die als spezielle Mediatoren – sogenannte Mikromediatoren – betrachtet werden.

Funktionalitäten bzw. Dienstleistungen, die der entsprechende Mediator erbringen soll. Wiederhold spricht in diesem Zusammenhang von sogenannten Zusatzdiensten (*value-added services*), die von den Mediatoren erbracht werden. Ein aus einer Menge von Mediatoren bestehendes Informationssystem wird so zu einem dienstzentrierten Informationssystem, in dem einzelne Dienste in unterschiedlichen Anwendungskontexten (wieder)benutzt werden. Die Generierung bzw. Realisierung von Informationssystemen durch eine adäquate Komposition von Mediatoren – also von Dienstleistungen – ist ein vielversprechender Ansatz, um die Herausforderungen zukünftiger verteilter Informationssysteme zu meistern. Die Einführung einer dienstleistungsorientierten Sichtweise unterstützt die Wiederbenutzbarkeit von Softwarekomponenten und die Steigerung der Softwarequalität (siehe Abschnitt 2.1.2.2).
Durch den Einsatz von Mediatoren werden Applikationen auf der oberen Ebene von den Datenbasen auf der unteren Ebene entkoppelt. Diese logische Entkopplung hat folgende Vorteile für den Entwurf von Informationssystemen:

- *Skalierbarkeit*. Je größer die Anzahl der beteiligten Datenbasen ist, desto größer wird die Anzahl der logischen Verstrebungen zwischen kundenspezifischen Applikationen und Datenbasen. Als vermittelnde Systemkomponente in der Mitte – zwischen kundenspezifischen Applikationen und Datenbasen – reduziert der Einsatz eines Mediators die Abhängigkeiten zwischen Applikationen und Datenbasen und ermöglicht so eine skalierfähige Realisierung von Informationssystemen. Diese zusätzliche Indirektion in der Informationssystemarchitektur birgt einerseits die Gefahr von Performanz- und Ausführungsverlusten, andererseits werden die notwendigen Grundlagen für Informationssysteme geschaffen, die sich aus einer großen Zahl an Datenbasen zusammensetzen.

- *Dynamische Komposition*. Unter der dynamischen Komposition versteht man die Möglichkeit, einzelne Dienste, welche durch Mediatoren verwaltet werden, zur Laufzeit in ein Informationssystem einzubinden. Die Entkopplung der oberen von der unteren Schicht erlaubt es Mediatoren, zur Laufzeit Entscheidungen hinsichtlich des Daten- und Kontrollflusses zu treffen. Anfragen auf hoher Abstraktionsstufe werden vom entsprechenden Mediator bearbeitet und gemäß den aktuellen System- und Umgebungszuständen an die betreffenden Datenbasen weitergeleitet. Die Festlegung, welche Datenbasen für die Erfüllung der Anfrage aus der oberen Schicht herangezogen werden, erfolgt durch den Mediator zur Laufzeit.

- *Unterstützung für autonome Datenbasen*. Mediatoren als Verwalter von Datenbasen unterstützen die Unabhängigkeit und Autonomie der jeweiligen Datenbasen. Beispielsweise können sich die Datenbasen getrennt von den Kundenanforderungen weiterentwickeln, da die Datenbasen von den kundenspezifischen Gegebenheiten entkoppelt sind.

Es ist die Aufgabe der beteiligten Mediatoren, diese Entwicklungen gegenüber anderen transparent zu verwalten. So können beispielsweise aus Wartungsgesichtspunkten bestimmte Softwareanomalien bei den Datenbasen verbessert bzw. beseitigt werden, ohne daß die eigentlich betroffenen kundenspezifischen Applikationen davon in irgendeiner Weise berührt werden. Dies ist ein wesentliches Argument für den Übergang von sogenannten 2-stufigen zu 3-stufigen Client/Server-Architekturen, die in Abschnitt 2.3.1 ausführlich vorgestellt werden.

- *Flexibilität*. Die Einführung von Indirektionen, wie die der Entkopplung von allgemeineren und speziellen Schichten innerhalb von Informationssystemen, bringt ein hohes Maß an Flexibilität hinsichtlich der Zusammenführung der Schichten. Durch die indirekten Verbindungen werden beispielsweise grundsätzlich unterschiedliche Schnittstellendefinitionen einheitlich durch die entsprechenden Mediatoren repräsentiert.

- *Wiederbenutzbarkeit*. Durch die Entkopplung benutzen Applikationen anstelle der direkten Zugangsschnittstelle der Datenbasen vielmehr die Dienstschnittstelle des betreffenden Mediators (siehe Abbildung 2.3). So kapseln und verwalten Mediatoren Dienste, die in unterschiedlichen Applikationen benutzt werden können. Diese Wiederbenutzbarkeit von Informationsdiensten in unterschiedlichen Anwendungen steigert sowohl die Softwareproduktivität als auch die Softwarequalität, da bereits existierende und getestete Mediatoren wieder zum Einsatz kommen.

Im folgenden Abschnitt werden Mediatoren hinsichtlich des Automatisierungspotentiales betrachet.

2.2.2 Beispiele für die Bedeutung von Mediatoren

Wiederhold [Wie92a] hebt die aktiven Eigenschaften von Mediatoren hervor. Als Verbindungsglied zwischen Applikationen und Datenbasen enthalten sie das notwendige Wissen um geeignete und passende Transformationen der Daten. Dieses Wissen beinhaltet sowohl Details hinsichtlich der Zugriffs- und Interpretationsmöglichkeiten auf der Ebene der Datenbasen als auch Einzelheiten der Applikationslogik auf der Ebene der Endkunden-Applikationen. Im folgenden sind mehrere Beispiele für die Bedeutung von Mediatoren in Informationssystemen angeführt.

2.2.2.1 Datentransformationen

Eine Datentransformation bezeichnet den Vorgang, in dem Daten in unterschiedliche Repräsentationen überführt werden. Dabei spielt nicht nur die syntaktische Darstellung, sondern auch die semantische Bedeutung der transformierten Daten

eine wichtige Rolle. Stellt beispielsweise ein Benutzer eine Anfrage an ein Informationssystem bzw. an eine Applikation, werden die Daten bezüglich der Anfrage aus der Repräsentation A der Datenbasis nach B transformiert, damit der Benutzer mit den Daten sinnvoll weiterarbeiten kann. Die syntaktischen Details der Repräsentation A liegen in der Verantwortung des Mediators und sind für den Benutzer unerheblich. Die Repräsentation B, die von der jeweiligen Anwendung abhängig ist, dient dabei als maßgebliches Datenformat zwischen den Anwendungen auf der Benutzerebene und dem Mediator.

2.2.2.2 Datensammlung

Der Prozeß der Datensammlung beinhaltet Methoden zum angemessenen Sammeln und Verarbeiten von Daten. So besitzen im allgemeinen Datenbanksysteme nicht die nötige Aktivität, um bestimmte Reihen von Abfragen an die Datenbasis zu starten. Deswegen ist es nötig, daß Mediatoren entsprechende Datenbankanfragen initiieren, die Ergebnisse entsprechend einsammeln und danach den jeweiligen Adressat weiterleiten bzw. übergeben. Das Zusammenführen von Ergebnissen kann unter Umständen sehr komplex werden, da vielfältige Transformationen und Interpretationen auf der Ebene der Mediatoren notwendig sind.
Generell sind algorithmische Vorgehensweisen in den Mediatoren erwünscht, die eine Datenbasis ohne weitere Unterstützung oft nicht erfüllen kann. In dieser Hinsicht können Mediatoren im Zusammenspiel mit Datenbasen sehr komplexe Anfragen von der Applikationsebene erfüllen, die über normale Anfragemechanismen – wie zum Beispiel das Absetzen von Anfragen in einer Datenbankanfragesprache wie SQL (Structured Query Language) [DD93] – der Datenbasen hinausgehen.

2.2.2.3 Applikationsserver

Ein Applikationsserver ist eine Middleware-Komponente [Ber96], die Dienstleistungen für die Applikationsebene zur Verfügung stellt, um auf Instanzen auf der Ebene der Datenbasen zuzugreifen[8]. Es ist die Aufgabe eines Applikationsservers, die Applikationslogik des Informationssystems zu verwalten. Darüber hinaus dient er als zentraler Zugangspunkt für Benutzeranfragen an das Informationssystem. Es liegt in der Verantwortung eines Applikationsservers zu entscheiden,

- welche Datenbasen für die Beantwortung einer Benutzeranfrage kontaktiert werden müssen und
- welche Datentransformationen und sonstige algorithmische Aktivitäten notwendig sind, um die Daten in der vom Benutzer erwarteten Form zurückzugeben.

[8] In Abschnitt 2.3.1.2 auf Seite 32ff werden Applikationsserver im Zusammenhang mit Client/-Server-Informationssystemen diskutiert.

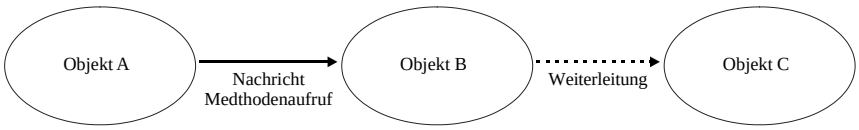


Abbildung 2.4: Delegationsprinzip in objektorientierten Systemen.

Viele der heute realisierten Informationssysteme benutzen einen Applikationsserver, um den erforderlichen Datentransformationen und den komplexen Anforderungen hinsichtlich einer flexiblen Systemarchitektur gerecht zu werden.
In Kapitel 4 wird detailliert auf den Entwurf und die Realisierung eines speziellen Applikationsservers eingegangen. Die Eigenschaften dieses Applikationsservers werden mit anderen Ansätzen zur Realisierung von Applikationsservern verglichen, die auf der *Common Object Request Broker Architecture* (CORBA) [Obj00a, MM97] oder auf den *Enterprise JavaBeans* (EJB) [MH99, Sun02a] basieren.

2.2.3 Delegation in objektorientierten Systemen

Ein Mediator gemäß Wiederhold [Wie92a] ist auf der Applikationsebene von Informationssystemen angesiedelt, wie es in Abbildung 2.3 dargestellt ist. Mediatorverwandte Konzepte innerhalb objektorientierter Paradigmen [Lie86] fokussieren im Gegensatz zur Applikationsebene auf die *Daten-* bzw. *Objektebene*. In diesem Abschnitt werden die gemeinsamen Eigenschaften der beiden Konzepte vorgestellt, um die grundsätzliche Bedeutung von Mediatoren beim Entwurf und der Realisierung von Informationssystemen zu unterstreichen.
Das *Delegationsprinzip* in objektorientierten Ansätzen entspricht einer Mediation auf der Daten- bzw. Objektebene. In Abbildung 2.4 ist der grundsätzliche Ablauf der Delegation illustriert: Bei der Delegation werden die Nachrichten, die von Objekten des Typs A nach B verschickt werden, nicht direkt bei Objekt B zur Ausführung gebracht. Objekt B leitet die Nachricht an Objekt C weiter. D. h. Objekte in objektorientierten Systemen können Nachrichten, die z. B. bestimmte Methodenaufrufe beinhalten, an andere Objekte weiterleiten. Der eigentliche Aufruf wird so durch das Prinzip der Delegation mediatiert.
Im Gegensatz zur Delegation ist das Vererbungsprinzip in objektorientierten Systemen an statisch festgelegte Konventionen gebunden, die während der Entwurfsphase des Informationssystems bzw. zum Erzeugungszeitpunkt (Kompilierzeit) festgelegt wurden [Lie86]. Die Delegation ermöglicht und unterstützt ein dynamischeres Verhalten als das Vererbungsprinzip, d. h. die Delegation unterstützt Entscheidungsprozesse, die erst zur Laufzeit adäquat berurteilt werden können [Lie86]. Demnach ist das Prinzip der Delegation in dynamischen, objektorientierten Systemen weniger restriktiv als das der Vererbung, da solche dynamischen Systeme ein gewisses Maß an Nichtdeterminismus beinhalten. Die Unbestimmtheiten von Ei-

genschaften und Zuständen von Objekten zu späteren Zeitpunkten ihres Lebenszyklusses machen den Entwurf und die Implementierung eines solchen Systems unter Verwendung des Delegationsprinzips unerläßlich. Zudem ist es unwahrscheinlich – auch aus Softwareentwicklungsgesichtspunkten – zum Erzeugungszeitpunkt der Objekte, alle möglichen Objektzustände und Interaktionsmuster zwischen Objekten vorherzusehen.
Gemäß Lieberman [Lie86] ist die Delegation außerdem sehr gut geeignet für den Entwurf und die Umsetzung von interaktiven und sich ändernden Softwareprozessen. Bei der Delegation werden Variablen und Methoden angemessener mit anderen Objekten geteilt als unter Verwendung von Vererbung. Das Vererbungsprinzip in objektorientierten Systemen unterstützt eher fixierte und zum Erzeugungszeitpunkt statisch festgelegte Interaktionsmuster.
Zusammenfassend läßt sich festhalten, daß Mediatoren einen wichtigen Bestandteil in der Architektur von Informationssystemen übernehmen. In den unten angeführten Abschnitten und Kapiteln werden einerseits der konkrete Einsatz von Mediatoren in Informationssystemen und andererseits weiterführende Aspekte des Mediatorkonzepts diskutiert:

- Im Abschnitt 2.3.1 über Client/Server-Informationssystemen wird erläutert, welche Bedeutung das Mediatorkonzept für den Übergang von 2-stufigen zu 3-stufigen Informationssystemen auf der Basis des Client/Server-Prinzips besitzt.

- In Kapitel 4 wird der Entwurf und die Realisierung eines speziellen Mediators, eines sogenannten leichtgewichtigen Applikationsservers, vorgestellt.

- Neue Aspekte der Mediation, wie beispielsweise der Entwurf von proaktiven Mediatoren, werden im Kapitel 9 erörtert.

2.3 Beispiele von Architekturen für Informationssysteme

Im folgenden Abschnitt werden zwei typische Vertreter heutiger Informationssystemarchitekturen vorgestellt. Sowohl Client/Server-Informationssysteme als auch netzwerkbasierte Informationssysteme beruhen in ihren prinzipiellen Eigenschaften auf dem im vorherigen Kapitel vorgestellten Mediatorkonzept. Neben der Einführung in den grundsätzlichen Aufbau solcher Systeme werden einige der jeweiligen konzeptionellen Eigenheiten vorgestellt. Die beiden betrachteten Architekturen bilden die Grundlage für den Entwurf von proaktiven Informationssystemen.

2.3.1 Informationssysteme auf der Grundlage von Client/Server

Client/Server-Architekturen sind eine natürliche Erweiterung der Prinzipien des modularen Entwurfs und der modularen Programmierung [Par72]. Die grundle-

gende Annahme des modularen Softwareentwurfs ist die Zerlegung des Gesamtsystems in einzelne, geeignete Bestandteile (Module), so daß die Entwicklung und Wartung des Gesamtsystems einfacher und besser möglich wird. Generell verbergen Module diejenigen Entscheidungen beim Entwurf von Softwaresystemen, welche als tendenziell veränderbar erkannt wurden. Insbesondere sind dies diejenigen Details, welche die Struktur und Umsetzung (Realisierung) der Funktionalität eines Moduls betreffen. Unter der *Modulkohäsion* versteht man den inneren Zusammenhalt eines Moduls. Inhaltlich zusammengehörende funktionale Beziehungen hinsichtlich des Softwaresystems sollten in einem Modul verankert bzw. gekapselt werden. Gemäß Yourdon und Constantine ist es das Ziel, eine möglichst hohe fachliche und logische Kohäsion in den einzelnen Modulen zu erreichen [YC79].
Generell erweitern Client/Server-Architekturen das modulare Prinzip bei der Realisierung von Informationssystemen um die Erkenntnis, daß Module mit hohen Kohäsionseigenschaften nicht notwendigerweise in einem einzigen Speicheradreßraum ausgeführt werden müssen, d. h. einzelne Module können verteilt – auf unterschiedlichen Rechnern – ablaufen. Dies bildet die Basis für verteilte Informationssysteme, in denen zusammengehörende funktionale Aspekte zu Modulen zusammengefaßt und auf räumlich verteilten Rechnern zur Ausführung gebracht werden.
Die beiden wesentlichen Module in Informationssystemen auf der Grundlage von Client/Server sind ein sogenannter Client und ein Server. Ein *Client* ist der Initiator einer Anfrage, die zunächst an einen Server geschickt wird und der daraufhin die erforderlichen Aktionen ausführt. Normalerweise übernimmt ein Client Funktionen hinsichtlich der grafischen Benutzeroberfläche, der Validierung der vom Benutzer eingegebenen Daten oder der Verwaltung von Anfragen an einen Server. Darüber hinaus ist er in manchen Anwendungsfällen für die Ausführung von Teilen der Applikationslogik des Informationssystems verantwortlich. Auf diese letztgenannte Eigenschaft eines Client wird in den folgenden Kapiteln noch ausführlich eingegangen, so daß an dieser Stelle die Aspekte der Ausführung der Applikationslogik im Client vernachlässigt werden.
Ein Client dient als sogenanntes *Front-End* zur Interaktion mit einem Informationssystem. Gemäß dem grundsätzlichen, schematischen Aufbau von Informationssystemen in Kapitel 2.1.1 Abbildung 2.1 kann die Rolle des Benutzers sowohl von Menschen als auch von Softwareprozessen übernommen werden. In Kapitel 2.3.1.3 über dienstzentrierte Client/Server-Informationssysteme wird auf diesen Aspekt besonders eingegangen, da eine von Softwareprozessen übernommene Benutzerrolle große Möglichkeiten bietet, Interaktionen und Informationsflüsse innerhalb von Informationssystemen zu automatisieren.
Ein *Server* ist für die Beantwortung bzw. Erfüllung der Anfragen eines Client verantwortlich. Nachdem er eine Anfrage eines Client erhalten hat, ermittelt er die dafür notwendigen Informations- und Dienstressourcen. Danach werden die entsprechenden Dienste oder Datenbasen am sogenannten *Back-End* kontaktiert, die gewonnenen Ergebnisse aufbereitet und an den Client zurückgeschickt. In dieser Hinsicht entspricht ein Server in Client/Server-Informationssystemen einem Mediator, der vielfältige Aufgaben übernehmen kann, wie im Kapitel 2.2 ausführlich

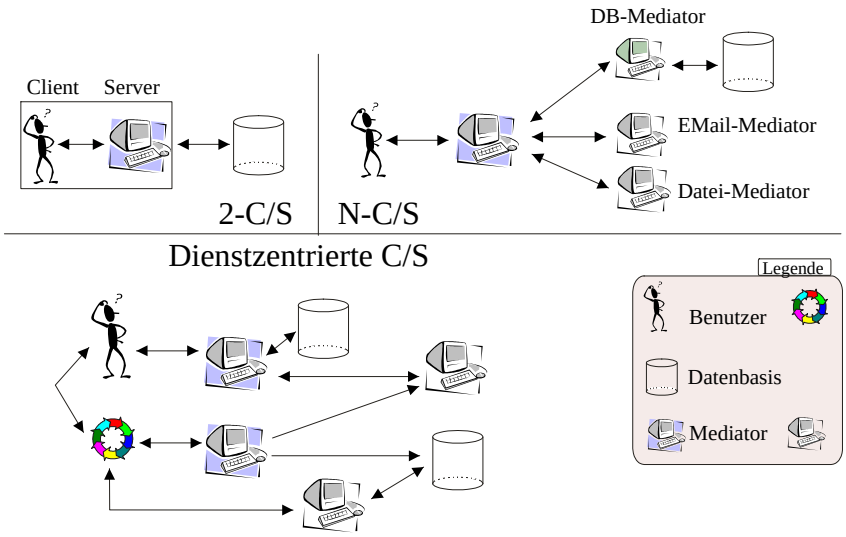


Abbildung 2.5: Übersicht von Informationssystemen auf der Grundlage von Client/Server-Architekturen.

erläutert wird. Eine der wichtigsten Aufgaben des Servers ist die Verwaltung und Ausführung der Applikationslogik des betreffenden Informationssystems, in der für die Applikation alle relevanten Merkmale und Eigenschaften der realen Diskurswelt abgebildet sind.
In den folgenden Kapiteln werden drei verschiedene Client/Server-Architekturen – sogenannte 2-stufige, n-stufige und dienstzentrierte Client/Server-Architekturen – vorgestellt. Dabei werden die jeweiligen Eigenschaften hinsichtlich ihrer Vor- und Nachteile zur Realisierung verteilter Informationssysteme diskutiert.

2.3.1.1 2-stufige Informationssysteme

In 2-stufigen Informationssystemen gibt es zwei logische Ebenen bzw. Module: Den Client und den Server. Dabei schickt der Client seine Anfragen direkt an den Server, wie aus Abbildung 2.5 ersichtlich wird. Der Client, der Server und die Dienste am Back-End (z. B. eine Datenbasis) werden häufig auf räumlich verteilten Lokationen installiert und ausgeführt.
Im folgenden werden einige spezielle Eigenschaften und potentielle Probleme von 2-stufigen Informationssystemen aufgeführt, die zu einer kritischen Beurteilung dieser Informationssystemarchitektur führen:

- *Fat-Client.* Durch die direkte Verbindung und enge Kopplung zwischen Client und Server entsteht am Front-End ein sogenannter schwergewichtiger Client, der auch als Fat-Client bezeichnet wird. Dieser Fat-Client zeichnet sich dadurch aus, daß er sowohl die oben aufgeführten Eigenschaften des

Clients als auch die des Servers in sich vereint. In Informationssystemen, die durch eine große Anzahl von Benutzern und dementsprechend zahlreichen Clients gekennzeichnet sind, birgt ein Fat-Client potentiell große Wartungsprobleme. Denn softwaretechnische Erweiterungen und Verbesserungen hinsichtlich der Applikationslogik des Informationssystems, die an sich nur den Server betreffen, ziehen unter Umständen aufgrund der engen Kopplung zwischen Client und Server komplette und teure Neu- oder Nachinstallationen des Fat-Clients bei allen Benutzern nach sich.

- *Simples Interaktionsprotokoll*. Die Interaktion zwischen Client und Server ist durch ein simples, synchrones Anfrage-Antwort-Interaktionsprotokoll gekennzeichnet. Der Client agiert als Initiator von Anfragen, die von Servern entsprechend beantwortet werden. Die Server sowie die Dienste am Back-End sind zunächst passive Komponenten, welche erst durch Anfragen von außen – des Clients – aktiv werden. Diese durch den Client initiierte Anfrage wird auch als sogenanntes *Pull*-Verfahren bezeichnet. Im Gegensatz dazu stehen sogenannte *Push*-Verfahren, die durch einen inversen Kontrollfluß gekennzeichnet sind, d. h. der Server initiiert die Interaktion mit dem Client und schickt diesem eine Anfrage. Eine wesentliche Voraussetzung für die Durchführung von Push-Verfahren ist die Registrierung des Clients beim Server. Bei der Registrierung werden dem Server client-spezifische Details – wie z. B. die aktuelle Lokation des Clients – mitgeteilt.

- *Grob-granulare Struktur*. Die grob-granulare Struktur der beiden Module bezieht sich – analog zur Argumentation des Fat-Clients – auf den Softwareentwurf von 2-stufigen Client/Server-Informationssystemen. Die geringe Modulanzahl eignet sich in der Regel nicht, um sehr komplexe funktionale Anforderungen in verteilten Informationssystemen adäquat abzubilden. Zudem sind dynamische Anpassungen an sich ändernde Bedingungen und Anforderungen nur in sehr beschränktem Maße möglich. Schließlich ist die in Kapitel 2.2 angesprochene dynamische Komposition von Diensten wegen der grob-granularen Struktur und starren Festlegung von Interaktionen nicht gegeben.

- *Passive Dokumente*. Wie in Abschnitt 2.1.2.1 bereits ausgeführt ist, bilden Dokumente und deren zugrundliegender Dokumentenbegriff eine wesentliche Säule beim Entwurf und der Realisierung von Informationssystemen. Die Dokumente bzw. die Information in 2-stufigen Architekturen sind in den Datenbasen am Back-End als passive und starre Informationsträger abgelegt. Die eigentliche Aktivität steckt im Client und Server. Beide verfügen aber ihrerseits nur über die oben beschriebenen eingeschränkten Aktivitäts- und Interaktionsmöglichkeiten.

Für performante, wenig komplexe Applikationen, die darüber hinaus nur von einer begrenzten Anzahl von Benutzern in Anspruch genommen werden, ist eine Rea-

lisierung eines Informationssystems auf der Basis einer 2-stufigen Client/Server-Architektur gut geeignet. Sobald allerdings komplexe funktionale Anforderungen in einer verteilten Umgebung mit vielen Benutzern durch ein Informationssystem abgebildet werden sollen, empfiehlt sich der Übergang von 2-stufigen zu sogenannten n-stufigen Client/Server-Architekturen, die im nächsten Abschnitt vorgestellt werden.

2.3.1.2 N-stufige Informationssysteme

Beim Übergang von 2- zu 3-stufigen Informationssystemen wird eine zusätzliche logische Ebene in die Architektur eingefügt, wie dies in Abbildung 2.5 illustriert ist. Diese Ebene wird als *middle tier* bezeichnet und steht zwischen dem Front- und Back-End. Diese mittlere Ebene wird durch einen speziellen Mediator repräsentiert, dem sogenannten *Applikationsserver*. Informationssysteme, die durch die Verallgemeinerung dieses Vorgehens bei der Einführung von zusätzlichen Ebenen in der Architektur entstehen, werden als n-stufige Informationssysteme auf der Grundlage von Client/Server bezeichnet, wobei *n* einen Indikator für die Anzahl der etablierten Ebenen darstellt.
Generell lassen sich die Aufgaben der drei Ebenen folgendermaßen charakterisieren:

Thin-Client Das Front-End (Client) dient ausschließlich zur benutzerorientierten Verarbeitung und zur Gestaltung grafischer Benutzeroberflächen. Diese Art des Clients wird als *Thin-Client* bezeichnet, da jegliche applikationsspezifische Logik und deren Eigenschaften auf den eingeführten Applikationsserver ausgelagert wird. Dadurch reduziert sich der Client in n-stufigen Informationssystemen im wesentlichen auf die Bewältigung benutzerspezifischer Anforderungen wie beispielsweise der grafischen Oberfläche. So wird in Schimkat et al. [SKK99] der Entwurf und die Implementierung eines objektorientierten Frameworks für die Visualisierung von grafischen Benutzeroberflächen in der Domäne Dokumentenmanagementsysteme beschrieben. Das Ziel ist eine erweiterbare und anpassungsfähige Architektur, die sich durch die Wiederbenutzbarkeit von Programmcode und Design auszeichnet. Voraussetzung dafür ist die Reduzierung des Clients auf primär präsentationsspezifische Funktionen, die dann einen domänenweit einheitlichen, konsistenten Einsatz des Frameworks innerhalb 3-stufiger Client/-Server-Architekturen ermöglichen.

Applikationsserver Zwischen Front- und Back-End befindet sich ein Applikationsserver, der als Mediator die betreffende Applikationslogik des Informationssystems enthält. Er dient als zentraler und singulärer Zugangspunkt für alle Benutzeranfragen. Zudem ist ein Applikationsserver in n-stufigen Informationssystemen räumlich vom Client getrennt und wird häufig auf einem separaten Rechner ausgeführt.

Back-End Das Back-End kapselt alle diejenigen grundsätzlichen Funktionalitäten als Mediatoren, die gemäß den Anforderungen an das Informationssystem zu erbringen sind. Dafür werden typischerweise am Back-End Mediatoren für den Datenbankzugriff oder für die Verwaltung von Dateien entworfen und realisiert. Häufig ist es außerdem das Ziel, nicht nur anwendungsabhängige, sondern anwendungsunabhängige Mediatoren am Back-End zu entwerfen, um große, gegebene Wiederbenutzbarkeitspotentiale auszuschöpfen. So lassen sich analog zum Design des oben beschriebenen Frameworks zur Gestaltung grafischer Benutzeroberflächen am Front-End auch Frameworks für einen domänenweiten Einsatz von Mediatoren am Back-End entwerfen.

Durch die Einführung eines mächtigen Mediators – des Applikationsservers – in der mittleren Ebene und durch die damit verbundene Entkopplung von Front-End und Back-End ergeben sich folgende Vorteile für den Entwurf und die Realisierung verteilter Informationssysteme:

- *Skalierbarkeit.* Der Entwurf des Front-End als Thin-Client ist ein skalierfähiger Ansatz, um eine sehr große Anzahl an Benutzern effizient zu verwalten. Allgemein versteht Neuman [Neu94] unter der Skalierbarkeit die Zunahme von bestimmten Eigenschaften, ohne daß dabei das Informationssystem ein abnormes bzw. ungewöhnliches Systemverhalten aufweist. Nach Neuman gibt es drei wesentliche Dimensionen der Skalierbarkeit: Die Anzahl der Benutzer, die räumliche Verteilung der Komponenten und die Wartbarkeit des Gesamtsystems. Die Anzahl der Benutzer als Einheit für die Skalierbarkeit des gesamten Informationssystems spielt in vielerlei Hinsicht eine wichtige Rolle:

  - Aus Wartungsgesichtspunkten hinsichtlich der Client-Installationen haben Änderungen und Erweiterungen der Applikationslogik nicht notwendigerweise unmittelbare Auswirkungen auf den Client. Folglich sind im Gegensatz zum Fat-Client bei 2-stufigen Client/Server-Systemen nicht zwingend Neuinstallationen notwendig.
  - Erweiterungen an der Applikationslogik werden zentral an einer Stelle, dem Applikationsserver, vorgenommen und verwaltet. Die Entkopplung der Applikationslogik vom Client und die damit verbundene Kapselung der Logik in einem separaten, eigenständigen Mediator reduziert den Wartungsaufwand.
  - Jegliche Client-Zugriffe auf die Ressourcen bzw. Mediatoren am Back-End werden durch den Applikationsserver geleitet. Dadurch wird der Applikationsserver in die Lage versetzt, die ihm zur Verfügung stehenden Ressourcen bestmöglich zu nutzen und eingehende Anfragen entsprechend auf die aktuell vorhandenen Ressourcen zu verteilen. Steigt die Anzahl der Benutzer bzw. der Anfragen an das Informationssystem,

so kann der Applikationsserver als zentraler Mediator definierte Lastverteilungen vornehmen und trägt somit zur verbesserten Stabilität und Zuverlässigkeit des gesamten Informationssystems bei.

- *Wiederbenutzbarkeit*. Die (fachgerechte) Zerlegung der Applikationslogik in geeignete Mediatoren fördert die Wiederbenutzbarkeit. Aus softwaretechnischer Sicht werden die entsprechenden Module gebildet und können in einem breiteren Anwendungskontext benutzt werden, d. h. die Kapselungseigenschaften von Mediatoren hinsichtlich funktionaler Aspekte ermöglichen eine Wiederbenutzung dieser Funktionalitäten in mehreren Anwendungsfällen.Dies betrifft vorwiegend die Mediatoren am Back-End, da diese nach primär funktionalen und anwendungsunabhängigen Gesichtspunkten entworfen sind. Wie in Abbildung 2.5 auf der rechten Seite dargestellt ist, entstehen mehrere fein-granulare Mediatoren, welche strikt nach funktionalen Kriterien entworfen sind. Dagegen besitzt der Applikationsserver spezifisches Wissen über bestimmte anwendungsabhängige Informationsflüsse und -logiken, die sich oft nicht in weiteren Anwendungskontexten wiederbenutzen lassen.

- *Wartung*. Die verbesserten Wartungsmöglichkeiten der Softwarekomponenten in 3-stufigen Client/Server-Systemen hängt mit den bereits oben erwähnten Argumenten des Thin-Clients und der Wiederbenutzbarkeit zusammen: Fein aufgeteilte Module, die funktional zusammengehörende Aspekte in einem Mediator kapseln, sind geeignete Kandidaten für die Reduzierung des Wartungsaufwands innerhalb des Informationssystems.

Den oben beschriebenen Vorteilen stehen die im folgenden skizzierten potentiellen Probleme von 3-stufigen Client/Server-Architekturen gegenüber. Dabei fällt auf, daß manche der im vorigen Anschnitt beschriebenen Nachteile von 2-stufigen Client/Server-Architekturen noch nicht gänzlich behoben sind.
Obwohl die Aktivität und die Kontrollflüsse im Vergleich zu 2-stufigen Architekturen zwar auf mehrere Mediatoren verteilt sind, ist die Art der Interaktion unverändert durch das Anfrage-Antwort-Protokoll bzw. durch pull-basierte Interaktionen bestimmt. Die klaren und strikten Rollenzuteilungen eines Clients und Servers behindern an dieser Stelle komplexere Interaktionsszenarien. In Kapitel II wird ausführlich auf komplexe Interaktionsverfahren eingegangen.
Zudem sind die Dokumente unverändert von hoher Passivität geprägt. Der Zugriff auf Dokumente erfolgt ausschließlich über die Datenbank bzw. das Datenbankmanagement-System (DBMS). Die limitierten, oben beschriebenen Aktivitäts- und Interaktionseigenschaften von 3-stufigen Client/Server-Informationssystemen haben direkte Auswirkungen auf die Rolle der Dokumente. Durch primär pull-basierte Interaktionen und restriktive Rollenzuweisungen innerhalb des Systems und dessen Umgebung wird die zuvor in Abschnitt 2.1.2 erläuterte, tragende Rolle der Dokumente nicht angemessen abgebildet und umgesetzt. Vor allem in sehr

dynamischen Umgebungen ist eine flexiblere Rollenzuweisung hinsichtlich der beteiligten Systemkomponenten wünschenswert.
Die potentiellen Performanzprobleme, die durch die zusätzliche Einführung von Architekturebenen bzw. Mediatoren entstehen können, spielen in 3-stufigen Architekturen aus Integrations- und Flexibilitätsaspekten eine untergeordnete Rolle. Denn je mehr Flexibilität eine Architektur bietet, desto mehr Spielräume für (spätere) Anpassungen an komplexe Anforderungen eines verteilten Informationssystems läßt sie den beteiligten Mediatoren, Datenbasen und deren Datenlieferanten.

2.3.1.3 Dienstzentrierte Informationssysteme

Ein dienstzentriertes Informationssystem ist durch die Verallgemeinerung des Client/Server-Prinzips und dessen kennzeichnende Eigenschaften charakterisiert:

- Beteiligte Systemkomponenten agieren einerseits als Client und andererseits als Server. In dieser Hinsicht gibt es keine starren Rollenzuteilungen wie bei 2-stufigen und 3-stufigen Client/Server-Informationssystemen.

- Dem Mediatorprinzip folgend werden fein-granulare, funktionale Zerlegungen von Systemkomponenten in Dienste vorgenommen, so daß letztlich ein loser Verbund von zahlreichen Mediatoren entsteht, so wie es in Abbildung 2.5 unter dienstzentrierten Client/Server-Systemen illustriert ist.

- Die Art und Weise der Interaktion und Kommunikation orientiert sich dabei immer noch am klassischen Anfrage-Antwort-Protokoll, wobei aber jede Systemkomponente als Initiator einer Anfrage in Erscheinung treten kann.

Eine fein-granulare, funktionale Zerlegung von Systemkomponenten in Dienste unterstreicht die Argumentation in Abschnitt 2 über eine dienstorientierte Sichtweise auf Informationssysteme. Im Mittelpunkt stehen dort große dynamische und verteilte Informationssysteme, bei denen es weniger um die Verwaltung und Reglementierung eines globalen Kontrollflusses als vielmehr um eine adäquate Koordinierung autonomer und vorwiegend lokal operierender Systemkomponenten geht.
Im Kapitel 9 werden sehr fein-granulare und lokal autonome Mediatoren vorgestellt, die als Mikromediatoren bzw. Mikroserver für Dokumente [SK02] fungieren. Die damit verbundene Aktivierung von Dokumenten ermöglicht u. a. neue, semi-automatisierte Arbeitsformen, wie sie in Abschnitt 2.1.2.3 beschrieben sind. In Kapitel 9 wird zudem erläutert, inwiefern diese Art von Mediatoren die Grundlage für eine Reihe von Automatismen in Informationssystemen bildet. Das Ausschöpfen von Automatsierungspotentialen, welches grundsätzlich im Konzept der Mediatoren liegt, wird nach Wiederhold [WG97] eine wichtige Rolle bei der Realisierung von Informationssystemen spielen.

2.3.2 Netzwerkbasierte Informationssysteme

Das Internet als global vernetzende Infrastruktur bietet eine Vielzahl von Kommunikations- und Kollaborationsmöglichkeiten für Menschen aus unterschiedlichen Ländern. Dabei dient das Internet unter anderem als Zugangsstelle (Zugangsmedium) für die Nutzung einer immensen Anzahl verschiedener Informationsquellen. Die Art der Informationsquellen variiert dabei von einfachen Homepages über wissenschaftliche Aufsätze bis hin zu Netzwerkdiensten oder Datenbanken von Firmen, welche ihre Produkte im Internet bzw. World-Wide-Web (WWW)[9] anbieten. Hervorzuheben ist die enorme Informationsvielfalt und die Masse an Information – die sogenannte Informationsflut – im Internet.

Die Erfolgsfaktoren des WWW von einst – beispielsweise eine einheitliche Dokumentenbeschreibungssprache wie die *Hypertext Markup Language* (HTML) [RHJ99] oder ein simples Zugangs- bzw. Kommunikationsprotokoll wie das *Hypertext Transfer Protocol* (HTTP) [Wor00] – werden nun zu potentiell limitierenden Faktoren in einem immer größer werdenden globalen und dynamischen Netzwerk von Informationsanbietern und -konsumenten. So wurde frühzeitig an der *Extensible Markup Language* (XML) [BPSMM00] als Nachfolger von HTML gearbeitet, um konzeptionelle Schwächen und Einschränkungen von HTML – wie beispielsweise die fest vorgegebene Definition von Dokumentenstrukturen – zu beseitigen. Auffallend ist dabei, daß sich die Haupttechnologien des WWW – wie beispielsweise HTML und XML – in Anwendungsbereichen durchsetzen, die nicht originär aus dem WWW-Umfeld kommen. So gibt es Bemühungen in fast jedem Anwendungsgebiet, domänenweit einheitliche Spezifikationen zu bestimmen und diese dann in Form von XML zu beschreiben bzw. abzulegen. Dabei entstehen proprietäre XML-Sprachen[10] wie die Mathematical Markup Language (MathML)[11], die Electronic Business Markup Language (ebXML)[12] oder die Gene Expression Markup Language (GEML)[13]. Generell gilt es festzuhalten, daß die Technologien des WWW immer mehr selbst traditionelle Anwendungsdomänen und deren Informationssysteme durchdringen. Folglich ist eine Analyse und Untersuchung der kritischen Erfolgsfaktoren sowohl aus technischer als auch aus allgemeiner Sicht plausibel und vielversprechend für den Entwurf und die Realisierung zukünftiger Informationssysteme.

Im folgenden wird der grundsätzliche Aufbau netzwerkbasierter Informationssysteme am Beispiel des WWW aufgezeigt und deren kritische Erfolgsfaktoren für zukünftige Informationssysteme erläutert.

---

[9] Im folgenden werden das Internet und das World-Wide-Web synonym verwendet, da beide die inhärente, weltweite Vernetzung digitaler Computer-Ressourcen beschreiben. Allerdings bestehen aus technischer Sicht einige Unterschiede, die an späterer Stelle erläutert werden.

[10] Siehe http://xml.coverpages.org/siteIndex.html#toc-applications für eine Auflistung aktueller Dokumentensprachen basierend auf XML. Im April 2002 waren dort ca. 509 unterschiedliche Domänensprachen, die auf XML basierten, aufgelistet.

[11] http://www.w3.org/Math

[12] http://www.ebxml.org

[13] http://www.geml.org

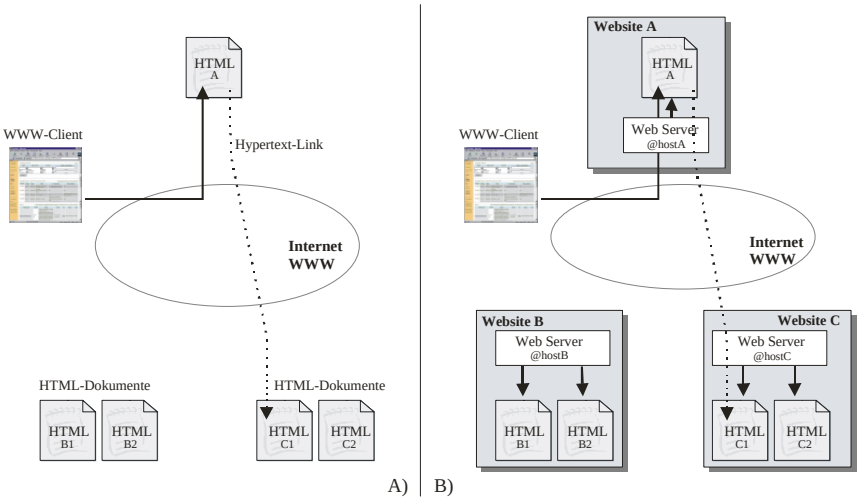


Abbildung 2.6: Schematischer Aufbau der netzwerkartigen Verknüpfung von Daten und Informationen im WWW.

2.3.2.1 Das Hypertextsystem WWW

Als Bush [Bus45] 1945 sein Informationssystem *memex* entwarf, stellte er sich ein individuell anpaßbares Informationssystem vor, welches auf Mikrofichen basierte. Dieses sollte ihm helfen, das Problem der bereits damaligen Informationsvielfalt zu lösen. Erst 1965 prägte T.H. Nelson den Begriff *Hypertext*, wobei damals primär die Nicht-Linearität von Textdokumenten gemeint war. Normalerweise werden Textdokumente in sequentieller Form realisiert, d. h. die darin enthaltenen Informationen - wie beispielsweise Sätze, Abschnitte oder Kapitel - werden nacheinander abgelegt. Die Definition eines Hypertext von Smith und Weiss [SW88] wird in dieser Arbeit als grundlegend vorausgesetzt, da diese einen Übergang bzw. eine Transformation von Hypertext-Konzepten auf unterschiedlichste Typen von Informationssystemen erlaubt:

> [...] hypertext is an approach to information management in which data is stored in a network of nodes connected by links [...]. Nodes can contain text, graphics, audio, video as well as source code or other forms of data.

Demnach ist ein Hypertext ein netzwerkbasiertes Informationssystem, in dem unterschiedlichste Arten von Daten – wie Text, Grafik oder Audiodaten – in einem Netzwerk von Datenknoten verwaltet werden. Dabei sind einzelne Knoten durch Links bzw. Verweise miteinander verbunden.
In Abbildung 2.6A sind Datenknoten als HTML-Dokumente repräsentiert, wobei die Dokumente *A* und *C1* unidirektional miteinander verlinkt sind: Dokument *A* enthält einen Hypertext-Link auf Dokument *C1*. Der konzeptionellen Sichtweise

in Abbildung 2.6A wird in Abbildung 2.6B überblicksartig die reale Umsetzung des WWW als Hypertextsystem gegenübergestellt:

- Der Zugriff auf einzelne Datenknoten bzw. deren Datenbasen (HTML-Dokumente *A*, *B1*, *B2*, *C1*, *C2*) erfolgt jeweils über eine spezielle Applikationskomponente, den Web-Server. Zugriffe auf geografisch bzw. inhaltlich zusammengehörende Datenknoten werden von einem gemeinsamen Web-Server (Web-Server *@hostB* und *@hostC*) verwaltet.

- Als einheitliche Beschreibungssprache der Dateninhalte innerhalb der Datenknoten im WWW wird HTML verwendet. Heterogene Daten- und Informationsquellen werden so wohl auf eine einfache Art und Weise dafür aber uniform repräsentiert.

- Jeder Datenknoten ist eindeutig innerhalb des WWW über einen sogenannten *Uniform Resource Identifier* (URI) bzw. *Uniform Resource Locator* (URL) [BLFM98] referenzierbar. Dadurch wird sowohl für lokale als auch für entfernte Datenknoten sowie für deren Inhalte ein einheitliches Adressierungsschema geschaffen, welches sehr gut in einer verteilten Umgebung wie dem Internet einsetzbar ist.

- Die Kommunikation von Dateninhalten zwischen Web-Servern innerhalb des WWW erfolgt stets über das zustandslose (und unidirektionale) Kommunikationsprotokoll HTTP. Zusammen mit dem Adressierungsschema URI gewährleistet ein Kommunikationsprotokoll wie HTTP, daß Benutzer solcher verteilter Informationssysteme leicht unterschiedliche Rollen innerhalb des Informationssystems einnehmen können. Einerseits fungieren sie als Informationslieferanten, die Daten und Informationen als HTML-Dokumente zur Verfügung stellen, andererseits agieren sie als Informationskonsumenten, die das vorhandene, verteilte Informationsangebot nutzen und in Anspruch nehmen.

Basierend auf den drei Säulen URI, HTML und HTTP gelang es T. Berners-Lee [BL99] das Hypertextsystem WWW zu schaffen, welches heute eine sehr hohe Durchdringung erreicht und eine immens große Anwendung im Alltag einer globalisierten Welt gefunden hat. Maßgeblich für den Erfolg und Durchmarsch des WWW als *das* globale Informationssystem unserer Zeit sind im wesentlichen zwei Eigenschaften[14]:

- Lokale Autonomie der Datenbasen.

- Dezentrale Systemarchitektur.

[14] Es sei darauf hingewiesen, daß selbstverständlich auch noch andere Faktoren zum Erfolg und zur großen Akzeptanz des WWW beigetragen haben. Aufgrund von geringerer inhaltlicher Relevanz zum Thema der vorliegenden Arbeit wird aber an dieser Stelle von einer ausführlicheren Diskussion abgesehen.

Lokale Autonomie der Datenbasen Unter der Autonomie der Datenbasen im WWW versteht man die Unabhängigkeit und die Selbständigkeit eines Informationslieferanten, Daten und Informationen in Form von HTML-Dokumenten weltweit zur Verfügung zu stellen. Aus allgemeiner Sichtweise entspricht das WWW einem integrativen Informationssystem, welches eine enorme Anzahl an räumlich verteilten Datenbasen in ein einziges Informationssystem, dem WWW, integriert. Für die Akzeptanz des WWW als Integrationsplattform war und ist es entscheidend, daß sich die einzelnen Datenbasen autonom voneinander weiterentwickeln können, d. h. die lokalen Dateninhalte unterliegen keinem global gelenkten und integrativen Vorgehen. Die einzige Voraussetzung ist dabei die Verwendung der einheitlichen Dokumentenbeschreibungssprache HTML.

Im Gegensatz dazu versuchen sogenannte föderierte Informationssysteme [SL90] als Integrationsplattform, für die partizipierenden, verteilten Datenbasen (zumeist Datenbank-Applikationen bzw. DBMS) ein global gültiges Datenschema zu finden bzw. zu generieren. Dieses wird dann systemweit dazu benutzt, um Informationsanfragen innerhalb des föderierten Informationssystems abzusetzen und abzuwickeln. Dabei werden globale Anfragen automatisch auf die jeweiligen lokal vorhandenen Datenschemata abgebildet. Im Vergleich zum Vorgehen innerhalb des WWW sind föderierte Informationssysteme meistens mächtiger und qualitativ besser in der Beantwortung von Informationsanfragen, da eine Menge von Metainformationen in den Informationsprozess miteinfließen. Im Gegensatz dazu bildet HTML eine wesentlich simplere Beschreibungsmöglichkeit für Dateninhalte, wobei Aspekte von Metainformationen nur rudimentär berücksichtigt werden.

Dezentrale Systemarchitektur Ein weiterer wichtiger Aspekt, der sich beim Entwurf des WWW als kritischer Faktor erwiesen hat, ist seine dezentrale Systemarchitektur. Eine solche Systemarchitektur zeichnet sich durch das Fehlen zentraler Systemkomponenten aus. Üblicherweise werden in modernen Informationssystemen wichtige Funktionen und Dienstleistungen von wenigen, oft zentralen Systemkomponenten übernommen. So ist ein Applikationsserver für ein Client/Server-Informationssystem oft die zentrale Systemkomponente, die jegliche Zugriffe auf Ressourcen innerhalb und außerhalb des betreffenden Informationssystems regelt. Fällt ein solcher zentraler Architekturbestandteil (single point of failure) aus, so wird die Anzahl der parallel anfragenden (aktiven) Benutzer oder das zu übertragende Datenvolumen zu groß, das die Gefahr von Teil- oder Totalausfällen des Informationssystems oder zumindest von Flaschenhälsen hinsichtlich der Performanz [Wie95] birgt. Deswegen ist der Entwurf geeigneter Systemkomponenten und der dazugehörigen Systemarchitektur unerläßlich.

Die grundlegenden Systemkomponenten des WWW sind HTML-Dokumente als Datenbasen und Web-Server als Zugriffsverwalter. Wie in Abbildung 2.6 veranschaulicht ist, bilden diese beiden Komponenten eine sogenannte Website. Eine Website ist demnach die eigentliche Systemeinheit innerhalb des WWW. Als verteiltes und verlinktes Informationssystem besteht das WWW aus einer sehr großen

Anzahl räumlich verteilter Websites, die allerdings keiner zentralen Kontrollinstanz unterliegen bzw. an keine zentrale Systemkomponente gebunden sind. So hat der Ausfall einzelner Websites keine fundamentalen Auswirkungen auf das verteilte Informationssystem als Ganzes.
Das Kommunikations- und Interaktionsprotokoll HTTP dient lediglich dazu, auf Daten bzw. Informationen von Websites zuzugreifen. Über die Verbindungsfunktion hinaus übernimmt das WWW weder Kontrollaufgaben noch Garantieregelungen. So werden beispielsweise hinsichtlich der inhaltlichen Qualität der Datenbasen oder der Zugriffszeiten auf einzelne Datenbasen und Websites keine Garantien übernommen.

2.4 Zusammenfassung

Abschließend läßt sich festhalten, daß eine Reihe von Anforderungen an zukünftige Informationssysteme gestellt werden, die beim Entwurf vieler heutiger Informationssysteme keine oder oft nur eine untergeordnete Rolle gespielt haben. In den Abschnitten 2.1.2.1-2.1.2.3 wurden konzeptionelle Kriterien für die Realisierung und Benutzung von Informationssystemen geschildert. In der folgenden Zusammenfassung sind die wesentlichen Herausforderungen an kommende Informationssysteme aufgelistet:

- Die Konzeption eines leichtgewichtigen Dokumentenbegriffs, der im Zentrum verteilter und dynamischer Informationssysteme steht, wird besonders hervorgehoben. Die Eigenschaft, Dokumente zu ändern und flexible Benutzungszeiten zu gestalten, wird in zukünftigen Informationssystemen immer mehr an Bedeutung gewinnen. Unterschiedlichste Sichten auf das gleiche Dokument und die Anpassungsfähigkeit von Dokumenten an bestimmte Anwendungskontexte – wie Benutzerverhalten oder örtliche Gegebenheiten der Ausführungsumgebung – werden in Zukunft von einem Dokument bzw. Dokumentenbegriff verlangt werden [Fuh00, AM00, SKN02].
- Das Überangebot von Informationen und die immense Steigerung der Anzahl an verfügbaren und aktiven Computerprozessoren werden neue Arbeitsformen und -weisen des Menschen erfordern [Ten00]. Im Mittelpunkt dabei steht die software-zentrierte Automatisierung von Interaktionen mit den jeweiligen Informationssystemen. Darüber hinaus sollten Informationssysteme Funktionalitäten hinsichtlich geeigneter Kollaborationsmöglichkeiten für die Benutzer zur Verfügung stellen.

In Abschnitt 2.2 wurde das wichtige Konzept der Mediatoren, die als verallgemeinerte Informationsbroker vielfältige Funktionen innerhalb von Informationssystemen einnehmen können, vorgestellt. Die meisten heutigen Informationssysteme auf der Grundlage von Client/Server beruhen letztlich auf dem Konzept von Mediatoren, wie sie Wiederhold [Wie92a, WRD+90] 1992 vorgestellt und in späteren Arbeiten [WG97, Wie01] erweitert hat.

In den Abschnitten 2.3.1 und 2.3.2 wurden ausführlich zwei typische Informationssystemarchitekturen – nämlich Client/Server-Informationssysteme und netzwerkzentrierte Informationssysteme – vorgestellt. Dabei wurde die Bedeutung und der Einsatz von Mediatoren in beiden Architekturtypen herausgearbeitet. Zudem wurden die Substanz und die Tragweite der Verwendung von standardisierten Kommunikationsprotokollen – wie beispielsweise HTTP – und Dokumentsprachen – wie beispielsweise HTML und XML – hervorgehoben. Die beiden betrachteten Architekturtypen bilden die Grundlage für den Entwurf und die Realisierung von proaktiven Informationssystemen.

Kapitel 3

Taxonomie für Informationssysteme

Um Informationssysteme hinsichtlich ihrer Eigenschaften und Systemarchitektur zu analysieren und klassifizieren, wird im folgenden Kapitel eine Taxonomie[1] für Informationssysteme erarbeitet. Diese Taxonomie orientiert sich dabei an den im vorhergehenden Kapitel 2 eingeführten drei wesentlichen Merkmalen, die nach Levy und Marshall [LM95] wichtige konzeptionelle Säulen im Bereich digitaler Informationssysteme bilden. Es sind dies die *Arbeitsweise* von Benutzern mit einem Informationssystem, der *Dokumenten-* bzw. *Informationsbegriff* und die *technologischen Gegebenheiten*.
Mit Hilfe der eingeführten Taxonomie werden in den darauf folgenden Kapiteln unterschiedliche Typen von Informationssystemen und deren Systemkomponenten klassifiziert und bewertet. Die vorgestellte Taxonomie ermöglicht eine klare Einordnung komplexer Sachverhalte innerhalb von Informationssystemen.
Konkret werden Informationssysteme und deren Komponenten anhand von drei Dimensionen beurteilt, wie es in Abbildung 3.1 illustriert ist:

- die *Benutzersicht*,
- der *Dokumentenbegriff* und
- die *technologische Infrastruktur*.

Die Einordnung der Komponenten in die Taxonomie erfolgt jeweils auch in grafischer Form basierend auf den drei Dimensionen.

3.1 Benutzersicht

Die Benutzersicht auf ein Informationssystem stellt den Benutzer und seine Arbeitsformen in den Mittelpunkt der Betrachtung. Im folgenden werden die inter-

[1] Taxonomie - griech.: Einordnung in ein bestimmtes System, [EHWM01].

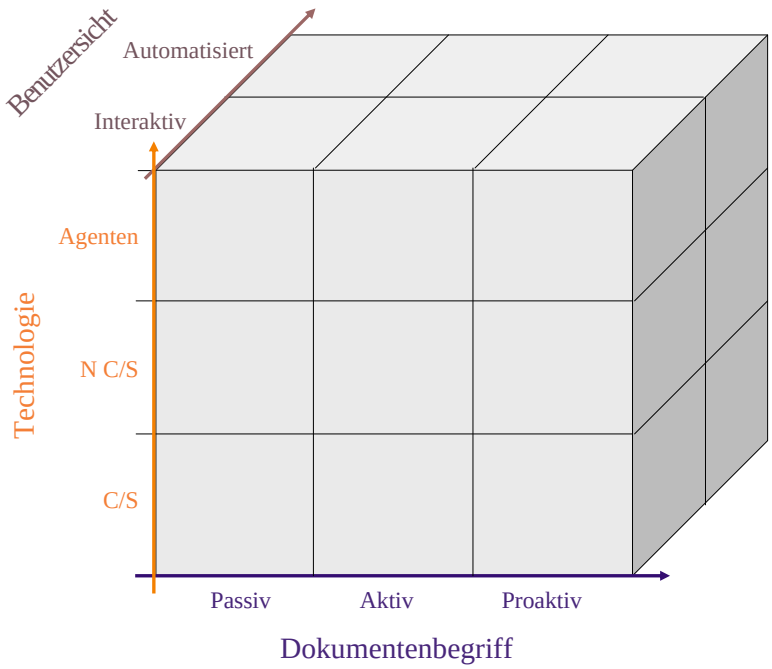


Abbildung 3.1: Taxonomie für Informationssysteme

aktive Benutzersicht als klassische Interaktionsform von Benutzern mit Informationssystemen und eine auf Automatismen in den Arbeitsweisen von Benutzern beruhende Sichtweise erörtert.

3.1.1 Interaktive Benutzersicht

Eine interaktive Benutzersicht ist typisch für viele verteilte Informationssysteme. Dabei steht der Mensch als Benutzer im Vordergrund der Betrachtung. Eine zentrale Rolle spielt dabei die Benutzerschnittstelle bzw. -oberfläche, welche dem Benutzer als zentrales Kommunikations- und Interaktionsmedium für den Umgang mit dem Informationssystem dient. Sowohl die von ihm initiierten Aktionen als auch die Präsentation der daraus entstandenen Ergebnisse werden durch die Benutzerschnittstelle aufgenommen und verarbeitet. In den folgenden Ausführungen werden die prinzipiellen Eigenschaften von Benutzerschnittstellen und -oberflächen vorgestellt.
Gemäß Moran [Mor81] versteht man unter einer Benutzerschnittstelle diejenigen Komponenten und Aspekte eines Informationssystems, mit denen die Benutzer begrifflich oder über ihre Sinne und ihre Motorik in Verbindung kommen. Zur Benutzerschnittstelle gehören damit auch das werkzeugspezifische und -unspezifische

Wissen des jeweiligen Benutzers, d. h. sowohl der Umgang und die Vertrautheit mit einer Benutzerschnittstelle als auch die anwendungsspezifischen Eigenheiten bilden die grundsätzlichen Anforderungen an einen menschlichen Benutzer hinsichtlich einer sinnvollen und zielorientierten Aufgabenbewältigung durch das jeweilige Informationssystem. Unter der Benutzeroberfläche versteht man alle Einheiten, Formen und Techniken, durch die Benutzer mit dem Informationssystem kommunizieren. Die Benutzeroberfläche ist der zu dem Informationssystem gehörende und für die Benutzer erkennbare bzw. sichtbare Teil der Benutzerschnittstelle [Wan93].
Zudem ist die klassische Sichtweise auf Benutzer von Informationssystemen gekennzeichnet durch sogenannte unmittelbare Interaktionen der Benutzer. Eine unmittelbare Interaktion hat zwei wesentliche Eigenschaften:

- *Unmittelbare Visualisierung*. Die angestoßenen Benutzeraktionen werden innerhalb der Benutzeroberfläche angemessen repräsentiert, d. h. der Benutzer erhält über die initiierten Interaktionen entsprechende Rückmeldungen, die durch die Benutzeroberfläche dem Benutzer unmittelbar visualisiert werden. Die Aktionen und Ergebnisse werden dabei weniger beschrieben, sondern möglichst konkret oder durch Analogien dargestellt.

- *Unmittelbare Ausführung*. Die Benutzeraktion wird durch die Benutzeroberfläche direkt zur Ausführung gebracht und damit an die entsprechenden, für die Ausführung verantwortlichen Systemkomponenten weitergeleitet. So stellt der Benutzer in Informationssystemen, die auf Client/Server-Architekturen basieren, das Front-End dar, welches eingehende Anfragen an die entsprechenden nachgelagerten Systemkomponenten weiterleitet[2]. In dieser Hinsicht entspricht eine unmittelbare Ausführung von Interaktionen dem Anfrage-Antwort-Interaktionsprotokoll (siehe Abschnitt 2.3.1.1), in dem der Benutzer als Initiator von Anfragen, die synchron ausgeführt werden, fungiert.

Shneiderman [Shn89] führte - basierend auf unmittelbaren Interaktionen - das Konzept sogenannter direkt manipulativer Benutzerschnittstellen ein, die dem Benutzer auf dem Bildschirm eine gegenständliche Modellwelt mit den zu bearbeitenden Objekten des Informationssystems bietet. Dabei findet eine Interaktion vor allem durch die Ausführung von grafischbasierten, räumlichen Operationen statt.
Abschließend lassen sich folgende Charakteristika einer interaktiven Benutzersicht auf Informationssysteme festhalten, die für die vorgestellte Taxonomie für Informationssysteme von Bedeutung sind:

1. *Menschliche Benutzer*. Gemäß der Abbildung 2.1 auf Seite 10 über den schematischen Aufbau von Informationssystemen gibt es unterschiedliche Arten von Benutzern. Eine interaktive Benutzersicht schiebt den Menschen

[2] Vgl. die Beschreibung von Client/Server-Informationssystemen in Kapitel 2.3.1.

als Benutzer eines Informationssystems in den Vordergund der Betrachtung. Gleichzeitig werden Benutzeraspekte von Softwareprozessen als sogenannte digitale Benutzer vernachlässigt oder gar nicht berücksichtigt. Der Mensch in einer interaktiven Benutzersicht fungiert als Aktor und Initiator, der so maßgeblichen Einfluß auf die Informations- und Kontrollflüsse eines Informationssystems ausübt.

2. *Benutzerschnittstelle*. Durch die starke Gewichtung des Menschen in einer interaktiven Benutzersicht gewinnen Aspekte des Entwurfs und der Realisierung von adäquaten Benutzerschnittstellen enorm an Bedeutung, da die Benutzerschnittstelle die primäre und oft einzige Systemkomponente zur Interaktion und Kommunikation mit einem Informationssystem darstellt. Es ist die Aufgabe der Benutzerschnittstelle, syntaktisch korrekte Kommunikationsschnittstellen mit dem Informationssystem bzw. den restlichen Systemkomponenten zu etablieren, die vom Menschen dann genutzt werden können.

3. *Implizites Wissen*. Die Kenntnis um inhaltliche Bedeutung der Daten und Vorgänge für eine sinnvolle Interaktion mit einem Informationssystems obliegt häufig dem Menschen selbst. Die Semantik der Anwendung ist oft in impliziter Form definiert, d. h. der Mensch kennt a priori die semantischen Schnittstellen, um ein Informationssystem und dessen Systemkomponenten – wie z. B. eine Benutzeroberfläche – sinnvoll zu benutzen. Falls Wissensdefizite in der inhaltlichen Bedeutung auf Seiten des Menschen bestehen, werden diese beispielsweise über Schulungen oder Fortbildungen behoben. Trotzdem bleiben Aspekte der inhaltlichen Bedeutungen der Anwendungsdomäne, -logik und deren Interpretationen in impliziter Form gegeben und werden vorwiegend über nicht automatisierte digitale Kommunikationskanäle behoben.

4. *Unmittelbarkeit*. Die initiierten Interaktionen werden – basierend auf direkt manipulativen Benutzerschnittstellen – zwischen Mensch und Informationssystem unmittelbar ausgeführt. Dies hat Auswirkungen sowohl auf die Darstellung in der Benutzeroberfläche als auch auf den Kontrollfluß des Informationssystems als Ganzes.

3.1.2 Automatisierte Benutzersichten

Eine von Automatismen geprägte Benutzersicht baut auf den in Abschnitt 2.1.2.3 vorgestellten kooperativen und automatisierten Arbeitsformen auf. In der folgenden Zusammenfassung werden die wesentlichen Eigenschaften nochmals hervorgehoben:

- Der Mensch fungiert im Gegensatz zu einer strikt interaktiven Benutzersicht primär als Administrator zahlreicher fortwährend ablaufender Softwarepro-

zesse. Dabei stehen für den Menschen Aspekte des Systemmanagements dieser Prozesse im Vordergrund.

- Automatisierte Arbeitsformen implizieren Interaktions- und Arbeitsformen, die weniger von menschlichen Benutzern direkt ausgelöst und ausgeführt werden. Vielmehr stehen dabei digitale Benutzer bzw. Softwareprozesse im Vordergrund, die über bekannte und einheitliche Kommunikationsschnittstellen miteinander und mit den Systemkomponenten von Informationssystemen interagieren.

Im allgemeinen setzen software-bezogene Automatismen einen bestimmten Grad an Aktivitäten voraus, die fortwährend und kontinuierlich betrieben werden. In dieser Hinsicht besteht eine Verbindung zu denjenigen Informationssystemarchitekturen, die geeignete Aktivitätseigenschaften vorweisen. Diese beruhen nicht nur auf klassischen, synchronen Anfrage-Antwort-Interaktionen[3], sondern ermöglichen auch asynchrone Interaktionen. Eine asynchrone Interaktion ist durch die zeitliche Entkopplung des Informationsflusses beim klassischen Anfrage-Antwort-Interaktionsprotokoll gekennzeichnet. Die Anfrage wird von der Antwort in zeitlicher Hinsicht getrennt. Anfragen und Antworten werden jeweils als unidirektionale Informationsflüsse betrachtet.
Generell schließt eine automatisierte Benutzersicht traditionelle, interaktive Benutzer nicht aus, sondern betont die Automatisierungsmöglichkeiten in den Arbeitsweisen von Benutzern. So wird selbstverständlich die Verwaltung und Administration der Softwareprozesse durch eine interaktive Benutzersicht gesteuert. In dieser Hinsicht ist die inhaltliche Nähe einer interaktiven und einer von Automatismen bestimmten Benutzersicht offensichtlich. Dennoch verfolgen automatisierte Benutzersichten und Arbeitsformen unterschiedliche Zielsetzungen, wie sie in Abschnitt 2.1.2.3 beschrieben sind.
Zusammenfassend lassen sich folgende wesentlichen Eigenschaften einer von Automatismen geprägten Benutzersicht festhalten:

1. *Mensch als Administrator*. Die Rolle des Menschen ändert sich von einem interaktiven Benutzer, der als Initiator sämtlicher Aktionen maßgeblich die Informations- und Kontrollflüsse innerhalb eines Informationssystems bestimmt, zu einem Manager von selbständig und fortwährend ablaufenden Softwareprozessen.

2. *Digitale Benutzer*. Neben dem menschlichen Benutzer als Administrator treten digitale Benutzer von Informationssystemen herausragend in Erscheinung. Die bei einer interaktiven Benutzersicht zentralen Aspekte und Fragestellungen des Entwurfs und der Realisierung adäquater Benutzerschnittstellen verlieren folglich an Bedeutung und treten eher in den Hintergrund.

[3] Synchrone Anfrage-Antwort-Interaktionen werden in Abschnitt 2.3.1.1 auf Seite 30ff im Zusammenhang mit Client/Server-Informationssystemen beschrieben.

Dagegen treten Aspekte flexibler, erweiterbarer und uniform erreichbarer Kommunikationsschnittstellen der beteiligten digitalen Benutzer in den Vordergrund. Diese Fragestellungen beziehen sich sozusagen auf die Benutzerschnittstelle digitaler Benutzer und bilden so das Pendant zur zentralen Bedeutung der Gestaltung von Benutzeroberflächen in der interaktiven Benutzersicht.

3. *Explizites Wissen*. Syntaktische und inhaltliche Bedeutungen der Daten und Informationen sind aus der Sicht eines digitalen Benutzers von großer Bedeutung. Im Vordergrund stehen dabei nicht nur die syntaktischen, korrekt ausgetauschten Dateninhalte zwischen den beteiligten Softwareprozessen, sondern es stellt sich auch die Frage der Bedeutung der Daten. In einer offenen und dynamischen Systemumgebung, in der eine potentiell immense Anzahl an Softwareprozessen aktiv ist, sind die Anforderungen an flexible und explizite Beschreibungsverfahren von Bedeutungsinhalten wichtig. Durch standardisierte und externalisierte Darstellungsformen können inhaltliche Bedeutungen und Zusammenhänge auf automatisierte Art und Weise verarbeitet werden.

4. *Mittelbarkeit*. Mittelbare Interaktionen haben analog zu unmittelbaren Interaktionen Auswirkungen sowohl auf die Visualisierung in einer Benutzeroberfläche für den Administrator als auch auf die Informations- und Kontrollflüsse in einem Informationssystem. Letzterer Aspekt ist besonders entscheidend, da nicht mehr nur synchrone, sondern auch asynchrone Kommunikations- und Interaktionsverfahren eine tragende Rolle einnehmen. Die durch die asynchrone Ausführung von Interaktionen resultierenden komplexen und nicht notwendigerweise deterministisch bestimmbaren Informationsflüsse müssen auf der Basis von geeigneten Kooperationsstrukturen für die aktiven Softwareprozesse ablaufen.

Zusammenfassend sind in Tabelle 3.1 die Eigenschaften einer interaktiven und einer von Automatismen geprägten Benutzersicht gegenübergestellt.

3.2 Dokumente

Dokumente bilden eine der drei wesentlichen Säulen von Informationssystemen, wie in Abschnitt 2.1.2.1 ausführlich erläutert wird. Dokumente als Ausprägungen digitaler Artefakte bilden einen sehr allgemeinen und weit gefaßten Dokumentenbegriff [LM95]. Basierend auf den Ausführungen über Eigenschaften und Aspekten von Dokumenten in Abschnitt 2.1.2.1 werden Dokumente in den folgenden Abschnitten in drei Kategorien eingeteilt: *Passive*, *aktive* und *proaktive* Dokumente. Diese gewählte Einteilung beruht auf dem Aktivitätsgrad eines Dokuments, der eine wesentliche Voraussetzung für die Automatisierung von Abläufen in Informationssystemen darstellt, so wie es bereits in Abschnitt 2.1.2.3 über automatisierte

Benutzersicht

interaktiv	automatisiert
• Mensch im Mittelpunkt • Benutzerschnittstelle • Implizites Wissen • Unmittelbarkeit der Visualisierung und synchronen Ausführung	• Mensch als Administrator • Digitale Benutzer • Explizites Wissen • Mittelbarkeit der Visualisierung und asynchronen Ausführung)

Tabelle 3.1: Gegenüberstellung der Eigenschaften verschiedener Benutzersichten.

Arbeitsformen erläutert ist. Zunächst werden anhand von unterschiedlichen Konzepten zur Dokumentenorganisation in Datei- und Datenbanksystemen einige bestehende Probleme erläutert, die auf einen mangelhaften bzw. nicht ausreichend flexiblen Dokumentenbegriff zurückzuführen sind.

3.2.1 Dokumentenorganisationen

Eine Dokumentenorganisation liefert die konzeptionellen Strukturen zur Verwaltung von mehreren Dokumenten. Die Organisation von Dokumenten in hierarchischen Dateisystemen ist ein typisches Beispiel für das Management von Dokumenten auf Arbeitsplatzrechnern. Generell werden Dokumente in einer Dokumentensammlung (z. B. in einem hierarchischen Dateisystem oder in einer Datenbank) zusammengeführt, um danach sinnvolle Informationsanfragen auf der Basis dieser Kollektionen zu stellen.
Eine Informationsanfrage orientiert sich dabei an den von der Dokumentenorganisation zur Verfügung gestellten Suchmöglichkeiten. So werden die gesuchten Dokumente über die entsprechende Spezifizierung der erwünschten Dokumentenattribute gefunden. Jede Dokumentenorganisation verwaltet entsprechende Zusatzinformationen und Metadaten bezüglich aller in der Dokumentensammlung befindlichen Dokumente.
In den folgenden Ausführungen werden konkrete Probleme und Fragestellungen bei der Dokumentenorganisation in Dateisystemen (Abschnitt 3.2.1.1) und in Datenbanken (Abschnitt 3.2.1.2) beispielhaft erläutert.

3.2.1.1 Probleme bei der Organisation in Dateisystemen

Sowohl der Zugriff als auch die Suche nach Dokumenten, die in Dateisystemen abgelegt sind, orientieren sich primär an der Lokation (Örtlichkeit) der Dokumente,

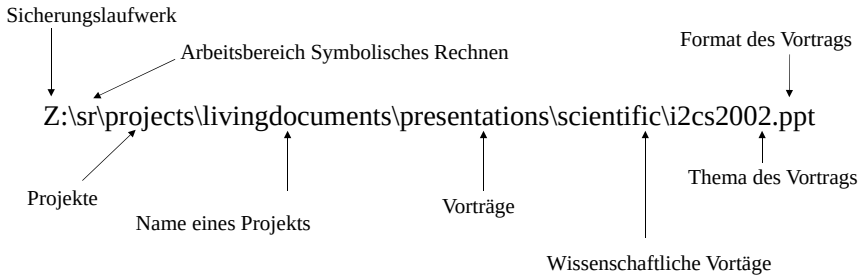


Abbildung 3.2: Zugriff auf Dokumente in hierarchischen Dateisystemen.

wie es in Abbildung 3.2 exemplarisch dargestellt ist. Dokumente in Dateisystemen werden in sogenannten Ordnern abgelegt. Jeder Ordner wird mit einem Namen versehen, der die darin enthaltenen Dokumente charakterisieren sollte. Dann werden die Ordner in einer hierarchischen Dateistruktur hinterlegt. Der Zugriff auf ein bestimmtes Dokument erfolgt nun über die Lokation innerhalb der Dokumenthierarchie.
Der Name jedes Ordners übernimmt dabei eine Metafunktion, d.h. über den Ordnernamen werden Dokumente und deren Eigenschaften zusätzlich beschrieben. In Abbildung 3.2 spezifiziert der Ordner mit dem Namen *projects* alle projektrelevanten Dokumente. Durch die hierarchische Anordnung stehen alle darin enthaltenen Dokumente und Unterordner inhaltlich miteinander in Beziehung.
Die Fokussierung auf die Lokation anstelle auf die Eigenschaften eines Dokuments hat einige potentielle Nachteile hinsichtlich einer skalierfähigen Dokumentenorganisation:

1. Die inhaltliche Bedeutung des Ordnernamens, die Semantik, ist implizit und ausschließlich für eine bestimmte Gruppe von Benutzern oder dem jeweiligen Dokumentenerzeuger zugänglich und bekannt.

2. Die Modifikation eines Ordnernamens kann zu Inkonsistenzen in der Dokumentenorganisation unterhalb der betroffenen Dokumentenhierarchie führen. Die weiter oben angeführte Struktur der inhaltlichen Beziehungen der Dokumente untereinander kann inkonsistent werden und folglich können bestimmte Dokumente nicht mehr oder nur schwer wiedergefunden werden. Die Wahrscheinlichkeit von Inkonsistenzen steigt mit der Größe und Komplexität der hierarchischen Ordnerstruktur.

3. Die Metadaten – also die einzelnen Ordnernamen und deren hierarchische Anordnung – sind von den eigentlichen Dokumenten getrennt. Auf der Basis eines Dokuments allein ist es schwer, Aussagen über die jeweiligen Metadaten und deren Eigenschaften zu treffen. Für eine sinnvolle Bearbeitung eines Dokuments ist seine aktuelle Lokation in der Hierarchie und damit die Beschreibung seiner Eigenschaften unbedingt erforderlich.

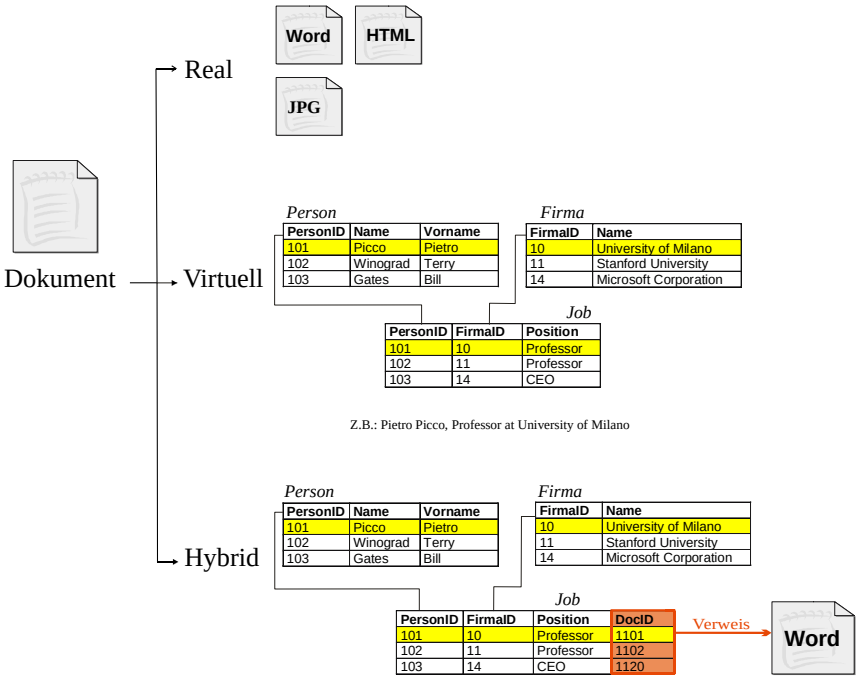


Abbildung 3.3: Reale, virtuelle und hybride Dokumente.

4. In einer verteilten Mehrbenutzerumgebung, in der Teile der hierarchischen Dokumentensammlung zwischen mehreren Benutzern geteilt werden, wiegen die unter (1-3) beschriebenen potentiellen Probleme um ein Vielfaches größer und komplexer.

3.2.1.2 Probleme bei der Organisation in Datenbanken

Die Organisation und Verwaltung von Dokumenten in Datenbanken ist seit geraumer Zeit ein etabliertes und renommiertes Forschungsfeld. Papazoglou [Pap93] hebt die Bedeutung traditioneller, datenbankzentrierter Informationssysteme hervor, die primär für informations- und datenintensive Aufgaben in verteilten Umgebungen konzipiert sind bzw. waren.
Der Begriff des Dokuments und der Daten wird im folgenden synonym verwendet. Digitale Daten, die in Datenbanken verwaltet werden, sind nach dem allgemeinen Dokumentenbegriff als Dokumente zu betrachten. Sie sind virtuelle Dokumente und besitzen im Gegensatz zu realen Dokumenten kein reales Gegenstück, wie es in Abbildung 3.3 illustriert ist. Ein virtuelles Dokument ergibt sich durch die Aggregation einzelner Datenbankattribute, die sich über mehrere Datenbanktabellen erstrecken können. In Abbildung 3.3 entspricht das virtuelle Dokument der Person *Picco* den Informationen, die sich in den normalisierten Datenbanktabellen *Person*,

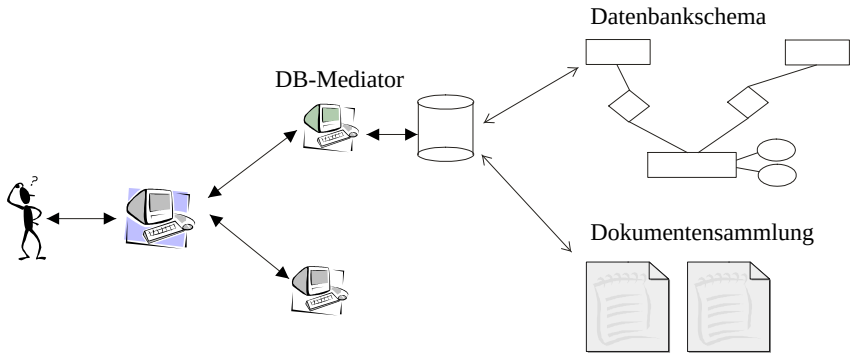


Abbildung 3.4: Überblick von Dokumentenorganisationen in Datenbanken.

Firma und *Job* befinden: *Pietro Picco ist Professor an der Universität Mailand*. Zudem gibt es hybride Dokumente, die eine Mischung aus realen und virtuellen Dokumenten darstellen. Der reale Teil wird über einen Verweis oder Link mit dem virtuellen Teil des hybriden Dokuments verbunden.

Kennzeichnend für die Systemarchitektur von Datenbanken sind drei unterschiedliche Abstraktionsebenen [Dat95]. Die physische Ebene ist für den normalen Benutzer nicht relevant, da dort etwaige Speicher- und Indexstrukturen festgelegt werden. Die logische Ebene bestimmt die Metaeigenschaften bzw. das Datenbankschema des jeweiligen Informationssystems. Die dritte Ebene der Sichten bietet die Möglichkeit, verschiedene Sichtweisen auf die logische Ebene zu generieren. So können benutzerspezifische Sichten auf das logische Datenmodell erzeugt werden. Das Hauptziel bei der Einführung unterschiedlicher Abstraktionsebenen ist die Datenunabhängigkeit. Durch die Etablierung klar definierter Schnittstellen zwischen den Ebenen kann die jeweilige tatsächliche Realisierung der Ebenen flexibel variieren[4].

Historisch betrachtet leisteten Codd [Cod70] und Chen [Che75] in den 70-er Jahren auf dem Gebiet der relationalen Datenbanken wegweisende Arbeiten, die Entwurf und Realisierung von Datenbanksystemen nachhaltig beeinflußt haben. Eine einfache und uniforme Darstellung von Daten - basierend auf einem soliden mathematischen Gerüst - ist dabei vor allem zu nennen.

Auf der Grundlage der skizzierten Systemarchitektur von Datenbanksystemen entstanden viele der heutigen Informationssysteme. In Abbildung 3.4 ist ein mehrstufiges Client/Server-Informationssystem mit einem Datenbanksystem am Back-End illustriert. Ein spezieller Mediator (*DB-Mediator*) ist für die Zugriffe auf die Datenbank und andere datenbankbezogenen mediatorischen Dienste verantwortlich. Zudem werden die tatsächlichen Dokumente in einer Dokumentensammlung bzw. in einem digitalen Archiv gehalten. Es wird lediglich ein Verweis auf die Loka-

[4]Die Einführung unterschiedlicher Abstraktionsebenen entspricht prinzipiell dem Mediatorkonzept, das in Abschnitt 2.2 vorgestellt wird.

tion des Dokuments in der Datenbank zusammen mit anderen Dokumenteneigenschaften mitgeführt und verwaltet. Der Einsatz eines separaten Dokumentenarchivs ist ein gängiges Vorgehen in großen wissenschaftlichen und kommerziellen dokumenten-bezogenen Informationssystemen und dient in Abbildung 3.4 zur Illustration potentieller Probleme bei der Organisation von Dokumenten in Datenbanken[5]:

- Die Dokumentensuche erfolgt strukturiert über die in der Datenbank abgelegten Metaeigenschaften. Diese Metaeigenschaften basieren – wie zuvor erläutert wurde – auf einem Datenbankmodell, welches bei der Erstellung des Informationssystems festgelegt wurde. Die Kopplung des Datenmodells an die Applikationslogik des Informationssystems ist eine sinnvolle und gängige Praxis, wenn es um die adäquate Abbildung von klar bestimmbaren Anforderungen an ein singuläres Informationssystem geht. Dabei werden allerdings die Bedeutung und die allgemeinen Eigenschaften der Dokumente in den Hintergrund gedrängt. Falls die gleichen Dokumente in unterschiedlichen Informationssystemen benutzt werden, ist diese konventionelle Vorgehensweise nicht optimal, da etwaige Dokumente in mehreren Informationssystemen unterschiedlich benutzt werden. Häufig werden dafür mehrere Instanzen des gleichen Dokuments erzeugt. An dieser Stelle wäre ein dokumentenzentrierter Ansatz wünschenswert, in dem in Abhängigkeit des aktuellen Benutzungkontextes die entsprechenden Eigenschaften einer singulären Dokumenteninstanz abgefragt werden können.

- Die Metaeigenschaften werden statisch festgelegt und durch die Generierung eines Datenbankmodells fixiert. Dieses Modell ist für die gesamte Dokumentensammlung maßgeblich und wird so auf alle Dokumente angewendet. Eine Änderung der Metaeigenschaften bzw. des Datenbankschemas ist aus Sicht eines normalen Benutzers oft gar nicht oder nur schwer möglich. In vielen Fällen ziehen Änderungen der Metaeigenschaften komplette und teure Neuentwicklungen der Applikationen nach sich.

- Wie in Abbildung 3.4 veranschaulicht ist, sind wichtige Bestandteile wie die Metadaten, die eigentlichen Dokumente und die Systemkomponenten (z. B. die *DB-Mediatoren*) voneinander getrennt. Während die Metadaten im Datenbankschema liegen, befinden sich die eigentlichen Dokumente in der Dokumentensammlung. Die Applikationslogik und Aktivität hinsichtlich des Informations- und Kontrollflusses sind in den Mediatoren lokalisiert, die räumlich voneinander getrennt sein können. Diese Art der Trennung bezeichnet Brodi [Bro99] als „künstliche Trennung von Daten und Applikationslogik, die Daten als Mitglieder zweiter Klasse behandelt“. Dokumente werden in dieser Hinsicht reduziert zu passiven Datencontainern.

[5] Es sei darauf hingewiesen, daß es im folgenden Verlauf primär darum geht, mögliche Gefahren oder Probleme bei der Organisation von Dokumenten zu beschreiben. Dabei wird keine komplette Ablösung bestehender und etablierter Datenbankkonzepte und -technologien gefordert.

Ein wichtiges Ziel sollte es sein, auf Dokumente ausschließlich über ihre Eigenschaften zuzugreifen. Auf der Grundlage der Beobachtungen in den vorhergehenden Abschnitten über die Organisation von realen, virtuellen und hybriden Dokumenten in Dateisystemen und Datenbanken werden in den folgenden Ausführungen drei Dokumentenbegriffe vorgestellt, welche die inhärente Bedeutung von Dokumenten innerhalb von Informationssystemen unterstreichen. Die vorgestellten Dokumentenbegriffe bilden eine wesentliche Säule in der Taxonomie für Informationssysteme.

3.2.2 Passive Dokumente

Generell entspricht ein passives Dokument einem Container, der Daten und Informationen beinhaltet. Dieser Container besitzt keine eigenen Aktivitätspotentiale. Dadurch ist er nicht in der Lage, selbständig Informations- und Kontrollflüsse innerhalb eines Informationssystems in irgendeiner Form zu beinflussen. In dieser Hinsicht ist ein passives Dokument von anderen Systemkomponenten abhängig, welche als aktive Entitäten in einem Informationssystem fungieren und bei Bedarf auf das passive Dokument und dessen Informationsinhalte zugreifen.
Die strukturellen Eigenschaften und Beziehungen passiver Dokumente werden als Metadaten erfaßt und beschrieben. Zwei typische Beispiele für reale und virtuelle Dokumente, die gemäß der oben angeführten Ausführungen als passive Dokumente klassifiziert werden können, sind die im vorigen Abschnitt über Dokumentenorganisationen beschriebenen Dokumente, welche in Dateisystemen und Datenbanken verwaltet werden:

1. *Dateisysteme*. Die Metadaten des realen Dokuments werden durch die hierarchische Anordnung bestimmt. Das reale Dokument enthält Informationen, die passiv auf dem Dateisystem abgelegt sind.

2. *Datenbanken*. Die Metadaten virtueller oder hybrider Dokumente werden in Datenbankschemata spezifiziert und verwaltet. Die einem virtuellen Dokument zugrundliegenden Datensätze sind dabei passiv in der Datenbank abgelegt[6].

Im allgemeinen unterliegen die Metadaten passiver Dokumente einer relativ starren Definition, d. h. sobald die strukturellen Eigenschaften der Dokumente beschrieben worden sind, werden diese zunächst als fix und unveränderbar betrachtet. Spätere Modifikationen an den Metadaten sind möglich, werden allerdings eher selten vollzogen. Typisch für passive Dokumente sind sowohl die örtliche Trennung der Me-

[6]Eine Erweiterung von traditionellen Datenbanken, die sogenannten aktiven Datenbanken [DHW95], ermöglicht über die Spezifikation von Triggern, daß unter bestimmten Bedingungen und Zuständen Daten automatisch hinsichtlich eines bestimmten Zieles modifiziert werden. Da dieser Aktivierungsprozeß nicht von den Daten bzw. Dokumenten erbracht wird, sondern von aktiven Systemkomponenten innerhalb des Datenbankmanagement-Systems, stellen aktive Datenbanken keine grundsätzliche Erweiterung des hier vorgestellten passiven Dokumentenbegriffs dar.

tadaten von den eigentlichen Dokumenteninhalten als auch der ausschließliche Zugriff auf die Metadaten passiver Dokumente von externen Systemkomponenten[7]. Letzteres ergibt sich offensichtlich aus den fehlenden Aktivitätseigenschaften passiver Dokumente. In Client/Server-Informationssystemen, die in Abschnitt 2.3.1 diskutiert werden, agieren datenbankspezifische Mediatoren (siehe Abschnitt 2.2) als primäre Träger von Aktivitäten.
Aufgrund der geringeren Änderungsraten der Dokumenteninhalte und Metadaten eignen sich passive Dokumente für die Realisierung von Informationssystemen, in denen sehr große und systematisch geordnete Datenmengen bestehen. Üblicherweise werden dort Teilinformationen der Dokumente in speziellen Datenstrukturen, in sogenannten *Indexen*, erfaßt und komprimiert, auf deren Basis Informationssysteme Anfragen an die Datenbasis bzw. Dokumentensammlung dann schnell und effektiv beantworten können [Pap93].
Konzeptionell stellt ein Index eine zentrale Datenstruktur dar, die einen Teilauszug der Datenbasis zu einem bestimmten Zeitpunkt als Kopie enthält. Durch die Passivität der Dokumente werden die benötigten Informationen über Pull-Verfahren in einen Index überführt. Es ist die Aufgabe des für die Indexerstellung verantwortlichen Dienstes, die Konsistenz und Aktualität des Indexes zu gewährleisten. Selbst Web-Suchmaschinen wie *Google* [Goo02,BP98b] oder *AltaVista* [Alt02] bedienen sich eines Indexes, um Informationsanfragen an passive, räumlich verteilte HTML-Dokumente effizienter und vor allem performanter beantworten zu können. Die Konsistenzanforderung an den erstellten Index wird dabei sehr schwach interpretiert. Innerhalb gewisser Zeitspannen – z. B. von mehreren Monaten – werden die verteilten Datenbasen (HTML-Dokumente) von Web-Crawlern heruntergeladen und erfaßt. Schließlich werden die gesammelten Informationen in den Index eingefügt. Dieser Crawling-Ansatz entspricht wiederum einem Pull-Verfahren, in dem spezielle, aktive Softwareprozesse (Crawler) Datenbasen im Web nach bestimmten Informationen abfragen. Die grundsätzliche Passivität von HTML-Dokumenten erfordert eine zusätzliche aktive Komponente, den Web-Crawler, für die Informationssammlung.
Abschließend lassen sich folgende Beobachtungen hinsichtlich eines passiven Dokumentenbegriffs festhalten:

1. Fehlende Aktivitätskomponenten in passiven Dokumenten erfordern zusätzliche, aus Sicht der Dokumente externe und aktive Systemkomponenten. Die Passivität führt in dieser Hinsicht zu einem nicht abgeschlossenen Dokumentenbegriff, d. h. hinsichtlich der Kapselungseigenschaften sind passive Dokumente von weiteren Komponenten abhängig. Diese aktiven Komponenten dienen als grob-granulare Verwalter und Mediatoren der Dokumenteninhalte und organisieren den Zugriff auf die Dokumente. Sie stellen das notwendige Bindeglied zwischen Dokumenteninhalt und Metadaten dar. Die

[7] Aus Sicht eines passiven Dokuments ist die Systemkomponente, die auf die Metadaten zugreift, nicht direkt zum betreffenden Dokument gehörend, sondern stellt vielmehr eine externe Systemkomponente dar.

angesprochene grob-granulare Eigenschaft der Mediatoren bezieht sich dabei auf die Anzahl der Dokumente, die einem solchen Mediator zugeordnet sind. Im Regelfall ist diese Anzahl der Dokumente sehr groß.

2. Die Passivität unterstützt die Bildung eines zentral verwalteten Indexes, der die Basis für eine performante Beantwortung von Informationsanfragen darstellt. Allerdings fördert solch ein Index keine lokale Autonomie von verteilten Datenbasen und deren passiver Dokumente, sondern betrachtet alle Dokumente hinsichtlich eines einheitlichen Schemas.

3. Ein passiver Dokumentenbegriff kommt in zahlreichen datenbankzentrierten Informationssystemen zum Einsatz. Dabei ist auffallend, daß diese Informationssysteme mit großen Datenmengen hantieren, die grundsätzlich in Datenbanken verwaltet werden. Die erforderlichen aktiven Systemkomponenten werden oft auch direkt in das Datenbankmanagement-System integriert und als spezieller Datenbankdienst zur Verfügung gestellt. Allerdings finden sich in der Literatur mehrere Artikel namhafter Autoren, die über den zukünftigen Entwurf von datenbankzentrierten Informationssystemen sowie über die Rolle der Daten bzw. Dokumente im allgemeinen elaborieren:

   - Papazoglou [Pap93] hebt hervor, daß die physische Verteilung der Dokumente bzw. der Datenbasis [8] ein wichtiges Kriterium für künftige Informationssysteme sei. Die räumliche Entkopplung der Datenbasis ermöglicht einen Typus von Informationssystemen, der sich durch zahlreiche lokale und autonome Systemkomponenten auszeichnet[9]. Die dafür nötigen Kooperationsinfrastrukturen erfordern ein hohes Maß an automatisierten und kooperativen Abläufen. In Zukunft, so Papazoglou, sollte es das Ziel sein, daß einzelne am Gesamtsystem teilhabende Informationssysteme selbst befähigt werden, als intelligente Informationsagenten zu fungieren, die komplexe und heterogene Aufgaben zu erfüllen haben. Die Basis hierfür bildet eine Erweiterung des passiven Dokumentenbegriffs.

     Auch Weikum [Wei00] sieht in der software-bezogenen Automatisierung von Abläufen – wie z. B. das Suchen von Informationen – eine der wesentlichen Chancen und Herausforderungen für datenbankzentrierte Informationssysteme.
   - Brodi [Bro99] stellt traditionelle Vorgehensweisen beim Entwurf von Informationssystemen in Frage, u.a. fordert er eine Gleichbehandlung von Daten, welche die „künstliche Trennung von Daten und Applikationslogik“ aufhebt. In dieser Hinsicht sollten Dokumente mehr als nur passive Datencontainer sein.

[8] Im betreffenden Artikel wird von Daten in diesem Zusammenhang gesprochen. Aufgrund der eingeführten Terminologie für Dokumente entspricht dies aber einem Dokument.

[9] Siehe auch die Diskussion über die Erfolgsfaktoren des WWW wie z. B. die lokale Autonomie der Datenbasen in Abschnitt 2.3.2.1.

- In [CW00] thematisieren Chaudhuri und Weikum mögliche zukünftige Architekturen von Datenbankmanagement-Systemen. Sie schlagen vor, wesentlich kleinere datenbankbezogene Systemkomponenten anstelle von großen und mächtigen Datenbankdiensten zu etablieren. Letztlich bedeutet dies einen Schritt in Richtung Einsatz von feingranularen Mediatoren, die jeweils strikt gekapselte Funktionalitäten zur Verfügung stellen, d. h. fein-granulare Datenbankdienste mit klar bestimmbaren und abgegrenzten Anwendungskontexten stehen im Mittelpunkt zukünftiger Datenbankmanagement-Systeme.

Die oben angeführten Ausführungen unterstützen die Betrachtung von Erweiterungen des passiven Dokumentenbegriffs zu aktiven Dokumenten, die im anschließenden Abschnitt vorgestellt werden.

3.2.3 Aktive Dokumente

Aktive Dokumente bestehen nicht nur aus passiven Datencontainern, sondern beinhalten auch aktive Komponenten. Eine aktive Komponente hat direkten Zugang zum Dokumenteninhalt oder zu den Metadaten eines Dokuments. Bei eingehenden Informationsanfragen an ein Dokument wird die aktive Komponente aktiviert und ist so in der Lage, den Dokumenteninhalt auf verschiedene Arten zu transformieren, bevor dieser dann letztlich der Anfrage in der aufbereiteten und aktualisierten Form präsentiert wird. In dieser Hinsicht entspricht eine solche aktive Komponente einem Mediator, welcher direkt einem Dokument zugeordnet ist. Prinzipiell kann dieser Mediator vielfältige dokumenten-spezifische Aufgaben übernehmen, die u.a. in Abschnitt 2.2 beschrieben sind.
Eng verbunden mit dem Konzept von aktiven Dokumenten sind sogenannte reaktive Dokumente. Reaktives Verhalten beschreibt dabei den Vorgang, daß in Abhängigkeit von externen Einflüssen bestimmte Aktionen erfolgen, d. h. ein reaktives Dokument reagiert ausschließlich auf das Eintreten gewisser Umwelteinflüsse und initiiert daraufhin Aktionen. In den folgenden Kapiteln werden aufgrund der inhaltlichen Verwandtschaft reaktive Verhaltensaspekte von Dokumenten unter aktiven Dokumenten subsumiert und nicht separat erörtert, es sei denn, daß explizit darauf hingewiesen wird.
Aus technischer Sicht ergeben sich zunächst mehrere Möglichkeiten, aktive Komponenten in ein Dokument einzubinden. In Abbildung 3.5 werden grundsätzlich drei verschiedene Typen der Einbindung unterschieden:

Typ A Eine aktive Komponente wird direkt in das Dokument eingebettet. Das aktive Dokument ist von der Ausführungsumgebung getrennt. Eine Ausführungsumgebung gewährleistest den eigentlichen Aktivierungsprozeß für ein aktives Dokument. In dieser Hinsicht ist ein aktives Dokument vom Typ A als passives Dokument zu betrachten, welches durch den Aktivierungsprozeß in eine Ausführungsumgebung überführt wird und erst danach aktiv

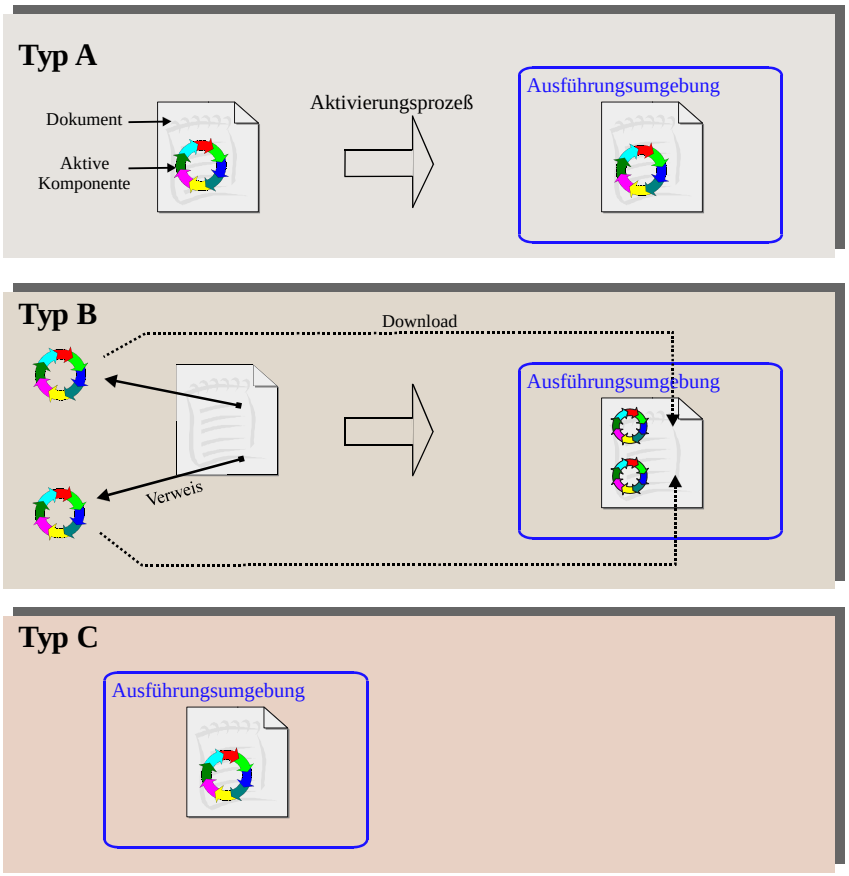


Abbildung 3.5: Verschiedene Möglichkeiten zur Realisierung von aktiven Dokumenten.

wird. Aktive Dokumente des Typs A sind abhängig von Ausführungsumgebungen, um ihr Aktivitätspotential nutzen zu können.

Auf dieser Grundlage können beispielsweise die in HTML-Dokumenten eingefügten JavaScript-Anweisungen [Fla01] direkten Einfluß auf den Darstellungsprozeß des HTML-Dokuments im Browser nehmen. Der Browser agiert hierbei als Ausführungsumgebung für das aktive Dokument, und die JavaScript-Anweisungen steuern die Visualisierung des HTML-Dokuments im Browser.

Typ B Eine aktive Komponente wird indirekt über Verweis- oder Linkstrukturen in das Dokument eingebettet. Dabei ist die aktive Komponente vom Dokument räumlich getrennt. Ansonsten entsprechen sich aktive Dokumente des Typs A und B, d. h. die Abhängigkeit zwischen Ausführungsumgebung und aktivem Dokument bleibt bestehen. Hinsichtlich des Aktivierungsprozesses

löst die Ausführungsumgebung die zusätzliche Indirektion auf, in dem es den Verweisen folgt, die entsprechenden aktiven Komponenten herunterlädt und lokal verfügbar macht. Anschließend ist das Dokument aktiv.

Typisch für aktive Dokumente des Typs B sind einerseits Java-Applets [AGH00], die in HTML-Dokumente über Linkstrukturen eingefügt, vom Browser heruntergeladen und schließlich im Browser zur Ausführung gebracht werden. Andererseits gibt es Ansätze [BCV99, BK01, Say01], die anstelle von HTML allgemeinere Markup-Sprachen und deren verwandte Technologien – wie XML (*Extensible Markup Langugage*) [BPSMM00], XLink (*XML Linking Language*) [DMO01] oder XPath (*XML Path Language*) [CD99] – verwenden. Das grundlegende Prinzip für aktive Dokumente des Typs B der Einbettung von aktiven Komponenten über zusätzliche Indirektionen bzw. über Verlinkungsmechanismen bleibt unverändert und ist für beide Ansätze kennzeichnend.

Typ C Dokumente des Typs C enthalten aktive Komponenten, die aus logischer Sicht direkt in das Dokument eingebettet sind. Im Gegensatz zu Dokumenten des Typs A und B entfällt ein separater Aktivierungsprozeß, da Dokumente des Typs C bereits eine eigene Ausführungsumgebung besitzen. Der fundamentale Unterschied zu den Typen A und B ist die Objektivierung der Dokumente des Typs C, d. h. Dokumente werden nicht mehr nur im klassischen Sinne als grob-granulare und häufig als textbezogene Datencontainer, sondern als aktive digitale Objekte der realen Welt betrachtet. Als aktive Objekte sind sie in der Lage zielgerichtete Tätigkeiten und Aufgaben zu verrichten[10]. Dabei sind sie unabhängig von den für Typ A und B typischen Aktivierungsprozessen, da sie selbst in der Lage sind, Informations- und Kontrollflüsse anzustoßen und zu übernehmen. In dieser Hinsicht bilden sie einen abgeschlossenen Dokumentenbegriff, der Aktivitäten mit in die Kapselungseigenschaften einer objektorientierten Sichtweise auf Dokumente aufnimmt und so die Unabhängigkeit und Selbständigkeit aktiver Dokumente des Typs C unterstützt.

Die Objektivierung des Dokumentenbegriffs impliziert einen allgemeineren Einsatz von Dokumenten, der nicht mehr nur primär auf die Darstellung von Dokumenteninhalten abzielt (Typ A), sondern Dokumente als feingranularere Organisationsformen für Informationssysteme betrachtet. Diese Argumentation wird grundsätzlich auch von Kaletta [Kal02] im Bereich digitaler Bibliotheken angeführt. Digitale Bibliotheken sind typische Repräsentanten einer Domäne, die dokumentenzentrierte Informationssysteme betrachten. Im Mittelpunkt zukünftiger digitaler Bibliotheken stünden die Verwaltung und Organisation digitaler Objekte, die hinsichtlich ihrer Quantität und Qualität neue Herausforderungen an traditionelle Systeme stellen wer-

[10] Aktive Dokumente des Typs C entsprechen am weitgehendsten der generellen Definition von *aktiv* gemäß dem Duden [EHWM01]: aktiv – zielgerichtete Tätigkeit aus eigenem Antrieb.

den. Heinreich und Maurer [HM00] fordern zudem eine Kommunikation und Interaktion innerhalb von Informationssystemen, die nicht nur Benutzer, sondern auch Dokumente explizit miteinschließt.

Chang und Znati [CZ01] beschreiben einen aktiven Dokumentenbegriff, der selbständig nach erwünschten Informationen sucht, sich dabei mit anderen aktiven Objekten zusammenschließt und gegebenfalls mit ihnen kooperiert.

Die dafür nötige Fähigkeit, Informations- und Kontrollflüsse zu initiieren ist dabei ein wesentliches Merkmal dieser aktiven Dokumente [11].

Der oben angesprochene Prozeß der Objektivierung von Dokumenten ist ein wichtiger Schritt zur Entwicklung eines verallgemeinerten, aktiven Dokumentenbegriffs. Deswegen werden im folgenden Teil zwei wesentliche Eigenschaften der Objektivierung von Dokumenten und deren Auswirkungen auf den Dokumentenbegriff erläutert:

1. *Granularisierung*. Dokumente bestehen nicht mehr nur aus einfachen Textinhalten, sondern stellen ein digitales Abbild unterschiedlichster Dinge der realen Welt dar, wie z. B. Grafiken-, Audio- und Videodateien. Edwards und LaMarca [EL99] gehen noch einen Schritt weiter und beschreiben einen universellen Dokumentenbegriff, der selbst aktive Entitäten der realen Welt miteinschließt. So sind physische Endgeräte (z. B. eine WebCam), Menschen (z. B. ein Benutzer) oder Softwareprozesse (z. B. Serverprozesse in einem Client/Server-Informationssystem) Ausprägungen eines universellen und aktiven Dokumentenbegriffs. Generell läßt sich festhalten, daß eine Objektivierung von Dokumenten sowohl grob als auch fein-granulare Entitäten der realen Welt berücksichtigt[12].

2. *Verhalten*. Auf der Grundlage objektorientierter Ansätze [Boo94, Mey97] bringt eine Objektivierung der Dokumente die Existenz sowohl daten- als auch verhaltensbezogener Eigenschaften von Dokumenten mit sich. So bestehen aktive Dokumente aus zwei Teilen: Einem Datenteil, der digitalisierte Informationen über reale Entitäten enthält, und einem Verhaltensteil, der Zugriffsmethoden auf das Dokument festlegt und dessen aktives Verhalten charakterisiert. Aktive Dokumente bilden so einen Dokumentenbegriff, der sich dem Konzept eines Objekts in objektorientierten Ansätzen annähert. Meyer [Mey98] betont die Allgegenwärtigkeit und längerfristige Gültigkeit objektorientierter Methodologien und Vorgehensweisen beim Entwurf von

[11] In Kapitel 9 werden sogenannte *Lebendige Dokumente* (*Living Documents*) ausführlich thematisiert und vorgestellt, die ihre konzeptionelle Basis in aktiven Dokumenten des Typs C haben. In [SKN02, SK02] stehen verteilte Informationssysteme im Mittelpunkt, deren Architektur auf *Living Documents* beruhen.

[12] Die Heterogenität der betrachteten Entitäten ist eng mit der Granularisierung verbunden. Dokumente als digitale Repräsentanten unterschiedlicher Typen von Entitäten dienen in dieser Hinsicht als geeignete Abstraktion, um Aspekte der Heterogenität zu bewältigen.

Softwaresystemen. Dies untermauert die Entwicklung eines verhaltensbezogenen, objektähnlichen Dokumentenbegriffs.

Generell bietet der Einsatz einer aktiven Komponente innerhalb eines Dokuments die Möglichkeit sowohl den Dokumenteninhalt als auch die Ausprägung der Metadaten zu bestimmten Zeitpunkten zu aktualisieren, anzupassen oder zu modifizieren. Aktive Komponenten sind dafür geeignet, unterschiedliche Sichten auf Dokumente dynamisch zu erzeugen. Eine Sicht entspricht einer speziellen Ausprägung oder Repräsentation eines Dokuments. Die dynamische Erzeugung von unterschiedlichen Sichten auf das gleiche Dokument durch die mediatorische Aktivierung in aktiven Dokumenten bietet mehrere Vorteile für die Realisierung von Informationssystemen im Vergleich zu statischen Vorgehensweisen:

- Generell sind die Anfragemöglichkeiten und Anfrageergebnisse an Dokumente bei der Verwendung von aktiven Komponenten größer, da Benutzer nicht notwendigerweise auf statisch vorgegebene Anfragekombinationen festgelegt sind.

- Aktive Dokumente sind in der Lage, aktualisierte Informationen bei der Beantwortung von Informationsanfragen zu berücksichtigen. Damit können Dokumente mit hohen Änderungsraten in sich schnell und häufig ändernden dynamischen Umgebungen adäquat modelliert und abgebildet werden.

- Die Präsentation der Ergebnisse einer Informationsanfrage an ein Dokument kann in Abhängigkeit von zahlreichen Kontexten erfolgen. Ein Kontext bestimmt den inhaltlichen Sach- und Sinnzusammenhang, der für die Beantwortung der Anfrage zur Verfügung stehenden Parameter. So kann beispielsweise die Darstellung[13] des Ergebnisses in Abhängigkeit bestimmter Benutzergruppen und -wünschen erfolgen, welches die Qualität der Beantwortung der Informationsanfrage wesentlich erhöht. Falls ein aktives Dokument zudem in mehreren Informationssystemen zur Anwendung kommt, kann das aktive Dokument in Abhängigkeit des jeweiligen Systems die erforderliche anwendungsspezifische Sicht auf das Dokument dynamisch generieren.

Neben den beschriebenen, auf dem Konzept eines Mediators basierenden Möglichkeiten für die dynamische Aufbereitung von Informationen ist das Automatisierungspotential von aktiven Komponenten in Dokumenten besonders hervorzuheben. Generell ist eine aktive Komponente in der Lage, Informations- und Kontrollflüsse anzustoßen und weiterzuleiten. Vor allem aktive Dokumente des Typs C, die weitestgehend unabhängig von Aktivierungsprozessen sind, können Aktionen

[13] Mit dem Begriff der Darstellung ist hier nicht notwendigerweise eine grafische Visualisierung gemeint, sondern generell die inhaltliche Gestaltung der Anfrageergebnisse. Dies ist von Bedeutung, wenn man unter Benutzer sowohl menschliche als auch digitale Benutzer versteht (siehe Abschnitte 2.1.1 und 2.1.2.3). So stehen bei digitalen Benutzern weniger grafische Gesichtspunkte als vielmehr die prinzipiell inhaltliche Gestaltung der Ergebnisse im Vordergund.

und damit verbundene Kontrollflüsse selbständig initiieren. Dies ist gleichbedeutend mit einem hohen Aktivierungsgrad von aktiven Dokumenten und unterstützt automatisierte Interaktionen innerhalb von Informationssystemen, die in Abschnitt 2.1.2.3 beschrieben sind. Die Informationsflüsse in Informationssystemen sind so nicht mehr strikt an Benutzeraktionen oder anderen – aus Sicht des Dokuments – externen Systemkomponenten gebunden, sondern können selbst von Dokumenten ausgeübt werden. Ein aktiver Dokumentenbegriff unterstützt in dieser Hinsicht den Einsatz von aktiven Datenbasen in Informationssystemen[14]

3.2.4 Proaktive Dokumente

Proaktive Dokumente sind spezielle aktive Dokumente des Typs C (gemäß Abbildung 3.5), wobei die aktive Komponente sich durch antizipierendes Verhalten auszeichnet[15]. Generell impliziert antizipierendes Verhalten die Ausrichtung einer zielgerichteten Handlung auf Gegebenheiten in der Zukunft. Proaktive Dokumente sind demnach in der Lage, selbständig und autonom vorgegebene Ziele zu erreichen. Die Selbständigkeit und Autonomie bezieht sich dabei auf die Wahlmöglichkeiten der Aktionen, die ein proaktives Dokument besitzt. In dieser Hinsicht ist die Reihenfolge der zur Zielerlangung notwendigen Aktionen nicht statisch vorgegeben, sondern liegt sozusagen im Ermessen des proaktiven Dokuments.
Im Gegensatz zu proaktiven Dokumenten besitzen aktive Dokumente eingeschränkte Handlungsmöglichkeiten. Aktive Dokumente sind in der Lage, Informations- und Kontrollflüsse für die Erreichung eines bestimmten Zieles zu initiieren. Dies geschieht aber ausschließlich unter Berücksichtigung eines statisch vorgegebenen Ablaufplans. So werden beispielsweise die in den aktiven Komponenten enthaltenen und spezifizierten Aktionen auf die vorherbestimmte Art und Weise zur Ausführung gebracht. Dies ist ein ausreichendes und adäquates Vorgehen, falls sich die Grundlagen und Annahmen für diesen starren Ablaufplan nicht ändern sollten. Dies ist z. B. der Fall für die Ausführung der `onClick()` JavaScript-Anweisung in HTML-Dokumenten (Typ A), die beim Klicken auf einen Hypertext-Link aktiviert wird. Ein weiteres Beispiel für einen einfachen und deswegen statisch bestimmbaren Ablaufplan ist die Benutzung von Java-Applets (Typ B) für die Validierung von HTML-Formularen bzw. den von den Benutzern getätigten Eingaben.
Falls sich vielfältige und komplexe Änderungen innerhalb eines Informationssystems ergeben sollten, beruht der statisch bestimmte Ablaufplan auf Annahmen, welche nicht notwendigerweise zur optimalen Zielerreichung führen, d. h. eine Anpassung des Ablaufplans wäre erforderlich, um den neuen und aktuellen Gegebenheiten gerecht zu werden.

[14] Eine ausführliche Beschreibung von automatisierten Informationssystemen findet sich für die Domäne Web-Informationssysteme in [KR97] und für die Domäne digitaler Bibliotheken in [MG01]. Wiederhold et al. diskutieren in [WG97] die grundsätzliche Bedeutung von Mediatoren im Zusammenhang mit der Automatisierung von Abläufen in Informationssystemen.

[15] Antizipation: Gedankliches Vorwegnehmen eines Geschehens (Duden)

Generell gilt es festzuhalten, daß proaktive Dokumente vor allem für die Erreichung komplexer Ziele sinnvoll einzusetzen sind. Handelt es sich um einfache Aufgaben einer aktiven Komponente – wie oben anhand von zwei Beispielen kurz illustriert – ist ein statisch bestimmter Ablaufplan ausreichend. Komplexe Ziele in Informationssystemen zeichnen sich u.a. durch folgende Eigenschaften aus:

- *Multiplizität*. Für die Bewältigung komplexer Aufgaben gibt es mehrere Möglichkeiten der Zielerreichung. A priori ist nicht bestimmbar, welcher der alternativen Lösungspfade der beste ist.
- *Abhängigkeiten*. Komplexe Ziele eines proaktiven Dokuments bestehen aus vielfältigen Abhängigkeiten zu anderen Systemkomponenten. Diese Abhängigkeiten sind nicht nur auf der logischen und konzeptionellen, sondern auch auf der temporalen und spatialen Ebene angesiedelt, d. h. auch aktuelle Ausprägungen von Zuständen der abhängigen Systemkomponenten nehmen Einfluß auf die Art und Weise der Zielerreichung.

Die wesentlichen Eigenschaften proaktiver Dokumente lassen sich wie folgt zusammenfassen:

1. Grundsätzlich baut der proaktive auf dem aktiven Dokumentenbegriff des Typs C auf. Proaktives Verhalten setzt in dieser Hinsicht eine Unabhängigkeit des Dokuments vom Aktivierungsprozeß voraus. Durch die konzeptionelle Nähe von proaktiven zu aktiven Dokumenten des Typs C gelten die gleichen Eigenschaften wie beispielsweise bei der Objektivierung von Dokumenten.

2. Die Komplexität der Aufgaben und Ziele proaktiver Dokumente erfordert häufig deklarative Mechanismen für die Formulierung der für die Zielerreichung notwendigen Aktionen und Ablaufpläne. Deklarative Ansätze fokussieren dabei auf der Beschreibung der grundsätzlichen Problematik, d. h. Beziehungen, Abhängigkeiten und Gegebenheiten werden in Form von deklarativen Aussagen beschrieben. Dabei wird keine Reihenfolge von Aktionen und Abläufen festgelegt, wie es beispielsweise bei prozeduralen Ansätzen der Fall ist[16].

3. Die Umgebung spielt eine wichtige Rolle, da umgebungs-relevante Parameter und Eigenschaften Einfluß auf die Aktivitäten eines proaktiven Dokuments nehmen. Umgebung ist dabei ein sehr weit gefaßter Begriff, der u. a. externe Systemkomponenten und weitere proaktive Dokumente einschließt. In Abhängigkeit von Umweltzuständen werden Entscheidungen getroffen und Aktionen selbständig initiiert[17]. Dies impliziert die Existenz geeigneter

[16] In Abschnitt 5.2.2 werden deklarative Programmieransätze näher beschrieben.

[17] In dieser Hinsicht sind proaktive Dokumente auch immer reaktiv, d. h. sie sind in der Lage, auf äußere Einflüsse in ihrer Umgebung zu reagieren.

Interaktionsmechanismen zwischen proaktiven Dokumenten und ihrer Umgebung, welche die Etablierung von bidirektionalen Informationsflüssen ermöglicht. Die Heterogenität der Umgebung ist dabei häufig ein limitierender Faktor[18].

4. Die in (3) angesprochene Relevanz adäquater Mechanismen zur Interaktion mit den umgebenden Systemkomponenten bezieht sich nicht nur auf die dafür notwendigen Zugriffs- und Kommunikationsmöglichkeiten. Neben diesen syntaktischen Aspekten – wie z. B. die genaue Spezifikation von Kommunikationsschnittstellen – spielen im Zusammenhang mit proaktiven Dokumenten auch Techniken zur automatisierten Bestimmung der inhaltlichen Bedeutungen eine große Rolle. So sollte ein proaktives Dokument in die Lage versetzt werden können, die Semantik der ausgetauschten Informationen auf eine automatisierte Art und Weise erfassen zu können. Manuelle Eingriffe in den Interpretationsprozeß sollten auf ein Minimum reduziert werden, da ansonsten ein proaktives Dokument in seinen Aktionen eingeschränkt werden würde. Ähnliche Fragestellungen ergeben sich im Zusammenhang mit dem sogenannten *Semantic Web* [BLHL01, BL99, Hen01, SEM01]. Innerhalb des *Semantic Webs* entstehen bedeutungsvolle Strukturen für den Inhalt von HTML-Dokumenten, die von Softwareprozessen verarbeitet werden können. Die externalisierte Darstellung von Bedeutungen ist dabei zentral für das *Semantic Web*. So werden nicht nur Dokumenteninhalte, sondern auch deren Bedeutung zwischen Softwareprozessen ausgetauscht. Dies bildet die Basis für eine automatisierte Form der Interaktion zwischen Softwareprozessen im allgemeinen und proaktiven Dokumenten im speziellen[19].

5. Die Selbständigkeit proaktiver Dokumente hinsichtlich ihres Verhaltens unterstützt eine erweiterte Form der Autonomie. Im Gegensatz zu aktiven besitzen proaktive Dokumente zusätzliche und größere Freiheitsgrade in der Wahl der auszuführenden Aktionen. Ihr konkretes Verhalten wird sozusagen von ihnen selbst bestimmt. Auf proaktiven Dokumenten basierende Informationssysteme bestehen aus einer Vielzahl von autonomen und aktiven Datenbasen, die sich durch ihre lokalen, unabhängigen Informations- und Kontrollflüsse auszeichnen. Bei einer globalen Betrachtung der Typen von Informationssystemen ergeben sich nicht-deterministische Informationsflüsse, die ihren Ursprung in der Interaktion und Kooperation der einzelnen proaktiven Datenbasen haben. Anstelle simpler und auf Benutzer fokussierte Interaktionen in traditionellen Client/Server-Informationssystemen treten nun vielfältige, komplexe und in hohem Maße nebenläufige Informations- und Kontrollflüsse.

[18] In Abschnitt 7 wird das Monitoringsystem *Specto* beschrieben, welches auf der Basis von XML-Dokumenten einen einfachen und interoperablen Austausch von Zuständen zwischen heterogenen Systemkomponenten ermöglicht.

[19] Einführende Diskussion über automatisierte Arbeitsformen (siehe Abschnitt 2.1.2.3).

Dokumentenbegriff

passiv	aktiv	proaktiv
• klassische Sichtweise • daten-zentriert	• Objektivierung: Eigenständigkeit, aktives und reaktives Verhalten, Kapselungseigenschaften • Daten + Verhalten • Initiator von Informationsflüssen • Mikromediator	• Autonomie • nicht-deterministische Informationsflüsse • deklarative Ansätze • Umgebung als relevanter Faktor

Tabelle 3.2: Gegenüberstellung der prinzipiellen Eigenschaften passiver, aktiver und proaktiver Dokumente

In Tabelle 3.2 werden die Eigenschaften unterschiedlicher Dokumentypen dargestellt.

3.3 Technologie

Der Begriff der Technologie umschreibt die Aspekte und Voraussetzungen für die technische Realisierung von Informationssystemen. Dabei stehen sowohl inhaltliche Aspekte der eingesetzten technischen Systemkomponenten als auch der Prozeß der eigentlichen Softwareentwicklung im Mittelpunkt. Da Wartungs- und Wiederbenutzbarkeitsaspekte im Softwareentwicklungsprozeß bereits im vorherigen Abschnitt 2.1.2.2 besprochen wurden, beschränkt sich die Diskussion technologischer Aspekte auf die Betrachtung von unterschiedlichen Systemkomponenten in Informationssystemen.
Die vorgestellte Taxonomie klassifiziert technologische Fragestellungen beim Entwurf und bei der Realisierung von Informationssystemen hinsichtlich dreier Typen von Systemkomponenten: 2-stufige, mehrstufige Client/Server-Architekturen und Software-Agenten.
Da Client/Server-Architekturen bezüglich ihrer Terminologie und Eigenschaften bereits ausführlich in Abschnitt 2.3.1 vorgestellt wurden, beschränkt sich die weitere Betrachtung in den folgenden Abschnitten auf technologische Aspekte sogenannter Software-Agenten[20]. Im Vordergrund stehen allgemeine Eigenschaf-

[20]In den folgenden Kapiteln wird der Begriff des Agenten synonym für Software-Agenten ver-

ten von Agenten und deren konzeptionelle Abgrenzung zu objektorientierten und mediator-basierten Technologien.

3.3.1 Agenten

Das Konzept eines Agenten läßt sich zurückführen auf Forschungsarbeiten im Bereich der Verteilten Künstlichen Intelligenz in den 70-er Jahren. So bildete beispielsweise das von Hewitt [Hew77] bereits 1977 vorgestellte Actor Modell einen ersten Vorläufer für heutige Agentensysteme. Im Mittelpunkt des Actor Modells von Hewitt steht ein sogenannter Actor, welcher ein sich abgeschlossenes, interaktives und nebenläufig ausgeführtes Objekt darstellt. Dieses Objekt hat einen gekapselten, internen Zustand und kann auf Nachrichten ähnlicher Objekte reagieren. In den 90-er Jahren wurde das Agentenkonzept wieder aufgegriffen und weiterentwickelt. Die Arbeiten von Genesereth und Ketchpel [GK94], Wooldridge und Jennings [WJ95], Nwana [Nwa95] und Bradshaw [Bra97] versuchen, Agenten und agentenbasierte Softwareentwicklung als eigenständige Disziplin in der Informatik zu etablieren. Die dafür notwendige Abgrenzung zu bestehenden Paradigmen – wie beispielsweise der Objektorientierung – wird vor allem von Jennings in [Jen00] veranschaulicht und diskutiert. In Abschnitt 3.3.2 wird detailliert auf die Unterschiede zwischen Objekten und Agenten eingegangen.
In der vorliegenden Arbeit orientiert sich die Definition eines Agenten an Jennings [Jen00], der sich wiederum auf die Ausführungen von Woolridge [Woo97] stützt:

> An agent is an encapsulated computer system that is situated in some environment and that is capable of flexible, autonomous action in that environment in order to meet its design objectives.

Gemäß oben genannter Definition ergeben sich folgende Aussagen:

1. Agenten sind klare bestimmbare Softwaresysteme mit wohldefinierten Grenzen und Schnittstellen.

2. Agenten sind eingebettet in eine spezielle Umgebung, deren Zustand über Nachrichten und Ereignisse an Agenten weitergeleitet werden. Auch Joseph und Kawamura [JK01] heben die Bedeutung der Umgebung von Agenten hervor: Um auf Nachrichten und Ereignisse in der Umgebung reagieren zu können, benötigt man Mechanismen und dementsprechende Infrastrukturen, die es Agenten ermöglichen, mit unterschiedlichen Komponenten der Umwelt zu interagieren.

3. Agenten werden entworfen, um spezielle Ziele zu erreichen.

4. Agenten sind gekennzeichnet durch ihre Autonomie, da sie sowohl die Kontrolle über ihren internen Zustand als auch über ihr Verhalten innehaben.

---

wendet.

5. Agenten sind fähig, komplexe Aufgaben zu erfüllen, da sie sowohl reaktive als auch proaktive Eigenschaften besitzen. Jennings [Jen01] rechtfertigt den Einsatz der Agententechnologie für die Realisierung von komplexen Informationssystemen, indem er zeigt, wie gängige Methoden für die Komplexitätsbewältigung [Boo94] in der Informatik von der Agententechnologie unterstützt werden:

   - *Dekomposition*. Bei der Dekomposition werden komplexe Probleme bzw. Systeme in kleinere und besser verwaltbare Teilstücke aufgeteilt. Diese Teilstücke können dann relativ isoliert betrachtet und eventuell sogar gelöst werden. Kriterien für die geeignete Zerlegung von komplexen Komponenten orientieren sich dabei grundsätzlich an den von Parnas [Par72] vorgestellten Modulkriterien. Jennings [Jen01] argumentiert, daß Agenten es ermöglichen, komplexe Systeme auf eine effektive Art und Weise zu zerlegen.
   - *Abstraktion*. Unter dem Begriff der Abstraktion zur Komplexitätsbewältigung versteht man eine vereinfachte Sicht auf ein komplexes System. Es wird ein Modell bestimmt, in dem essentielle Eigenschaften hervorgehoben, während andere Details vernachlässigt werden. Jennings [Jen01] argumentiert, daß Agenten passende Abstraktionen anbieten, um die Modellierung komplexer Systeme zu vereinfachen.
   - *Organisation*. Eine Organisation bezeichnet den Prozeß der Spezifikation und Verwaltung von Beziehungen zwischen verschiedenen Systemkomponenten oder -modulen. Jennings [Jen01] wählt bewußt den allgemeineren Begriff der Organisation anstelle des von Booch [Boo94] angeführten Terminus der Hierarchie. Für Jennings impliziert eine hierarchische Anordnung einen vorgegebenen Kontrollfluß (z. B. vom Wurzelelement zu den einzelnen Blättern), der hinsichtlich dynamischer, komplexer Informationssysteme zu restriktiv und statisch bestimmt sei.

Zusammenfassend läßt sich festhalten, daß die Agententechnologie und die damit verbundene agentenbasierte Softwarenentwicklung geeignete Mechanismen und Vorgehensweisen zur Verfügung stellt, um komplexe Systeme und insbesondere Informationssysteme zu realisieren. An dieser Stelle sei darauf hingewiesen, daß sehr wohl auch andere Technologien, wie z. B. das objektorientierte Paradigma, sich einerseits für den Entwurf und die Realisierung komplexer Informationssysteme eignen und andererseits in der Lage sind, spezifische konzeptionelle Eigenheiten agentenbasierter Vorgehensweisen nachzubilden. In dieser Hinsicht erweitern Agenten die Palette konzeptioneller Möglichkeiten und Vorgehensweisen, um komplexe Informationssysteme zu entwerfen und zu realisieren.
In Kapitel 5 wird im Zusammenhang mit dem Entwurf eines Agentensystems ausführlich auf die speziellen Eigenschaften von Agenten eingegangen. In den folgenden Abschnitten werden Agenten und deren generelle Eigenschaften mit Kon-

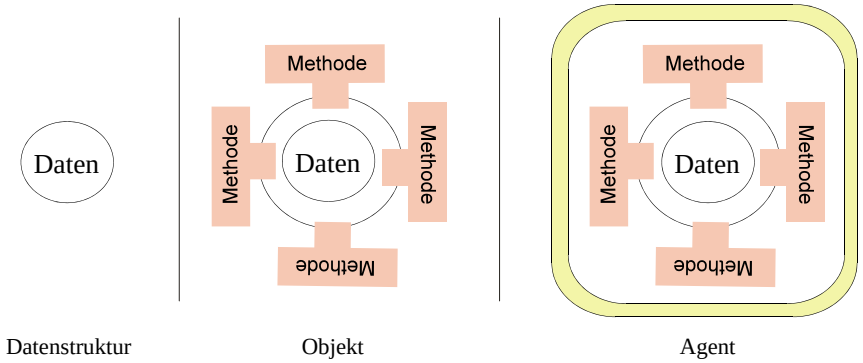


Abbildung 3.6: Datenstrukturen, Objekte und Agenten gemäß Joseph und Kawamura [JK01].

zepten objektorientierter Methodologien verglichen. Es wird konkret untersucht, inwiefern Agenten als eine Erweiterung des klassischen Objektbegriffs verstanden werden können. Anschließend werden Agenten im Kontext von Wiederholds [Wie92a] Mediatoren betrachtet. Im Mittelpunkt steht eine Fortführung und Weiterentwicklung des Mediator-Konzepts, welche Agenten als miniaturisierte Mediatoren ansieht.

3.3.2 Agenten und Objekte

In diesem Abschnitt wird herausgearbeitet, inwiefern das Konzept eines Agenten als Erweiterung eines Objekts im objektorientierten Paradigma betrachtet werden kann. Andererseits werden prinzipielle Unterschiede zwischen Agenten und Objekten ausführlich dargelegt.

Objekte Im objektorientierten Paradigma spielt das Objekt die zentrale Rolle [GR83, Str88, Boo94, Mey97]. Ein Objekt zeichnet sich durch zwei prinzipielle Eigenschaften aus: *Zustand* und *Verhalten*. Der Zustand eines Objekts wird – aus Sicht des Objekts – in internen Datenstrukturen abgelegt. Der Zugriff auf die Datenstrukturen und auf den Objektzustand erfolgt über Nachrichten, die an das Objekt gesendet werden. In der Nachricht ist festgelegt, welche Methoden oder Datenstrukturen des Objekts aufgerufen werden. Die reglementierte Art des Zugriffsmechanismuses auf Objekte, wie es in Abbildung 3.6 illustriert ist, ermöglicht eine adäquate Kapselung von Daten bzw. Datenstrukturen in Objekten.

Agenten In objektorientierter Hinsicht wird ein Agent als ein Objekt mit einer zusätzlichen Kapselungsschicht verstanden. Diese zusätzliche Schicht kontrolliert alle Zugriffe auf die Methoden und Datenstrukturen des Agenten, wie es in Abbildung 3.6 verdeutlicht ist. So werden sämtliche Nachrichten an einen Agenten durch diese Schicht geleitet. Dort wird entschieden,

welche Methoden eines Agenten tatsächlich ausgeführt bzw. aktiviert werden. Diesen Entscheidungsprozeß in einem Agenten bezeichnet man auch als Verhaltensaktivierung.

Genesereth und Ketchpel [GK94] betonen die Ähnlichkeit der Eigenschaften von Objekten und Agenten: „Beide stellen eine nachrichtenbasierte Schnittstelle zu ihren internen Datenstrukturen und Algorithmen (Methoden) zur Verfügung.“ Dabei ist zu beachten, daß der Begriff der Nachricht unterschiedliche Bedeutungen bei Objekten und Agenten besitzt:

- Eine Nachricht im objektorientierten Sinne entspricht einem Aufruf von Methoden oder Datenstrukturen, die zu einem Objekt gehören[21]. Dieser Aufruf setzt voraus, daß die syntaktischen Details der Methodensignatur und Datenstruktur bekannt sind. Falls sich beispielsweise die syntaktische Reihenfolge der Übergabeparameter einer Methode ändern sollte, sind alle Nachrichten bezüglich dieser Methode entsprechend anzupassen. Genesereth und Ketchpel heben zudem hervor, daß die Bedeutung einer Nachricht im objektorientierten Kontext aufgrund des Prinzips des Polymorphismus[22] von Objekt zu Objekt variieren kann.

- Nachrichten werden zwischen Agenten in einem einheitlichen Format verschickt, d. h. jeder Agent benutzt unabhängig von der inhaltlichen Bedeutung die gleiche syntaktische Schnittstelle für den Austausch von Nachrichten. Nachdem der Inhalt der Nachricht von der zusätzlichen Zugriffsschicht eines Agenten analysiert worden ist, wird das entsprechende Verhalten im Agenten aktiviert, in dem die dafür notwendigen Methoden aufgerufen werden (siehe Abbildung 3.6). Nach dem Prinzip der Verhaltensaktivierung bei Agenten übernimmt der „aufgerufene“ Agent die Auswahl der entsprechenden Aktionen. Im Vergleich zu „objektorientierten Nachrichten“ werden in den Nachrichten von Agenten keine aufzurufenden Methodensignaturen, sondern bedeutungsvolle und komplexe Nachrichteninhalte hinterlegt[23].

Auf der Grundlage der oben beschriebenen Eigenschaften und Charakteristika von Objekten und Agenten werden in der folgenden Übersicht drei wesentliche Unterschiede zwischen diesen beiden erläutert:

[21] Im objektorientierten Paradigma können nur speziell gekennzeichnete Datenstrukturen direkt aufgerufen werden. Diese Differenzierung ist für die grundsätzliche Gegenüberstellung der Eigenschaften von Objekten und Agenten aber nicht entscheidend, so daß sie an dieser Stelle nicht weiter thematisiert wird.

[22] Generell beschreibt das Prinzip des Polymorphismus im objektorientierten Kontext die Möglichkeit, von Daten und Methoden unterschiedliche Formen einzunehmen. Die Festlegung der tatsächlich zu benutzenden Form wird erst zur Laufzeit bestimmt. Für eine ausführliche Erläuterung des Prinzips des Polymorphismus sei auf Booch [Boo94], Meyer [Mey97], Cardelli und Wegner [CW85] verwiesen.

[23] Für Details des Nachrichtenaufbaus und -austausches zwischen Agenten gibt es ausführliche Erläuterungen in Abschnitt 5.2.1. Dort wird im Zusammenhang mit dem Entwurf eines Agentensystems ausführlich auf die nachrichtenbasierte Kommunikation zwischen Agenten eingegangen.

- Im allgemeinen sind Objekte passive Repräsentanten eines Abbilds der realen Welt. Damit ein Objekt selbst aktiv wird, muß zuerst eine Nachricht an das entsprechende Objekt gesendet werden. Im Gegensatz dazu sind Agenten durch ihre aktiven und proaktiven Eigenschaften wesentlich unabhängiger von den Kontroll- und Informationsflüssen ihrer Umgebung.

- Obwohl Objekte ihre Daten und Verhaltenseigenschaften in geeigneter Weise kapseln, bleiben die Aspekte der Verhaltensaktivierung davon unberücksichtigt. Analog zur Argumentation in Punkt (3.3.2) werden Methoden eines Objekts von außen aufgerufen, d. h. jede als öffentlich deklarierte Methode kann von anderen Objekten direkt aktiviert werden. Objekte besitzen im Gegensatz zu Agenten keine weiteren Wahlmöglichkeiten, die Zugriffe auf ihre Methoden bzw. ihr Verhalten zu beeinflussen[24].

- Der dritte Unterschied bezieht sich auf die Form der Kommunikation zwischen Objekten und Agenten untereinander. Im Mittelpunkt der Kommunikation zwischen Objekten stehen die syntaktischen Schnittstellen der Methodensignaturen und die internen Datenstrukturen. Der Aufruf einer bestimmten Objektmethode oder Datenstruktur impliziert gleichzeitig deren inhaltliche Bedeutung, d. h. zum Zeitpunkt des Methodenaufrufs müssen nicht nur die syntaktischen, sondern auch die semantischen Bedeutungen des Aufrufs bereits bekannt sein.

  Im Gegensatz dazu trennen Agenten syntaktische und semantische Details der Kommunikation. Agenten benutzen ein einheitliches und von der Bedeutung konkreter Nachrichteninhalte unabhängiges Nachrichtenformat. Dies gewährleistet die langfristige Gültigkeit von Kommunikationsschnittstellen im Hinblick auf die möglichen Änderungen der Methodensignaturen im Laufe eines Agentenlebens. Die inhaltliche Bedeutung einer Nachricht wird in einem separaten Schritt verarbeitet.

Darüber hinaus würden nach Booch [Boo94] „Objekte, Klassen und Module zwar essentielle, aber nicht ausreichende Abstraktionen bieten, um komplexe Softwaresysteme zu entwerfen“. Bestehende und anerkannte Konzepte im Bereich der Softwareentwicklung basieren primär auf den Prinzipien der Lokalisierung und der Kapselung. In dieser Hinsicht führen Agenten bzw. eine auf Agenten basierte Softwareentwicklung [Jen00] das Prinzip der Lokalisierung einen Schritt weiter: Agenten lokalisieren Absichten in einem Agenten selbst, d. h. jeder Agent erhält seinen eigenen Kontrollfluß und die Fähigkeit, Entscheidungen – wie beispielswei-

[24] Im Mittelpunkt der Betrachtung stehen die konzeptionellen Eigenschaften des objektorientierten Paradigmas. Sicherlich kann man objektorientierte Programme entwickeln, welche die Aspekte der Verhaltensaktivierung berücksichtigen bzw. nachimplementieren. Dadurch würde sich die Diskussion über den Vergleich zwischen Agenten und Objekten auf benutzerdefinierte, objektorientierte Programme reduzieren.

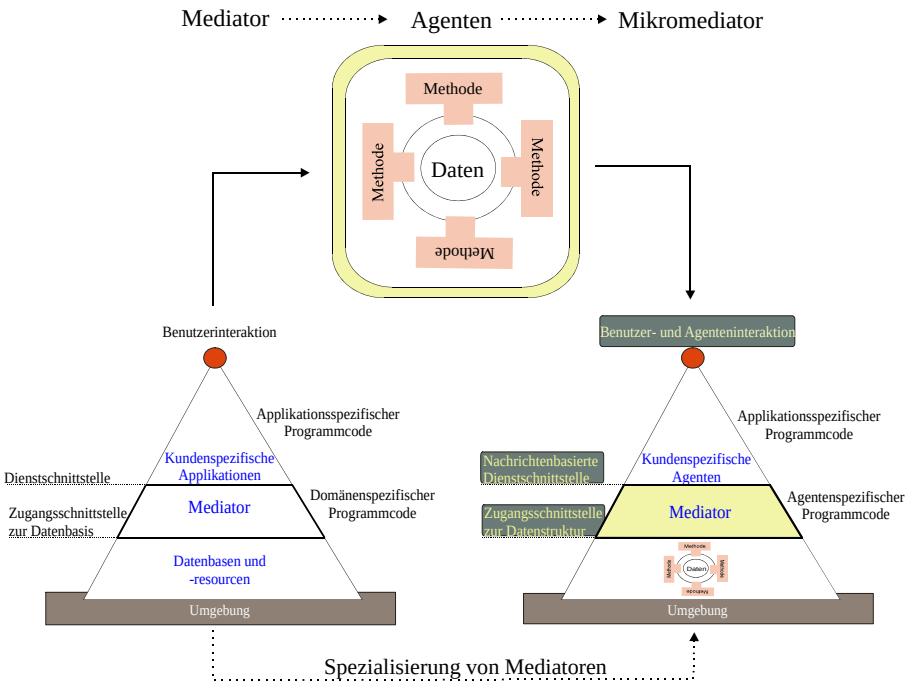


Abbildung 3.7: Agenten und Mediatoren.

se der Verhaltensaktivierung – selbständig zu treffen[25].

3.3.3 Agenten und Mediatoren

In diesem Abschnitt wird untersucht, inwieweit Agenten als Mediatoren [Wie92a] betrachtet werden können. Die Diskussion in diesem Abschnitt stützt sich auf die generelle Beschreibung von Mediatoren in Abschnitt 2.2 auf Seite 22ff.
In Abbildung 3.7 ist ausgehend von der allgemeinen Darstellung verschiedener Mediationsschichten in einem Informationssystem illustriert, wie ein Agent die Funktionen eines (speziellen) Mediators – *Mikromediator*s – übernimmt[26]:

- Anstelle der allgemeinen Form einer Datenbasis verwaltet ein Mikromediator die Zugriffe auf die Methoden und Datenstrukturen eines Agenten. Der Begriff des Mikromediators leitet sich von der Größe der Datenstrukturen eines Agenten ab, welche in der Regel um ein Vielfaches kleiner sind als normale Datenbasen.

[25] Das Prinzip der Lokalität war treibende Kraft beim Entwurf der *Living Documents* , die in Kapitel 9 vorgestellt und diskutiert werden.

[26] Die allgemeine Darstellung der Mediationsschichten in Informationssystemen orientiert sich an der Abbildung 2.3 auf Seite 22.

- Agenten als Mikromediatoren besitzen analog zu normalen Mediatoren eine zusätzliche Kapselungsschicht (siehe Abbildung 3.6), um alle Zugriffe auf die Methoden und Datenstrukturen eines Agenten zu verwalten.

- Die *Dienstschnittstelle* eines Mikromediators ist gekennzeichnet durch eine ausschließlich nachrichtenbasierte Kommunikation. Die Benutzung eines einheitlichen Nachrichtenformats sichert die langfristige Gültigkeit der Kommunikationsschnittstellen von Mikromediatoren. Durch die Aktivitätseigenschaften von Agenten (siehe Abschnitt 3.3.1) sind in Informationssystemen, die auf der Grundlage von Agenten realisiert sind, komplexe Interaktionsverfahren möglich.

- Die *Zugangsschnittstelle* zu Methoden und zur Datenstruktur ist abhängig von der jeweiligen Funktion des Mikromediators. Es obliegt dem Mikromediator gemäß dem Prinzip der Verhaltensaktivierung selbst zu bestimmen, welche Methoden oder Datenstrukturen bei eingehenden Nachrichten aktiviert werden.

Generell haben Mikromediatoren und traditionelle Mediatoren ähnliche Aufgaben und Funktionen. Die in den Abschnitten 2.2.1 und 2.2.2 beschriebenen Eigenschaften und Aufgaben von Mediatoren – wie beispielsweise die *Skalierbarkeit, dynamische Komposition*, die *Unterstützung für autonome Datenbasen* oder die *Flexibilität* beim Entwurf von Informationssystemen – treffen prinzipiell auch für Mikromediatoren zu. Der wesentliche Unterschied liegt in der Art und der Größe der Datenbasis, die einem Mikromediator zugeordnet ist: Mikromediatoren sind fein-granularen Datenbasen – den Methoden und Datenstrukturen eines Agenten – zugeordnet.
Abschließend sind in Tabelle 3.3 überblicksartig die wesentlichen Eigenschaften und Aspekte von Technologien für die Realisierung von Informationssystemen gegenübergestellt, die in Abschnitt 3.3 beschrieben sind. Dies sind 2- bzw. n-stufige Client/Server-Informationssysteme und Informationssysteme, die auf Agenten basieren[27].

3.4 Zusammenfassung

Die in diesem Kapitel vorgestellte Taxonomie ermöglicht die Klassifizierung von Informationssystemen hinsichtlich ihrer Eigenschaften und ihrer Systemarchitektur. Die Klassifizierung von Informationssystemen orientiert sich dabei an drei Dimensionen:

[27] Es sei darauf hingewiesen, daß die Eigenschaften und Aspekte von Client/Server-Informationssystemen bereits in Abschnitt 2.3 beschrieben und diskutiert werden. Deswegen beschränkt sich die Betrachtung technologischer Aspekte zur Realisierung von Informationssystemen in Abschnitt 3.3 auf Agenten.

Technologien		
2-cs	n-cs	Agenten
• Entwurf von verteilten Systemen im Vordergrund • Fat-Client • Simples Interaktionsprotokoll • Grob-granulare Struktur	• Einführung eines Thin-Clients und Applikationsservers • Skalierbarkeit • Wiederbenutzbarkeit • Fein-granulare Mediatorisierung	• Granularität • Aktivität • Kapselung • Mikromediator

Tabelle 3.3: Gegenüberstellung der Eigenschaften von Informationstechnologien auf der Grundlage von Client/Server und Agenten.

Benutzersicht Die Benutzersicht auf ein Informationssystem stellt den Benutzer und seine Arbeitsformen in den Mittelpunkt der Betrachtung bei der Realisierung von Informationssystemen. In Abschnitt 3.1 wurden zwei verschiedene Benutzersichten erörtert:

- Eine *interaktive Benutzersicht* als klassische Interaktionsform von Benutzern mit Informationssystemen fokussiert auf den Menschen und dessen Möglichkeiten zur Interaktion mit dem Informationssystem. Dementsprechend stehen die Aspekte für den Entwurf geeigneter *Benutzerschnittstellen* im Vordergrund, da diese das primäre Interaktionsmedium für den Menschen darstellen. In Abschnitt 3.1.1 wurden mehrere Eigenschaften einer interaktiven Benutzersicht herausgearbeitet wie z. B. die *Unmittelbarkeit* der Visualisierung und die Ausführung von Benutzeraktionen.
- Eine *automatisierte Benutzersicht* ist gekennzeichnet durch bestimmte Automatismen in den Arbeitsformen von Benutzern. Der Mensch fungiert primär als *Administrator* zahlreicher fortwährend ablaufender Softwareprozesse. In Abschnitt 3.1.2 wurde erläutert, wie die Bedeutung sogenannter *digitaler Benutzer* zunimmt. Eine von Automatismen gekennzeichnete Benutzersicht ist zudem geprägt von einer *mittelbaren* Visualisierung und Ausführung der vom Benutzer initiierten Aktionen. Ein weiterer wesentlicher Aspekt ist die *externalisierte Form* der Darstellung von Bedeutungsinhalten. Diese externalisierte Form

ermöglicht eine Automatisierung von Interaktionen zwischen Softwareprozessen, welche durch den Benutzer als Administrator verwaltet werden.

Dokumentenbegriff Der Dokumentenbegriff ist von zentraler Bedeutung für die Klassifizierung von Informationssystemen. Er gibt Auskunft über die Art und Weise der Modellierung von Informationen in Informationssystemen.

In Abschnitt 3.2.1 wurden anhand von zwei unterschiedlichen Arten von Dokumentenorganisationen – Dokumentenaorganisation in *Dateisystemen* und in *Datenbanken* – die Probleme von mangelhaften und restriktiven Dokumentenbegriffen dargelegt.

Anschließend wurden drei verschiedene Dokumentenbegriffe im Detail erörtert. Die dabei gewählte Einteilung beruhte auf dem Aktivitätsgrad eines Dokuments, der eine wesentliche Voraussetzung für die Automatisierung von Abläufen in Informationssystemen darstellt:

- *Passive Dokumente* sind Container, die Daten und Informationen beinhalteten. Sie besitzen keine eigenen Aktivitätspotentiale. In Abschnitt 3.2.2 wurde erläutert, inwiefern Informationssysteme, die auf einem passiven Dokumentenbegriff basieren, primär datenzentrierte Anwendungen unterstützen.
- *Aktive Dokumente* stellen eine Erweiterung des passiven Dokumentenbegriffs dar. Aktive Dokumente bestehen nicht nur aus passiven Datencontainern, sondern beinhalten auch aktive Komponenten. Eine aktive Komponente hat direkten Zugang zum Dokumenteninhalt oder zu den Metadaten eines Dokuments.

  In Abschnitt 3.2.3 wurden drei unterschiedliche Typen von aktiven Dokumenten untersucht. Kennzeichnend für aktive Dokumente ist die *Objektivierung* des Dokumentenbegriffs, d. h. Dokumenten werden nicht mehr nur Daten, sondern auch verhaltensspezifische Eigenschaften zugeordnet. Die Objektivierung von Dokumenten hat ihren konzeptionellen Ursprung in den objektorientierten Vorgehensweisen.
- *Proaktive Dokumente* entsprechen einer speziellen Form von aktiven Dokumenten. Die proaktive Komponente zeichnet sich durch antizipierendes Verhalten aus. Proaktive Dokumente sind in der Lage, *selbständig* und *autonom* vorgegebene Ziele zu erreichen.

  In Abschnitt 3.2.4 wurde erläutert, inwiefern proaktive Dokumente den Entwurf und die Realisierung von *dynamischen* Informationssystemen unterstützen.

Technologische Infrastruktur Die technologische Infrastruktur umschreibt die Aspekte und Voraussetzungen für die technische Realisierung von Informationssystemen. Die vorgestellte Taxonomie klassifiziert technologische

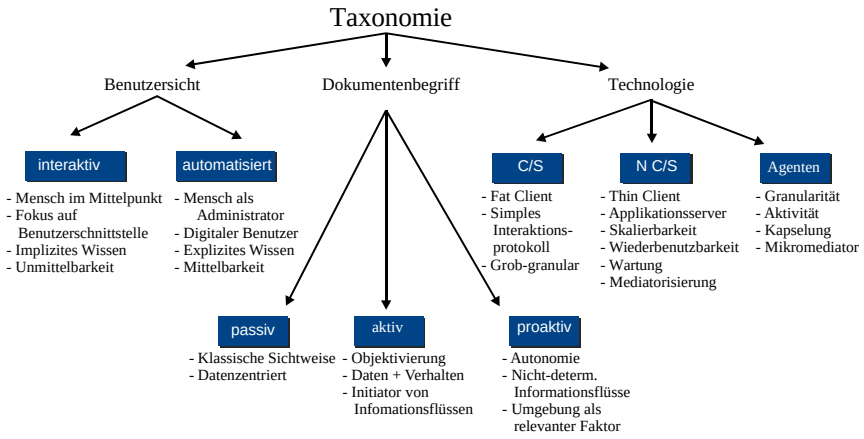


Abbildung 3.8: Zusammenfassung der Taxonomie.

Fragestellungen beim Entwurf und bei der Realisierung von Informationssystemen hinsichtlich dreier Typen von Systemkomponenten: *2-stufige, mehrstufige Client/Server-Architekturen* und *Agenten*. Da Client/Server-Architekturen bereits ausführlich in Abschnitt 2.3.1 auf Seite 28ff vorgestellt wurden, beschränkte sich die Betrachtung technologischer Aspekte im Abschnitt 3.3.1 auf Agenten.

Nach einer kurzen historischen Betrachtung des Agentenkonzepts wurden in Abschnitt 3.3.1 die allgemeinen Eigenschaften von Agenten beschrieben. Im Anschluß wurden Agenten mit Objekten in objektorientierten Paradigmen verglichen, um die grundsätzliche Bedeutung von Agenten zu unterstreichen. In Abschnitt 3.3.3 wurde das Agentenkonzept im Zusammenhang mit Mediatoren betrachtet. Es wurde erläutert, inwiefern Agenten einen speziellen Typus von Mediatoren – sogenannte *Mikromediatoren* – darstellen.

In Abbildung 3.8 wird die Taxonomie für Informationssysteme überblicksartig zusammengefaßt.

Teil II

Leichtgewichtige Informationsinfrastrukturen

Kapitel 4

Respondeo – ein leichtgewichtiger Applikationsserver

Das Client/Server-Modell ist ein logisches Modell für den Entwurf von Informationssystemen. Es enthält mit dem *Client* und dem *Server* zwei Komponenten mit unterschiedlichen Aufgaben und Funktionen. Um einen Dienst des Informationssystems auszuführen und in Anspruch zu nehmen, übernehmen die beiden Komponenten folgende Rollen:

- Der Client ist ein *Dienstnehmer*. Normalerweise ist ein Client ein Programm, das direkt mit dem Benutzer interagiert. Der Client löst Anfragen nach einer konkreten Dienstleistung bei Servern aus und wartet jeweils auf die Antwort zu den gestellten Anfragen.
- Der Server ist der *Diensterbringer*. Er hält bestimmte Dienstleistungen bereit, z. B. das Versenden von E-Mails oder die Durchführung einer Banktransaktion. Grundsätzlich handelt es sich dabei um einen Softwareprozeß, der auf Anfragen von Clients wartet, diese ausführt und den wartenden Clients die Antworten und Ergebnisse zurückschickt.

Ein Applikationsserver ist eine Middleware-Komponente [Ber96], die Dienstleistungen für die Applikationsebene zur Verfügung stellt, um mit den Instanzen auf der Ebene der Datenbasen zu interagieren[1]. Die Terminologie der Datenbasis ist ein sehr weit gefaßter Begriff für eine Systemkomponente, die (digitale) Informationen enthält[2]. In dieser allgemeinen Form subsumiert eine Datenbasis im wesentlichen zwei Arten von Komponenten:

- *Informationssystem*. Ein Informationssystem und seine Systemkomponenten werden ganzheitlich als Datenbasis betrachtet, in dem der Benutzer

[1] In Abschnitt 2.3.1.2 werden Applikationsserver im Zusammenhang mit Client/Server-Informationssystemen diskutiert.

[2] Vgl. Abschnitt 2.1.

durch das Stellen von Anfragen in die Lage versetzt wird, (digitale) Informationen über eine bestimmte Problemdomäne des Informationssystems zu erhalten.

- *Dienst*. Ein Dienst wird als Datenbasis für eine – im Vergleich zu der ganzheitlichen Betrachtung – stark eingeschränkte bzw. spezialisierte Problemdomäne betrachtet.

Eine ganzheitliche und spezielle Interpretation einer Datenbasis erweitert den Anwendungs- und Einsatzbereich eines Applikationsservers. Einem Applikationsserver, der einen wichtigen Mediator [Wie92a, WG97, RW91, WRD+90][3] darstellt, kommt eine große Bedeutung sowohl beim Entwurf von mehrstufigen Client/-Server-Informationssystemen als auch für die Realisierung netzwerkartiger und dienstbasierter Informationssysteme zu.
In diesem Kapitel wird der Entwurf und die Realisierung des Applikationsservers *Respondeo* [4] diskutiert, der sowohl für den Einsatz in mehrstufigen Client/Server-Informationssystemen (vgl. Abschnitt 2.3.1.2) als auch in dienstzentrierten Informationssystemen (vgl. Abschnitt 2.3.1.3) geeignet ist. Damit unterstützt er u. a. die Forderungen, die in Abschnitt 2.1.2.2 über *dienstbasierte Technologien* diskutiert werden, einer Integration von verteilten und autonomen Softwarekomponenten. Aufgrund der flexiblen Verknüpfung existierender Dienste durch ein Netzwerk lassen sich neue Informationssysteme einfach entwickeln bzw. aus bestehenden Komponenten aggregieren.
Respondeo [SMKK00] ist ein sogenannter leichtgewichtiger Applikationsserver. Der Begriff der *Leichtgewichtigkeit* beinhaltet folgende Eigenschaften:

- Die Kernkomponente *Respondeo*s ist von geringer Codegröße[5]. Damit besitzt ein leichtgewichtiger Applikationsserver – wie *Respondeo* – im Gegensatz zu den meisten kommerziellen Applikationsservern[6] einen kleinen sogenannten Memory-Footprint.

- Die Palette an verfügbaren Diensten ist in einem leichtgewichtigen Applikationsserver eingeschränkt. Bestimmte Dienste – wie beispielsweise ein Transaktionsmanagement – werden von der Kernkomponente *Respondeo*s nicht angeboten. Derartige Dienste werden aus der Sicht von *Respondeo* als Zusatzdienste kategorisiert und können als spezielle Applikationsobjekte in *Respondeo* eingebunden werden. Allerdings sind diese Zusatzdienste nicht Bestandteil der Kernkomponente, damit das oben angesprochene Kriterium der geringen Codegröße erfüllt wird. Gemäß der Diskussion von Mediatoren

[3] In Abschnitt 2.2 werden die Eigenschaften von Mediatoren erörtert.

[4] *respondere* (lat.) – antworten, beantworten, entgegnen, erwidern; *respondeo* (1. Person Singular) – *ich antworte*.

[5] Die Kernkomponente *Respondeo*s ist ca. 60 KByte groß. Die Größe der Applikationsobjekte variiert und ist abhängig vom jeweiligen Informationssystem.

[6] Vgl. Abschnitt 4.6.

in Abschnitt 2.2 stellt ein Applikationsobjekt einen Mediator dar, der für die Verwaltung und für die spezifischen Datentransformationen der jeweiligen Dienste verantwortlich ist.

- Die einfache Integration in bereits bestehende Applikationen.

Das vorliegende Kapitel befaßt sich mit dem Entwurf und der Realisierung von *Respondeo*. In Abschnitt 4.1 werden die spezifischen *Ziele* beim Entwurf und der Realisierung von *Respondeo* erörtert. Danach wird in Abschnitt 4.2 die *Architektur* vorgestellt, in deren Mittelpunkt das *Applikationsmodell* und der *Nachrichtenbus* von *Respondeo* stehen. Abschnitt 4.3 befaßt sich mit den Details der *Implementierung* des leichtgewichtigen Applikationsservers. Abschnitt 4.4 beschreibt mehrere Informationssysteme, die *Respondeo* und dessen Dienste benutzen. In Abschnitt 4.5 wird eine Einordnung von *Respondeo* in die in Kapitel 3 diskutierte Taxonomie vorgenommen. Schließlich werden in Abschnitt 4.6 vergleichbare Projekte beschrieben.

4.1 Ziele

In diesem Abschnitt werden die Entwurfsziele *Respondeo*s erörtert. Generell orientiert sich der Entwurf *Respondeo*s an den Anforderungen, die sich an einen Applikationsserver in mehrstufigen Client/Server-Informationssystemen stellen (vgl. Abschnitt 2.2.2.3).
Als Applikationsserver bietet *Respondeo* Dienste zur Verwaltung der Applikationslogik von Informationssystemen an. Die konkrete Implementierung der Applikationslogik liegt in der Verantwortung des Informationssystems oder der Systemkomponente, die in *Respondeo* eingebunden ist. In diesem Zusammenhang wird von der Integration von Informationssystemen oder Systemkomponenten in das Softwareframework von *Respondeo* gesprochen. Damit soll zum Ausdruck gebracht werden, daß *Respondeo* lediglich einen Rahmen für die Verwaltung und Wiederbenutzung von Applikationsobjekten vorgibt. Die eigentliche inhaltliche Ausprägung eines Applikationsobjekts obliegt dem jeweiligen Informationssystem, welches in das Softwareframework integriert ist.
Die Ziele beim Entwurf des leichtgewichtigen Applikationsservers *Respondeo* sind im folgenden dargelegt:

- *Generalisierung von Dienstleistungen*. Für die Generalisierung von Diensten oder Dienstleistungen ist es kennzeichnend, daß Funktionalitäten, die in unterschiedlichen Anwendungskontexten verlangt sind, allgemein entworfen und implementiert werden [Vin02, SK01]. Diese Funktionalitäten werden als Dienste dergestalt zur Verfügung gestellt, daß der gleiche Dienst von unterschiedlichen Anwendungen benutzt werden kann.

Beispielsweise sind die Aspekte einer parallelen Verarbeitung[7] von Benutzeranfragen oft unabhängig von dem konkreten Informationssystem, in dem der Applikationsserver eingesetzt wird. Deswegen ist ein Ziel des Entwurfs von *Respondeo* die Etablierung eines Parallelisierungsdienstes, der von allen in *Respondeo* integrierten Informationssystemen und Systemkomponenten einheitlich genutzt werden kann.

- *Integrative Kommunikationsparadigmen.* Unter einem integrativen Kommunikationsparadigma werden folgende Eigenschaften verstanden:

  - Die *Einfachheit* der Integration eines Informationssystems oder einer Systemkomponente in das Softwareframework von *Respondeo.* Eine einfache Einbindung verkürzt den Integrationsprozeß und unterstützt so die Akzeptanz bei Entwicklern und Benutzern von *Respondeo.*
  - Die *Dauerhaftigkeit* der benutzten Kommunikationsschnittstellen. Damit eine möglichst große Zahl an Systemkomponenten in *Respondeo* integriert werden kann, sind Schnittstellen erforderlich, die sich nicht fortwährend ändern. Durch die Änderung, welche durch die Modifikation einer Schnittstelle hervorgerufen worden ist, müßten die bereits integrierten Systeme an die „neuen“ Schnittstellen von *Respondeo* angepaßt werden. Dieses Vorgehen ist nicht wünschenswert im Rahmen einer integrativen Systemkomponente wie *Respondeo.*

- *Skalierbarkeit.* Das Kriterium der Skalierbarkeit [Neu94] tritt in zwei Ausprägungen in Erscheinung:

  - Anzahl der Anfragen, die von *Respondeo* parallel bearbeitet werden können.
  - Anzahl der in *Respondeo* eingebundenen Informationssysteme oder Systemkomponenten.

- *Flexible Konfigurationsmöglichkeiten.* Die Anpassung der von *Respondeo* angebotenen Dienste und die Einbindung weiterer Informationssysteme erfolgt durch eine geeignete Konfiguration der bereits existierenden Komponenten. An die Stelle der Programmierung neuer Funktionen und weiterer Einbindungen tritt ein Prozeß, der sich dadurch auszeichnet, daß existierende und konfigurierbare Komponenten wiederbenutzt werden. Generell wird die Zeit für die Einbindung neuer Systeme in *Respondeo* wesentlich reduziert, indem eine konsequente Trennung zwischen der Entwicklung (Programmierung) und der eigentlichen Benutzung vorgenommen wird [Vin02]. Zudem

[7] Man spricht in diesem Zusammenhang auch von sogenannten *multi-thread* fähigen Applikationsservern, d. h. jede eintreffende Benutzeranfrage erhält ihren eigenen Kontrollfluß (thread of control) zugewiesen.

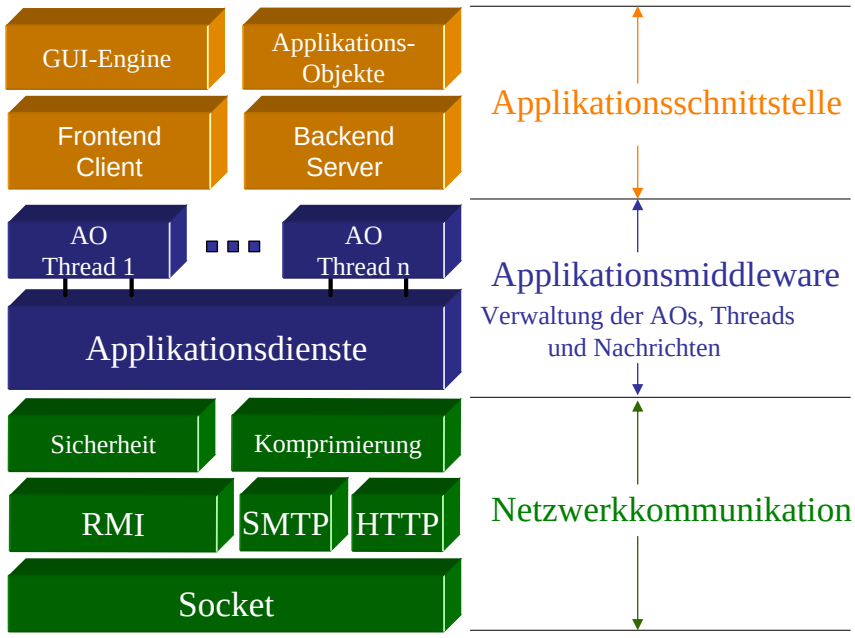


Abbildung 4.1: Die Architekturschichten *Respondeo*s.

wird der Wartungsaufwand für das Softwareframework von *Respondeo* gesenkt, da existierende und bereits getestete Komponenten zum Einsatz kommen.

4.2 Architektur

In diesem Abschnitt werden die Hauptkomponenten der objektorientierten Frameworkarchitektur von *Respondeo* vorgestellt. Ein objektorientiertes Framework ist eine wiederbenutzbare, halbfertige Applikation, die Anpassungsmöglichkeiten bietet, um konkrete Applikationen zu generieren [FS97](vgl. Abschnitt 2.1.2.2). Es ist charakterisiert durch eine Menge von abstrakten Klassen und Schnittstellen, die als Ganzes eine generische Software-Architektur für eine Reihe von ähnlichen Applikationen darstellt [JF88, JR91]. Dabei werden zwei Applikationen als ähnlich bezeichnet, sofern sie der gleichen oder einer ähnlichen Applikationsdomäne angehören.
Eine Schnittstelle definiert in diesem Zusammenhang eine objektorientierte Klasse, in der alle Methoden abstrakt spezifiziert sind. Folglich unterstützen Frameworks eine klare Unterscheidung zwischen der Spezifizierung von Funktionalitäten und ihrer jeweiligen Implementierung[8].
Generell orientiert sich die Architektur *Respondeo*s an einem Schichtenmodell ge-

[8] Siehe auch die Diskussion von Frameworks in Abschnitt 2.1.2.2.

mäß Abbildung 4.1. Grundsätzlich gibt es drei Schichten im Softwareframework von *Respondeo*:

- *Netzwerkkommunikation*. Die Netzwerkkommunikation ist die unterste Schicht des Frameworks und basiert auf der Kommunikation mit *Sockets* [CS93]. Die *Remote Method Invocation* (RMI) [Sun01] bietet in Java die Möglichkeit, Methoden von entfernten Objekten transparent aufzurufen. In dieser Hinsicht basiert RMI auf dem allgemeinen Prinzip des *Remote Procedure Call* (RPC), das erstmals von Birell und Nelson [BN84] 1984 vorgestellt wurde. RMI benutzt Sockets auf der untersten Ebene der Datenübertragung.

  Auf der Basis von RMI bietet *Respondeo* zwei verschiedene Typen von Kommunikationskanälen für die Datenübertragung an. *Respondeo* gewährleistet eine sichere und komprimierte Datenübertragung. Durch die Komprimierung wird das Datenvolumen verringert und die Netzwerklast gesenkt. Neben RMI bietet *Respondeo* auch die Möglichkeit, das *Simple Mail Transfer Protocol* (SMTP) [Kle01] (siehe Abschnitt 4.4.2) oder das *Hypertext Transfer Protocol* (HTTP) [Wor00] (siehe Abschnitt 4.4.3) für die Netzwerkkommunikation zu benutzen.

- *Applikationsmiddleware*. Die mittlere Schicht bietet verschiedene Dienste für die Verwaltung der integrierten Komponenten an. Sobald eine Komponente in *Respondeo* integriert ist, steht diese prinzipiell als Dienst in einem breiten Anwendungskontext zur Verfügung. Ein solcher Dienst wird als Applikationsobjekt (AO) bezeichnet und von *Respondeo* verwaltet. Gemäß den Ausführungen über Dienste und deren Spezifikationen in Abschnitt 2.1.2.2 ist ein Dienst durch die Spezifikation seiner Schnittstelle gegeben. Da ein Zugriff auf einen Dienst über *Respondeo* erfolgt, ist die Spezifikation der Applikationsschnittstelle von *Respondeo* maßgeblich. Diese wird auf der obersten Ebene der Schichtenarchitektur gemäß Abbildung 4.1 festgelegt.

  Neben den applikationsspezifischen Diensten stellt *Respondeo* einen Parallelisierungsdienst zur Verfügung, der jedem Applikationsobjekt eine Menge von Kontrollflüssen zuordnet (siehe `AO Thread 1` bzw. `AO Thread n` in Abbildung 4.1). Dieser spezielle Dienst kann über Konfigurationsmechanismen an das jeweilige Applikationsobjekt angepaßt werden (siehe Abschnitt 4.3.1).

- *Applikationsschnittstelle*. Die oberste Schicht des Frameworks bestimmt die Applikationsschnittstelle von *Respondeo*. Diese Schnittstelle wird sowohl von Clients als auch von Backend-Servern benutzt, um auf *Respondeo* und dessen Dienste zuzugreifen. Die Applikationsschnittstelle bestimmt Nachrichten, die an *Respondeo* geschickt werden. Der Zugriff auf *Respondeo* und dessen Dienste erfolgt *ausschließlich* über die nachrichtenbasierte Schnittstelle, die in Abschnitt 4.2.1 genauer erörtert wird.

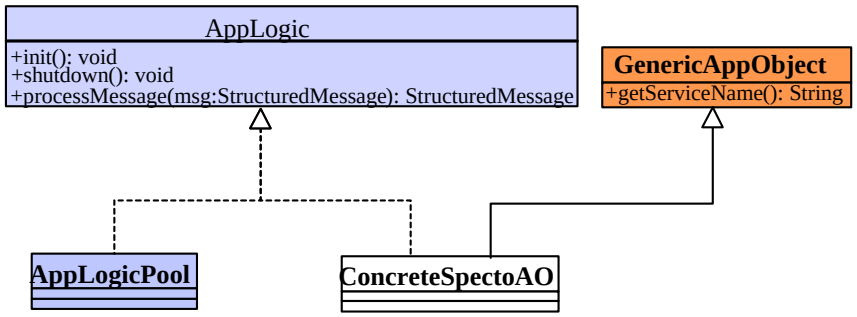


Abbildung 4.2: Das Applikationsmodell von *Respondeo*.

4.2.1 Applikationsmodell

Das Applikationsmodell von *Respondeo* legt den Rahmen für die Bestimmung und die Spezifikation von Applikationsobjekten innerhalb von *Respondeo* fest. Ein Applikationsobjekt ist eine Abstraktion für Dienste bzw. Informationssysteme, die in *Respondeo* integriert sind. Um ein neues Applikationsobjekt bzw. einen neuen Dienst in *Respondeo* zu integrieren, wird eine neue Klasse entworfen (z. B. die Klasse `ConcreteSpectoAO` gemäß Abbildung 4.2), die eine Generalisierung der Klassen `AppLogic` und `GenericAppObject` darstellt. Im folgenden werden die Eigenschaften der am Applikationsmodell beteiligten Klassen näher beschrieben:

AppLogic Im Mittelpunkt des Applikationsmodells von *Respondeo* steht das Applikationsobjekt. Gemäß Abbildung 4.2 ist jedes Applikationsobjekt vom Typ `AppLogic`. `AppLogic` ist eine Java-Schnittstelle (Interface), die zwei Methoden für die Verwaltung eines in *Respondeo* integrierten Dienstes spezifiziert[9]:

- Die Methode `init()` wird bei der Initialisierung des Applikationsobjekts von *Respondeo* direkt ausgeführt. In dieser Methode werden die spezifischen Eigenschaften des jeweiligen Applikationsobjekts hinterlegt.
- Die Methode `shutdown()` wird bei der Beendigung des Dienstes aufgerufen.

  Die Methoden `init()` und `shutdown()` dienen als *innere* Schnittstellen zwischen den Applikationsobjekten und dem Kernframework von *Respondeo*.

[9]Eine Java-Schnittstelle ist eine objektorientierte Klasse, die nur abstrakte Methoden spezifiziert. Im Gegensatz zu einer Java-Schnittstelle erlaubt eine abstrakte Klasse in Java die Spezifikation auch von nicht-abstrakten Methoden, die gegebenenfalls überladen werden können. Eine Java-Schnittstelle unterstützt die Trennung zwischen der Spezifikation und der Implementierung von Diensten in *Respondeo*.

- Die Methode `processMessage()` bestimmt die nachrichtenbasierte Kommunikationsschnittstelle von *Respondeo*. Jede Nachricht ist vom Typ `StructuredMessage` und entspricht einer strukturierten Nachricht, die aus einem einheitlichen Nachrichtenkopf und -rumpf besteht[10]. Die nachrichtenbasierte Kommunikationsschnittstelle zeichnet sich dadurch aus, daß ausschließlich über den Austausch von standardisierten Nachrichten auf die Applikationsobjekte von *Respondeo* zugegriffen wird. Jedes Applikationsobjekt besitzt somit die gleiche syntaktische Kommunikationsschnittstelle, was die Integration von verschiedenen Diensten wesentlich erleichtert.

  Die Methode `processMessage()` bildet die *äußere* Schnittstelle[11], auf die Clients und Server auf *Respondeo* bzw. auf dessen Dienste zugreifen.

GenericAppObject Die Klasse `GenericAppObject` bietet zusätzliche Funktionalitäten für Applikationsobjekte an. So wird beispielsweise über die Methode `getServiceName()` der Name des integrierten Dienstes zurückgegeben. Generell ist es für die Generierung von Applikationsobjekten aber nicht zwingend erforderlich, die Funktionalitäten von `GenericAppObject` zu erben[12]. `GenericAppObject` dient lediglich als Container für zusätzliche Funktionalitäten, die allgemein benutzbar sind. Diese Funktionalitäten sind nicht an spezifische Eigenschaften einzelner Applikationsobjekte gebunden. Die Klasse `GenericAppObject` unterstützt so die Wiederbenutzbarkeit von Funktionen auf der Entwurfsebene der (abstrakten) Applikationsobjekte anstelle der konkreten Applikationsobjekte (z. B. `ConcreteSpectoAO` in Abbildung 4.2).

AppLogicPool Die Klasse `AppLogicPool` ist ein Applikationsobjekt, welches von *Respondeo* selbst als allgemeiner Dienst zur Verfügung gestellt wird. `AppLogicPool` ermöglicht die parallele Ausführung von mehreren Instanzen eines bestimmten Typs von Applikationsobjekten. Die Klasse `AppLogicPool` kapselt alle Zugriffe auf die betreffenden Applikationsobjekte und weist jeder Instanz des Applikationsobjekts einen eigenen Kontrollfluß (thread of control) zu. Die zugewiesenen Kontrollflüsse werden in einem Pool verwaltet. Die Anzahl der maximal verfügbaren Kontrollflüsse (siehe `poolSize` in Listing 4.1) ist für ein konkretes Applikationsobjekt konfigurierbar (siehe auch Abschnitt 4.3.1).
Die Methode `init()` in der Klasse `AppLogicPool` lädt das zuvor festgelegte Applikationsobjekt (siehe `member` in Listing 4.1) dynamisch zur Laufzeit. Ein weiterer Bestandteil der Initialisierungsphase des speziellen Applikationsobjekts `AppLogicPool` ist die oben angesprochene Zuweisung eines Pools von Kontrollflüssen, der ausschließlich für Anfragen an diesen Dienst zur Verfügung steht.

[10] In Abschnitt 4.3.2 wird der Aufbau von Nachrichten des Typs `StructuredMessage` ausführlich erörtert.

[11] Dies entspricht der Applikationsschnittstelle in Abbildung 4.1.

[12] Die einzige Restriktion für den Entwurf von Applikationsobjekten bzw. Diensten innerhalb des Applikationsmodells *Respondeo*s ist die Generalisierung der Klasse `AppLogic`.

Falls für die Bearbeitung einer eintreffenden Dienstanfrage aktuell kein weiterer Kontrollfluß zur Verfügung steht, wird jene Anfrage blockiert, bis im Pool wieder ein Kontrollfluß vorhanden ist. Generell wird nach Beendigung einer Anfrage der dann zur Verfügung stehende Kontrollfluß wieder in den Pool eingefügt, damit er für weitere Anfragen genutzt werden kann.
Die Klassen `AppLogicPool` und `GenericAppObject` können durch Generalisierung bzw. Vererbung problemlos erweitert werden. Dadurch können beliebig weitere Parallelisierungsstrategien oder generelle Funktionalitäten in das Framework von *Respondeo* eingebunden werden. Das dynamische Laden konkreter Applikationsobjekte zur Laufzeit durch die Klasse `AppLogicPool` oder künftiger weiterer `Pool`-Klassen fördert sowohl eine einfache auf Konfigurationstechniken basierende Integration als auch eine hohe Wiederbenutzbarkeit bereits vorhandener Softwarekomponenten.
Die Konfiguration und Initialisierung von Applikationsobjekten wird in Abschnitt 4.3.1 ausführlich geschildert und ist in Abbildung 4.4 illustriert.

4.2.2 Nachrichtenbus *MBus*

Eine wichtige Komponente in *Respondeo* ist der sogenannte Nachrichtenbus *MBus*, der für den im vorherigen Abschnitt 4.2.1 angesprochenen Austausch von Nachrichten verantwortlich ist. *MBus* ist ein nachrichtenbasierter Softwarebus, der objektorientiert in Java implementiert ist. Er ermöglicht die Realisierung von Informationssystemen, deren Systemkomponenten in einer verteilten Umgebung zum Einsatz kommen.
Folgende Eigenschaften sind für den nachrichtenbasierten Softwarebus *MBus* charakteristisch:

- Die Teilnehmer bzw. die Kommunikationspartner des Softwarebusses werden durch symbolische Namen eindeutig repräsentiert[13]. Jeder Teilnehmer wird *indirekt* über seine Referenz, den symbolischen Namen, angesprochen. Es erfolgt keine direkte Kommunikation zwischen den beteiligten Kommunikationspartnern.

- Die Kommunikation zwischen den Teilnehmern erfolgt ausschließlich durch den Austausch von standardisierten Nachrichten[14]. Es werden keine proprietären Kommunikationsschnittstellen der jeweiligen Teilnehmer aufgerufen oder benutzt.

- *Respondeo* übernimmt die Verwaltung der ausgetauschten Nachrichten, d. h. *Respondeo* ermittelt zur Laufzeit den Adressaten einer Nachricht und übergibt anschließend den Nachrichteninhalt an den Empfänger. Die Ziele

[13] Zur Klarstellung der verwendeten Terminologie eines *Teilnehmers*: Ein Teilnehmer im Zusammenhang mit *MBus* entspricht einem Dienst, der in *Respondeo* integriert worden ist. Nach dieser Integration wird daraus ein auf eintreffende Nachrichten wartender Teilnehmer – im Sinne von *MBus*.
[14] In Abschnitt 4.3.2 wird das in *Respondeo* benutzte Nachrichtenformat im Detail beschrieben.

und die Transportwege von Nachrichten in einem Softwarebus werden auf Applikationsebene bestimmt. Das grundsätzliche Vorgehen entspricht dem Nachrichtenversand auf der Netzwerkebene unter Verwendung des TCP/IP-Protokolls.

Der Entwurf von *MBus* basiert im wesentlichen auf dem Prinzip eines *dynamischen Attributes*. Eine Kommunikations- und Applikationsschnittstelle wie die Klasse `AppLogic` mit der Methode `processMessage()` in Abbildung 4.2 beinhaltet als Übergabeparameter lediglich ein Objekt der Klasse `StructuredMessage`[15]. Anstelle der Definition von separaten Schnittstellen für jedes mögliche Attribut oder für jeden möglichen Parameter wird ein allgemeiner (Daten-)Container `StructuredMessage` definiert. `StructuredMessage` ist in der Lage, beliebig komplexe Attribute oder Parameter zu transportieren. Die folgenden Vorteile ergeben sich durch den Entwurf auf der Basis von dynamischen Attributen:

- Für die Integration eines neuen Teilnehmers in *MBus* ist keine Re-Kompilierung von etwaigen neuen Schnittstellen notwendig. *MBus* besitzt nur eine einzige Schnittstelle, die auch nach der Integration von neuen Teilnehmern unverändert bleibt. Während sich der Nachrichteninhalt ändert bzw. ändern kann, bleibt die Methodensignatur unverändert. Die syntaktischen Schnittstellen sind so über längere Zeiträume hinweg stabil und ermöglichen die Realisierung robuster Informationssysteme[16].

- Bei potentiellen Anpassungen an zukünftige Anforderungen ist die Kommunikationsschnittstelle von *MBus* nicht betroffen. Analog zur vorherigen Argumentation bleiben die syntaktischen Schnittstellen unverändert stabil. Dies führt nicht nur zu stabilen, sondern auch zu flexibel erweiterbaren Kommunikationsschnittstellen.

- Die Anzahl der benötigten Schnittstellen zwischen Kommunikationspartnern ist sehr gering. Dementsprechend reduziert sich der Aufwand für die Wartung und Validierung von zahlreichen Schnittstellendefinitionen.

4.2.3 Prinzipieller Ablauf

In Abbildung 4.3 ist der grundsätzliche Ablauf der Verarbeitung einer Benutzeranfrage dargestellt:

1. Der Benutzer initiiert einer Anfrage. In Abbildung 4.3 wird die Anfrage beispielhaft aus einem Web-Client gestellt. Die Anfrage wird in eine Nachricht

[15] Siehe Listing 4.2 für eine detaillierte Beschreibung der Klasse `StructuredMessage`.

[16] Siehe für die Diskussion der Trennung von syntaktischen und semantischen Details der Kommunikationsschnittstellen auch den Abschnitt 3.3.2 über eine agentenbasierte Kommunikation.

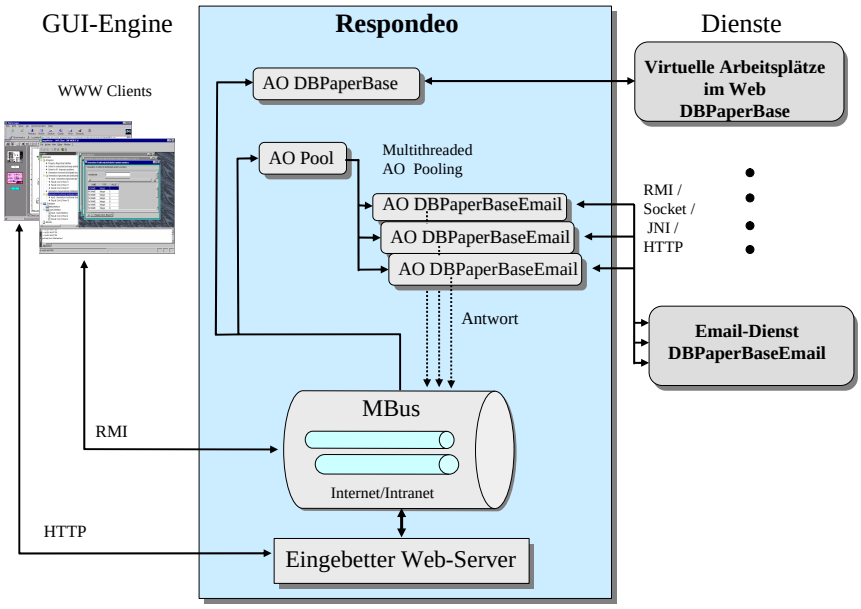


Abbildung 4.3: Systemüberblick und Verarbeitungsablauf von Anfragen an *Respondeo*.

des Typs `StructuredMessage` verpackt. In Abhängigkeit vom benutzten Kommunikationsprotokoll wird die Nachricht direkt (*RMI*) oder indirekt (*HTTP*) an den Nachrichtenbus *MBus* verschickt.

2. Die Anfrage wird auf den Nachrichtenbus *MBus* (vgl. Abschnitt 4.2.2) gelegt und an das betreffende, für die Beantwortung der Benutzeranfrage zuständige Applikationsobjekt weitergeleitet. Die Kernkomponente *Respondeo*s prüft die Verfügbarkeit des betreffenden Applikationsobjekts. Falls eine Instanz dieses Applikationsobjekts verfügbar ist, wird diese aus dem Pool von verfügbaren Applikationsobjekten entfernt und der aktuellen Anfrage zugeteilt.

3. Das jeweilige Applikationsobjekt prüft den Nachrichteninhalt. Dies liegt nicht mehr im Verantwortungsbereich des Kernframeworks. Nach der Bestimmung der erforderlichen Aktionen werden die notwendigen Schritte durch das jeweilige Applikationsobjekt eingeleitet.

4. Das Ergebnis der Anfrage wird in eine Nachricht des Typs `StructuredMessage` verpackt und an den Client bzw. Benutzer zurückgeschickt. Nach Beendigung der Anfrage wird das betreffende Applikationsobjekt in den Pool von verfügbaren Applikationsobjekten eingefügt und steht somit für die Verarbeitung neuer Anfragen bereit.

4.3 Implementierung

In den folgenden zwei Abschnitten werden spezifische Eigenschaften der Implementierung und Benutzung von *Respondeo* dargelegt:

- In Abschnitt 4.3.1 werden die Details des Konfigurationsmechanismus von *Respondeo* und dessen integrierten Applikationsobjekten erörtert. Der Einsatz von Konfigurationsmechanismen unterstützt die einfache und schnelle Integration von Applikationsobjekten in das Framework von *Respondeo.*
- Abschnitt 4.3.2 beschreibt den Aufbau des Nachrichtenformats, das innerhalb des gesamten Frameworks zum Einsatz kommt.

4.3.1 Konfiguration

Die Konfiguration und Initialisierung der Applikationsobjekte in *Respondeo* erfolgt in mehreren Schritten:

1. *Einlesen der Konfigurationsparameter*. *Respondeo* liest eine strukturierte Konfigurationsdatei ein, welche die notwendigen Informationen über die Applikationsobjekte enthält.
2. *Bestimmen der Applikationsobjekte und von deren Klassenpfaden*. Die Eigenschaften jedes Dienstes werden über Name-Wert-Paare bestimmt. Der Name des ersten Paares bestimmt den symbolischen Namen des Dienstes, unter dem er bei *Respondeo* angemeldet ist. Im oberen Teil der Abbildung 4.4 sind dies beispielsweise zwei Dienste mit Namen `DBPaperbase` und `DBPaperBaseEmail`. Der Wert des ersten Paares gibt die Klasse an, welche die Applikationslogik des Dienstes implementiert. Dies sind im oben genannten Beispiel die Klassen `api.djenterprise.appserv.logic.DBserialized` bzw. `api.djenterprise.appserv.logic.RespondeoMailGateway`.
3. *Dynamisches Laden der Applikationsobjekte durch Respondeo.* Das Laden der Klasse, welche die Applikationslogik für den Dienst enthält, erfolgt dynamisch zur Laufzeit und wird vom Kernframework von *Respondeo* vorgenommen.
4. *Initialisieren und Starten der Applikationsobjekte*. Nachdem die spezifizierten Klassen geladen worden sind, werden sie durch den Aufruf der jeweiligen `init()`-Methode (siehe auch Abbildung 4.2) initialisiert und gestartet.

In Abbildung 4.4 ist der Konfigurationsmechanismus von *Respondeo* veranschaulicht. Ein bestimmter Teil der Name-Wert-Paare (`member` und `poolSize`) wird durch das Kernframework von *Respondeo* verarbeitet. Die übrigen Parameter liegen in dem Verantwortungsbereich der jeweiligen Applikationsobjekte.

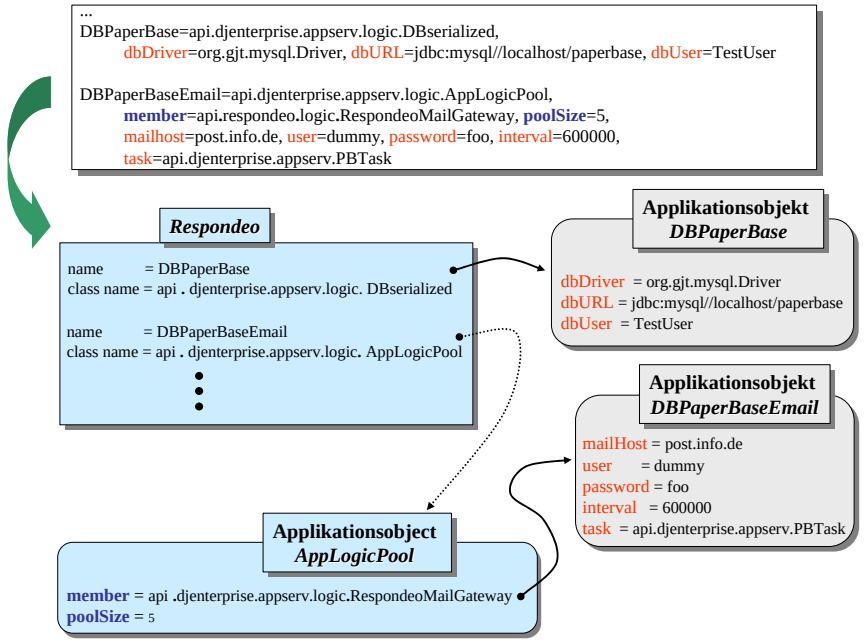


Abbildung 4.4: Konfiguration und Initialisierung von Applikationsobjekten in *Respondeo*.

Grundsätzlich gibt es zwei Typen von Parametern in *Respondeo*, wobei jeder Parameter durch ein Name-Wert-Paar repräsentiert wird:

- *Allgemeine Parameter*. Ein allgemeiner Parameter enthält Informationen, die unabhängig von Applikationsobjekten sind. In Abbildung 4.4 sind zwei allgemeine Parameter – `member` und `poolSize` – dargestellt. Diese Parameter beziehen sich auf die Klasse `AppLogicPool`. Der Parameter `member` gibt den Klassenpfad des Applikationsobjekts an, welches vom Parallelisierungsdienst (`AppLogicPool`) dynamisch erzeugt und verwaltet wird. Der Parameter `poolSize` bestimmt die maximale Anzahl von Kontrollflüssen, die dem im Parameter `member` bestimmten Applikationsobjekt zugeordnet werden.

- *Spezielle Parameter*. Ein spezieller Parameter ist immer einem bestimmten Applikationsobjekt zugeordnet. Die Bedeutung und die Anwendbarkeit spezieller Parameter in *Respondeo* ist somit immer auf einen bestimmten Typ von Applikationsobjekt beschränkt. Das Kernframework von *Respondeo* behandelt spezielle Parameter als sogenannte Black-Box, d. h. jeder spezielle Parameter wird direkt an das Applikationsobjekt übergeben.

Diese Aufteilung bzw. Klassifizierung von Parametern unterstützt den Einsatz von sogenannten *hot spots* [Pre94] innerhalb von Frameworks. Generell bietet ein *hot*

```
# Auszug aus der
#   Konfiguration und Parametrisierung der  Applikationsobjekte
#
# Die Applikationobjekte lesen die Parameter ein und üfhren
# entsprechende Aktionen aus

# PaperBase
DBPaperBase=api.djenterprise.appserv.logic.DBserialized
    ,dbDriver=org.gjt.mysql.Driver
    ,dbURL=jdbc:mysql://localhost/paperbase
    ,dbUser=TestUser
    ,dbPwd=foo

#  PaperBase mit Anbindung an Mailgateway
DBPaperBaseEmail=api.djenterprise.appserv.AppLogicPool
    ,member=api.djenterprise.appserv.logic.RespondeoMailGateway
    ,poolSize=2
    ,mailHost=post.info.de
    ,user=dummy
    ,password=foo
    ,interval=6000000
    ,PBHost=dilbert
    ,task=api.djenterprise.appserv.PaperbaseTask
```

Listing 4.1: Applikationsobjekte für das Informationssystem *PaperBase*.

spot die Möglichkeit, an vorgegebenen Punkten des Frameworks, Erweiterungen vorzunehmen. Im Zusammenhang mit dem Framework von *Respondeo* stellt die Konfiguration der Dienste über Name-Wert-Paare und deren Klassifizierung in allgemeine und spezielle Parameter eine Erweiterungsmöglichkeit dar, um beliebige Dienste einheitlich in *Respondeo* einzubinden.
In Listing 4.1 sind beispielhaft die Spezifikationen für die Applikationsobjekte `DBPaperBase` und `DBPaperBaseEmail` illustriert, die auch in der Abbildung 4.4 zur Darstellung des Initialisierungsmechanismus in *Respondeo* verwendet werden:

DBPaperBase DBPaperBase ist ein Dienst, der die Datenbankanbindung an das Informationssystem *PaperBase* realisiert[17]. `DBPaperBase` ist der symbolische Name des Dienstes, der bei *Respondeo* angemeldet ist. Jede Nachricht mit dem Empfänger `DBPaperBase`, die über den Nachrichtenbus *MBus* eintrifft, wird von *Respondeo* an den Dienst `DBPaperBase` weitergeleitet. Die in Java implementierte Applikationslogik liegt in der Klasse `DBserialized`. Zudem gibt es eine Reihe von speziellen Parametern, welche die Datenbankverbindung zum Informationssystem *PaperBase* näher spezifizieren:

- `dbDriver` bestimmt den Datenbanktreiber. Innerhalb des Dienstes `DBPaperBase` wird ein MySQL-Treiber [MyS02a] benutzt.

[17] Vgl. Abschnitt 10.1 hinsichtlich einer ausführlichen Diskussion des Informationssystems *PaperBase*.

- `dbURL` bestimmt das Protokoll zur Kommunikation mit der Datenbank.
- `dbUser` bzw. `dbPwd` bestimmen den Benutzernamen und das Kennwort für die Authentifizierung mit der Datenbank. Alle Datenbankzugriffe des Dienstes `DBPaperBase` erfolgen unter diesem Benutzer.

DBPaperBaseEmail DBPaperBaseEmail ist ein Dienst, der vom Informationssystem *PaperBase* angeboten wird, um E-Mails zu empfangen und zu verschicken. So werden beispielsweise die Benutzer der *PaperBase* über aktuelle Änderungen per E-Mail informiert oder können neue Dokumente per E-Mail in das Informationssystem einfügen. Im Gegensatz zum Dienst `DBPaperBase` werden Anfragen an `DBPaperBaseEmail` über einen Pool von mehreren Instanzen dieses Dienstes beantwortet. Für die parallele Verarbeitung der Anfragen bzw. für die Verwaltung des Pools ist das Applikationsobjekt `AppLogicPool` verantwortlich. In Listing 4.1 sind zwei allgemeine Parameter dargestellt, die der Klasse `AppLogicPool` zugeordnet sind:

- `member` bestimmt den Klassenpfad bzw. die Klasse, welche die Applikationslogik zum Empfangen und Verschicken von E-Mails implementiert (`RespondeoMailGateway`).
- `poolSize` bestimmt die maximale Anzahl der Instanzen vom Typ `DBPaperBaseEmail`. Die Instanzen stehen gleichzeitig für die Verarbeitung von Anfragen zur Verfügung.

Neben den allgemeinen Parametern der Klasse `AppLogicPool` gibt es spezielle Parameter, welche die Applikationslogik für das Empfangen und Senden von E-Mails näher spezifizieren:

- `mailHost` bestimmt den Hostnamen des Mailrechners.
- `user` bzw. `password` bestimmen den Benutzernamen und das Kennwort für die Authentifizierung mit dem E-Mail-System auf dem Rechner `mailHost`.
- `interval` bestimmt das Intervall in ms, in dem unter dem Namen `user` auf dem Rechner `mailHost` neue, verfügbare E-Mails abgerufen werden.
- `PBHost` spezifiziert den Hostnamen der Datenbank für das Informationssystem *PaperBase*.
- `task` bestimmt die Klasse, die für das Versenden von E-Mails verantwortlich ist.

Weitere Applikationsobjekte sind im Anhang A auf Seite 243ff abgebildet.

```
   .. Die Klasse StructuredMessage ..

   @author Ralf Schimkat
   @version November 1998

*/ public class StructuredMessage implements java.io.Serializable{

   /** Referenz auf Kopf (Header) der Nachricht */
   public MessageHeader header;
   /** Referenz auf Rumpf (Body) der Nachricht */
   public Object body;

   /** Konstruktor
   @param originator Quelle der Nachricht
   @param target Ziel der Nachricht
   @param messageName Name der Nachricht
   @param i Anzahl der Name-Value-Paare im MessageHeader
   */
   public StructuredMessage(
            String originator,
            String target,
            String messageName,
            int i)
   {
      header = new MessageHeader(originator, target, messageName, i);
      body = null;
   }

   /** Konstruktor
   @param target Ziel der Nachricht
   @param messageName Name der Nachricht
   @param i Anzahl der Name-Value-Paare in Header
   */
   public StructuredMessage(String target, String messageName, int i) {
    this("", target, messageName, i);
   }
}
```

Listing 4.2: Einheitliches Nachrichtenformat in *Respondeo* durch die Klasse `StructuredMessage`.

4.3.2 Einheitliches Nachrichtenformat

Die Applikationsschnittstelle, das Applikationsmodell und der Nachrichtenbus *MBus*, die in Abschnitt 4.2 diskutiert werden, beruhen auf einer einheitlichen, nachrichtenbasierten Kommunikation. Demzufolge ist der Aufbau einer Nachricht von großer Bedeutung und wird in diesem Abschnitt näher beschrieben.
In Listing 4.2 ist ein Auszug der Klasse `StructuredMessage` dargestellt, die das Format der Nachrichten innerhalb von *Respondeo* bestimmt. Folgende Eigenschaften sind charakteristisch für Nachrichten des Typs `StructuredMessage`:

- *Serialisierbarkeit.* Jede Nachricht vom Typ `StructuredMessage` ist serialisierbar in Java. Deswegen implementiert die Klasse `StructuredMessage` das Java-Interface `Serializable`.

- *Metadaten*. Die Metadaten einer Nachricht haben innerhalb des Frameworks einen vorgegebenen, einheitlichen Aufbau (vgl. Listing 4.3[18]). Jede Nachricht vom Typ `StructuredMessage` besteht aus einem Kopf- (Header) und einem Rumpfteil (Body). Der Kopfteil besitzt fixe (`originator`, `target`, `messageName`) und flexibel erweiterbare Attribute (`NameValuePair`):

  - Die Quelle einer Nachricht wird über `originator` festgelegt. Das Ziel einer Nachricht wird über einen symbolischen Namen bestimmt und in `target` abgelegt. Normalerweise wird der Name eines Dienstes, der für die inhaltliche Bearbeitung der Nachricht zuständig ist, als Ziel angegeben. Zudem erhält jede Nachricht einen Namen (`messageName`).
  - Erweiterbare Attribute werden über Name-Wert-Paare spezifiziert (siehe `NameValuePair` in Listing 4.3). Für einfache Nachrichten reicht es beispielweise aus, ausschließlich über Name-Wert-Paare zu kommunizieren. Dabei wird kein Objekt im Rumpfteil mitverschickt. Erweiterbare Attribute werden nicht vom Kernframework *Respondeo*s interpretiert, sondern dienen dazu, zusätzliche Informationen auf der Ebene der Dienste (bzw. Applikationsobjekte) mitzuführen. So werden dienstspezifische Parameter oder Eigenschaften in den erweiterbaren Attributen einer Nachricht vom Typ `StructuredMessage` abgelegt.

- *Dateninhalt*. Der eigentliche Dateninhalt wird im Rumpfteil der Nachricht versendet. Als Dateninhalte kommen dafür alle serialisierbaren, digitalen Objekte in Frage, wie beispielsweise Instanzen der Klasse `String` in Java oder anderer Datenstrukturen. Zudem können Grafiken, Java-Programme[19] oder E-Mails im Rumpfteil einer Nachricht vom Typ `StructuredMessage` enthalten sein. Die Interpretation und die Verarbeitung des Dateninhalts obliegt dem Empfänger einer Nachricht.

4.4 Integrierte Subsysteme und Applikationen

In den folgenden Abschnitten werden einige der in *Respondeo* eingebundenen Informationssysteme bzw. Dienste näher beschrieben, um den vorteilhaften Einsatz von *Respondeo* zu unterstreichen. Charakteristisch für den Großteil der integrierten Dienste ist die generelle Anwendbarkeit und Einsetzbarkeit in einer verteilten

[18] Im Anhang unter Listing A.2 auf Seite 244 findet sich eine detaillierte Darstellung der Klasse `MessageHeader`.

[19] In diesem Zusammenhang sind Java-Programme gemeint, die ihren internen Zustand bzw. ihre Datenstrukturen in serialisierter Form abgelegt haben.

```
/**
   ... Die Klasse MessageHeader ...

   @author Ralf Schimkat
   @version Dezember 1998

*/ public class MessageHeader implements java.io.Serializable{
      /** Quelle der Nachricht */
      public String originator;
      /** Ziel der Nachricht */
      public String target;
      /** Name der Nachricht */
      public String messageName;
      /** Name-Wert-Paare */
      public NameValuePair[] variableHeader;
      /** Client Hostname */
      public String clientHost;

      /** Konstruktor
      @param orginator Quelle der Nachricht
      @param target Ziel der Nachricht
      @param messageName Name der Nachricht
      @param i Anzahl der Name-Value-Paare
      */
      public MessageHeader(String originator,
        String target, String messageName, int i);

      ...
}
```

Listing 4.3: Metadaten einer Nachricht – Die Klasse `MessageHeader`.

Umgebung, dank derer sich auf einfache Weise ein Netzwerk von Diensten und Informationssystemen aufbauen läßt. Typisch für dieses Netzwerk von Diensten ist die uniforme Benutzung von Nachrichten des Typs `StructuredMessage`. Die potentiellen Schwierigkeiten und Probleme mit proprietären Schnittstellen spielen in diesem Netzwerk keine Rolle, da über den in Abschnitt 4.2.2 beschriebenen Nachrichtenbus *MBus* kommuniziert wird.

4.4.1 JDBC – Java Database Connectivity

Die *Java Database Connectivity* (JDBC) [Ree00, Sun02b] Technologie ermöglicht den Zugriff auf relationale Datenbanken unter der Benutzung der Programmiersprache Java. Kennzeichend für JDBC ist die Unabhängigkeit dieser Datenbank-Middleware von konkreten Datenbankherstellern. Generell stellt JDBC ein Verbindungsglied zwischen Java und Datenbanken dar, insofern als JDBC die Datenbankanweisungen bzw. -programme nach Java konvertiert. Der Standard für Datenbanksprachen ist die *Structured Query Language* (SQL) [DD93, Fed93].
Für die Integration in *Respondeo* wurde ein Datenbankdienst entworfen und implementiert, der den Zugriff auf Datenbanken unter Benutzung von JDBC ermöglicht. Der realisierte Datenbankdienst ist durch seine umfassende Anwendbarkeit

gekennzeichnet. Er enthält keine anwendungs- oder datenbankabhängigen Eigenschaften. Erst durch die Initialisierung des Datenbankdienstes unter der Benutzung des Konfigurationsmechanismus (siehe Abschnitt 4.3.1) von *Respondeo* wird der Datenbankdienst an eine konkrete Datenbank gebunden.
Durch die flexible Parametrisierbarkeit des Datenbankdienstes können auf einfache Art und Weise beliebige Datenbanken in einer verteilten Umgebung *uniform* angebunden werden. Dadurch entsteht ein Netz von Mediatoren bzw. Datenbankdiensten, die in *Respondeo* eingebunden sind. Die uniforme Anbindung der Datenbanken bezeichnet dabei die Eigenschaft, daß für den Zugriff auf Datenbanken, die potentiell auf entfernten Rechnern installiert sind, lediglich eine Nachricht vom Typ `StructuredMessage` an den betreffenden Mediator (Datenbankdienst) geschickt werden muß. Es sind weder aufwendige und datenbankspezifische Installationen notwendig, noch müssen bestimmte proprietäre Datenbankschnittstellen oder die Lokation der Datenbank berücksichtigt werden.
Aus Implementierungsgesichtspunkten besteht die Schnittstelle des Datenbankdienstes aus der in Abschnitt 4.2.2 beschriebenen nachrichtenbasierten Kommunikation. Die Nachrichten des Typs `StructuredMessage` werden vom Datenbankdienst verarbeitet und die darin enthaltenen Datenbankanweisungen ausgeführt. Das Ergebnis der Datenbankanfrage wird danach in eine Nachricht des Typs `StructuredMessage` verpackt und an den Absender der Anfrage zurückgeschickt.

4.4.2 E-Mail

E-Mails sind elektronische Nachrichten und basieren auf dem SMTP-Protokoll [Kle01]. Über E-Mails können beliebige digitale Inhalte zwischen Teilnehmern in einem Netzwerk verschickt werden. Typischerweise bildet der Mensch einen solchen Teilnehmer, der seine E-Mails über spezielle E-Mail-Clients (z. B. Microsoft Outlook) liest bzw. an andere Teilnehmer verschickt.
Für die Integration in *Respondeo* wurde ein allgemeiner E-Mail-Dienst[20] realisiert, der im wesentlichen folgende zwei Funktionen erfüllen kann:

- *Verschicken* von Informationen an Benutzer eines Informationssystems. Diese Informationen beziehen sich beispielsweise auf die Aktualität bestimmter Dateninhalte in einem Informationssystem.

- *Empfangen* von neuen Daten- bzw. Dokumenteninhalten, die Benutzer in den aktuellen Datenbestand eines Informationssystems einfügen möchten. Der Benutzer hat die Möglichkeit, über das Versenden von E-Mails auf einfache Art und Weise neue Dateninhalte in ein Informationssystem einzufügen.

[20] Im Abschnitt 4.3.1 werden bei der Beschreibung des Dienstes `DBPaperBaseEmail` einige der Konfigurationsparameter des E-Mail-Dienstes beschrieben.

Aus technischen Erwägungen benutzt der E-Mail-Dienst SMTP anstelle von RMI, um Informationen mit anderen Teilnehmern bzw. Systemkomponenten auszutauschen. Das Standardprotokoll für die Kommunikation mit *Respondeo* und dessen Diensten ist RMI, wie in Abschnitt 4.2 und in Abbildung 4.1 erörtert wird. In dieser Hinsicht stellt der E-Mail-Dienst auch eine Erweiterung der innerhalb von *Respondeo* zur Verfügung stehenden Palette von Kommunikationsprotokollen auf der Netzwerkebene dar.

4.4.3 Integration eines Web-Servers

Jetty ist ein leichtgewichtiger Web-Server für die Generierung von statischen und dynamischen Inhalten im Web. Seine geringe Codegröße von ca. 300 Kbyte ermöglicht die Einbettung in andere Java-Applikationen – wie beispielsweise *Respondeo* – auf einfache Art und Weise.
Durch die Integration von *Jetty*[21] wurde die Anzahl der zur Verfügung stehenden Kommunikationsprotokolle in *Respondeo* um das *Hypertext Transfer Protocol* (HTTP) erweitert. Beliebige Nachrichten können so auch unter Benutzung von HTTP kommuniziert werden.

4.4.4 Routing von Nachrichten

Das Routing von Nachrichten in *Respondeo* bezeichnet den Prozeß, in dem die ankommenden Nachrichten bei einem Applikationsobjekt[22] direkt zu einem weiteren – dem eigentlichen Adressaten – geleitet werden.
Der Routing-Dienst von *Respondeo* kommt vor allem bei Informationssystemen und Diensten zum Einsatz, die durch eine sogenannte *Firewall* geschützt werden. In solchen Systemumgebungen ist der Zugriff auf Dienste bzw. Systemkomponenten nur eingeschränkt möglich. Durch die Benutzung des Routing-Dienstes wird dem Benutzer die Möglichkeit gegeben indirekt auf Dienste, die durch *Respondeo* verwaltet werden, zuzugreifen. Der Zugriff auf Dienste in *Respondeo* ist unabhängig davon, ob die gewünschten Dienste direkt oder indirekt (über den Routing-Dienst) angesprochen werden. Es wird die gleiche Nachricht an den betreffenden Dienst verschickt. Der angeforderte Dienst entscheidet, ob die Nachricht bearbeitet werden kann oder ob sie entsprechend weitergeleitet wird. Dieser Routing-Mechanismus bietet eine große Transparenz für die Benutzung von Diensten in einem Netzwerk.

[21] Mittlerweile ist *Jetty* auch ein wesentlicher Bestandteil von Applikationsservern wie *JBoss* (siehe http://www.jboss.org) oder *JOnAS* (siehe auch http://www.objectweb.org/jonas/index.html), so daß sich die Vorteile von leichtgewichtigen Komponenten auch zunehmend im kommerziellen Umfeld durchzusetzen beginnen.

[22] Der Autor weist darauf hin, daß ein Applikationsobjekt die gewählte Abstraktion im Framework von *Respondeo* ist, um die integrierten Dienste zu repräsentieren. Wenn in dieser Hinsicht von einem Applikationsobjekt die Rede ist, so ist immer der entsprechende, in *Respondeo* integrierte Dienst gemeint.

4.4.5 *FireStorm* – strukturierte Modellsuche

FireStorm [MS01, Mül01] ist ein System zur Verwaltung und zum strukturierten Suchen von Modellen wie z. B. mathematische, UML- [Obj00b] oder ER-Modelle. Die Modelle werden dabei durch spezielle 3-stufige Graphen repräsentiert. Für die Suche kommen verschiedene Algorithmen zur Bestimmung von Graphenähnlichkeiten zum Einsatz, die parametrisiert auf die speziellen Problemdomänen angepaßt sind.

Aus Systemsicht ist *FireStorm* ein mehrstufiges Informationssystem. *Respondeo* bildet den Applikationsserver, der auf der mittleren Ebene die Zugriffe auf den *FireStorm*-Server verwaltet. Jede Problemdomäne – in *FireStorm* auf einen speziellen 3-stufigen Graphen abgebildet – wird als eigenständiger Dienst in *Respondeo* integriert. Da *FireStorm* in einer Mehrbenutzerumgebung installiert ist, benutzen alle *FireStorm*-Dienste die parallele Ausführungsmöglichkeit von Anfragen durch den Dienst `AppLogicPool` (siehe Abschnitt 4.2.1).

4.4.6 *PaperBase* – Verwaltung von virtuellen Arbeitsräumen im Web

PaperBase[23] ist ein Informationssystem zur Erstellung und Verwaltung von sogenannten *virtuellen Arbeitsräumen* im Web. Diese Arbeitsräume lassen sich individuell anpassen. Zudem können Teile des Arbeitsraums mit anderen Benutzern geteilt werden, so daß kooperative Arbeitswelten entstehen. Im Abschnitt 10.1 werden der Aufbau und die Eigenschaften des Informationssystems *PaperBase* ausführlich erläutert.

PaperBase ist ein mehrstufiges Client/Server-Informationssystem mit einer relationalen Datenbank am Back-End. Für die Zugriffe auf die relationale Datenbank im Informationssystem *PaperBase* wird der Datenbankdienst (siehe Abschnitt 4.4.1) benutzt. Der Datenbankdienst ist die wichtigste Systemkomponente für das Suchen von Dokumenten, Einfügen neuer Dokumente und Aktualisieren der Informationen in der *PaperBase*.

Zum Empfangen bzw. Versenden von E-Mails wird der E-Mail-Dienst (siehe Abschnitt 4.4.2) benutzt. Der Versand von E-Mails richtet sich vor allem an die kooperativen Eigenschaften der *PaperBase*. Sobald beispielsweise ein neues Dokument oder ein Kommentar eines Benutzers zur Verfügung steht, werden die Mitglieder der betroffenen kooperativen Arbeitswelt davon per E-Mail unterrichtet. Die Fähigkeit der *PaperBase*, E-Mails zu empfangen, wird dazu benutzt, Dokumente per E-Mail in die *PaperBase* einzuspeisen. Die neuen Dokumente werden an eine bestimmte E-Mail-Adresse gesendet, wo sie vom E-Mail-Dienst abgeholt werden. Nachdem die E-Mails heruntergeladen worden sind, werden die enthaltenen Dokumente in die Datenbank bzw. in das Informationssystem *PaperBase* eingefügt.

[23] Siehe auch die Homepage der *PaperBase* unter http://www-sr.informatik.uni-tuebingen.de/~schimkat/pb.

4.4.7 *Specto* und *Tempto*

Specto [SHKK00] und *Tempto* [SSDKK00] sind mehrstufige Client/Server-Informationssysteme, die jeweils XML-basierte Datenbasen verwalten.
Specto ist ein System zum plattformunabhängigen Monitoring von Applikationen in einer verteilten Umgebung. Die Informationen über die Applikationszustände, welche die Basis für das Monitoring bilden, werden in der *Specto Markup Language* (*SpectoML*)[24] festgehalten. Eine ausführliche Diskussion des Monitoringsystems *Specto* erfolgt in Kapitel 7.
Tempto ist ein Framework zum Testen von Applikationen. Zudem bietet es eine Infrastruktur zur Verwaltung von Testdokumenten, die auf der Basis der *SpectoML* erstellt werden. Das Informationssystem *Tempto* wird in Abschnitt 8.1 genauer erörtert.
Sowohl *Specto* als auch *Tempto* benutzen *Respondeo*, um auf die XML-basierten Dokumente bzw. Datenbasen zuzugreifen und die darin enthaltenen Informationen herauszufiltern. *Respondeo* bietet für diese beiden verteilten Informationssysteme einen Dienst an, der sich vor allem durch den *transparenten* und *parallelen* Zugriff auf die Datenbasen auszeichnet. Die Probleme und Gefahren, die sich durch die große Anzahl der XML-basierten Datenbasen und deren räumliche Verteilung ergeben, werden durch *Respondeo* überwunden. Es wird ein Netzwerk von Datenbasen geschaffen, auf die einheitlich und transparent zugegriffen wird.

4.4.8 *Progress* – Integrationsplattform für mathematische Dienste

Progress [Bec95] ist eine Integrationsplattform zur Generierung und Benutzung von mathematischen Diensten im Internet. Es enthält eine Skript-Sprache, welche die Definition und die Verarbeitung komplexer Datenstrukturen (z. B. Listen, Tupel) erlaubt. Der Zugriff auf die Integrationsplattform von *Progress* ist über eine Client/Server-Architektur realisiert.
Um Teilkomponenten von *Progress* in *Respondeo* zu integrieren, wurde ein spezielles Applikationsobjekt entworfen[25]. Dieses agiert als Mediator für Zugriffe auf *Progress*, indem es die Anfragen, die in Nachrichten des Typs `StructuredMessage` enthalten sind, transformiert und danach den *Progress*-Server kontaktiert. Für Details der Integration von *Progress* in *Respondeo* sei auf Schimkat et al. [SMKK00] verwiesen.

4.4.9 *JIma* – Java ImageMaster

JIma [SKK99] bezeichnet das Client-Framework des Dokumentenverwaltungssystems *ImageMaster* der Fa. T-Systems. ImageMaster besteht aus einer mehrstufigen Client/Server-Architektur, wobei ein objektorientierter Applikationsserver, der

[24] Der Aufbau und die Eigenschaften der *SpectoML* werden in Kapitel 6 diskutiert.

[25] Die Integration von *Progress* in *Respondeo* beschränkt sich auf bestimmte Teilkomponenten von *Progress* zum Suchen von mathematischen Modellen.

in der Programmiersprache C++ implementiert ist, und das Client-Framework *JIma* die beiden Kernkomponenten bilden. Die Haupteinsatzgebiete von ImageMaster liegen im Management von sehr großen Dokumentbeständen wie beispielsweise der Archivierung von Telefonabrechnungen verschiedener deutscher Mobilfunkfirmen.
Der Einsatz des Frameworks *JIma* auf Client-Seite ermöglicht die einfache Generierung von unterschiedlichen, an den konkreten Kundenbedürfnissen orientierten grafischen Benutzeroberflächen[26]. *JIma* ist komplett in Java implementiert.
Respondeo und dessen Routing-Dienst wurden eingesetzt, um Anfragen der Java-basierten Clients an die entsprechenden – in C++ implementierten – Server innerhalb des ImageMasters weiterzuleiten. Zudem wurde ein Dienst generiert, der die Java-basierten Anfragen in eine unternehmensweit einheitliche und plattformunabhängige Darstellung transformiert. Diese dient als Basis zur Kommunikation zwischen Systemkomponenten, die in unterschiedlichen Programmiersprachen implementiert sind.

4.4.10 *VISU* – automatisch erzeugte Schaubilder

VISU ist ein kleines, mehrstufiges Informationssystem zur dynamischen Generierung von grafischen Abbildungen im Web wie z. B. XY-Diagrammen. Die zugrunde liegenden Dateninhalte sind in XML-Dokumenten hinterlegt, die auf *SpectoML* (siehe Abschnitt 6) beruhen. Die dynamisch generierten Abbildungen auf Basis von XML-basierten Datenbasen kommen vor allem bei der Visualisierung von Prozessen in sehr dynamischen Umgebungen und Applikationen zum Einsatz, wie beispielsweise bei einer verteilten und autonomen Berechnung eines irregulären Problems (siehe Abschnitt 8.4). *VISU* liefert in solchen komplexen Umgebungen wertvolle Zusatzinformationen, die für den Gesamtablauf von Bedeutung sind.
Ähnlich wie in Abschnitt 4.4.7 erläutert stellt *Respondeo* einen Dienst für den Zugriff auf die Datenbasen und einfache Algorithmen für die statistische Auswertung der Dateninhalte zur Verfügung. Der Zugriff im Web erfolgt ausschließlich über die HTTP-Schnittstelle *Respondeo*s. Auf der Seite des Clients werden die grafischen Abbildungen als JPG-Grafik direkt im Browser visualisiert.

4.5 Einordnung in die Taxonomie

Die in Kapitel 3 vorgestellte Taxonomie bezieht sich auf die Einordnung und Klassifizierung von Informationssystemen. Da *Respondeo* für sich kein komplettes Informationssystem darstellt, sondern nur wichtige Infrastrukturen und Funktionen für die Realisierung und Umsetzung von Informationssystemen übernimmt, ist eine Einordnung in die Taxonomie nur schwer möglich.
Versucht man dennoch, eine Klassifizierung anhand der Taxonomie und deren drei wesentlichen Dimensionen – Benutzersicht, Dokumentenbegriff und Technologie–

[26] Vgl. Abbildung A.1 im Anhang.

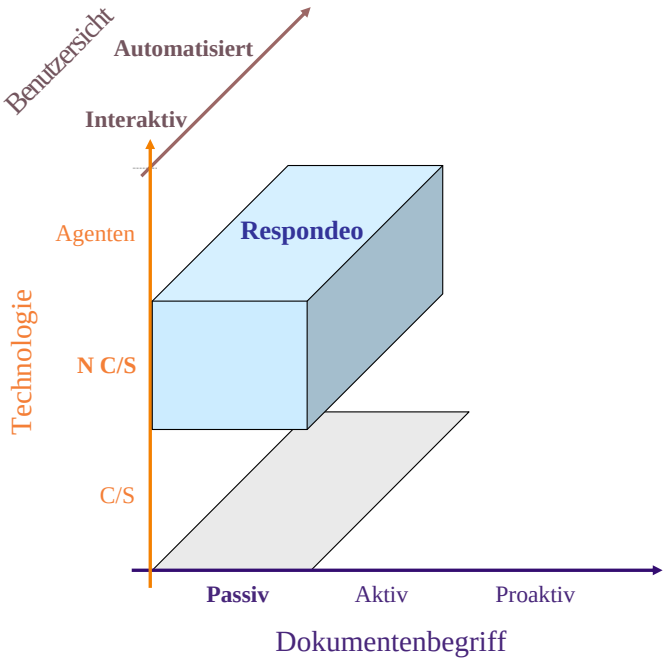


Abbildung 4.5: Prinzipielle Einordnung *Respondeo*s in die Taxonomie.

vorzunehmen, so sind *Respondeo* und dessen Dienste aus *technologischen* Gesichtspunkten im Bereich der mehrstufigen Client/Server-Informationssysteme anzusiedeln. *Respondeo* übernimmt dabei die Rolle eines Mediators für die verschiedenen in *Respondeo* integrierten Dienste[27].

In Abbildung 4.5 bildet die Einordnung *Respondeo*s in die Taxonomie einen Quader, der sich über die beiden Merkmale der interaktiven und automatisierten *Benutzersicht* erstreckt. Innerhalb von *Respondeo* wird nicht zwischen menschlichen und digitalen Benutzern unterschieden. Generell bindet die Architektur von *Respondeo* die Benutzersicht an seine Applikationsschnittstelle (siehe Abbildung 4.1). Da dort eine uniforme, nachrichtenbasierte Schnittstelle etabliert ist, die sowohl für klassische (interaktive) Benutzeranfragen als auch für (automatisierte) Dienstanfragen am Back-End Gültigkeit besitzt, werden beide Benutzersichten unterstützt[28].

[27] In Abschnitt 2.2 werden Mediatoren eingeführt und deren Eigenschaften diskutiert. In Abschnitt 2.3 werden mehrstufige Client/Server-Informationssysteme sowie u. a. die Rolle von Applikationsservern erörtert.

[28] In Abhängigkeit von den Eigenschaften des Informationssystems, in dem *Respondeo* zum Einsatz kommt, werden verschiedene Benutzersichten unterstützt: So werden beispielsweise innerhalb des ImageMasters (siehe Abschnitt 4.4.9) nur interaktive Anfragen, die vom Benutzer gesendet werden, realisiert. Dagegen treten in der *PaperBase* (siehe Abschnitt 4.4.6 und 10.1) auch automatisierte Dienstanfragen in Erscheinung, die nicht originär von einem (menschlichen) Benutzer initiiert worden sind.

Hinsichtlich des *Dokumentenbegriffs* unterstützt *Respondeo* ausschließlich passive Dokumente, wobei eine exakte Klassifizierung in dieser Dimension nur schwer möglich ist. Dokumente bzw. der Dokumentenbegriff spielen bei *Respondeo* eine untergeordnete Rolle. Dokumente treten erst in den Informationssystemen, für die *Respondeo* Infrastrukturen zur Verfügung stellt, in Erscheinung.

4.6 Vergleichbare Ansätze

In den folgenden Abschnitten werden ähnliche und verwandte Projekte beschrieben. Dabei stehen vor allem große und teilweise kommerziell verfügbare Softwareframeworks im Vordergrund der Betrachtung.

4.6.1 CORBA und EJB

Enterprise Java Beans (EJB) [MH99, Sun02a] und *Common Object Request Broker Architecture* (CORBA) [Obj00a, MM97] sind die beiden bekanntesten Frameworks für die Realisierung von großen und meist kommerziell angewendeten Informationssystemen.

EJB Die in den letzten Jahren entstandene Spezifikation der EJB unterstützt primär den Entwurf von Softwarekomponenten in Java auf der Server-Seite von mehrstufigen Client/Server-Informationssystemen . Dabei enthalten die Komponenten die Applikationslogik eines Informationssystems. EJB stellt nun verschiedene systemnahe Dienste zur Verfügung, um den Entwickler des Informationssystems zu entlasten. Das Ziel ist, daß sich ein Entwickler auf die Implementierung der Applikationslogik und weniger auf die Details eines verteilten Transaktionsmanagements oder eines ausgefeilten Sicherheitskonzepts konzentriert.
Durch die Fokussierung der EJB auf mehrstufige Client/Server-Informationssysteme bieten EJB die Möglichkeit, sehr schlanke Thin-Clients (siehe Abschnitt 2.3.1) zu generieren, da die Applikationslogik konsequent auf Server-Seite verwaltet wird. Dies ist besonders wichtig für Clients, die auf kleinen Geräten wie z. B. PDAs (Personal Digital Assistant) operieren.
Generell ist die Bedeutung des Mediatorkonzepts (siehe Abschnitt 2.2) in den EJB hervorzuheben: Der Zugriff auf ein Bean auf der Server-Seite erfolgt nicht direkt, sondern durch einen speziellen Mediator, den sogenannten *Bean Container*. Dieser verwaltet alle Zugriffe auf die betreffende Ressource. Dadurch wird eine einheitliche, logische Ebene geschaffen, welche die Etablierung eines Komponentenmodells – wie das der EJB – erst ermöglicht. Ohne diese zusätzliche Ebene würden applikationsspezifische Details zumeist im Vordergrund stehen und ein einheitliches Applikationsmodell verhindern.
Respondeo unterstützt ebenfalls den Entwurf und die Realisierung von mehrstufigen Client/Server-Informationssystemen. Das Mediatorkonzept ist bei der Integration von Diensten in *Respondeo* ebenso von großer Bedeutung, da dadurch

flexible und dynamisch anpaßbare Informationssysteme entstehen. Im Gegensatz zum komplexen Applikationsmodell der EJB unterstützt *Respondeo* ein weniger mächtiges, aber dafür einfacheres Applikationsmodell.

CORBA Für die Realisierung von verteilten Informationssystemen stellt CORBA eine Spezifikation eines programmiersprachenunabhängigen Objektmodells dar. Im Mittelpunkt steht ein Objektmodell, das nicht an eine bestimmte Programmiersprache gebunden ist.
Objekte in CORBA stellen Funktionen und Dienste zur Verfügung. Die Beschreibung der Dienste erfolgt auf der Schnittstellenebene, d. h. die syntaktischen Details der Schnittstelle – wie z. B. die Typen und die Reihenfolge der Übergabeparameter – werden einheitlich in der *Interface Definition Language* (IDL) definiert. Die semantischen Bedeutungen bzw. applikationsabhängigen Eigenschaften werden in CORBA nur rudimentär berücksichtigt.
CORBA fokussiert im Unterschied zu den EJB oder *Respondeo* auf die Ebene der Objekte. Bei der Realisierung von Informationssystemen stehen die entfernten Aufrufe der Funktionen und Methoden eines Objekts im Mittelpunkt. Der entfernte, objektbasierte Methodenaufruf orientiert sich dabei strikt am Prinzip des entfernten Methodenaufrufs (RPC) [BN84].

4.6.2 Web-Services, XML-RPC, SOAP, WSDL und UDDI

Generell sind Web-Services Dienste, die im Web angeboten werden. Das W3C definiert die Eigenschaften eines Web-Services folgendermaßen:

> Ein Web-Service ist eine Applikation, die mittels einer URI (*Uniform Resource Identifie*r) [BLFM98] identifiziert und deren Schnittstellen mittels XML definiert, beschrieben und ermittelt werden. Ein Web-Service unterstützt die direkte Kommunikation zwischen Softwarekomponenten. Dabei werden XML-basierte Nachrichten über standardisierte Kommunikationsprotokolle des Internets ausgetauscht [AGB+02][29]

Web-Services und die Benutzer von Web-Services sind typischerweise Applikationen, d. h. im Vordergrund steht eine von Automatismen geprägte Interaktion zwischen Softwarekomponenten[30]. Applikationen werden durch die Aggregation und durch den Zusammenschluß mehrerer Web-Services dynamisch generiert.
In technischer Hinsicht benutzen Web-Services mehrere, XML-basierte und zum Standard erklärte Spezifikationen des Webs, die im folgenden kurz erläutert werden:

[29] Siehe auch die Homepage des W3C unter http://www.w3.org/2002/ws/ für weitere Informationen zu Web-Services.

[30] Siehe auch Abschnitt 3.1.2: Diskussion von automatisierten Interaktionen und deren Einordnung in die Taxonomie für Informationssysteme.

- *XML-RPC* [Use02] stellt eine Infrastruktur zur Verfügung, die es Softwareprozessen ermöglicht, Prozeduren über das Internet aufzurufen. In dieser Hinsicht entspricht XML-RPC einem entfernten Prozeduraufruf [BN84], der HTTP als Transportschicht und XML als Datenkodierungsformat benutzt. Grundsätzlich besteht XML-RPC aus einer Spezifikation, für die eine Menge von Implementierungen existiert.

  XML-RPC war der Vorgänger des sogenannten *Simple Object Access Protocol* (SOAP) [BEK+00]. SOAP wird unter dem W3C standardisiert. SOAP ist – ähnlich wie XML-RPC – ein leichtgewichtiges Kommunikationsprotokoll für den Austausch von Informationen in einer verteilten Umgebung. SOAP definiert weder ein Programmiermodell für Applikationen noch werden irgendwelche Implementierungen vorgegeben. Es ist das Hauptziel von SOAP, einen einfachen Mechanismus zur Verfügung zu stellen, damit unterschiedliche Bedeutungsinhalte in XML beschrieben bzw. kommuniziert werden können.

- *Web Service Description Language* (WSDL) [CCMW01] ist eine auf XML basierende Sprache zur Beschreibung von Netzwerkdiensten im Web. Dabei werden Netzwerkdienste als Kommunikationsendpunkte im Web betrachtet, die durch den Austausch von Nachrichten miteinander kommunizieren. Generell stellt WSDL eine Infrastruktur zur Beschreibung der Eigenschaften und Funktionalitäten von Netzwerkdiensten zur Verfügung. Durch WSDL sollen Softwarekomponenten in die Lage versetzt werden, die Eigenschaften von Netzwerkdiensten in automatisierter Form zu erfragen. WSDL soll die Basis für eine direkte und automatisierte Interaktion zwischen Netzwerkdiensten im Web bilden.

- *Universal Description, Discovery and Integration* (UDDI) [BHH+02] ist ein im Jahre 2000 gestartetes Projekt mit dem Ziel, eine Spezifikation für ein globales Verzeichnis von web-basierten Netzwerkdiensten zu definieren.

XML, SOAP, WSDL, UDDI und HTTP bilden die Basis für ein Netzwerk von Web-Services, in dem der Zugriff auf entfernte Dienste ausschließlich über standardisierte und herstellerunabhängige Infrastrukturen erfolgen soll.
Die Größe des Projekts und die Vielzahl der partizipierenden Firmen in den Konsortien lassen eine definitive Beurteilung des Erfolgs von Web-Services nur schwer zu. Bis die Web-Services und die beteiligten Spezifikationen den vielerorts prognostizierten Reifegrad erreichen werden, dürfte es noch einige Zeit dauern. Trotzdem sind Web-Services ein vielversprechender Ansatz, um die Benutzung und den Einsatz von Softwarekomponenten im Web vehement zu beeinflussen.
Respondeo und dessen Architektur unterstützen ebenfalls den netzwerkartigen Aufbau von Diensten, wie es in den vorhergehenden Abschnitten erläutert wurde. Die konzeptionelle Einfachheit und der Aspekt der Leichtgewichtigkeit *Respondeo*s sind die wesentlichen Unterschiede zu den Web-Services. Beim Entwurf von *Respondeo* standen die komplexen Aspekte von herstellerunabhängigen

Spezifikationen nicht im Mittelpunkt. Hinsichtlich der Implementierung ist eine Anbindung an die Infrastrukturen von Web-Services denkbar und die Benutzung von XML als Datentransportformat kann ein Bestandteil der zukünftigen Arbeit sein.
In Schimkat et al. [SNB00] wurde bereits im Jahr 2000 die wesentlichen Eigenschaften eines dienstzentrierten Netzwerks im Web diskutiert. Eine wesentliche Systemkomponente bei der vorgestellten Dienstarchitektur war *Respondeo*.
Aus historischer Sicht läßt sich abschließend festhalten, daß im Projekt *Respondeo* bereits im Jahre 1998 mit dem Aufbau eines Netzwerks von dienstzentrierten Softwarekomponenten begonnen wurde.

Kapitel 5

Okeanos – ein Framework für mobile Agenten

In diesem Kapitel wird der Entwurf und die Realisierung des Agentenframeworks *Okeanos* [1] erörtert. *Okeanos* ermöglicht die Gestaltung von mobilen und stationären Agenten in Java, die sich u. a. durch regelbasierte Eigenschaften auszeichnen. Zudem stellt *Okeanos* eine Reihe von Infrastrukturen zur Verfügung, die von Agenten und anderen Softwarekomponenten benutzt werden, um sogenannte *Multiagentensysteme* [Wei99] zu realisieren. In [SBS+00, SFK01a, HS01] wird auf Basis von *Okeanos* ein Multiagentensystem im Bereich des *Symbolischen Rechnens* geschaffen, das die Anwendbarkeit von Agenten in verteilten, dynamischen Systemumgebungen unterstreicht.
Die Definition eines Agenten in diesem Kapitel stützt sich auf Wooldridge [Woo97] und Jennings [Jen00]:

> An agent is an encapsulated computer system that is situated in some environment and that is capable of flexible, autonomous action in that environment in order to meet its design objectives [Woo97].

Diese Definition beruht auf der einführenden Diskussion von Agenten in Abschnitt 3.3.1 im Zusammenhang mit der Erörterung technischer Aspekte der in Kapitel 3 entwickelten Taxonomie. Kapitel 3 befaßt sich zudem mit einem Vergleich zwischen den Objekten im objektorientierten Paradigma und den Agenten. Ferner werden in Abschnitt 3.3.3 Agenten als spezielle Mediatoren [Wie92a] (vgl. Abschnitt 2.2) interpretiert und deren Eigenschaften für die Gestaltung von Informationssystemen erläutert.

[1] *Okeanos* ist aus dem griechischen *Oceanus* abgeleitet und steht für „den Gott des großen Flusses Ozean, der kreisförmig die gesamte Erde umgibt“ [THE02]. *Okeanos* war der Ursprung allen frischen Wassers, der Flüsse und des Regens. Mit dem Namen soll zum Ausdruck gebracht werden, daß *Okeanos* eine Infrastruktur für permanent aktive Agenten zur Verfügung stellt. Derartige fortwährende („fließende“) Aktivitäten sind „essentiell“ für dynamische und verteilte Softwaresysteme.

Im vorliegenden Kapitel stehen die Aspekte des Entwurfs eines Softwareframeworks für Agenten im Vordergrund. Zudem werden die konkreten Eigenschaften von *Okeanos*-Agenten wie z. B. die *Mobilität der Agenten* und die *unterschiedlichen Programmiermodelle*, die von Agenten in *Okeanos* unterstützt werden, vorgestellt. Einen weiteren Schwerpunkt dieses Kapitels bildet die Mobilität von Agenten und deren allgemeinere Form der *Software-Mobilität*. Der Einsatz von *Okeanos*-Agenten ist aber nicht auf mobile Agenten beschränkt. Die in *Okeanos* entwickelten Agenten können auch ausschließlich stationäre Eigenschaften besitzen. In dieser Hinsicht bietet das Framework *Okeanos* unterschiedliche Arten von Agenten an, die mobile Eigenschaften enthalten können, aber nicht müssen.
Dieses Kapitel über das Agentenframework *Okeanos* ist wie folgt aufgebaut: Abschnitt 5.1 gibt eine Einführung in die Konzepte der Software-Mobilität. Generell werden unterschiedliche Entwurfsparadigmen für verteilte Informationssysteme erörtert, die alle eine bestimmte Form der Software-Mobilität benutzen. In Abschnitt 5.2 werden die Ziele beim Entwurf des Softwareframeworks von *Okeanos* beschrieben. Neben der Möglichkeit zur Realisierung von mobilen Agenten, die für eine bestimmte Form der Software-Mobilität stehen, werden die Offenheit des Gesamtsystems und die Bereitstellung verschiedener Programmiermodelle genauer betrachtet. Abschnitt 5.3 schildert die spezifischen Eigenschaften der Architektur und der Implementierung. Schließlich wird in Abschnitt 5.4 die Einordnung von *Okeanos* in die in Kapitel 3 erörterte Taxonomie diskutiert.

5.1 Einführung in die Software-Mobilität

Die Entwicklung und Realisierung komplexer und verteilter Informationssysteme erfordert ein hohes Maß an Flexibilität, Erweiterbarkeit und Dynamik. Der hohe Durchdringungsgrad und die große Anzahl unterschiedlicher Benutzer in verteilten Informationssystemen erfordern zudem eine große Anpaß- und Erweiterbarkeit der angebotenen Dienste. Der Zugriff auf die Dienste und deren Funktionalität sollte in Abhängigkeit von den allgemeinen Präferenzen und speziellen Anforderungen eines jeweiligen Benutzers erfolgen.
Klassische Ansätze für die Bewältigung der Anforderungen basieren auf dem Client/Server-Prinzip, in dem der (entfernte) Server eine Menge von Diensten zur Verfügung stellt, die über ein Netzwerk transparent aufgerufen und benutzt werden. In Systemen wie beispielsweise CORBA [Obj00a], die auf dem Prinzip des entfernten Methodenaufrufs (Remote Procedure Call – RPC [BN84]) basieren, wird die Lokation der beteiligten Systemkomponenten als Implementierungsdetail betrachtet. Im Vordergrund steht der transparente Zugriff auf entfernte Dienste und Ressourcen. Die Lokationstransparenz von Daten, Diensten oder Servern ist ein adäquates Entwurfsziel, sofern das zu realisierende Informationssystem aus homogenen und statischen Teilkomponenten besteht. In einem solchen System sind die Interaktionsszenarien und Kommunikationsflüsse vorhersehbar, so daß das Verhalten des Gesamtsystems absehbar ist.

Innerhalb großer, verteilter und vor allem heterogener, dynamischer Informationssysteme ist die Vorhersehbarkeit von Ereignissen und folglich das Verhalten des Gesamtsystems nicht mehr ohne weiteres gegeben [Weg97][2]. Zudem sind die partizipierenden Dienste in einem solchen dynamischen und verteilten Informationssystem als selbständige und autarke Komponenten nicht in ein restriktives und rigides Gesamtsystem zu packen. Dabei würde die (lokale) Autonomie des einzelnen Dienstes zu stark eingeschränkt. Es ist das Ziel, die Autonomie der einzelnen Systemkomponenten beizubehalten und durch geeignete Kooperationsmöglichkeiten ein adäquates Gesamtsystem aus (lose) miteinander verbundenen und kooperierenden Teilkomponenten zu gestalten.
Waldo et al. [WWWK94] heben hervor, daß es unter bestimmten Bedingungen erstrebenswert ist, die Lokation von Diensten und Systemkomponenten *explizit* zu berücksichtigen. Das Konzept von mobilen Softwareeinheiten oder der *Software-Mobilität* stellt dem Entwicklungsprozeß von verteilten Informationssystemen erweiterte Möglichkeiten in bezug auf das eher starre Client/Server-Prinzip zur Verfügung. Die Erweiterung bezieht sich dabei auf die Möglichkeiten, die Lokation von Systemkomponenten explizit in die Entwurfs- und Implementierungsentscheidungen einzubinden.
Carzaniga et al. [CPV97] definieren Software-Mobilität wie folgt[3]:

> code mobility is the capability to dynamically change the bindings between code fragments and the location where they are executed.

Gemäß Carzaniga et al. bezeichnet die Software-Mobilität die Möglichkeit, *dynamisch* zur Laufzeit sowohl die *Bindung* zwischen Softwarekomponenten als auch deren physische *Lokation* innerhalb eines verteilten Systems neu zusammenzustellen.

5.1.1 Klassifizierung von Software-Mobilität

Fugetta et al. [FPV98] stellen eine Klassifizierung von verschiedenen Vorgehensweisen für die Benutzung von Software-Mobilität beim Entwurf von Applikationen und deren Komponenten in verteilten Umgebungen vor. In der folgenden Auflistung werden die Abstraktionen[4] dargelegt, die für die Klassifizierung in Abbildung 5.1 maßgeblich sind:

- *Code-Komponenten – Code*. Eine Code-Komponente beinhaltet das Wissen um die Ausführung von bestimmten Programmen. Im Zusammenhang

[2] Vgl. Abschnitt 2.1.2.2.

[3] Software-Mobilität ist die in dieser Arbeit gewählte Übersetzung aus dem Englischen für *code mobility*.

[4] Eine *Abstraktion* im Zusammenhang mit der Klassifizierung von Software-Mobilität repräsentiert eine allgemeine Komponente in der Software-Architektur eines verteilten Systems. Anhand der vorgestellten Abstraktionen sollen die grundsätzlichen Eigenschaften und Einsatzmöglichkeiten der Software-Mobilität im Zusammenhang mit der Realisierung von verteilten Informationssystemen verdeutlicht werden.

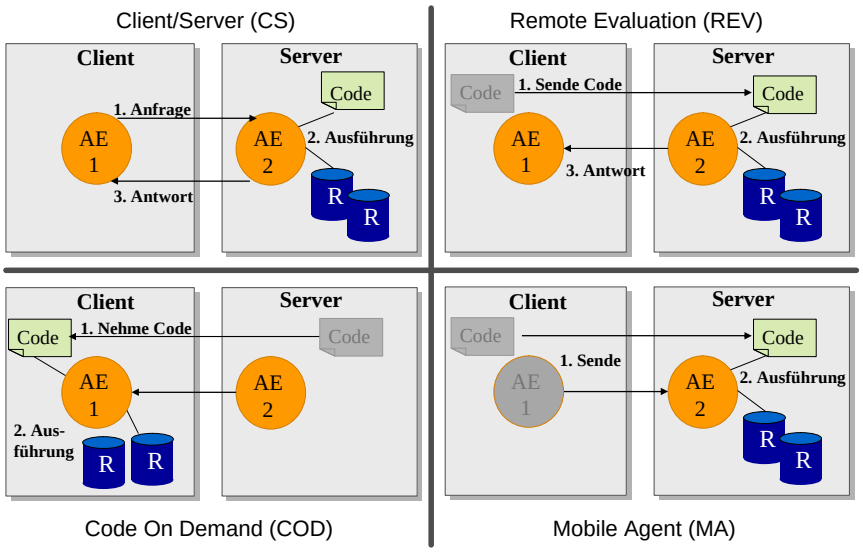


Abbildung 5.1: Die Klassifizierung der Software-Mobilität basierend auf Fugetta et al. [FPV98].

mit dem Entwurf von Informationssystemen ist eine Code-Komponente der Wissensträger für die Verarbeitung der Applikationslogik.

- *Ausführungseinheiten – AE.* Eine Ausführungseinheit ist in der Lage, Programme und Komponenten auszuführen. Sie stellt eine aktive Einheit dar.

- *Ressourcen – R.* Ressourcen repräsentieren Datenbasen und Dienste, die zur Erfüllung einer Dienstanfrage benötigt werden.

- *Ausführungsumgebungen – Client/Server.* Ausführungsumgebungen (AU) sind Orte, an denen Code-Komponenten, Ausführungseinheiten und Ressourcen installiert sind. AUs bilden die Grundlage für die Ausführungseinheiten. Eine AU entspricht der intuitiven Vorstellung von einer Lokation, die durch räumliche Eigenschaften definiert ist.

- *Interaktionen.* Eine Interaktion ist ein Ereignis zwischen den Komponenten. Dabei werden Nachrichten zwischen Komponenten oder Ausführungseinheiten ausgetauscht, die miteinander interagieren.

Die folgende Annahme bildet die Grundlage für die Klassifizierung (siehe Abbildung 5.1) verschiedener Vorgehensweisen bei der Ausnutzung der Software-Mobilität: Eine Ausführungseinheit `AE 1`, die sich in der Ausführungsumgebung `Client` befindet, benötigt das Ergebnis eines bestimmten Dienstes. Für die erfolgreiche Abwicklung der Dienstanfrage ist eine weitere, nicht lokale Aus-

führungsumgebung `Server` involviert[5].

Client/Server – C/S Bei *C/S*[6] bietet eine Ausführungseinheit `AE 2` in der Ausführungsumgebung `Server` eine Menge von Diensten an. Für eine erfolgreiche Ausführung der Dienste sind die Ressourcen `R` und das Wissen `Code` um die korrekte Bearbeitung notwendig. Beim C/S-Vorgehen sind sowohl `R` als auch `Code` auf dem `Server` vorhanden.

Die Ausführungseinheit `AE 1` in der Ausführungsumgebung `Client` initiiert nun eine Dienstanfrage, in der eine Interaktion mit `AE 2` vollzogen wird. Daraufhin führt `AE 2` den angeforderten Dienst aus, indem die entsprechenden Wissensträger `Code` und Ressourcen `R` aktiviert werden. Das Ergebnis der Anfrage wird an `AE 1` zurückgesendet.

Obwohl die Aspekte der Software-Mobilität beim klassischen C/S-Vorgehen eine untergeordnete Rolle spielen, werden sie aus Gründen der Übersicht und der Vollständigkeit hier aufgeführt.

Remote Evaluation – REV Bei der Vorgehensweise *REV* befindet sich der Wissensträger (`Code`) für die Ausführung des Dienstes auf der Seite des `Clients`. Allerdings fehlen auf der Client-Seite die für eine erfolgreiche Bearbeitung des Dienstes notwendigen Ressourcen `R`. Diese befinden sich in einer anderen (entfernten) Ausführungsumgebung – dem `Server` (siehe Abbildung 5.1 rechts oben).

Für eine erfolgreiche Verarbeitung der Dienstanfrage wird die Code-Komponente – mit dem Wissen um die korrekte Verarbeitung der Dienstanfrage – von der Ausführungseinheit `AE 1` des Clients zu `AE 2` auf dem Server übertragen. `AE 2` bringt die Code-Komponente zur Ausführung, indem die (lokalen) Ressourcen auf der Server-Seite genutzt werden. Schließlich sendet `AE 2` die Antwort an `AE 1` zurück.

Code On Demand – COD Bei *COD* ist die Ausführungseinheit `AE 1` in der Ausführungsumgebung `Client` bereits in der Lage, auf die Ressourcen `R` zuzugreifen. Allerdings fehlt `AE 1` das Wissen (`Code`) um die korrekte Bearbeitung. Da `Code` in der (entfernten) Ausführungsumgebung `Server` zur Verfügung steht, wird eine Interaktion zwischen `AE 1` und `AE 2` initiiert. Dabei wird schließlich der Wissensträger `Code` vom `Server` zum `Client` verschickt, so daß `AE 1` die Anfrage beantworten kann.

Ein typisches Beispiel für COD sind die sogenannten Java-Applets [AGH00], die in HTML-Seiten eingebettet sind. Auf Anfrage werden dynamisch die Code-Komponenten in die Ausführungsumgebung `Client` geladen und ausgeführt. Gemäß der Klassifizierung von aktiven Dokumenten

[5] Vgl. Kapitel 2.3.1 über die Diskussion verschiedener Client/Server-Informationssysteme und die Einleitung in Kapitel 4 über den Applikationsserver *Respondeo*.

[6] Siehe auch Abschnitt 2.3.1 über eine Diskussion der Eigenschaften von Client/Server-Informationssystemen.

(vgl. Abschnitt 3.2.3 und Abbildung 3.5) wäre COD eine Realisierungsmöglichkeit für aktive Dokumente des Typs B.

Mobile Agent – MA Bei *MA* befinden sich der Wissensträger `Code` und die Ausführungseinheit `AE 1` auf dem `Client`. Jedoch existieren die für die Dienstanfrage nötigen Ressourcen `R` nur in der (entfernten) Ausführungsumgebung `Server`. Bei der MA-Vorgehensweise migriert die Ausführungseinheit `AE 1`, die den Wissensträger `Code` beinhaltet, vom `Client` zum (entfernten) `Server`. Nach der Migration vollzieht `AE 1` die Beendigung der Dienstanfrage, in dem die (lokal) vorhandenen Ressourcen `R` in der Ausführungsumgebung `Server` benutzt werden.

Die MA-Vorgehensweise unterscheidet sich wesentlich von den anderen, da in MA komplette Ausführungseinheiten (z. B. `AE 1`) zwischen Ausführungsumgebungen bzw. Systemkomponenten migrieren[7]. Während in REV und COD der Fokus auf der Übertragung von Code-Komponenten (`Code`) liegt, die als Wissensträger agieren, werden in MA Wissensträger *und* Ausführungseinheiten zwischen entfernten Ausführungsumgebungen (`Client` und `Server`) transferiert.

5.1.2 Diskussion

Auf der Basis der im vorherigen Abschnitt erörterten Klassifizierung werden nun allgemeine Vorteile für den Entwurf von verteilten Informationssystemen erläutert, bei denen bestimmte Formen (REV, COD, MA) der Software-Mobilität zum Einsatz kommen.
Die folgende Auflistung von Vorteilen orientiert sich an den Interaktionsmustern der Ausführungseinheiten und der flexiblen Ortszuweisung der Code-Komponenten:

- *Anpaßbarkeit und Aggregation von Diensten.* In traditionellen Client/Server-Informationssystemen stellt der Server einen Dienst (z. B. `AE 2` in Abbildung 5.1) zur Verfügung, der in den meisten Fällen durch eine statisch definierte Schnittstelle vorgegeben ist. Beim Eintreten von neuen und unvorhergesehenen Änderungen auf der Seite der Clients sind unter Umständen Anpassungen an den Server notwendig. Dieses Vorgehen erhöht die Komplexität und vermindert die Flexibilität des Servers.

  Durch die Möglichkeit, Software in entfernten Ausführungsumgebungen (REV, COD, MA) auszuführen, wird die Flexibilität des Servers erhöht, da er nur noch sehr einfache Dienste zur Verfügung stellen muß. Die Einfachheit garantiert, daß solche Dienste stabile Schnittstellen besitzen. Clients

[7] Im allgemeinen kapselt ein mobiler Agent die Applikationslogik (Code-Komponente als Wissensträger), die Daten und die Ausführungseinheit.

sind nun in der Lage, neue und angepaßte Dienste durch die dynamische *Aggregation* von einfachen Diensten zu generieren[8]. Die aggregierten Dienste sind auf die jeweiligen Anforderungen und Erfordernisse der Clients genau abgestimmt[9].

- *Autonomie*. Mobile Umgebungen [FZ94] sind dadurch gekennzeichnet, daß viele der partizipierenden, verteilten Systemkomponenten nicht permanent über eine Verbindung zum Netzwerk verfügen. Eine dauerhafte Verbindung ist aber die Grundvoraussetzung in traditionellen, verteilten Umgebungen für das Stellen einer Anfrage an einen bestimmten Dienst, der von einer entfernten Systemkomponente erbracht wird.

  Durch die Software-Mobilität sind sogenannte verbindungslose Operationen möglich, in denen für die Dauer der Bearbeitung der Dienstanfrage keine dauerhafte Verbindung des Clients oder Servers zum Netzwerk existieren muß. Der anfänglich entfernte Dienst und die Anfrage, die auf den Server geladen wird, interagieren lokal miteinander. Sie sind in dieser Hinsicht autonom von den anderen Systemkomponenten und deren aktuellen Zuständen.

  Die Fähigkeit der lokalen Interaktion anstelle des entfernten Austausches von Daten über ein Netzwerk ermöglicht eine drastische Reduzierung des Datenverkehrs, wie vor allem von Gray et al. [GKP+01] argumentiert wird[10].

- *Flexibilität*. In traditionellen verteilten Umgebungen bestimmen die jeweiligen Systemkomponenten das Management der verschickten Daten. Jede Komponente enthält die für die Interpretation und Kommunikation der Daten notwendige Software (die Wissensträger `Code`).

  Falls sich die syntaktischen oder semantischen Schnittstellen der Daten häufig ändern sollten, ist eine Fixierung des Wissens um die korrekte Bearbeitung der Daten in den jeweiligen (verteilten) Systemkomponenten (z. B. `Code` in Abbildung 5.1) nicht erstrebenswert. Die Software-Mobilität unterstützt einen flexiblen und effizienten Ansatz, in dem der Wissensträger (`Code`) mit den Daten direkt verschickt wird. Auf diese Weise enthält eine Systemkomponente immer den aktuellen Wissensträger, der für die korrekte Bearbeitung der Daten notwendig ist.

[8] Die klassische, strikte Rollenzuteilung in Clients und Server verschwindet zunehmend in verteilten Informationssystemen, die sich durch die Benutzung von Software-Mobilität auszeichnen. So können Clients multiple Rollen in einem Informationssystem übernehmen, die denen eines Dienstnehmers order eines Diensterbringers entsprechen können (vgl. Kapitel 4).

[9] Der in Kapitel 4 diskutierte Applikationsserver *Respondeo* verfolgt einen ähnlichen Ansatz zur Generierung von neuen Diensten durch die Aggregation einfacher, existierender Dienste. Die Aspekte und die Vorteile der Software-Mobilität spielen bei *Respondeo* aber eine untergeordnete Rolle.

[10] Baldi und Picco [BP98a] erörtern die Kriterien und die Umstände für einen effektiven Einsatz von Komponenten, die sich durch Software-Mobilität auszeichnen.

5.2 Ziele

In diesem Abschnitt werden die generellen Ziele für den Entwurf von *Okeanos* beschrieben. Danach werden in Abschnitt 5.3 die wichtigsten Eigenschaften der Architektur und Implementierung erörtert.

5.2.1 Offenheit

Die *Offenheit* von *Okeanos* bezieht sich auf den flexiblen, erweiterbaren und stabilen Einsatz des Agentenframeworks. Es ist das Ziel, eine allgemeine Infrastruktur zur Realisierung von Multiagentensystemen zu schaffen, die nicht an bestimmte Anwendungsdomänen gebunden sind.

- Die Implementierung der Agenten in Java [AGH00] bildet die Basis für einen *plattformunabhängigen* Einsatz von *Okeanos*, da Java als eine Programmiersprachenumgebung auf mehreren Plattformen zur Verfügung steht[11]. Zudem unterstützt *Okeanos* die Anbindung von Diensten und Anwendungssystemen, die in anderen Programmiersprachen (z. B. C++ [Str97]) implementiert sind, über das *Java Native Interface* [Lia99] von Java.

- Die Unterstützung von *stationären* und *mobilen Agenten* in *Okeanos*[12]. Dadurch wird ein breites Spektrum möglicher Informationssysteme abgedeckt, die mit Hilfe von *Okeanos* realisiert werden können.

  Ein mobiler Agent in *Okeanos* basiert auf dem MA-Vorgehen (Abbildung 5.1), in dem die Ausführungseinheiten (`AE 1`) und die Wissensträger (`Code`) zwischen Ausführungsumgebungen (`Client` bzw. `Server`) migrieren können.

  Ein stationärer Agent wird auf der Basis einer dienstorientierten Vorgehensweise als sogenannter *Okeanos*-Dienst in das Framework eingebunden[13]. Jeder *Okeanos*-Dienst wird in einem speziellen Verzeichnis (siehe Abschnitt 5.3.3) verwaltet.

- Die Realisierung von Agenten mit einem kleinen Memory-Footprint[14]. Ein geringer Memory-Footprint unterstützt den Einsatz von *Okeanos* auch in Umgebungen, die nur über knappe Ressourcen verfügen wie z. B. den limitierten Speicherplatz auf einem PDA (*Personal Digital Assistant*).

- Die Benutzung von *einfachen* und *stabilen* Schnittstellen. In Abschnitt 5.3.1 wird die nachrichtenbasierte Kommunikationsschnittstelle in *Okeanos* erörtert. Sie ist durch eine einfache Schnittstelle definiert, die auf der Basis eines

[11] Vgl. auch Homepage von Java unter http://java.sun.com.

[12] Ein *stationärer Agent* ist ein Agent, der hinsichtlich seiner Lokation an einen bestimmten Ort gebunden ist.

[13] Siehe auch die Abschnitte 2.1.2.2 und 2.3.1.3.

[14] Ein *Memory-Footprint* steht für die Größe des Speicherabbildes von *Okeanos*-Agenten.

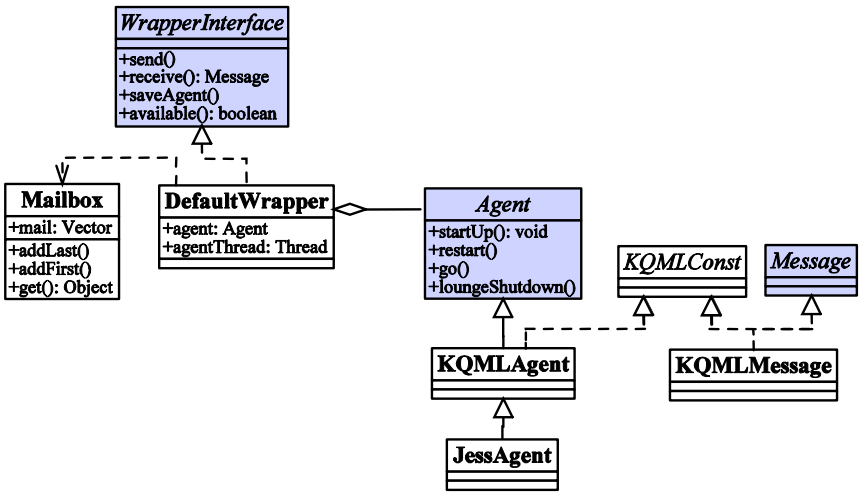


Abbildung 5.2: Agenten in *Okeanos*.

standardisierten Nachrichtenformats den strukturierten Austausch von beliebigen Nachrichteninhalten ermöglicht.

5.2.2 Unterstützte Programmiermodelle

In den folgenden beiden Abschnitten werden zwei unterschiedliche Programmiermodelle vorgestellt, die von *Okeanos* unterstützt werden. Ein Programmiermodell gibt den Rahmen für die Programmierung von Agenten in *Okeanos* vor.

5.2.2.1 Objektorientierte Programmierung

Die objektorientierte Programmierung ermöglicht die Realisierung von Agenten und Multiagentensystemen durch die *modulare Zerlegung* in Komponenten, die in geeigneter Weise *wiederbenutzt* werden können[15].
Abbildung 5.2 gibt einen Überblick über den Aufbau und die Beziehungen von Agenten in *Okeanos*. Jeder Agent unterstützt eine nachrichtenbasierte Kommunikationsschnittstelle. Jede Nachricht ist vom allgemeinen Typ `Message`. Das Framework bietet einen konkreten Nachrichtentyp `KQMLMessage` für die Kommunikation (siehe Abschnitt 5.3.1) zwischen Agenten an. Grundsätzlich sind aber auch weitere Nachrichtenformate in *Okeanos* integrierbar, sofern diese auf der Klasse `Message` aufbauen.
Der Zugriff auf einen Agenten in *Okeanos* erfolgt indirekt über eine genau definierte Schnittstelle (`WrapperInterface`). Analog zur Einbindung verschiedener Nachrichtenformate stellt *Okeanos* eine Standardimplementierung

[15] Siehe auch den Vergleich zwischen objektorientierten und agentenbasierten Ansätzen in Abschnitt 3.3.2.

`DefaultWrapper` zur Verfügung. Generell lassen sich aber auch weitere Implementierungen der Schnittstelle `WrapperInterface` in das Softwareframework von *Okeanos* einbinden.
Durch `DefaultWrapper` erhält jeder Agent einen sogenannten *Briefkasten* (`Mailbox`) zugeordnet, der seine gesamten Nachrichten verwaltet. Die bei `DefaultWrapper` gewählte Abstraktion eines Briefkastens entkoppelt den eigentlichen Kern eines Agenten von den Details der Nachrichtenverwaltung. Durch diese Entkopplung können Agenten asynchron miteinander interagieren: Die gesendeten Nachrichten werden in dem Briefkasten zwischengespeichert, bis der jeweils empfangende Agent die Nachrichten abruft. Die Zeitpunkte des Verschickens und Empfangens können dabei variieren, so daß der Sender einer Nachricht nicht an die Kontrollflüsse bzw. an die Empfangsbereitschaft des anderen Agenten gebunden ist.
Den Kern des Frameworks, der für die Realisierung von Agenten verantwortlich ist, besteht aus den Schnittstellen `WrapperInterface`, `Message` und der abstrakten Klasse `Agent`, wie in Abbildung 5.2 illustriert ist. Der Kern legt auf einer hohen Abstraktionsstufe die Eigenschaften und Strukturen von Agenten in *Okeanos* fest. Auf der nächsten, konkreteren Ebene bietet *Okeanos* eine (mögliche) Implementierung der spezifizierten Schnittstellen an. Dieser Ansatz bietet im wesentlichen zwei Vorteile:

- Aus der Sicht eines Agentenentwicklers wird die Benutzung des Frameworks erleichtert und der Entwicklungsprozeß beschleunigt, da für alle (abstrakten) Agentenkonzepte bereits Implementierungen zur Verfügung stehen.

- Falls die standardmäßigen Implementierungen – wie z. B. `KQMLMessage` oder `DefaultWrapper` – den Anforderungen eines bestimmten Informationssystems nicht genügen sollten, können weitere Implementierungen bzw. Spezialisierungen von `Agent`, `WrapperInterface` oder `Message` jeweils transparent in das Framework eingebunden werden. Durch diesen Ansatz wird eine bestmögliche Wiederbenutzbarkeit der Komponenten in *Okeanos* erreicht.

Konkrete Agenten, die in den Informationssystemen zum Einsatz kommen, entstehen durch eine weitere Spezialisierung der Klassen `KQMLAgent` oder `JessAgent`. Während die Eigenschaften eines `KQMLAgent` in Abschnitt 5.3.1 erörtert wird, steht ein `JessAgent` für einen regelbasierten Agenten, der im folgenden Abschnitt besprochen wird.

5.2.2.2 Regelbasierte und deklarative Programmierung

Die regelbasierte Programmierung ermöglicht die Repräsentierung von Wissen durch eine Heuristik, d. h. eine Menge von Regeln beschreibt die Aktionen, die in einer gegebenen Situation ausgeführt werden.

```
(defrule agent-migrate-rule-1
  (agent ?ID completed)
  (destination ?ID ?DESTHOST)
  (host ?DESTHOST ?IP)
=>
  (send-move-advice ?ID ?IP)
)
```

Listing 5.1: Eine Regel für einen Agenten zum Migrieren zwischen zwei Lokationen.

Regelbasierte Programme stellen einen deklarativen Ansatz für die Spezifikation der Applikationslogik dar: Es wird lediglich das Wissen in deklarativer Form zur Verfügung gestellt. Eine sogenannte *Inferenzmaschine* entscheidet auf der Basis der existierenden Regeln und der aktuellen Fakten, welche Teile der Applikationslogik benutzt werden. Eine Inferenzmaschine ist ein Programm, das automatisch durch die Anwendung von Pattern-Matching-Algorithmen bestimmt, welche Regeln unter den aktuellen Umständen angewendet werden müssen.
Im Zusammenhang mit dem Agentenframework *Okeanos* sind Agenten regelbasierte Programme, welche die Applikationslogik eines Agenten in Form von Regeln bestimmen. Dies führt zu einer klaren Trennung zwischen der Applikationslogik einerseits und den zugrunde liegenden Daten andererseits. Die konkrete Ausführung der Applikationslogik (z. B. die Bestimmung der Reihenfolge der auszuführenden Aktionen) wird durch die Inferenzmaschine bestimmt, welche zur Laufzeit auf der Basis der aktuellen Daten und Zustände den Ablauf des Agenten festlegt[16].
In *Okeanos* werden regelbasierte Agenten (vgl. die Klasse `JessAgent` in Abbildung 5.2) durch die Integration der *Java Expert System Shell* (JESS) [FH99][17] in das Framework integriert. JESS ist eine Inferenzmaschine, die komplett in Java implementiert ist. Sie unterstützt die Realisierung von regelbasierten Expertensystemen in Java[18]. Die Generierung neuer Fakten in JESS durch die Inferenzmaschine beruht auf dem *Rete-Algorithmus* [For82].
In Abbildung 5.1 ist ein Beispiel für eine JESS-Regel gegeben. Es wird eine Regel `agent-migrate-rule-1` definiert, welche die Umstände (`completed`) bestimmt, unter denen ein Agent (`IP`) auf einen anderen Host (`DESTHOST`) migrieren sollte.

[16] Siehe auch Abschnitt 3.2.4 über *Proaktive Dokumente*.

[17] Die Autoren von JESS, das bereits im Jahre 1999 in *Okeanos* eingebunden wurde, sind mittlerweile ein aktives Mitglied der Spezifizierung einer *Java Rule Engine API* [Jav02]. Deren Ziel ist die Spezifikation einer leichtgewichtigen und standardisierten Programmierschnittstelle für die Benutzung regelbasierter Inferenzmaschinen in Java.

[18] JESS ist eine Erweiterung und Portierung der *CLIPS*-Umgebung (siehe http://www.ghg.net/clips/CLIPS.html bzw. http://www.siliconvalleyone.com/clips.htm für weitere Informationen). CLIPS ist eine Umgebung zur Realisierung von Expertensystemen, die von der NASA entwickelt wird.

```
(tell
    :sender     Agent1
    :receiver   Agent2
    :ontology   "okeanos"
    :language   xml
    :content    "<action>
                    <job>move</job>
                    <parameter>dilbert</parameter>
                 </action>"
)
```

Listing 5.2: Ein Beispiel für eine KQML-Nachricht.

In *Okeanos* führt jeder mobile `JessAgent` seine aktuellen und zustandsbehafteten Fakten- und Regelbasen mit sich, wenn er zwischen zwei Ausführungsumgebungen migriert. Zu jedem Zeitpunkt besitzt ein mobiler Agent in *Okeanos* seine eigene Regel- und Wissensbasis unabhängig von der Lokation einer Ausführungsumgebung.

5.3 Architektur und Implementierung

In diesem Abschnitt werden einige der wichtigen Eigenschaften der Architektur und der Implementierung von *Okeanos* erörtert. Für weitere Details der Architektur und der Implementierung wird auf [SBS+00, SFK01a, HS01, Fri99] verwiesen.

5.3.1 Nachrichtenbasierte Kommunikation

Agenten in *Okeanos* interagieren durch den Austausch von Nachrichten miteinander. Die *Knowledge Query and Manipulation Language* (KQML) [FLM97][19] definiert eine Syntax zur Formulierung von Nachrichten, die zwischen Agenten in *Okeanos* ausgetauscht werden. Dabei läßt KQML den transportierten Nachrichteninhalt unberührt, weil KQML ausschließlich als mächtiger und standardisierter Container für den Nachrichtenaustausch dient. KQML legt sozusagen einen „Briefumschlag“ für Daten fest, die zwischen Agenten verschickt werden.
Der Inhalt einer KQML-Nachricht läßt sich in drei Teile unterteilen (siehe Listing 5.2):

- *Kommunikation*. Hierzu gehören u. a. Informationen über den Sender (`sender`) und den Empfänger (`receiver`) einer Nachricht.
- *Nachricht*. Hierzu gehören Informationen über die benutzte Ontologie [Gru93][20] (`ontology`) und Inhaltssprache (`language`) der Nachricht.

[19] KQML wurde bereits 1990 von der *External Interfaces Working Group* des *DARPA Knowledge Sharing Effort*-Programmes entwickelt und gilt als ein Standard im Bereich der Agentenkommunikation.

[20] Vgl. Abschnitt 6.2.2.

Eine Inhaltssprache enthält Informationen über die eigentlichen Daten (`content`), die zwischen Agenten ausgetauscht werden.

- *Inhalt.* Dieser Teil entspricht dem eigentlichen Inhalt (`content`) der Nachricht, der von KQML unberücksichtigt bleibt.

Die Schlüsselwörter `sender`, `receiver`, `ontology`, `language` und `content` sind in KQML vorgegeben und dienen zur einheitlichen Beschreibung einer Nachricht[21]. KQML ermöglicht zudem die Festlegung weiterer, applikationsabhängiger Schlüsselwörter.
Jede Nachricht in KQML gehört einer sogenannten *Performative*[22] an. Eine Performative bestimmt den Typ einer KQML-Nachricht. KQML legt eine Reihe von Performativen standardmäßig fest (wie z. B. `tell` in Abbildung 5.2). Darüber hinaus können weitere Performativen hinzugefügt werden, die für den Nachrichtenaustausch zwischen Agenten von Bedeutung sind.
Wiederhold [Wie92b] betont die Bedeutung von KQML als allgemeinen Nachrichtencontainer für die Realisierung stabiler Kommunikationsschnittstellen von Agenten:

- Sicherung der Kommunikationsschnittstellen durch die Festlegung einer *einfachen* Spezifikation für das Senden und Empfangen von Nachrichten, die immer dem gleichen Nachrichtenformat (KQML) entsprechen[23].

- Unterstützung einer *automatisierten* Interaktion zwischen Agenten, da jeder Agent über die gleiche Kommunikationsschnittstelle verfügt. Dadurch verlieren die syntaktischen Details der Interaktion an Bedeutung[24]. Der Fokus der Interaktion liegt ausschließlich auf dem Inhalt der Nachricht.

- Benutzung eines *standardisierten* und *erweiterbaren* Nachrichtenformats.

- Transport von Nachrichten mit *beliebigen* Inhalten. So können neben textbasierten Inhalten auch Grafiken, Videos oder serialisierte Agenten transportiert werden. Zudem lassen sich XML-Dokumente als Inhalt einer KQML-Nachricht kommuniziert.

In Abbildung 5.2 ist verdeutlicht, inwiefern Agenten in *Okeanos* ausschließlich über Nachrichten kommunizieren. Ein Agent, der im Hinblick auf ein zu realisierendes Informationssystem zu entwickeln ist, entsteht durch die Spezialisierung

[21] Ferner gibt es weitere Schlüsselwörter in KQML [FLM97] wie beispielsweise `reply`, `reply-with` oder `in-reply-to`.

[22] *Performative* – „Eine sprachlich zum Ausdruck gebrachte Handlung, damit auch vollziehend“ [Dud96].

[23] Siehe auch die Diskussion des Nachrichtenformats beim Applikationsserver *Respondeo* in Kapitel 4. Dieser bietet ebenfalls eine nachrichtenbasierte Kommunikationsschnittstelle an.

[24] Bei traditionellen, auf dem RPC basierenden Client/Server-Systemen liegt der Schwerpunkt auf der syntaktischen Definition der Schnittstelle (siehe auch Abschnitt 5.1).

der Klasse `KQMLAgent`, wie in Abbildung 5.2 dargestellt. Dieser Agent erbt alle Eigenschaften und Fähigkeiten, um KQML-Nachrichten zu empfangen und zu versenden. Die Klasse `KQMLConstants` enthält bereits eine Vielzahl der in KQML benutzten Schlüsselwörter, so daß die Entwicklung eines KQML-Agenten erleichtert wird.
Eine alternative Realisierungsmöglichkeit ist die Spezialisierung des Agenten von der Klasse `JessAgent`, wobei diese Klasse die in Abschnitt 5.2.2.2 beschriebenen regelbasierten Agenten in *Okeanos* repräsentiert. Dadurch werden Agenten geschaffen, die über komplexe nachrichten- *und* regelbasierte Mechanismen miteinander kommunizieren. In [SBS+00, SFK01a] werden komplexe Agenten vom Typ `JessAgent` für die verteilte Berechnung eines irregulären Problems, das durch seinen nicht deterministischen bzw. nicht vorhersehbaren Ablaufplan gekennzeichnet ist, eingesetzt.

5.3.2 Agentenort – *Lounge*

Gemäß der Definition in Abschnitt 5.1.1 ist eine Ausführungsumgebung (AU) ein Ort, an dem wichtige Komponenten eines Agenten – wie z. B. die Code-Komponenten oder Ausführungseinheiten – sowie Ressourcen installiert sind (siehe auch Abbildung 5.1). Eine Ausführungsumgebung oder ein Agentenort wird innerhalb des Frameworks *Okeanos* als sogenannte *Lounge* bezeichnet.
Eine *Lounge* hat folgende Eigenschaften:

- Jeder Agent ist immer einer bestimmten *Lounge* zugeordnet. Bei der Kommunikation zwischen Agenten werden grundsätzlich zwei Fälle unterschieden:
  - *Lokale Interaktion.* Bei der lokalen Interaktion kommunizieren Agenten direkt über den Austausch von KQML-Nachrichten miteinander, die im Briefkasten des jeweiligen Agenten gelegt werden.
  - *Entfernte Kommunikation.* Agenten auf verschiedenen *Lounges* kommunizieren indirekt durch Migration miteinander, d. h. um mit einem Agenten auf einer entfernten *Lounge* zu interagieren, ist es erforderlich, daß der betreffende Agent zuvor dorthin migriert. Jede *Lounge* besitzt einen speziellen, stationären Agenten *PortalAgent*, der ausschließlich für die Migration der Agenten in *Okeanos* zuständig ist.
- Joseph und Kawamura [JK01] betonen die Bedeutung der Miteinbeziehung der Umwelt in ein Multiagentensystem. Eine *Lounge* stellt das Bindeglied zwischen Agenten und ihrer Umgebung dar. Die Umgebung liefert wichtige Informationen für Agenten oder für Dienste, die in *Okeanos* angebunden sind. Der Begriff der Umgebung ist dabei sehr weit gefaßt und beinhaltet u. a. externe Systemkomponenten oder weitere Agenten. In Abhängigkeit von den Umweltzuständen werden Entscheidungen bei Agenten getroffen und Aktionen selbständig initiiert.

Jede *Lounge* legt ihre Zustände in *SpectoML*-Dokumenten (siehe Kapitel 6) ab, die als leichtgewichtige Datenbasen für die Ermittlung der *Lounge*-Zustände dienen. In Schimkat et al. [SFK01a] wird der Einsatz von Zustandsinformationen für die optimale Ausführung von Multiagentensystemen auf der Basis von *Okeanos* in verteilten Umgebungen erörtert. Die Fallstudie in Kapitel 8.2 über eine verteilte Berechnung im Bereich des Symbolischen Rechnens unterstreicht die Bedeutung umwelt-relevanter Zustände für Agentensysteme.

5.3.3 Verzeichnisdienst

Der Vezeichnisdienst *DirectoryAgent* ist neben dem Kommunikationsdienst *PortalAgent*, der im vorherigen Abschnitt kurz beschrieben wurde, ein weiterer spezieller Agent, der von einer *Lounge* zur Verfügung gestellt wird.
Der Verzeichnisdienst stellt das wichtige Bindeglied zwischen der agenten- und der dienstorientierten Sicht auf *Okeanos* dar. Jeder Dienst, der auf einer *Lounge* verfügbar ist, wird vom Verzeichnisdienst mitgeführt. Dienste in *Okeanos* werden über einen Namen identifiziert. Dieser Name ist im Kontext einer *Lounge* eindeutig. Das Paar <*Dienstname, Lounge*> bildet einen global eindeutigen Schlüssel für die Referenzierung eines Dienstes in einem Informationssystem, das aus mehreren verteilten *Lounges* besteht.
Da die Verfügbarkeit von Diensten in dynamischen Umgebungen und Informationssystemen fortwährenden Änderungen unterzogen ist, wird eine Infrastruktur innerhalb des Agentensystems *Okeanos* notwendig, welche die aktuell verfügbaren Dienste in einer speziellen Datenstrukur – dem Verzeichnisdienst – verwaltet. Der Verzeichnisdienst hält Informationen über lokale und globale Gegegebenheiten für *Okeanos*-Agenten bereit:

- Eine Liste von Agenten, die *lokal* einen bestimmten Dienst anbieten.
- Eine Liste von *Lounges*, an denen ein bestimmter *globaler* Dienst angeboten wird.
- Eine Liste aller bekannten *Lounges*.

Jeder Dienst in *Okeanos* ist einem Agenten zugeordnet. Entweder stellt ein Agent selbst einen Dienst zur Verfügung oder er agiert als Mediator für den eigentlichen Dienst, der potentiell in verschiedenen Programmiersprachen implementiert ist. Es ist die Aufgabe des Agenten, als Repräsentant des eigentlichen Dienstes für die Transformationen zwischen den (internen) KQML-basierten Nachrichten in *Okeanos* und den (externen) heterogenen Dienststrukturen zu vermitteln. Die strikte Anbindung von Diensten in *Okeanos* über das in Abschnitt 2.2 vorgestellte Mediatorkonzept ermöglicht eine flexible und einheitliche Einbindung von unterschiedlichen Diensten in das Agentensystem *Okeanos*.

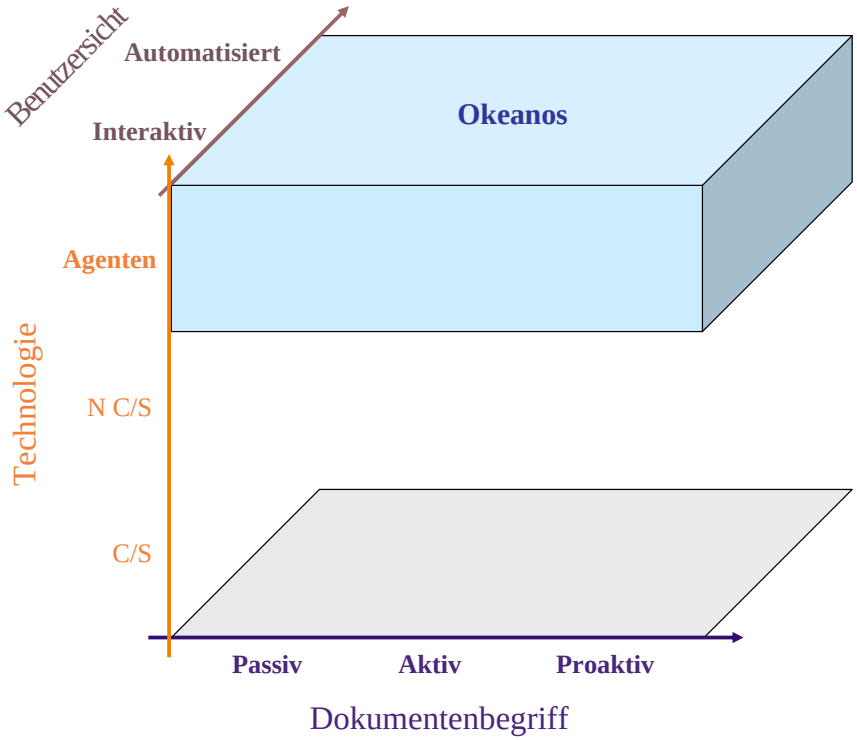


Abbildung 5.3: Einordnung von *Okeanos* in die Taxonomie.

5.4 Einordnung in die Taxonomie

Die Einordnung von *Okeanos* in die in Kapitel 3 erörterte Taxonomie für Informationssysteme ist nur schwer möglich, da *Okeanos* lediglich Infrastrukturen für die Realisierung von Informationssystemen zur Verfügung stellt. Dennoch wird im folgenden gemäß Abbildung 5.3 eine *mögliche* Einordnung des Agentenframeworks *Okeanos* in die Taxonomie vorgenommen.

Benutzersicht *Okeanos* unterstützt sowohl *interaktive* (vgl. Abschnitt 3.1.1) als auch *automatisierte* (vgl. Abschnitt 3.1.2) Benutzersichten. Durch eine nachrichtenbasierte Kommunikation unter Verwendung von KQML-Nachrichten als standardisiertes und erweiterbares Nachrichtenformat (vgl. Abschnitt 5.3.1) verfügen *Okeanos*-Agenten über stabile Kommunikationsschnittstellen, die in interaktiver und automatisierter Form benutzt werden können. Die Benutzung von KQML fördert eine inhaltsbasierte Kommunikation zwischen Agenten. Dabei wird der Fokus auf den Inhalt der Nachrichten und weniger auf die Festlegung der syntaktischen Kommunikationsschnittstellen gelegt.

Dokumentenbegriff Bei der in Abbildung 5.3 vorgenommenen Einordnung ist zu

beachten, daß *Okeanos* für *passive*, *aktive* und *proaktive* Dokumente einsetzbar ist[25].

In Informationssystemen mit einem *passiven Dokumentenbegriff* werden *Okeanos*-Agenten als Mediatoren der Dokumente eingesetzt[26]. Dabei wird einem Mediator eine bestimmte Anzahl von Dokumenten zugeordnet. Falls alle Dokumente von einem einzigen Mediator verwaltet werden, entspricht dieser Agent einem Applikationsserver, wie er in Kapitel 4 vorgestellt wird. In diesem Falle ist die Terminologie eines „Agenten“ irreführend. Falls jedes Dokument genau einem Agenten zugeordnet ist, entsteht ein „echtes“ Multiagentensystem. Die Kriterien für die Auswahl einer der jeweiligen Realisierungsvarianten hängt von den konkreten Anforderungen des jeweiligen Informationssystems ab.

Generell ist die Umsetzung und Realisierung eines *aktiven Dokumentenbegriffs* an das Konzept der Software-Mobilität (vgl. Abschnitt 5.1) gebunden:

- Die Umsetzung von aktiven Dokumenten des *Typs C* (vgl. Abbildung 3.5) wird durch den Einsatz von mobilen Agenten erleichtert. Der *MA*-Ansatz (vgl. Abschnitt 5.1) in Bezug auf die Software-Mobilität unterstützt den mobilen Einsatz von aktiven Dokumenten, die nicht mehr an bestimmte Lokationen gebunden sind. Typische Lokationen für reale und virtuelle Dokumente (vgl. Abbildung 3.3) sind Dateisysteme und Datenbanken. Durch die flexible Wahl der Lokation von (aktiven) Dokumenten ergeben sich eine Reihe interessanter Einsatzmöglicheiten, die u. a. in Abschnitt 10.2 in einer Fallstudie über einen sogenannten *Living Hypertext* [SKN02] erörtert werden.
- Aktive Dokumente des *Typs B* werden über den *COD*-Ansatz unterstützt. Die aktive Komponente des Dokuments wird über Verweisstrukturen eingebunden. Sie entspricht dem Wissensträger, der dynamisch in ein Dokument eingebettet wird.

Ein *proaktiver Dokumentenbegriff* wird in *Okeanos* durch die Möglichkeit einer regelbasierten Programmierung von Agenten (vgl. Abschnitt 5.2.2.2) unterstützt. Die Applikationslogik der proaktiven Komponente eines Dokuments wird über Regeln in JESS deklarativ festgelegt.

Technologie Die technologische Einordnung von Agenten bzw. von agentenbasierten Informationssystemen in die Taxonomie ist offensichtlich, da Agenten explizit als ein Klassifizierungsmerkmal in Erscheinung treten (vgl. Abschnitt 3.3.1).

[25] Vgl. die Diskussion der verschiedenen Dokumentenbegriffe in Abschnitt 3.2.
[26] Vgl. die Diskussion des Mediatorkonzepts in Abschnitt 2.2.

Kapitel 6

SpectoML – eine spezielle Auszeichnungssprache

Traditionell werden digitale Dokumente als fixierte und statisch definierte Ausprägungen von digitalen Artefakten betrachtet [LM95] (vgl. den Dokumentenbegriff in Abschnitt 2.1.2.1). Dokumente, die zeitlich bezogene Eigenschaften beschreiben, spezifizieren einen *Zustand*, der wichtige Informationen für verschiedene Anwendungsbereiche enthält:

- *Monitoring*. Im Bereich des Monitoring wird auf der Basis von Zustandsinformationen versucht, den Ablauf eines Systems zu verfolgen. In verteilten Informationssystemen liefert die Ablaufverfolgung wichtige Informationen über den globalen Systemzustand und bietet die Möglichkeit für vielfältige Optimierungen des Gesamtablaufs.

- *Debugging*. Im Bereich des Debugging liegt der Schwerpunkt auf der Erkennung von Fehlern und deren Ursachen. Die Basis für die Erkennung bilden Zustandsinformationen der beteiligten Softwarekomponenten.

- *Kontextabhängige Verarbeitung*. Bei der kontextabhängigen Verarbeitung geht es darum, aktuelle Zustände von Softwarekomponenten, Informationssystemen und deren relevanten Umgebungen in den Verarbeitungsprozeß miteinzubeziehen. So kann die Präsentation der Ergebnisse einer gestellten Informationsanfrage an ein Informationssystem abhängig von mehreren, aktuellen Kontexten sein. So spielen die aktuelle Lokation des Benutzers, dessen Präferenzen für die Visualisierung der Ergebnisse oder die Präsenz anderer Benutzer im Informationssystem eine wichtige Rolle bei der Rückgabe der Ergebnisse an den Benutzer[1].

  Bei der Informationssuche im Web fordern mehrere Autoren einen Dokumentenbegriff, der unterschiedliche Kontexte berücksichtigt und so die Qua-

[1] Vgl. Abschnitt 3.2.3 über *aktive Dokumente* für eine Thematisierung von Anwendungskontexten.

lität der Informationssuche verbessert [Fuh00, AM00]. In Kapitel 10.2 wird anhand einer Fallstudie dargelegt, wie unterschiedliche Kontexte bei der Informationssuche im Web mitberücksichtigt werden.

In diesem Kapitel wird eine spezielle Auszeichnungssprache *Specto Markup Language* (*SpectoML*)[2] vorgestellt, wobei gleichzeitig ihre speziellen Eigenschaften diskutiert werden. *SpectoML* [SHKK00, SFK01a] wird für die Repräsentierung von Zuständen unterschiedlicher Applikationen und Softwarekomponenten benutzt.
Eine *Auszeichnungssprache*[3] hat die Aufgabe, die logischen Bestandteile eines Dokuments zu beschreiben. Die *Extensible Markup Language* (XML) [BPSMM00], die vom *World-Wide-Web Consortium* (W3C) konzipiert und standardisiert wird, stellt eine *erweiterbare* Auszeichnungssprache dar. Sie teilt die wesentlichen Freiheiten der *Standard Generalized Markup Language* (SGML) [ISO86], ohne die Kompliziertheit dieses ISO-Standards zu besitzen.
Generell ist eine Auszeichnung eine Information in der Gestalt von Symbolen, die zu einem Dokument hinzugefügt werden. Sie wird dazu benutzt, die einzelnen Bestandteile eines Dokuments zu identifizieren und die Beziehungen der Bestandteile untereinander festzulegen. Demnach besteht eine Auszeichnungssprache aus einer Menge von Auszeichnungssymbolen.
In der XML-Terminologie werden die Auszeichnungssymbole als *Tags* bezeichnet. Tags stellen eine Form von Metadaten [Mar98] dar, die explizite Informationen über bestimmte Textstellen beinhalten. Generell sind Metadaten Daten, die andere Daten beschreiben. Sie liefern zusätzliche Informationen über die Struktur und den Inhalt der Daten und verbessern dadurch die Nutzbarkeit der Daten[4]. So werden Softwarekomponenten in die Lage versetzt, auf die in den XML-Dokumenten enthaltenen Informationen einfacher und in automatisierter Form zuzugreifen.
XML gestattet es dem Autor, eine eigene Grammatik in Form einer *Document Type Definition* (DTD) oder eines Schemas [TBMM01, BM01] zu entwerfen. Darin werden Inhalte und logische Bestandteile des XML-Dokuments beschrieben. Die zwei wesentlichen Komponenten einer DTD sind die Elemente und Attribute[5]. Ein XML-*Element* besteht aus einem Paar von Tags und den Daten, die durch die Tags festgelegt werden. Die Daten können ihrerseits wiederum andere Elemente beinhalten. Ein XML-Element kann *Attribut*e enthalten, die zusätzliche Informationen über ein Element liefern.
Eine DTD ist eine normative Definition, die strukturierte Informationen beinhaltet. Sie gibt Auskunft über folgende Fragen:

[2] *spectare* (lat.) – anblicken, ansehen, blicken, schauen, zuschauen, zusehen; *specto* (1. Person Singular) – *ich schaue zu*.

[3] Auszeichnungssprache – engl.: markup language.

[4] Vgl. auch Abschnitte 2.1.1 und 2.1.2.1 für eine allgemeine Diskussion der Beziehungen zwischen Dokumenten und Metadaten.

[5] Für eine komplette Beschreibung der Bestandteile einer DTD bzw. eines XML-Dokuments wird auf die Spezifikation in Bray et al. [BPSMM00] verwiesen.

- Welche Tags gibt es in einem XML-Dokument?
- In welchem Kontext dürfen diese Tags vorkommen?
- Welche Attribute sind erlaubt?
- Welche weiteren Tags und Daten sind innerhalb eines Tags erlaubt?

Die Daten, die durch und mit XML repräsentiert sind, werden auch als *semi-strukturierte Daten* [Abi97] bezeichnet. Semi-strukturierte Daten sind Daten, die keine streng vorgegebene Struktur wie beispielsweise in relationalen Datenbanken besitzen. Trotzdem besteht die Möglichkeit, auf die einzelnen Bestandteile von semi-strukturierten Daten über entsprechende Anfragesprachen zuzugreifen. In dieser Hinsicht stellen XML-Dokumente – wie beispielsweise *SpectoML*-Dokumente – semi-strukturierte Datenbasen dar. Dadurch wird die konzeptionelle Verwandtschaft zwischen XML-Dokumenten und Datenbanken zum Ausdruck gebracht. Die in diesem Kapitel vorgestellten *SpectoML*-Dokumente stellen demnach eine semi-strukturierte Datenbasis für die Repräsentierung von Zuständen unterschiedlicher Applikationen und Softwarekomponenten dar.
Dieses Kapitel ist folgendermaßen aufgebaut: Im nächsten Abschnitt werden die wesentlichen Ziele beim Entwurf der *SpectoML* erörtert. Nach der Einführung der Terminologie für Zustände befaßt sich Abschnitt 6.2 mit den charakteristischen Details der Architektur von *SpectoML*. Es werden konkret die Elemente und Attribute der Document Type Definition sowie der Mechanismus für die getrennte, explizite Darstellung der inhaltlichen Zustandsbedeutungen veranschaulicht. In Abschnitt 6.3 werden die Eigenschaften der *SpectoML* beschrieben, wobei der Schwerpunkt auf der kompakten Repräsentation von XML-basierten Zuständen liegt. Abschnitt 6.4 befaßt sich mit dem speziell an die *SpectoML* angepaßten Parser, der die Kernkomponente der Anfragesprache für *SpectoML*-Dokumente bildet. Schließlich werden in Abschnitt 6.5 einige der Einsatzgebiete der *SpectoML* beschrieben.

6.1 Ziele

6.1.1 Allgemeinheit

Die *Allgemeinheit* betrifft die Vielzahl der Anwendungsmöglichkeiten der *SpectoML*. Ein wesentliches Ziel beim Entwurf war die Auszeichnung von Applikationszuständen, die *unabhängig* von Systemplattformen, Programmiersprachen und Applikationsdomänen[6] benutzt werden können.
Einen plattform- und programmiersprachenunabhängigen Einsatz unterstützt die Verwendung von XML als ein textbasiertes Format. Zudem existiert für die Verarbeitung von XML-Dokumenten eine große Palette an Werkzeugen, die auf verschiedenen Plattformen zur Verfügung stehen.

[6] In den Kapiteln 7, 8.1 und 10.2 wird *SpectoML* für das Monitoring von verschiedenen Applikationen, für das Management von Testdokumenten bzw. für die erweiterte Suche nach Informationen in einem Hypertext benutzt.

Der spezielle Entwurf der *SpectoML* (siehe Abschnitt 6.2) ermöglicht die effiziente Repräsentation von Zuständen der Applikationen. Dabei ist *SpectoML* an keine bestimmte Applikation oder Applikationsdomäne gebunden und ermöglicht einen uniformen Zugriff (siehe Abschnitt 6.3) auf die in *SpectoML*-Dokumenten enthaltenen Informationen.

6.1.2 Erweiterbarkeit

Die oben beschriebenen Ziele der Allgemeinheit und des breiten Anwendungsspektrums bedingen erweiterbare, flexible Strukturen beim Entwurf der *SpectoML*. Die Erweiterbarkeit garantiert, daß die individuellen und applikationsspezifischen Eigenheiten adäquat in *SpectoML* eingebunden werden.
Analog zum Entwurf eines objektorientierten Frameworks [JF88, FS97][7] gibt der Entwurf der *SpectoML* einen systematischen Rahmen vor, der die Basis für die Einbindung unterschiedlicher Anforderungen seitens der Applikationen darstellt. In den Abschnitten 6.2 und 6.3 werden einige der Mechanismen und Eigenschaften in *SpectoML* beschrieben, die für eine erweiterbare Auszeichnung von Zuständen in XML benutzt werden.

6.1.3 Kompaktheit

Das Ziel der *Kompaktheit* bezieht sich auf die Größe und Effizienz der Repräsentation von Zuständen in *SpectoML*. Eine kompakte Repräsentation ermöglicht eine speichergünstige Inanspruchnahme der vorhandenen Systemressourcen. Dadurch wird gewährleistet, daß die Verarbeitung von Applikationszuständen durch die Benutzung der *SpectoML* keine Seiteneffekte auf die jeweiligen Applikationen ausüben[8]. In Abschnitt 6.3 wird erläutert, inwiefern *SpectoML* eine kompakte Darstellung unterstützt.
Eine effiziente Repräsentation ermöglicht die Benutzung von ausdrucksstarken Mitteln zur Beschreibung von Zuständen. Neben der Beschreibung ist die dynamische Extraktion der in den XML-Dokumenten enthaltenen Informationen von großer Bedeutung. Die *SpectoML*-Dokumente werden als Datenbasen verstanden, die wichtige Informationen über die Applikationen enthalten. Die dynamische Extraktion bezeichnet den Vorgang, in dem zunächst eine Anfrage an eine solche (XML-basierende) Datenbasis gestellt wird. Danach werden die Informationen entsprechend aus der Datenbasis extrahiert und zurückgegeben. Die für die Verarbeitung von Anfragen notwendigen Werkzeuge sollen effizient im Hinblick auf ihre Größe und Verarbeitungsgeschwindigkeit sein. In Abschnitt 6.4 wird die spezielle Anfragesprache für *SpectoML*-Dokumente erörtert, die diesen Kriterien genügen soll.

[7] Vgl. auch die Abschnitte 2.1.2.2 und 4.2.

[8] Falls die vorhandenen Speicherressourcen für die Verarbeitung von Zuständen stark limitiert sein sollten, würde eine speicherintensive Kodierung der Zustände in XML zusätzliche Ressourcen in Anspruch nehmen, die eventuell direkte, negative Auswirkungen auf die Applikation hätten.

6.2 Architektur

Für die Erörterung des Aufbaus und der Eigenschaften der *SpectoML* ist es notwendig, folgende Terminologie für die Beschreibung von Zuständen festzulegen:

Zustandstyp Ein *Zustandstyp* legt den Wertebereich der möglichen Zustände fest. Ein Zustandstyp ist immer benutzerdefiniert.

Beispiel: Ein Zustandstyp, der mit `StartDienst` bezeichnet wird, definiert für den Applikationsserver *Respondeo* (vgl. Kapitel 4) den Zustand, daß ein Dienst von *Respondeo* erfolgreich gestartet wurde.

Zustandsgruppe Eine *Zustandsgruppe* bestimmt eine Menge von Zustandstypen, die zueinander in Beziehung stehen. Die Beziehung definiert Abhängigkeiten auf der Ebene der Applikation. Ein Zustandstyp kann mehreren Zustandsgruppen angehören.

Beispiel: Der Zustandstyp `StopDienst` für *Respondeo* steht mit dem Typ `StartDienst` in einer simplen Abhängigkeitsbeziehung, d. h. `StopDienst` setzt immer einen Zustand `StartDienst` voraus. Diese beiden Zustandstypen bilden eine Zustandsgruppe.

Zustandsparameter Ein Zustandstyp kann *Zustandsparameter* enthalten, die zur Laufzeit den aktuellen Zustand näher spezifizieren.

Beispiel: Die Zustandstypen `StartDienst` bzw. `StopDienst` besitzen jeweils Parameter, die den Namen und die Lokation des Dienstes beinhalten.

Zustand Der *Zustand* einer Applikation entspricht einem digitalen Artefakt zu einem bestimmten Zeitpunkt. Ein Zustand ist immer genau einem Zustandstypen zugeordnet und entspricht einer Instanz seines Zustandstyps. Der Begriff der Instanz bezeichnet dabei die Eigenschaft, daß die konkrete Ausprägung des Zustands erst zur Laufzeit feststeht bzw. dynamisch ermittelt wird.

Beispiel: Um `07:22 Uhr` ist der Zustand vom Typ `StartDienst` mit den Parametern `EmailDienst` und `dilbert.informatik.uni-tuebingen.de` eingetreten, d. h. der Dienst `EmailDienst` ist auf dem Rechner mit der IP-Adresse `dilbert.informatik.uni-tuebingen.de` erfolgreich gestartet worden.

Die Architektur, die in den folgenden Abschnitten dargelegt wird, umfaßt die Details des grundsätzlichen Aufbaus der *SpectoML* (siehe Abschnitt 6.2.1) und die Beschreibung des Ansatzes für die getrennte Darstellung der semantischen Zustandsbedeutungen (siehe Abschnitt 6.2.2).

```
<!-- DTD for Knowledge Repositories of Living Documents
<!ELEMENT events (preamble, (log | warn | error)*)>

<!ELEMENT preamble (version, timeZone?, datePattern?,
                    messageType?, source?, startDate?,
                    info?)>

<!ELEMENT version (#PCDATA)>
<!ELEMENT timeZone (#PCDATA)>
<!ELEMENT datePattern (#PCDATA)>
<!ELEMENT messageType (#PCDATA)>
<!ELEMENT startDate (#PCDATA)>
<!ELEMENT info (#PCDATA)>

<!ELEMENT log (source?, date, msg, param*)>
...
<!ELEMENT source EMPTY>
<!ELEMENT date (#PCDATA)>
<!ELEMENT msg (#PCDATA)>
<!ELEMENT param EMPTY>
...
```

```
...
<!ATTLIST log     sev        CDATA   #IMPLIED
                  id         ID #IMPLIED
                  fatherID   IDREF #IMPLIED>
...
<!ATTLIST source  app        CDATA #REQUIRED
                  appID      CDATA #IMPLIED
                  class      CDATA #IMPLIED
                  IP         CDATA #REQUIRED>

<!ATTLIST msg id             CDATA #IMPLIED
              para1          CDATA #IMPLIED
              para2          CDATA #IMPLIED
              para3          CDATA #IMPLIED
              para4          CDATA #IMPLIED
              para5          CDATA #IMPLIED>

<!ATTLIST param  type        CDATA #IMPLIED
                 name        CDATA #IMPLIED
                 value       CDATA #REQUIRED>
```

Abbildung 6.1: Auszug aus der Document Type Definition der *SpectoML*.

6.2.1 Document Type Definition

In Abbildung 6.1 ist ein Auszug aus der DTD der *SpectoML* abgebildet[9]. In den folgenden Punkten sind einige der wichtigsten Elemente der DTD näher beschrieben[10]:

preamble `preamble` gibt globale Eigenschaften an, die für das gesamte Dokument bzw. für alle Einträge gelten.

- **timeZone** `timeZone` ist ein *optionales Element* (OE) und bestimmt die Zeitzone, auf die sich die Einträge im Dokument beziehen.
- **datePattern** `datePattern` (OE) bestimmt das Format für die Datums- bzw. Zeitangaben. Durch die Einführung dieses Elements können verschiedene Formate benutzt und ineinander überführt werden.
- **messageType** `messageType` (OE) bestimmt das Fragment (z. B. eine objektorientierte Klasse), das für die Erstellung der eindeutigen Zustandstypen zuständig ist.
- **info** `info` (OE) dient zur Spezifikation von zusätzlichen Informationen über das XML-Dokument.

log | warn | error `log`, `warn` und `error` definieren die erste Klassifizierungsebene von Zuständen. Jeder Zustand ist vom Typ `log`, `warn` oder `error`. Ferner gibt es eine Reihe von Attributen, die im folgenden beschrieben sind:

[9] Im Anhang A wird in den Listings A.4-A.6 eine mögliche Transformation der DTD in ein entsprechendes XML-Schema [TBMM01, BM01] dargestellt.

[10] Eine komplette und ergänzende Beschreibung der Elemente und Attribute findet sich in [Häu99].

sev `sev` bestimmt die Dringlichkeit[11] eines Zustands. Dieses Attribut ermöglicht eine weitere Unterteilung der Zustände in bestimmte Zustandsgruppen. Eine Zustandsgruppe wird auf der Ebene der Applikation festgelegt[12].

id Falls die Zustände automatisch durchnumeriert werden, erhält jeder Zustand eine eindeutige Nummer.

fatherID `fatherID` bestimmt die Nummer des Zustands, der zum aktuellen Zustand in einer Vater-Kind-Beziehung steht. Mit den Attributen `id` und `fatherID` werden hierarchische Beziehungen und Zusammengehörigkeiten zwischen Zuständen festgelegt. Zustände, die via `fatherID` und `id` zueinander in Beziehung stehen, bilden eine hierarchische Zustandsgruppe.

source `source` (OE) spezifiziert die Quelle, welche für die Generierung des aktuellen Zustands verantwortlich ist. Im folgenden sind die Attribute `app`, `appID` und `IP` des Elements `source` beschrieben:

app `app` bestimmt den symbolischen Namen der Applikation, in welcher der Zustand eingetreten ist.

appID `appID` steht in direktem Zusammenhang mit `app` und gibt eine Nummer für die jeweilige Applikation an. Jeder Applikation, die für die Kodierung ihrer Zustände *SpectoML* benutzt, wird eine eindeutige Nummer zugewiesen. Unter dieser Nummer werden die für die jeweilige Applikation definierten Zustandstypen hinterlegt.

IP `IP` gibt die IP-Adresse an, unter welcher der Zustand eingetreten ist.

date `date` spezifiziert den Zeitpunkt und das Datum für das Eintreten des Zustands. Die Zeitangaben werden gemäß dem im `datePattern` bestimmten Format abgelegt.

msg `msg` bestimmt die textuelle Beschreibung des Zustands. Falls kein `id`-Attribut (vgl. Beschreibung des folgenden Attributs) spezifiziert worden ist, bestimmt ausschließlich `msg` den textuellen Inhalt des Zustands.

id `id` ist eine eindeutige Identifizierungsnummer für einen bestimmten Zustandstyp innerhalb einer Applikation. Zusammen mit dem Attribut `appID` identifiziert `id` jeden Zustandstyp eindeutig. Das Tupel (`id`, `appID`) bildet einen Primärschlüssel für alle existierenden Zustandstypen.

[11] engl. *severity*.

[12] Siehe auch das Beispiel bei der Einführung der Terminologie der *Zustandsgruppe* zu Beginn dieses Abschnitts: `StartDienst` und `StopDienst` bilden eine applikationsabhängige Zustandsgruppe.

```
<!— DTD der Bedeutungen von SpectoML-Dokumenten —>

<!ELEMENT spectoProperties
 (package?, loggerClass, applicationName, applicationID?, logfileName?
    , appendFlag?, echoFlag?, info?, bufferSize?, logMessage+)>

<!ELEMENT package (#PCDATA)>
<!ELEMENT loggerClass (#PCDATA)>
<!ELEMENT applicationName (#PCDATA)>
<!ELEMENT applicationID (#PCDATA)>
<!ELEMENT logfileName (#PCDATA)>
<!ELEMENT appendFlag (#PCDATA)>
<!ELEMENT echoFlag (#PCDATA)>
<!ELEMENT info (#PCDATA)>
<!ELEMENT bufferSize (#PCDATA)>

<!ELEMENT constant (#PCDATA)>
<!ELEMENT severity (#PCDATA)>

<!ELEMENT logMessage (constant, severity?,
     (english | german | french | italian)+)>

<!ELEMENT german (#PCDATA)>
<!ELEMENT english (#PCDATA)>
<!ELEMENT french (#PCDATA)>
<!ELEMENT italian (#PCDATA)>
```

Listing 6.1: DTD der Bedeutungen

paraN `para`*N* bestimmen die direkt abhängigen Zustandsparameter. In *SpectoML* werden maximal 5 direkt abhängige Zustandsparameter erlaubt. Für weitere oder komplexe Parameter wird der unten beschriebene, erweiterbare Tripel-Mechanismus benutzt.

param `param` (OE) bestimmen Listen von Tripeln (`type`, `name`, `value`). Jedes Tripel beschreibt ein Attribut, das für die Beschreibung des aktuellen Zustands zusätzliche Informationen liefert. `type` bestimmt den Typ, `name` den Namen und `value` den Wert des Attributs. Dieser Tripel-Mechanismus ermöglicht die Verwendung von beliebigen, zusätzlichen Attributen und Parametern für die jeweilige Applikation.

6.2.2 Explizite Darstellung der Bedeutungen

Menschen und Softwaresysteme müssen miteinander kommunizieren. Dabei treten häufig Verständnisprobleme auf, da unterschiedliche Ansichten und unterschiedliche Verständnisse von Begriffen aufeinander treffen. Die Einführung einer sogenannten *Ontologie* ist ein Weg, um sich auf ein einheitliches Vokabular zu einigen. In [Gru93] definiert Gruber den Begriff der Ontologie[13] folgendermaßen:

[13] Ursprünglich entstammt der Begriff *Ontologie* der Philosophie, wo er soviel wie „die Lehre vom

> An ontology is an explicit specification of a conceptualization [Gru93].

Gruber beschreibt eine Ontologie als eine explizite Spezifikation einer Konzeptualisierung. Unter einer Konzeptualisierung ist eine abstrakte und vereinfachte Sichtweise auf einen realen Weltausschnitt zu verstehen. Durch eine Ontologie werden die Ausdrücke bzw. das Vokabular zur Beschreibung dieser Konzeptualisierung definiert.
Eine Ontologie als eine explizite und einheitliche Darstellung von inhaltlichen Bedeutungen unterstützt die *automatisierte* Verarbeitung und Interpretation von Bedeutungen durch Softwareprozesse [FvHH+01]. Dies fördert einen hohen Interaktionsgrad zwischen Softwarekomponenten (vgl. Abschnitt 2.1.2.3 über *automatisierte Arbeitsformen*).
Aus konzeptioneller Sicht unterstützt eine Ontologie den Einsatz eines Mediators in ungleichen bzw. heterogenen Systemlandschaften. In bezug auf das Ziel eines allgemeinen und interoperablen Einsatzes der *SpectoML* für beliebige Applikationen bedeutet dies, daß eine Ontologie die Wiederverwendung von existierenden Komponenten fördert sowie das Wissen um bestehende Zusammenhänge und Zustände der realen Welt für unterschiedliche Applikationen einheitlich zugänglich macht.
Der in der *SpectoML* verfolgte Ansatz für die Einführung einer Ontologie basiert auf dem Prinzip eines *Katalogs*. Ein Katalog für *SpectoML* enthält alle gültigen und möglichen Zustandstypen für eine Applikation (evt. für eine ganze Applikationsdomäne). Dieser Katalog ist seinerseits als XML-Applikation abgelegt. Die DTD für die Einträge der Bedeutungen von Zuständen bzw. für die Festlegung der Zustandstypen ist in Listing 6.1 dargestellt. In Abbildung 6.2 ist exemplarisch ein Ausschnitt der Spezifikation der Zustandstypen für den leichtgewichtigen Applikationsserver *Respondeo* (siehe Kapitel 4) illustriert.
Ein Beispiel eines XML-basierten Katalogs für die automatisierte Verarbeitung von semantischen Inhalten, die in *SpectoML*-Dokumenten abgelegt sind, ist in Abbildung 6.3 veranschaulicht:
Auf der linken Seite der Abbildung 6.3 sind drei *SpectoML*-Dokumente dargestellt, die Informationen über zwei verschiedene Applikationen beinhalten:

- `KR@host1` und `KR@host2` sind der Applikation `A` zugeordnet, d. h. beide sind von der der gleichen Applikation generiert worden. Allerdings handelt es sich bei den zwei XML-Dokumenten um verschiedene Instanzen der gleichen Applikation: Während `KR@host1` auf Rechner `host1` gelaufen ist, ist `KR@host2` auf dem Rechner `host2` gestartet worden.

- `KR@host3` spiegelt die Zustände einer Instanz der Applikation `B` wider, die auf dem Host `host3` abgelaufen ist.

Sein, von den Ordnungs-, Begriffs- und Wesensbestimmungen des Seienden“ [EHWM01] bedeutet.

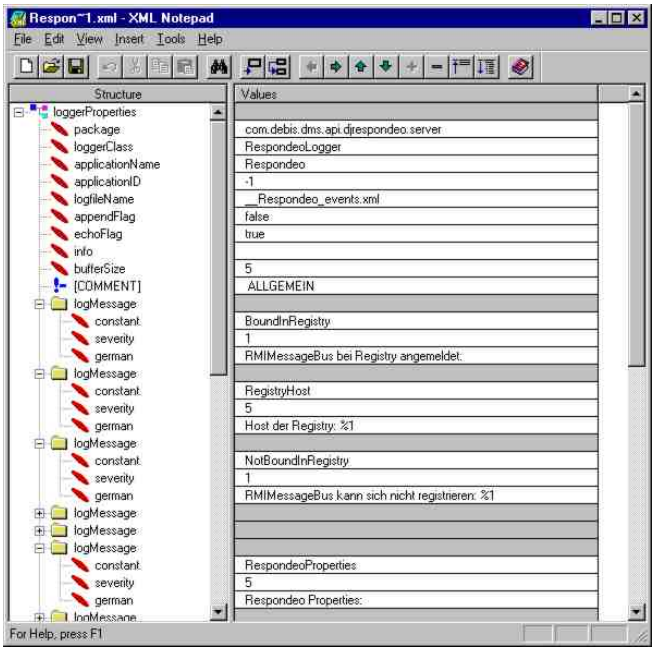


Abbildung 6.2: Spezifizierung der Bedeutungen für *Respondeo.*

Abbildung 6.3 verdeutlicht, daß die Einführung einer einheitlichen und expliziten Beschreibung der Zustandsbedeutungen sinnvoll ist. Softwarekomponenten und Informationssysteme sind in der Lage, unabhängig von einer bestimmten Applikation nicht nur auf einzelne Zustände zuzugreifen, sondern auch deren Bedeutungen zu interpretieren.

6.3 Eigenschaften

In diesem Abschnitt werden die allgemeinen Eigenschaften der *SpectoML* beschrieben:

- *Mehrsprachigkeit.* Die Beschreibung von Zuständen bzw. von Zustandstypen erfolgt transparent in mehreren Sprachen. Die Sprachversion kann dynamisch gewechselt werden, was unterschiedliche Benutzersichten hinsichtlich der verwendeten Zustandbeschreibungen ermöglicht. So wird der gleiche Zustand für Benutzer mit verschiedenen Sprachen gemäß ihrer gewählten Sprachversion dynamisch aufbereitet.

- *Hierarchische Zustände.* Hierarchische Abhängigkeiten zwischen Zuständen werden über den in Abschnitt 6.2.1 beschriebenen Vater-Kind-Mechanismus abgebildet. Diese zusätzlichen Ausdrucksmöglichkeiten erweitern die strukturellen Merkmale der *SpectoML.*

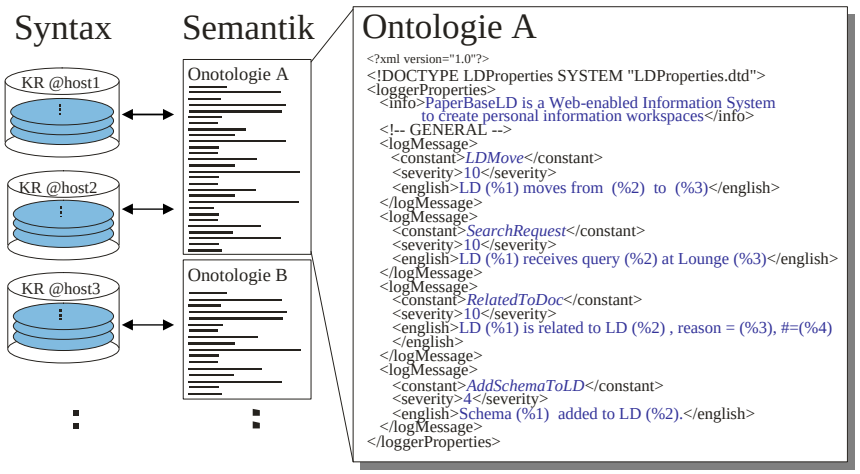


Abbildung 6.3: Zusammenhang zwischen den syntaktischen und semantischen Spezifikationen der Einträge in der *SpectoML*.

- *Erweiterungsfähige Zustandsbeschreibungen*. Standardmäßig werden die Zustandsparameter durch die Attribute `paraN` repräsentiert, wie es in Abschnitt 6.2.1 erläutert wird. Bei einer großen Anzahl an Parametern oder falls sehr komplexe Parameterbeschreibungen erforderlich sind, werden Listen von Tripeln (`type`,`name`,`value`) benutzt (siehe Abschnitt 6.2.1). Der Tripel-Mechanismus ist sehr ausdrucksstark und mächtig. Die Listen können beliebig viele Tripel enthalten.

- *Kompakte XML-Repräsentierung*. Eine kompakte Repräsentierung von Zuständen unterschiedlicher Applikationen erfolgt in der *SpectoML* in mehreren Schritten:

  Zunächst werden Informationen, die alle Zustände betreffen, an einer einzigen Stelle in dem XML-Dokument – der *Präambel* – abgelegt. Die Präambel wird durch ein spezielles XML-Element `preamble`, das in Abschnitt 6.2.1 beschrieben ist, repräsentiert.

  Durch die Einführung einer Präambel wird ein *SpectoML*-Dokument in zwei Teile unterteilt. Neben der Präambel existiert der *Rumpf* eines *SpectoML*-Dokuments, der eine Liste konkreter Zustände enthält.

  Die Ausgangsbasis für die nächsten Schritte bildet der Rumpf der *SpectoML*, der seine Zustände zunächst in einer tabellenartigen Struktur ablegt. Prinzipiell entspricht ein Zustand genau einer Zeile in einer Tabelle, wobei die Spalten die Metadaten bestimmen. In Abbildung 6.4 sind mehrere Transformationen illustriert, die zu einer kompakten und interoperablen Darstellung von Zuständen in der *SpectoML* führen:

1. *Vanilla-Transformation T1.* In `T1` werden die Metadaten `Tabelle1` bzw. `colA` zu Elementen in der XML-Darstellung, wobei die Elemente hierarchisch angeordnet sind. Die Tabellenwerte werden zu Werten der XML-Elemente abgebildet.

   Unterschiedliche Tabellen, die ihren Ursprung in verschiedenen Applikationen haben, besitzen nach der Transformation `T1` unterschiedliche XML-Darstellungen.

2. *Wertobjekt-Transformation T2.* In `T2` werden die Metadaten `Tabelle1` bzw. `colA` zu Wertobjekten transformiert. Die zuvor als Metadaten kodierten Informationen werden nun selbst zu Wertobjekten. Für die Durchführung von `T2` werden neue Elemente eingeführt, die global für alle Metadaten gelten. Dies entspricht der Einführung eines Katalogs bzw. einer Ontologie, die in Abschnitt 6.2.2 näher beschrieben sind.

   Unterschiedliche Darstellungen in der Ausgangsituation haben nach der Transformation von `T1` nach `T2` den gleichen Aufbau. Sie unterscheiden sich nur durch die konkreten, jeweiligen Ausprägungen der Elemente. Dies hat den Vorteil, daß auf die Elemente der XML-Darstellung – unabhängig von der Art der zugrunde liegenden Applikation – uniform zugegriffen werden kann. Die Benutzung von Wertobjekten hat den Vorzug, daß die Schnittstellen (z. B. die Programmierschnittstellen) für die Benutzung der *SpectoML* stabil sind.

   Ein einheitlicher Zugriff auf unterschiedliche Darstellungen ist mit der Middleware-Komponente *Java Database Connectivity* (JDBC) [Sun02b, Ree00] zu vergleichen. JDBC ermöglicht den Zugriff in Java auf Datenquellen wie z. B. auf die Tabellen in relationalen Datenbanken in einer datenbankunabhängigen Art und Weise. JDBC abstrahiert von den konkreten Eigenschaften der Datenquelle, indem spezifische Details der jeweiligen Tabellen oder der Datenbankhersteller durch eine allgemeingültige Beschreibung ausgedrückt werden. So ist beispielsweise der Zugriff auf beliebige Spalten von Tabellen möglich. Eine Spalte, die u. a. durch einen Namen und eine Datentypbeschreibung spezifiziert wird, ist ein Bestandteil der Metadaten in einer Datenbankapplikation.

3. *Kompaktifizierung T3.* Die Transformation `T3` hat eine möglichst kompakte Darstellung des XML-Dokuments zum Ziel. Zu diesem Zweck wurde beim Entwurf der *SpectoML* die Reihenfolge für die Repräsentation von Parametern festgelegt. Dafür wurden an vielen Stellen Attribute anstelle von Elementen in der XML-Darstellung benutzt[14]. Empirische Erfahrungen beim Einsatz der *SpectoML* haben gezeigt, daß die meisten Parameter nur wenig Speicherplatz für ihre Darstellung benötigen. Zudem treten viele Zustände eines bestimmten Zustandstyps

[14] Vgl. die unterschiedliche Darstellung von Parametern `paraN` in `T3`.

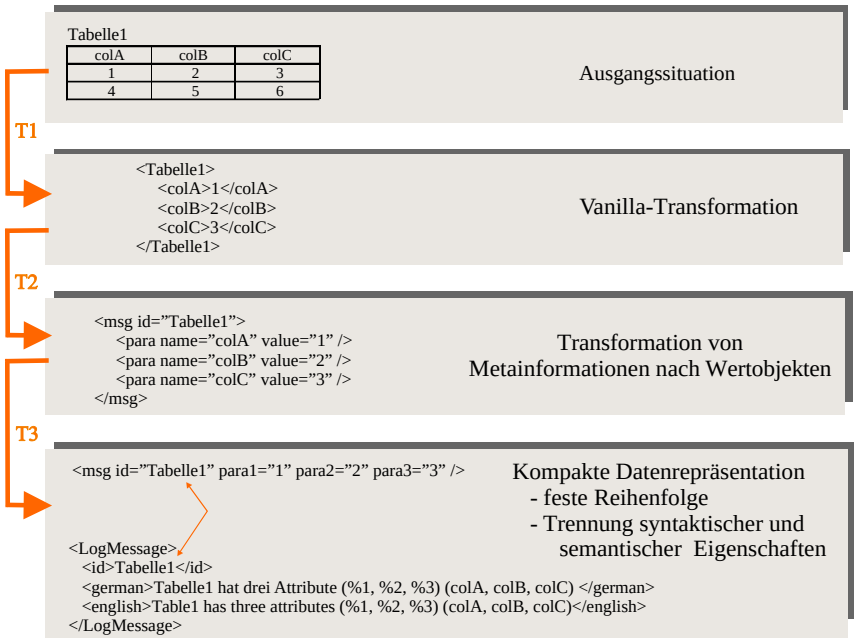


Abbildung 6.4: Transformationen für die kompakte Darstellung von *SpectoML*-Dokumenten.

mehrfach auf, so daß sich gewisse Redundanzen in der Darstellung ergeben.

Des weiteren wurden die semantischen Details der einzelnen Zustände getrennt von der eigentlichen Kodierung des Zustands vorgenommen (vgl. Abschnitt 6.2.2).

Diese beiden Vorgehensweisen führen zu einer kompakten und XML-basierten Repräsentation von Zuständen für verschiedene Applikationen.

Die einzelnen, beschriebenen Transformationen sind nicht voneinander abhängig und können somit auch separat durchgeführt werden.

6.4 Anfragesprache für *SpectoML*-Dokumente

Um auf die in XML-Dokumenten enthaltenen Informationen zuzugreifen, werden sogenannte *XML-Prozessoren* und *XML-Anfragesprachen* benötigt. Ein XML-Prozessor bearbeitet ein XML-Dokument und hat so Zugriff auf dessen strukturelle und inhaltliche Bestandteile wie z. B. einzelne XML-Elemente und deren konkrete Daten. Eine Anfragesprache ermöglicht einen strukturellen Zugriff und die Ex-

traktion von Daten, indem Ausdrücke und Funktionen auf das XML-Dokument angewandt werden.
Bestehende Anfragesprachen für XML-Dokumente – wie z. B. *XQuery* [CCF+01], *XQuilt* [RCF00, CRF00], *XML-QL* [DFF+98] – basieren auf *XML Path Language* (XPath) [CD99] des W3C[15]. Das primäre Ziel von XPath ist die Adressierung einzelner Teile eines XML-Dokuments. Auf Basis des in der XPath-Spezifikation festgelegten Adressierungsmechanismus werden die Daten aus XML-Dokumenten in einer strukturierten Art und Weise extrahiert[16].
Für die Bearbeitung der Anfragen an XML-Dokumente sind XML-Prozessoren nötig. Da XPath zunächst nur eine Spezifikation und keine Implementierungen beinhaltet, existieren verschiedene Implementierungen der XML-Prozessoren bzw. der Anfragesprachen für unterschiedliche Plattformen. Solch ein Anfragesystem gilt für alle Auszeichnungssprachen auf der Basis von XML und ist nicht auf eine bestimmte Menge von Auszeichnungssprachen beschränkt.
Reduziert man den generellen Einsatz eines XML-Prozessors und einer Anfragesprache für XML-Dokumente auf eine bestimmte Auszeichnungssprache – wie z. B. *SpectoML* – ergeben sich mehrere potentielle Vorteile, die im folgenden aufgeführt sind:

- *Leichtgewichtiger XML-Prozessor*. Der Einsatz einer speziell an *SpectoML* und deren Eigenschaften (siehe Beschreibung der DTD in Abschnitt 6.2.1) angepaßten Anfragesprache reduziert zwangsläufig die Codegröße des dazugehörigen XML-Prozessors, da nur eine Untermenge der in XML möglichen Beschreibungs- und Strukturmerkmale benutzt wird. Die geringe Codegröße ermöglicht die direkte Einbindung in andere Softwarekomponenten[17].

- *Geschwindigkeit*. Der spezielle Aufbau der DTD ermöglicht einen optimierten Zugriff auf die einzelnen Bestandteile eines *SpectoML*-Dokuments. Dadurch läßt sich neben der geringen Größe auch ein performanter XML-Prozessor entwerfen und implementieren.

Der Nachteil der eingeschränkten Verwendung eines proprietären XML-Prozessors sowie einer stark angepaßten Anfragesprache relativiert sich für *SpectoML*, weil die Geschwindigkeit und vor allem die Größe die dominierenden Faktoren für die Verwendung von *SpectoML* sind.
Der für *SpectoML* entworfene XML-Prozessor ist ein in Java implementierter Parser, der mit Hilfe von JavaCC (*Java Compiler Compiler*)[18] entwickelt wurde.

[15] *XQuery* basiert im wesentlichen auf den Eigenschaften und Konzepten von *XQuilt* und wird vom W3C standardisiert.

[16] Für eine detaillierte Beschreibung der Datenextraktion und des Adressierungsmechanismus wird auf die Spezifikation [CD99] von XPath vom W3C verwiesen.

[17] In Teil IV über *Living Documents* werden Zustandsinformationen von Agenten und deren Umgebung in *SpectoML*-Dokumenten abgelegt und direkt in jeden Agenten eingebunden. Eine geringe Codegröße ist für einen flexiblen und breiten Einsatz dieser Agenten unerläßlich.

[18] Siehe auch die Homepage von JavaCC unter http://www.webgain.com/products/java_cc.

JavaCC ist ein Generator für Parser, die in Java-Applikationen eingesetzt werden. Generell ist ein Parsergenerator ein Werkzeug, das auf der Basis einer spezifizierten Grammatik ein Programm (Parser) erstellt.
Dieses Programm überprüft die Eingabedaten hinsichtlich dieser Grammatik. Die Eingabedaten entsprechen dabei einer Anfrage an ein *SpectoML*-Dokument, wobei die Anfrage der in der Grammatik spezifizierten Anfragesprache genügen muß. Abbildungen und Listings der Parserdefinitionen bzw. Grammatiken für die Anfragesprache in Bezug auf *SpectoML*-Dokumente sind im Anhang A dargestellt.
In Abschnitt 8.4 über *XRePro* werden auf der Basis von *Prolog* [SS86, CH01] weitere, mächtige Anfragewerkzeuge für XML-Dokumente im allgemeinen und für *SpectoML*-Dokumente im besonderen erörtert.

6.5 Einsatzgebiete

In diesem Abschnitt werden einige der Applikationsdomänen beschrieben, in denen *SpectoML* für die Beschreibung von Zuständen auf der Ebene der Applikationen zum Einsatz kommt[19]. Die Heterogenität bzw. Unterschiedlichkeit in den vorgestellten Applikationsdomänen unterstreicht die breite und generelle Anwendbarkeit der *SpectoML.*

6.5.1 Monitoring von Applikationen

Das Monitoring von Applikationen umfaßt die dynamische *Generierung*, *Interpretation* und *Präsentation* von Informationen, die in direktem Zusammenhang mit den zu beobachteten Softwarekomponenten und -systemen stehen [JLSU87]. Den komplexen Prozeß des Monitorings unterstützt die *SpectoML* auf die folgende Art und Weise:

- *Generierung*. Die für das Monitoring wichtigen Informationen werden als Zustände gemäß der *SpectoML* in XML abgelegt. Die Benutzung von XML als Auszeichnungsprache für die Zustände ist unabhängig von Systemplattformen. Der Aufbau der *SpectoML* ermöglicht den flexiblen Einsatz in unterschiedlichen Applikationen (vgl. Abschnitte 6.2.1 und 6.3).

- *Interpretation*. Die in *SpectoML*-Dokumenten enthaltenen Informationen werden über den im Abschnitt 6.4 beschriebenen, speziellen XML-Prozessor und die Anfragesprache interpretiert.

- *Präsentation.* Die strukturierte Interpretation und Verarbeitung der Zustandsinformationen dient als Basis zur Erzeugung von dynamischen

[19] Zudem bildet *SpectoML* die Basis für die Beschreibung unterschiedlicher Anwendungs- und Umgebungskontexten für die in Teil IV diskutierten *Living Documents.*

Sichten auf ein *SpectoML*-Dokument. Eine dynamische Sicht liefert ein zur Laufzeit generiertes Abbild der zugrunde liegenden Informationen[20].

In Kapitel 7 über das Monitoring-Framework *Specto* [SHKK00, SFK01a] werden die Vor- und Nachteile des Einsatzes der *SpectoML* für das Monitoring in verteilten Informationssystemen diskutiert.

6.5.2 Darstellung von Testdokumenten

Ein *Testdokument* oder *Testreport* enthält aktuelle Informationen über die Ausführung von Tests, die einen wesentlichen Beitrag zur Qualitätssicherung im Software-Entwicklungszyklus leisten. Auf der konzeptionellen Ebene entsprechen die Informationen den Zuständen, die von den Testfällen während der Testausführung eingenommen werden. In dieser Hinsicht gelten die selben Vorteile der *SpectoML*, die im vorhergehenden Abschnitt 6.5.1 für die Applikationsdomäne des Monitorings angeführt sind.
In Abschnitt 8.1 über das Testmanagement-Framework *Tempto* [SSDKK00] wird der Einsatz von *SpectoML* für die Beschreibung, Verarbeitung und Präsentation von Testdokumenten genauer erläutert.

6.5.3 Visualisierung von verteilten Prozessen

Die Ausführung von verteilten Softwareprozessen ist durch einen komplexen und oft nicht vorhersehbaren Ablauf gekennzeichnet. Um potentielle Probleme und Fehler, die bei den Ausführungen derartiger Prozesse auftreten, besser verstehen und nachvollziehen zu können, sind primär folgende Schritte erforderlich:

- Aufbau von geeigneten Datenbasen, welche die Informationen über die Prozeßabläufe enthalten.
- Realisierung eines Informationssystems, dessen grafische Benutzeroberfläche in der Lage ist, komplexe Systemabläufe verständlich zu visualisieren. Die Basis für die Visualisierung bilden die Informationen in den Datenbasen.

Vor diesem Hintergrund wurde im Rahmen einer weiteren Fallstudie für die Einsatzmöglichkeiten der *SpectoML* in verteilten Umgebungen das Softwareframework *Represento* für die Visualisierung von verteilten Prozessen geschaffen. *Represento* [Cas01] wurde im Rahmen einer Diplomarbeit entworfen und in Java implementiert.
Es ist als eigenständiges Softwareframework konzipiert, d. h. *Represento* ist selbst ein Informationssystem, welches unterschiedliche Datenquellen bzw. Datenbasen

[20] Im folgenden Abschnitt 6.5.3 werden beispielsweise die Informationen zur dynamischen Visualisierung von Softwareprozessen in einem Java-Framework benutzt.

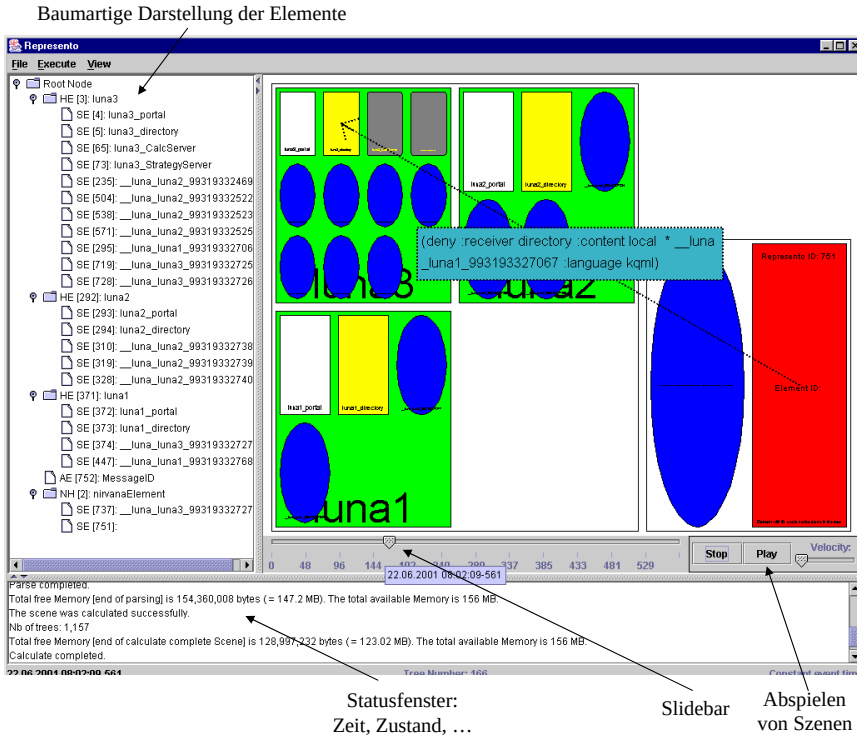


Abbildung 6.5: Übersicht des grafischen Aufbaus von *Represento.*

benutzt. Eine *Datenbasis* innerhalb von *Represento* entspricht einem *SpectoML*-Dokument, welches die Zustände und Eigenschaften des jeweiligen Softwareprozesses enthält. *SpectoML* bildet die Systemschnittstelle zwischen dem Softwareframework *Represento*s und dem entsprechenden Softwareprozeß.
Bei der Visualisierung in *Represento* werden anspruchsvolle, grafische Elemente mit in den Darstellungsprozeß einbezogen, wie es in Abbildung 6.5 veranschaulicht ist. So erhält der Benutzer z. B. ein Filmwerkzeug an die Hand, welches ihm das Abspielen von beliebigen Zustandssequenzen auf einfache Art und Weise ermöglicht. Zudem existieren mehrere Filtermöglichkeiten in *Represento*, um das Volumen der visualisierten Informationen entsprechend anzupassen. Ein Filter definiert eine Menge von Anforderungen, die von den Daten in den *SpectoML*-Dokumenten erfüllt werden müssen.

Teil III

Fallstudien

Kapitel 7

Specto – eine flexible Monitoring-Infrastruktur

Specto [SHKK00, Häu99, SFK01a][1] stellt Infrastrukturen zum Monitoring von Applikationen in verteilten Systemen zur Verfügung. Abdu et al. [ALB96] bezeichnen Monitoring als wichtigen Prozeß, in dem Informationen über benötigte Softwarekomponenten eines verteilten Systems gesammelt und aufbereitet werden. Das Monitoring bildet so die Basis, um Kontrollentscheidungen für die korrekte Ausführung der Softwarekomponenten zu treffen.
Ferner wird von folgender Definition von Monitoring gemäß Joyce et al. [JLSU87] ausgegangen:

> Monitoring ist the process of dynamic collection, interpretation and presentation of information concerning objects or software processes under scrutiny.

Im Mittelpunkt des Monitorings steht das zu beobachtende und zu verwaltende Objekt (VO). Ein VO bildet die Abstraktion für reale Objekte oder Softwareprozesse. Die über ein VO gesammelten Informationen entsprechen einzelnen Zuständen, die von einem VO durchlaufen werden. Gemäß der Definition von Joyce et al. ist es die Aufgabe des Monitorings, die Zustände eines VOs zu *sammeln*, korrekt zu *interpretieren* und schließlich dem Benutzer in einer angemessenen Form zu *präsentieren*.
Die Monitoring-Infrastruktur *Specto* besteht aus mehreren Komponenten, die in den folgenden Abschnitten näher erläutert werden:

- Die Zustände eines VOs werden gemäß der *SpectoML* (vgl. Kapitel 6) einheitlich und strukturiert in XML [BPSMM00] beschrieben. Snodgrass

[1] *spectare* (lat.) – sehen, betrachten, anschauen; *specto* (1. Person Singular) – *ich betrachte*. Durch den Namen *Specto* soll die Intention zum Ausdruck gebracht werden, die Zustände in komplexen, verteilten Umgebungen und die darin enthaltenen Systemkomponenten transparenter erscheinen zu lassen.

[Sno88] hinterlegt die Zustände ebenfalls in einem strukturierten, auf dem relationalen Modell [Cod70] basierenden Format. Während Snodgrass eine zentrale Datenbank für die Verwaltung aller Zustände benutzt, stellt *Specto* keine solchen restriktiven Anforderungen an die Persistenz der Monitoring-Informationen[2].

- Der Applikationsserver *Respondeo* (vgl. Kapitel 4) dient als Mediator (vgl. Abschnitt 2.2) für die Verwaltung der Monitoring-Informationen in *Specto.*

- Jedes VO wird mit den zuvor spezifizierten Zuständen instrumentalisiert (vgl. Abschnitt 7.2.1). Dafür stellt *Specto* eine Programmierschnittstelle (*Application Programming Interface – API*) zur Verfügung, welche die Benutzung der *SpectoML*-basierten Zustände auf der Ebene der Programmiersprache erleichtert[3].

Ein wichtiger Aspekt beim Monitoring ist die Miteinbeziehung umweltrelevanter Informationen. Zambonelli [Zam01] führt unter dem Schlagwort *situatedness* an, daß die Modellierung der Umgebung eines Informationssystems ihrerseits sehr dynamisch sein kann. Deswegen sollten keine Annahmen getroffen werden, welche die Softwaresysteme – Applikationen, die vom Monitoring-System beobachtet werden – und die dazugehörige Umgebung konzeptionell eng miteinander verbinden. Zambonelli argumentiert, daß die Monitoring-Infrastruktur derartig flexibel sein sollte, daß hinsichtlich der Eigenschaften der Umgebung – wie z. B. die Wahl der Systemplattform und der Programmiersprache – keine Einschränkungen gemacht werden sollten. Der Entwurf und die Realisierung von *Specto* entspricht diesen Kriterien, wie es in den folgenden Abschnitten erörtert wird.

7.1 Ziele

In diesem Abschnitt werden die generellen Einsatzmöglichkeiten von *Specto* als Monitoring-Infrastruktur beschrieben. Außerdem werden die spezifischen Eigenheiten und Ziele, welche dem Entwurf von *Specto* zugrunde liegen, erläutert.
Der Einsatz der Monitoring-Infrastruktur *Specto* erstreckt sich u. a. über die folgenden Anwendungsmöglichkeiten:

- *Wartung*. Hinsichtlich der Wartung liefert *Specto* wertvolle Informationen, über die Zustände und das Verhalten der beobachteten Softwaresysteme bzw.

[2] Generell ermöglicht *Specto* die Benutzung einer Datenbank zur Verwaltung der XML-basierten Monitoring-Informationen. Die Systemarchitektur *Specto*s ist aber nicht an die Existenz einer solchen Systemkomponente gebunden.

[3] Zum aktuellen Stand der Implementierung bietet *Specto* eine Java- [AGH00] und C++-API [Str97] an. Dabei beinhaltet die C++-API nur einen limitierten Funktionsumfang. Da *Specto* an keine bestimmte Programmiersprache gebunden ist, können weitere APIs einfach hinzugefügt werden.

Applikationen. Auf der Basis der Monitoring-Informationen können fehlerhafte Komponenten und Systemabläufe erkannt werden. Insbesondere in dynamischen Systemen, die sich durch irreguläre Abläufe und Interaktionen auszeichnen, ist es häufig schwierig, ohne eine adäquate Monitoring-Infrastruktur Aussagen über die Fehlerquellen treffen zu können [SFK01a].

- *Umwelt*. Unter dem Begriff Umwelt sind allgemein die Softwaresysteme und -komponenten zusammengefaßt, die aus dem Blickwinkel einer bestimmten Applikation wertvolle zusätzliche Informationen für dessen Ablauf liefern können. Dafür ist es entscheidend, daß die Umweltzustände festgehalten und an interessierte Applikationen automatisiert weitergereicht werden. Der allgemeine Monitoring-Ansatz von *Specto* unterstützt ein solches Vorgehen, wie es u. a. in Schimkat et al. [SFK01a] unter Verwendung von *SpectoML*-Dokumenten erörtert wird.

  Im Zusammenhang mit proaktiven Informationssystemen, die in Kapitel 9 erörtert werden, spielen Umweltzustände und deren Veränderungen eine wichtige Rolle für die Abläufe in Informationssystemen. Neben der Wahrnehmung ist auch die korrekte Interpretation der Umweltzustände von Bedeutung. Nwana und Ndumu [NN99] heben ebenfalls die Bedeutung des Monitorings für automatisierte Interaktionen zwischen Softwareprozessen – wie z. B. Agenten (vgl. Abschnitt 3.3.1) – und ihrer Umwelt hervor.

- *Programmvisualisierung*. Die Monitoring-Informationen enthalten das Wissen um die jeweiligen Zustände der Applikationen. In komplexen und vor allem dynamischen Systemen ist es hilfreich, Systemabläufe und -zustände grafisch zu visualisieren, um beispielsweise ein besseres Verständnis von potentiell auftretenden Fehlern zu bekommen. Auf der Grundlage der Monitoring-Informationen, die in *SpectoML*-Dokumenten beschrieben sind, wurde im Rahmen einer Diplomarbeit *Represento* [Cas01] – ein Werkzeug für die grafische und animierte Visualisierung von dynamischen Prozessen – entwickelt (vgl. Abschnitt 6.5.3).

Im folgenden werden die konkreten Ziele beim Entwurf und der Realisierung von *Specto* als flexibel einsetzbare Monitoring-Infrastruktur beschrieben:

- *Einfache Benutzung* und *simple Integration* in bestehende Softwaresysteme. Der Begriff der Benutzung bezieht sich dabei einerseits auf den Entwickler, der *Specto* in die betreffende Applikation einbindet, und andererseits auf den Benutzer, der in einem zeitlich nachgelagerten Prozeß auf die erzeugten Monitoring-Informationen zugreift und diese auswertet[4].

- *Standardisiertes Datenformat* für die Beschreibung von Monitoring-Informationen. Die Benutzung von XML als Beschreibungsformat für die

[4] In Abbildung 7.1 ist der prinzipielle Ablauf für die Bestimmung von Monitoring-Informationen und ihre Einbindung in die jeweilige Applikation illustriert.

Informationen, die Aussagen über das VO treffen, ermöglicht eine standardisierte Darstellung. In *Specto* wird die Auszeichnungssprache *SpectoML* (vgl. Kapitel 6) als Beschreibungsformat für Monitoring-Informationen benutzt.

- *Automatisierte Kontroll- und Interaktionsmöglichkeiten* mit dem Monitoring-System. Dies ist insbesondere von Bedeutung, wenn Softwareprozesse wie z. B. Agenten mit dem Monitoring-System interagieren und kommunizieren. Durch diesen Automatisierungsschritt sind Softwareprozesse in der Lage, ihre Umwelt zu beobachten und beim Eintreten bestimmter Umweltzustände entsprechend zu reagieren.

- Zugriff über *web-basierte Schnittstellen*. Der client-seitige Zugriff eines Benutzers auf die Monitoring-Informationen erfolgt über das Web. Dafür bietet *Specto* eine web-basierte, grafische Benutzerschnittstelle, wie sie in Abbildung 7.4 dargestellt ist. Anfragen des Benutzers werden über den Applikationsserver *Respondeo* (vgl. Kapitel 4) an die entsprechenden Monitoring-Informationen weitergeleitet. Für eine automatisierte Bearbeitung werden die generierten Monitoring-Informationen via E-Mail bzw. SMTP [Kle01] an entsprechende Adressaten weitergeleitet.

- *Unabhängigkeit von speziellen Applikationsdomänen*. Die Monitoring-Infrastruktur ist nicht an spezielle Eigenschaften von Applikationsdomänen gebunden. So gehen beispielsweise keine domänenabhängige Eigenschaften in das Beschreibungsformat für die Monitoring-Informationen mit ein. *SpectoML* verwendet allgemein benutzbare Elemente in ihrem Aufbau. Es gibt keine *Tags*, die speziell für bestimmte Domänen konzipiert sind (vgl. Abschnitt 6.1 über die *Ziele* beim Entwurf der *SpectoML* und Abschnitt 6.2.2 über die *explizite Darstellung* von Zustandsmeldungen).

- *Unabhängigkeit von speziellen Programmiersprachen*. Die Kodierung der Monitoring-Information erfolgt nicht in der Programmiersprache des VOs, sondern in einem separaten, standardisierten Beschreibungsformat. Durch diese Trennung zwischen der Programmiersprache und dem Beschreibungsformat für Monitoring-Informationen ist *Specto* nicht an spezielle Programmiersprachen gebunden.

7.2 Monitoring-Modell

Ein Objekt, das durch *Specto* beobachtet und verwaltet wird, wird als VO bezeichnet. Typischerweise ist ein VO eine Applikation oder ein Softwareprozeß. Die aktuellen Zustände eines VOs werden in Form von sogenannten Logmeldungen festgehalten. Eine Logmeldung entspricht einer Monitoring-Information bezüglich eines bestimmten VOs. Jede Logmeldung in *Specto* ist ein *Zustand* gemäß

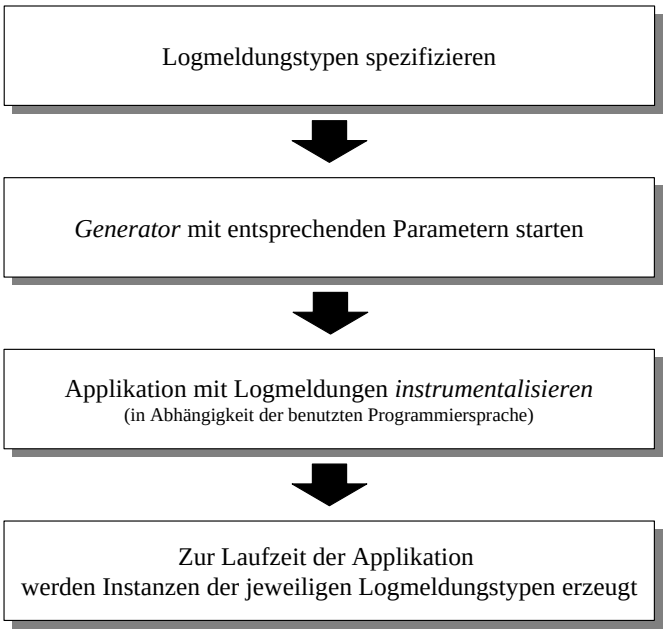


Abbildung 7.1: Prinzipieller Ablauf zur Generierung von Monitoring-Informationen bzw. Logmeldungen.

der in Abschnitt 6.2 vorgenommen Begriffsdefinition in *SpectoML*-Dokumenten. Prinzipiell übernehmen Logmeldungen in *Specto* alle Eigenschaften – wie z. B. die Mehrsprachigkeit und die hierarchische Zuordnung – von *SpectoML*-Dokumenten, die in Abschnitt 6.3 beschrieben sind.
Ein Benutzer, der an dieser Information interessiert ist, versucht auf diese Logmeldungen zuzugreifen und die Logmeldungen entsprechend zu interpretieren. Beim Benutzer wird prinzipiell zwischen einem menschlichen und einem digitalen Benutzer unterschieden (vgl. Abschnitt 2.1.1 über den schematischen Aufbau von Informationssystemen). Der Prozeß der Interpretation der Logmeldungen bzw. der Zustände orientiert sich an dem Vorgehen zum strukturierten Anfragen der Informationen in *SpectoML*-Dokumenten (vgl. Abschnitt 6.4).
Das in diesem Abschnitt erläuterte Monitoring-Modell von *Specto* basiert auf Joyce et al. [JLSU87] und enthält die drei Phasen der *Generierung*, der *Verwaltung* und schließlich der *Präsentation* der Monitoring-Informationen. In den folgenden Abschnitten werden die einzelnen Phasen genauer erläutert.

7.2.1 Generierung

In Abbildung 7.1 ist der prinzipielle Ablauf der Generierung von Monitoring-Informationen bzw. Logmeldungen dargestellt. Der Ablauf ist durch eine einheitliche und systematische Methodik gekennzeichnet:

```
<?xml version="1.0"?>

<!DOCTYPE loggerProperties SYSTEM"loggerProperties.dtd">

<loggerProperties>
  <package>com.api.djenterprise.appserv</package>
  <loggerClass>RespondeoLogger</loggerClass>
  <applicationName>Respondeo</applicationName>
  <applicationID>-1</applicationID>
  <logfileName>Respondeo_events.xml</logfileName>
  <appendFlag>false</appendFlag>
  <info>Lightweight Application Server</info>
  <bufferSize>5</bufferSize>

  <!-- ALLGEMEIN -->
  <logMessage>
     <constant>BoundInRegistry</constant>
     <severity>1</severity>
     <german>MessageBus bei Registry angemeldet.</german>
  </logMessage>

  <logMessage>
    <constant>RegistryHost</constant>
    <severity>5</severity>
    <german>Host der Registry: %1</german>
  </logMessage>

...

</loggerProperties>
```

Listing 7.1: Ein Beispiel für Logmeldungstypen im Applikationsserver *Respondeo*.

Logmeldungstypen spezifizieren Im ersten Schritt spezifiziert der Entwickler die Logmeldungstypen, die grundsätzlich für das Monitoring eines VOs relevant sind. Dafür definiert er eine Menge von Zustandstypen gemäß der *SpectoML* und bestimmt deren Semantik. Die explizite Beschreibung der inhaltlichen Bedeutungen erfolgt gemäß der DTD in Listing 6.1 in Abschnitt 6.2.2.

Listing 7.1 dient als ein einfaches Beispiel für die Spezifikation von zwei Logmeldungstypen (`BoundInRegistry` und `RegistryHost`), die für den Applikationsserver *Respondeo* relevant sind. Insgesamt enthält *Respondeo* 20 verschiedene Logmeldungstypen, die alle einheitlich in XML beschrieben sind.

Generator Der Generator erzeugt auf der Basis der im vorherigen Schritt spezifizierten Logmeldungstypen und unter Berücksichtigung der Sprachversion eine API für die betreffende Programmiersprache des VOs. Diese automatisch generierte Programmierschnittstelle erleichtert die nächste Phase der Instrumentalisierung.

So wird beispielsweise für den Applikationsserver *Respondeo* eine Java-

```
...
try {
   MessageBusImpl obj = new MessageBusImpl(properties);
   Naming.rebind("//myHOST/MessageBus", obj);
   RespondeoLogger.log(RespondeoLogger.RegistryHost, "myHOST");
} catch (Exception e) {
   ...
   throw new Exception("Network Problem");
}
...
```

Listing 7.2: Ein Beispiel für die Instrumentalisierung der Logmeldungen im Applikationsserver *Respondeo*.

Klasse `RespondeoLogger.java`, in welcher die in Listing 7.1 spezifizierten Logmeldungstypen in entsprechende Datenstrukturen und Objekte der Programmiersprache Java transformiert worden sind, erzeugt.

Instrumentalisierung In dieser Phase instrumentalisiert der Entwickler das betreffende VO mit den bereits spezifizierten Logmeldungstypen. Durch entsprechende Aufrufe der API werden an den jeweiligen Programmstellen des VOs die gewünschten Zustände gemäß der *SpectoML* hinterlegt.

In Listing 7.2 wird der Programmcode von *Respondeo* u. a. mit der Logmeldung `RespondeoLogger.RegistryHost` instrumentalisiert.

Erzeugung von konkreten Logmeldungen Falls zur Laufzeit ein VO die in der Phase der Instrumentalisierung bestimmten Zustände durchläuft, werden konkrete Instanzen der entsprechenden Logmeldungen erzeugt und mit den notwendigen Laufzeitparametern versehen.

Zur Laufzeit wird die Logmeldung `RespondeoLogger.RegistryHost` mit dem Parameter `myHOST` versehen (vgl. Listing 7.2).

In den ersten drei Phasen des Generierungsprozesses geht es um die prinzipielle Bestimmung der Monitoring-Informationen hinsichtlich eines VOs. Dabei werden die Monitoring-Informationen einheitlich gemäß der *SpectoML* beschrieben. In der letzten Phase werden schließlich die aktuellen Zustände des VOs erfaßt und in XML abgelegt.
Die erste Phase der Spezifizierung der Logmeldungstypen ist plattform- und programmiersprachenunabhängig, da die Bestimmung in XML erfolgt. Der Generator in der zweiten Phase unterstützt den Entwickler bei der Instrumentalisierung eines VOs hinsichtlich einer bestimmten Programmiersprache. Sofern bereits ein Generator für die Programmiersprache des VOs existiert, entsteht für den Entwickler praktisch kein zusätzlicher Aufwand für die Transformation der Logmeldungstypen. Die Zustände in der letzten Phase sind ebenfalls wiederum unabhängig von der gewählten Plattform und Programmiersprache, da sie in *SpectoML*-Dokumenten abgelegt werden.

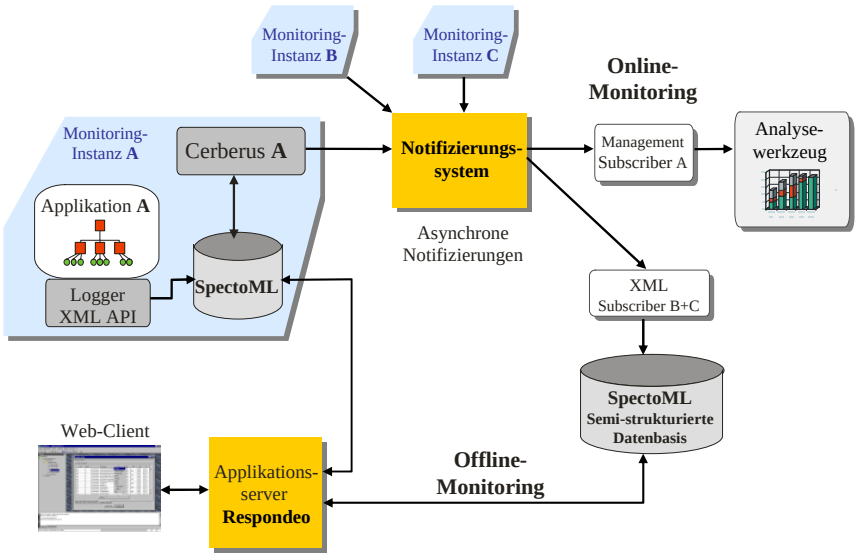


Abbildung 7.2: Systemüberlick der Monitoring-Infrastrukturen von *Specto.*

7.2.2 Verwaltung

In der Verwaltungsphase des Monitoring-Modells von *Specto* werden die erzeugten Monitoring-Informationen weiterverarbeitet, damit der Zugriff von anderen Systemkomponenten auf die Informationen erleichtert wird. Eine sogenannte *Monitoring-Instanz* ist für die Verwaltung und Verteilung seiner Informationen zuständig (vgl. Abbildung 7.2). Eine Monitoring-Instanz besteht aus einer Menge von VOs (`Applikation A`), die über die automatisch erzeugte Programmierschnittstelle (`Logger XML API`) ihre Zustände in einem *SpectoML*-Dokument ablegen (vgl. Abschnitt 7.2.1). Ein sogenannter *Cerberus* ist für die Weiterleitung dieser Zustände innerhalb eines verteilten Systems verantwortlich. Zur Laufzeit der `Applikation A` publiziert ein Cerberus die aktuellen Zustände in ein Notifizierungssystem. Im Rahmen dieser Fallstudie wurde das nachrichtenbasierte Notifizierungssystem *Mitto* entwickelt, das ähnliche Funktionalitäten bietet wie *Siena* [CRW01, CRW00][5]. *Mitto* basiert auf dem sogenannten *publish/subscribe* Kommunikationsparadigma. Die Cerberi der jeweiligen Monitoring-Instanzen treten dabei als Sender von Nachrichten, die in *Mitto* publiziert werden, auf. Desweiteren gibt es eine Menge von Empfängern (z. B. `Managment Subscriber A` und `XML Subscriber B+C`), die sich für bestimmte Nachrichten interessieren. Generell bestimmt ein Empfänger über eine Menge von Filtern diejenigen

[5]Im Zusammenhang mit den Fallstudien bezüglich der *Living Documents* in Kapitel 10 wurde *Siena* als nachrichtenbasierte Notifizierungssystem eingesetzt. Generell bietet *Siena* eine größere Palette an Funktionalitäten zum Publizieren von Nachrichten als *Mitto.* Außerdem basiert *Siena* auf einer dezentralen Systemarchitektur, welches ihren Einsatz in den Fallstudien zu *Living Documents* rechtfertigt. Für das Monitoring in *Specto* sind prinzipiell beide Notifizierungssysteme einsetzbar.

Nachrichten, die *Mitto* an ihn weiterleiten soll. In *Specto* enthält jede publizierte Nachricht eine Menge von Zuständen, die zu einem bestimmten VO bzw. zu einer Monitoring-Instanz gehören.
Die in Abbildung 7.2 dargestellten Empfänger sind typisch für den Ablauf innerhalb des sogenannten *Online-Monitorings* in *Specto.* Beim Online-Monitoring werden die Zustände der VOs *zur Laufzeit* und *asynchron* an die potentiellen Empfänger weitergeleitet. Im folgenden werden zwei typische Arten von Empfängern in *Specto* näher beschrieben. Für beide Arten ist es u. a. kennzeichnend, daß die Sender (`Monitoring-Instanzen A`, `B` und `C`) und die Empfänger in einem verteilten System organisiert sind:

- Der `Management Subscriber A` ist ausschließlich an den Zuständen der `Monitoring-Instanz A` interessiert. Beim asynchronen Eintreffen einer neuen Monitoring-Information wird diese in eine Software zur Visualisierung der aktuellen Zustände der `Monitoring Instanz A` eingebunden.

- Der `XML Subscriber B+C` interessiert sich für die Zustände der `Monitoring-Instanzen A` und `B`. Sein Ziel ist die Generierung einer *neuen Datenbasis*, die durch die Aggregation und Zusammenführung der Daten aus den `Monitoring-Instanzen A` und `B` entsteht.

Über das Online-Monitoring in *Specto* ist es möglich, die Zustände beliebiger Systemkomponenten und Softwareprozessen in aktueller Form zu verteilen. Deswegen eignet sich *Specto* als Infrastruktur für die flexible Integration von sogenannten *umweltrelevanten Informationen* in Informationssystemen. Umweltrelevante Informationen ergeben sich aus den publizierten Zuständen der jeweiligen Monitoring-Instanzen. Systemkomponenten, welche die Rolle eines Empfängers beim Online-Monitoring einnehmen, sind so in der Lage, ihre (dynamische) Umgebung „wahrzunehmen“ und in entsprechender Form auf Umwelteinflüsse zu reagieren. Die Einbettung umweltrelevanter Einflüsse ist ein Ziel des Entwurfs von *Specto* gewesen (vgl. Abschnitt 7.1).
Neben der automatisierten Interaktion, die typisch beim Online-Monitoring in *Specto* ist, besteht die Möglichkeit aus der Sicht eines Benutzers, interaktiv auf die Monitoring-Informationen in den *SpectoML*-Dokumenten zuzugreifen. Beim sogenannten *Offline-Monitoring* (siehe Abbildung 7.2) initiiert der Benutzer eine Anfrage über den Applikationsserver *Respondeo,* der in Kapitel 4 ausführlich erörtert wird. *Respondeo* stellt dafür spezielle Dienste zur Verfügung, die einen Zugriff auf die verteilten Monitoring-Informationen über das Web ermöglichen.
Generell sind hinsichtlich der Verwaltung und Verteilung von Monitoring-Informationen in *Specto* zwei Eigenschaften besonders hervorzuheben:

- *Strukturierte Zugriffsmöglichkeiten.* Ein wichtiger Bestandteil für die Interpretation der Monitoring-Informationen in *Specto* ist der strukturierte, datenbankähnliche Zugriff auf die Daten. Über die Anfragekomponente bzw.

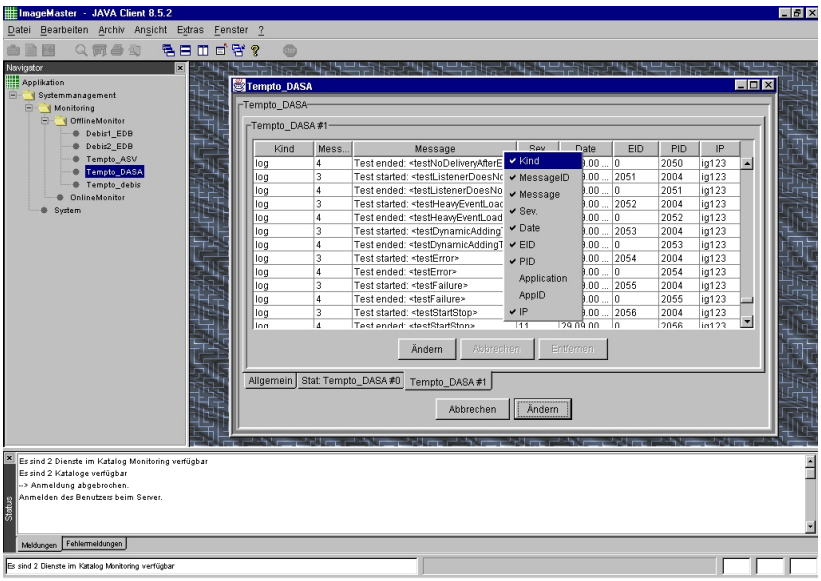


Abbildung 7.3: Visualisierung der *SpectoML*-Dokumente im Systemmanagement-Framework der Fa. T-Systems.

-sprache von *SpectoML* (vgl. Abschnitt 6.4) können mächtige Anfragen sowohl in interaktiver als auch automatisierter Form gestellt werden.

- *Filtern von Monitoring-Informationen*. Um das auftretende Datenvolumen hinsichtlich der generierten Monitoring-Informationen beim Online-Monitoring zu reduzieren, bestimmt jeder Empfänger eine Menge von Filtern. Dadurch werden effektiv nur diejenigen Monitoring-Informationen von *Mitto* weitergeleitet, die auch tatsächlich im Interesse des jeweiligen Empfängers sind.

7.2.3 Präsentation und Visualisierung

In dieser Phase des Monitoring-Modells werden die gesammelten und verarbeiteten Monitoring-Informationen den Benutzern präsentiert. Ein Benutzer – z. B. ein Administrator, der für die Überwachung einer Menge von VOs verantwortlich ist – interagiert mit *Specto* über eine Benutzerschnittstelle. Generell sind seine Interaktionen dadurch gekennzeichnet, daß er bestimmte Monitoring-Informationen, die ihm einen Einblick über Zustände und Abläufe im System geben, nachfrägt. Die Ergebnisse seiner initiierten Anfragen werden *unmittelbar* und *synchron* ausgeführt. In der Benutzerschnittstelle von *Specto* (siehe Abbildung 7.4) werden die Ergebnisse, sobald sie verfügbar sind, entsprechend visualisiert (vgl. Abschnitt 3.1.1 über die Merkmale einer *interaktiven Benutzersicht*).

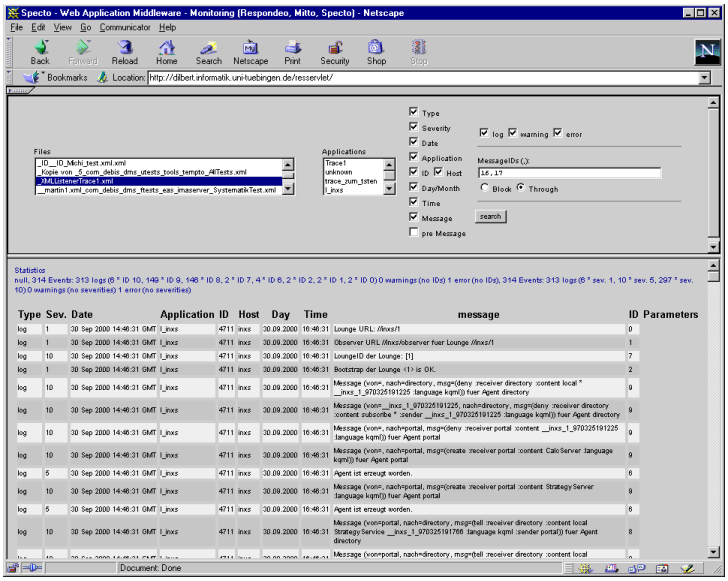


Abbildung 7.4: Visualisierung der Monitoring-Informationen in HTML.

Grundsätzlich gibt es vielfältige Möglichkeiten einem Benutzer, die Monitoring-Informationen auf der Basis von *SpectoML*-Dokumenten zu präsentieren. Im folgenden werden zwei verschiedene Visualisierungsformen auf der Basis von HTML [RHJ99] und Java beschrieben:

- In Abbildung 7.3 ist die Visualisierung der Monitoring-Informationen auf der Basis von *SpectoML* in Java illustriert. *Specto* wurde dafür in das Systemmanagement-Framework, das in Java implementiert ist, der Fa. *T-Systems* integriert.

  Im Navigationsbereich bestimmt der Benutzer das *SpectoML*-Dokument des jeweiligen VOs. Die Monitoring-Informationen des ausgewählten VOs werden tabellarisch im Arbeitsbereich der Benutzerschnittstelle gemäß Abbildung 7.3 aufgelistet.

- In Abbildung 7.4 ist die Darstellung der Monitoring-Informationen in HTML illustriert. Generell bietet *Specto* die Möglichkeit über das Web auf die Monitoring-Informationen zuzugreifen (vgl. Abschnitt 7.2.2).

  Die generelle Benutzerführung des Web-Clients von *Specto* ist ähnlich realisiert wie der Java-Client. Nachdem der Benutzer über das Web ein VO ausgewählt hat, werden die entsprechenden Monitoring-Informationen tabellarisch in HTML dargestellt.

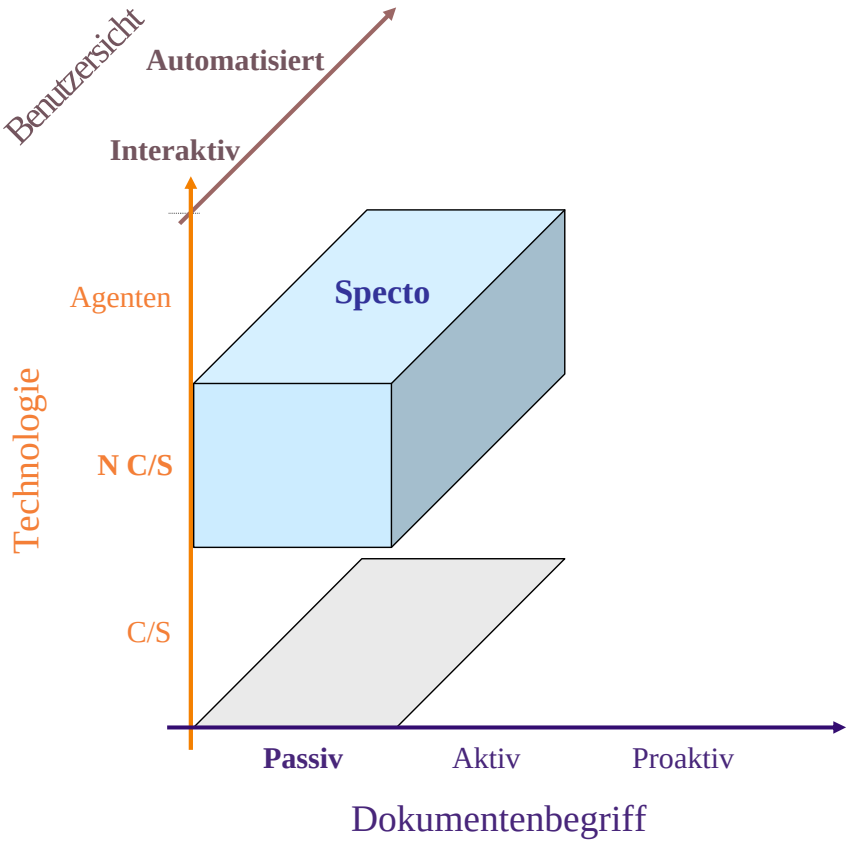


Abbildung 7.5: Einordnung von *Specto* in die Taxonomie.

7.3 Einordnung in die Taxonomie

Die in Kapitel 3 erörterte Taxonomie klassifiziert Informationssysteme hinsichtlich der *Benutzersicht* (vgl. Abschnitt 3.1), des zugrunde liegenden *Dokumentenbegriffs* (vgl. Abschnitt 3.2) und der verwendeten *Technologie* (vgl. Abschnitt 3.3). In Abbildung 7.5 ist die Einordnung *Specto*s in die Taxonomie veranschaulicht. Die Infrastrukturen von *Specto* bilden ein Informationssystem, das hinsichtlich der Einordnung einem Quader entspricht. Dieser erstreckt sich über die Merkmale einer *interaktiven* (vgl. Abschnitt 3.1.1) und *automatisierten* (vgl. Abschnitt 3.1.2) Benutzersicht. Zudem ist *Specto* von einem passiven Dokumentenbegriff (vgl. Abschnitt 3.2.2) und einer mehrstufigen Client/Server-Architektur (vgl. Abschnitt 2.3.1) geprägt:

Benutzersicht *Specto* unterstützt sowohl eine interaktive als auch automatisierte Benutzersicht. Beim Offline-Monitoring (vgl. Abschnitt 7.2.2) stellt der Benutzer über die Benutzerschnittstelle des Clients eine Anfrage, die von *Respondeo* entgegengenommen und weiterverarbeitet wird. Die Ergebnis-

se werden synchron an den Client zurückgeschickt. Der Client präsentiert und visualisiert die Ergebnisse in unmittelbarer Form, das typisch für eine interaktive Benutzersicht ist.

Beim Online-Monitoring (vgl. Abschnitt 7.2.2) in *Specto* treten verschiedene Informations- und Kontrollflüsse, die sich dynamisch zur Laufzeit ergeben, auf. Die Rolle des Initiators von Anfragen an Monitoring-Informationen werden hier u. a. von Softwareprozessen übernommen. Die automatisierte Benutzersicht in *Specto* ist von asynchronen Benachrichtigungen und automatisierten Interaktionen zwischen den beteiligten Softwareprozessen gekennzeichnet.

Dokumentenbegriff Der Informationsträger in *Specto* sind die *SpectoML*-Dokumente, welche die Monitoring-Informationen der jeweiligen VOs enthalten. In diesem Zusammenhang sind die *SpectoML*-Dokumente von einem *passiven Dokumentenbegriff* geprägt, weil sie keine eigenen Aktivitätspotentiale besitzen. Da sie ausschließlich als semi-strukturierte Datencontainer für Monitoring-Informationen fungieren, sind sie nicht in der Lage, selbständig Informations- und Kontrollflüsse innerhalb eines Informationssystems wie *Specto* zu initiieren. In dieser Hinsicht sind die Monitoring-Informationen in *Specto* von anderen Systemkomponenten – dem Applikationsserver *Respondeo* und dem Notifizierungssystem *Mitto* – abhängig, welche die Rolle aktiver Entitäten in einem Informationssystem übernehmen und bei Bedarf auf das passive Dokument und dessen Inhalte strukturiert zugreifen.

Technologie Die in Abbildung 7.2 illustrierte Systemarchitektur *Specto*s besteht aus einer mehrstufigen Client/Server-Architektur, in dessen Mittelpunkt der Applikationsserver *Respondeo* (vgl. Kapitel 4) und das Notifizierungssystem *Mitto* stehen. *Respondeo* fungiert als Mediator (vgl. Abschnitt 2.2), der Anfragen der Thin-Clients – wie beispielsweise des in Abbildung 7.4 dargestellten Web-Clients von *Specto* – beantwortet. Der Zugriff auf die Monitoring-Informationen in *Specto* erfolgt stets über *Respondeo* oder *Mitto*. Diese Unterteilung der Systemarchitektur ist kennzeichnend für mehrstufige Client/-Server-Informationssysteme.

Kapitel 8

Weitere Fallstudien

Im folgenden Kapitel werden weitere Fallstudien zur Untersuchung der Anwendbarkeit der in den vorherigen Kapiteln vorgestellten Infrastrukturen des Agentenframeworks *Okeanos* (vgl. Kapitel 5) oder der Auszeichnungssprache *SpectoML* (vgl. Kapitel 6) für die Beschreibung von Zuständen behandelt.

- In Abschnitt 8.1 wird ein XML-basiertes Testmanagement-Framework vorgestellt, das Testdokumente auf der Basis von *SpectoML*-Dokumenten erzeugt und verwaltet. Die Generierung und die Verwaltung der Testdokumente erfolgt analog zum Monitoring-Modell *Specto*s (vgl. Abschnitt 7.2). Eine ausführliche Diskussion des Frameworks findet in [SSDKK00, SD00] statt.

- Im Mittelpunkt der Fallstudie, die in Abschnitt 8.2 erläutert wird, steht eine verteilte, irreguläre Berechnung aus dem Bereich des Symbolischen Rechnens. Die Berechnung wird dynamisch in entsprechende Teilprobleme zerlegt, wobei jedes Teilproblem von einem mobilen *Okeanos*-Agenten berechnet wird.

  Jeder Agent entscheidet autonom über die Wahl der Ressourcen, die für die Berechnung notwendig sind [SBS+00]. Es gibt keine globalen und zentralen Systemkomponenten, welche die einzelnen Abläufe und Informationsflüsse während der Berechnung koordinieren.

  Zudem wird beschrieben, wie die Systemzustände einzelner Agenten sinnvoll für eine effiziente Ausführung der verteilten Berechnung benutzt werden können [SFK01a]. Diese Fallstudie unterstreicht die umweltrelevanten Aspekte für eine optimierte Ausführung von Informationsflüssen (vgl. Abschnitt 7.1).

  Eine ausführliche Diskussion der algorithmischen Details der verteilten Berechnung und der Umsetzung auf der Basis mobiler *Okeanos*-Agenten findet in [SFK01a, SBS+00, Fri99] statt.

- Schließlich wird in Abschnitt 8.4 ein erweiterter Ansatz für die effektive Verwendung umweltbezogener Informationen angeführt. Neben den strukturierten Zugriffsmöglichkeiten auf *SpectoML*-Dokumente, die im Kapitel 6 über *SpectoML* vorgestellt werden und beispielsweise beim Monitoring in *Specto* zum Einsatz kommen, werden nun weitere Verarbeitungsmöglichkeiten der Informationen in den *SpectoML*-Dokumenten untersucht. Auf der Basis von Prolog [SS86] bilden die XML-basierten Informationen die Grundlage für logische Inferenzverfahren wie z. B. der Deduktion.

Generell gilt für die erläuterten Fallstudien, daß diese nur Teilaspekte zur Realisierung proaktiver Informationssysteme behandeln und abdecken. Bei der Konzeption der Fallstudien in dieser Arbeit werden spezifische Aspekte exemplarisch in einem genau bestimmten Anwendungsbereich eruiert. Dennoch sind die gewonnen Erkenntnisse auf andere Anwendungsdomänen übertragbar. So können die auf der Basis von *SpectoML* erstellten Testdokumente (vgl. Abschnitt 8.1) von den Deduktionsdiensten (vgl. Abschnitt 8.4) in Prolog profitieren. Ähnliches gilt für die in Kapitel 9 erörterten *Living Documents*, welche die Grundlage zur Realisierung von proaktiven Informationssystemen bilden.

8.1 *Tempto* – ein XML-basiertes Testmanagement-Framework

Tempto [SSDKK00] [1] ist ein Framework [JF88], das die Erstellung von Testfällen erleichtert und das Management von XML-basierten Testdokumenten in einer verteilten Umgebung unterstützt[2]. Generell bildet das Testen von Software einen wichtigen Bestandteil innerhalb der Softwareentwicklung. Ein systematischer Ansatz beim Testen erhöht die Qualität der zu entwickelnden Software und stellt einen kritischen Faktor für die Reduzierung der Wartungskosten dar [Fow99].
Ein Testfall bzw. ein Test in *Tempto* ist ein objektorientiertes Programm, das in Java implementiert ist. Jeder Testfall wird systematisch und einheitlich in das Framework von *Tempto* eingebunden. Dafür stellt das Framework definierte Schnittstellen, die von den Testfällen implementiert werden, zur Verfügung. Durch diese systematische Einbettung in *Tempto* wird die automatisierte Ausführung der Testfälle ermöglicht, d. h. es obliegt nicht dem Entwickler, die Testfälle manuell anzustoßen und auszuführen, sondern das Framework übernimmt die Ausführung aller Testfälle.
Jeder Testfall beinhaltet ein *Testkriterium*, das ein Maß für die Zielerreichung des Testfalls darstellt. Ein Testkriterium gibt einen *Soll-Wert*, der vom zu testenden Objekt (TO) erreicht werden sollte, vor. Durch einen Soll-Ist-Vergleich werden Abweichungen und die daraus resultierenden Fehler gemessen und bestimmt. Ein

[1] *temptare* (lat.) – versuchen, ausprobieren; *tempto* (1. Person Singular) – *ich versuche*.

[2] Vgl. Abschnitt 2.1.2.2 über *Technologien* in Informationssystemen für eine allgemeine Erörterung objektorientierter Frameworks.

Testdokument in *Tempto* enthält die *Ist-Werte*, die bei der Ausführung des jeweiligen Testfalls, festgehalten werden. Der Generierungs- und Verwaltungsprozeß von Testdokumenten in *Tempto*, die auf der *SpectoML* basieren, wird in Abschnitt 8.1.3 genauer geschildert.
Generell gibt es verschiedene Arten von TOs. Für jedes TO werden Testfälle bestimmt und entwickelt, die anschließend in *Tempto* automatisiert ausgeführt werden:

- *Systemebene*. Auf dieser Ebene wird das Gesamtsystem getestet. Die Systemspezifikation dient dabei als Testkriterium. Die Testfälle auf der Systemebene werden als sogenannte *Acceptance Tests* bezeichnet.
- *Modulebene*. Auf dieser Ebene werden bestimmte Funktionen einzelner Module getestet. Die Testfälle auf der Modulebene werden auch als sogenannte *Functional Tests* bezeichnet.
- *Objektebene*. Auf der untersten Ebene der Systemhierarchie stehen die entwickelten Klassen und Objekte, die hinsichtlich ihrer geforderten Methoden- bzw. Verhaltenseigenschaften getestet werden (*Unit Tests*).

Teilaspekte der Bestimmung und Entwicklung von Testfällen werden in Abschnitt 8.1.2 näher erläutert[3]. Generell steht jedoch das Management der Testdokumente in diesem Abschnitt im Vordergrund der Betrachtung.

8.1.1 Ziele

Im folgenden werden einige der wichtigsten Ziele beim Entwurf *Tempto*s erläutert:

- *Frühzeitige Fehlererkennung*. Auf der Objektebene von TOs geht es primär um die Erkennung von Fehlern in den implementierten Klassen. *Tempto* unterstützt die Erstellung von Testfällen auf der Objektebene, um möglichst frühzeitig im Entwicklungsprozeß auf (potentielle) Fehler aufmerksam zu machen.
- *Qualitätssicherung durch Automatisierung*. Durch die automatisierte Ausführung der Testfälle werden die Zeit und die Kosten eines Testdurchlaufs wesentlich reduziert. Zudem ermöglicht die Automatisierung die Generierung einer Vielzahl von Testfällen[4]. Die Existenz einer großen Anzahl von positiven Testfällen ist letztendlich ein Indikator für die Qualität der entwickelten Software.

[3] Für die kompletten Details des Frameworks, welches für die Erstellung von Testfällen verantwortlich ist, wird auf [SSDKK00, SD00] verwiesen. Das grundsätzliche Vorgehen in *Tempto* bei der Testerzeugung basiert auf *JUnit* [BG02]. Eine ausführliche Beschreibung der in *Tempto* vollzogenen Erweiterungen findet sich bei Schmidt-Dannert [SD00].

[4] Bei der automatisierten Testausführung sind mehrere hundert Testfälle, die fortwährend zur Ausführung kommen, keine Seltenheit.

- *Standardisierte Testdokumentation auf der Basis von XML*. Die Testdokumente, welche die Ergebnisse der jeweiligen Ausführung eines Testfalls beinhalten, werden als *SpectoML*-Dokument (vgl. Kapitel 6) standardisiert abgelegt.

- *Testdokumentation als Wissensbasis*. Die Testdokumentation dient als Wissensbasis und Informationsquelle zur Analyse des Entwicklungsprozesses und zur Qualitätssicherung der entwickelten Software, wie es in Schimkat et al. [SFK01b] erläutert wird.

- *Zielgerichtete und automatisierte Fehlersuche*. Durch die strukturierten Zugriffsmöglichkeiten auf die XML-basierten Testdokumente werden automatisierte Auswertungen der Testergebnisse ermöglicht. Analog zur Verteilung der Monitoring-Informationen in *Specto* (vgl. Abschnitt 7.2), die ebenfalls in *SpectoML*-Dokumenten abgelegt sind, werden Softwareprozesse in die Lage versetzt, auf die aktuellen Testergebnisse in automatisierter Form zu reagieren (siehe auch Abschnitt 8.1.3).

8.1.2 Spezifische Eigenschaften

In diesem Abschnitt werden einige der spezifischen Eigenschaften *Tempto*s geschildert. Grundsätzlich basiert jener Teil von *Tempto*, der für die Erstellung und für die Ausführung von Testfällen verantwortlich ist, auf dem Framework *JUnit* [BG02]. Dagegen ist das Management der Testdokumente in *Tempto* unabhängig von JUnit realisiert (vgl. Abschnitt 8.1.3):

- *Parallele Ausführung von Tests*. Eine Menge von Testfällen kann über die Angabe von Parametern parallel ausgeführt werden.

- *Dynamisches Hinzufügen von Tests*. Zur Laufzeit besteht die Möglichkeit eine bedingte Ausführung von Testfällen vorzunehmen. Dadurch werden dynamische Aspekte bei der Auswahl der Testfälle berücksichtigt.

- *Entkopplung des Testlaufs von der Verarbeitung der Resultate*. Die eigentliche Ausführung eines Testfalls ist von der Verarbeitung der Ergebnisse getrennt. Diese Trennung vereinfacht die Erstellung eines Testfalls für den Entwickler, da die Ergebnisverwaltung nicht direkt in seinen Verantwortungsbereich fällt.

  Zudem ermöglicht die Trennung zwischen der Ausführung und der Ergebnisverwaltung einen unabhängigen Einsatz der Komponenten in *Tempto*, d. h. für die Benutzung *Tempto*s ist die Existenz der Komponente zur Verwaltung der Testdokumente nicht zwingend erforderlich.

8.1.3 Verwaltung der Testdokumente

Die Verwaltung der Testdokumente in *Tempto* orientiert sich am Monitoring-Modell *Specto*s, das in Abschnitt 7.2 ausführlich erläutert wird. Konzeptionell entspricht ein Testfall hinsichtlich eines bestimmten TOs in *Tempto* einem VO in der Terminologie *Specto*s. In dieser Hinsicht werden beim Ausführen eines Testfalls wichtige Informationen über die aktuellen Zustände in XML-basierten Testdokumenten festgehalten.
Die Abstraktion einer Monitoring-Instanz in *Specto*, die in Abbildung 7.2 dargestellt ist, läßt sich auf das Testframework *Tempto* einfach übertragen, indem anstelle der Menge von VOs nun eine Menge von Testfällen tritt[5]. Die Prozesse der Generierung, Verwaltung und Präsentation von Testdokumenten in *Tempto* sind analog zu denen im Abschnitt 7.2 beschriebenen bei *Specto*:

- *Generierung*. Es werden die Wertebreiche und die Zustandstypen der Testfälle in XML definiert. Danach wird für die Programmiersprache Java eine Programmierschnittstelle, welche die zuvor spezifizierten Zustandstypen enthält, automatisch generiert.

- *Verwaltung*. Ein Cerberus ist für die Weiterleitung der aktuellen Zustände bzw. der Testdokumente verantwortlich. Analog zum Offline- und Online-Monitoring bestehen interaktive und automatisierte Zugriffsmöglichkeiten auf die Testdokumente. Demzufolge eröffnet *Tempto* interaktive und automatisierte Formen der Fehlersuche auf der Basis der XML-basierten Testdokumente.

- *Präsentation*. Es werden die gleichen Benutzerschnittstellen (vgl. Abbildungen 7.3 und 7.4) wie bei *Specto* benutzt, da beide Systeme auf der *SpectoML* basieren. Dies unterstreicht die Mächtigkeit, Anwendbarkeit und Wiederbenutzbarkeit der *SpectoML* zur Beschreibung von Zuständen in verschiedenen Applikationsdomänen.

8.2 Eine verteilte Berechnung auf der Basis von mobilen und autonomen Agenten

Im Mittelpunkt der Fallstudie steht eine verteilte, irreguläre Berechnung aus dem Bereich des Symbolischen Rechnens. Das Gesamtproblem wird dynamisch in entsprechende Teilprobleme zerlegt, wobei jedes Teilproblem von einem mobilen *Okeanos*-Agenten (vgl. Kapitel 5) berechnet wird. An jeder *Lounge* ist ein spezieller Berechnungsdienst, welchem die Agenten ihre Teilprobleme zur Berechnung übergeben, installiert. Grundsätzlich entsteht während der Berechnung ein nebenläufiger und konkurrierender Zugriff auf die aktuell verfügbaren Berechnungsdien-

[5] Aus der Sicht *Specto*s werden die Testfälle in *Tempto* „beobachtet“ und deren Zustände in *SpectoML*-Dokumenten ablegt.

ste. Für die Details der verteilten Berechnung wird auf Schimkat et al. [SBS+00] verwiesen.
Bei der Durchführung der Fallstudie stehen folgende Ziele im Vordergrund der Betrachtung:

- *Realisierung eines autonomen und dezentralen Systems*. Jeder *Okeanos*-Agent entscheidet autonom über die Wahl der Ressourcen, die für die Berechnung notwendig sind. Es gibt keine globalen, zentralen Systemkomponenten, welche die einzelnen Abläufe und Informationsflüsse während der verteilten Berechnung koordinieren oder vorgeben. Jeder *Okeanos*-Agent trifft seine Entscheidungen auf der Grundlage der Informationen, die *lokal* an der jeweiligen *Lounge* verfügbar sind.

- *Dynamische Informations- und Kontrollflüsse*. Die verteilte Berechnung beruht auf einem irregulären Problem aus dem Symbolischen Rechnen. Die Aufteilung des Gesamtproblems in entsprechende Teilprobleme und die Dauer für die Berechnung sind nicht a priori bestimmbar. Durch die Zuweisung eines Teilproblems zu einem autonomen *Okeanos*-Agenten entstehen dynamische Informations- und Kontrollflüsse, die zumeist im einzelnen nicht exakt determinierbar sind. Lediglich der konzeptionelle Rahmen für den grundsätzlichen Ablauf der Informations- und Kontrollflüsse kann deklarativ vorgegeben werden. Dabei wird aber keine Reihenfolge von Aktionen und Abläufen bestimmt, wie es beispielsweise bei prozeduralen Vorgehensweisen der Fall ist[6].

- *Verwendung eines regelbasierten Programmiermodells*. Die Applikationslogik für die verteilte Berechnung ist auf der Basis von JESS [FH99] (vgl Abschnitt 5.2.2.2 über die *regelbasierte Programmierung* von *Okeanos*-Agenten) realisiert. Die Applikationslogik zeichnet sich durch eine Menge von JESS-Regeln aus, die das Verhalten eines *Okeanos*-Agenten in deklarativer Form festlegen. So gibt es beispielsweise Regeln, die den Zeitpunkt und die Umstände bestimmen, wenn ein *Okeanos*-Agent seine aktuelle `Lounge A` verlassen sollte, da nun auf der `Lounge B` der Berechnungsdienst wieder zur Verfügung steht und auf neue Anfragen wartet.

Diese Fallstudie bildet die Basis für weitere Untersuchungen, die in den folgenden Abschnitten 8.3 und 8.4 erörtert werden.

8.3 Optimierung von verteilten Abläufen durch die Verwendung von Zustandsinformationen

Die Ausgangssituation für diese Fallstudie bildet die verteilte Berechnung auf der Grundlage von mobilen *Okeanos*-Agenten, die im vorherigen Abschnitt 8.2 be-

[6] Vgl. Abschnitt 3.2.4 über *proaktive Dokumente* und Abschnitt 5.2.2 über *Programmiermodelle in Okeanos.*

schrieben ist. Um die dynamischen und verteilten Abläufe während der Berechnung zu optimieren, werden zusätzliche Informationen über die aktuellen Zustände der mobilen *Okeanos*-Agenten und *Lounges* generiert. In Schimkat et al. [SFK01a] findet eine ausführliche Erörterung der Fallstudie statt.
Im folgenden werden die wesentlichen Ziele bei der Durchführung dieser Fallstudie erläutert:

- Die Berücksichtigung von Umweltinformationen als zusätzliche Informationsquellen für optimierte Abläufe in einem autonomen und dezentralen System.
- Die Plausibilisierung des Einsatzes von *Specto* und dessen Infrastrukturen zur Modellierung, Generierung und Verteilung von Umweltinformationen.

Zur Modellierung, Generierung und Verteilung der Zustandsinformationen sind folgende Schritte notwendig:

1. Die grundsätzlichen Zustandstypen der autonomen und mobilen *Okeanos*-Agenten, die an der verteilten Berechnung teilnehmen, werden systematisch bestimmt (vgl. Abschnitt 7.2 über den *prinzipiellen Ablauf für die Generierung von Monitoring-Informationen* in *Specto*).
2. Danach werden die Agenten mit den entsprechenden Zustandsmeldungen versehen und instrumentalisiert.
3. Die Monitoring-Informationen bzw. Zustände werden zur Laufzeit gesammelt, an bestimmte Interessenten über ein Notifizierungssystem weitergeleitet und ausgewertet (vgl. `XML Subscriber A+B` in Abbildung 7.2 über den *Systemüberblick Spectos*). Die Auswertung übernimmt ein spezieller Agent *Promoter-Agent*, welcher die Zustandsinformationen der Agenten und *Lounges* in einem neuen *SpectoML*-Dokument aggregiert. Nach der Zusammenführung der Informationen stellt der Promoter-Agent strukturierte Anfragen an die neu geschaffene Datenbasis gemäß der *SpectoML* in automatisierter Form (vgl. Abschnitt 6.4).
4. Im letzten Schritt initiiert der Promoter-Agent auf der Basis der dynamisch abgefragten und ausgewerteten Zustandsinformationen sogenannte *Folgeaktionen*. Eine Folgeaktion bestimmt eine bestimmte Anzahl an Agenten, die von einer *Lounge* `Quelle` zu einer *Lounge* `Ziel` migrieren sollten, damit das verteilte System adäquater ausgelastet ist[7]. Der Promoter-Agent bestimmt über die Zustandsinformationen die aktuelle Performanz und Auslastung einer *Lounge*. Die Auslastung ist ein Maß, das in den Auswertungsprozeß des Promoter-Agenten miteinfließt und demnach in einer Folgeaktion

[7] Prinzipiell sind auch andere Folgeaktionen bestimmbar. Hinsichtlich der Ziele, die in dieser Fallstudie verfolgt werden, ist diese einfache Bestimmung einer Folgeaktion ausreichend und sinnvoll.

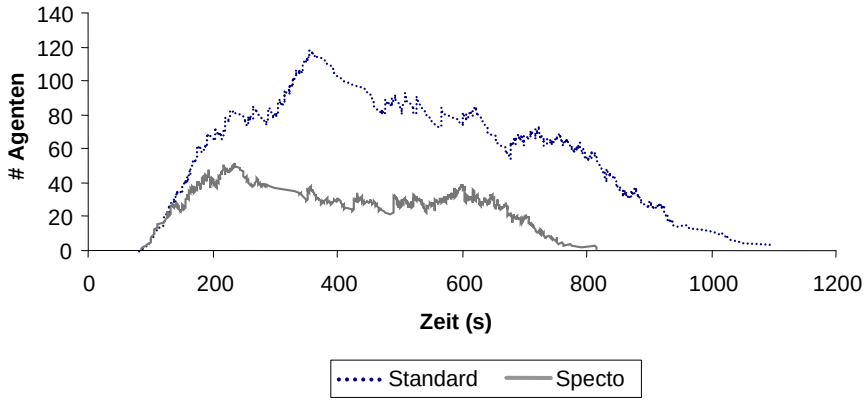


Abbildung 8.1: Die Anzahl mobiler Agenten und die Laufzeit einer verteilten, irregulären Berechnung.

bereits berücksichtigt ist. Schließlich schickt der Promoter-Agent eine Nachricht an die betreffende *Lounge* `Quelle` mit dem entsprechenden Migrationsvorschlag. Generell liegt es im Verantwortungsbereich jeder *Lounge*, die Vorschläge eines Promoter-Agenten zu befolgen. Die Folgeaktionen besitzen lediglich ein Vorschlagsrecht, d. h. die Autonomie und die dezentrale Organisation der Agenten bleibt unverändert bestehen.

Generell werden also neue Informationsflüsse angestoßen, die auf der Grundlage der Zustände getroffen werden. Dadurch werden das System und seine Abläufe positiv beeinflußt. Gemäß Abbildung 8.1 ist die Berechnung durch den Einsatz von *Specto* bzw. durch die Mitberücksichtigung von Zustandsinformationen (`Specto`) performanter im Vergleich zum Standardvorgehen (`Standard`), das in Abschnitt 8.2 beschrieben ist. Die Anzahl der erzeugten Agenten, die jeweils eine Teilaufgabe hinsichtlich der verteilten Berechnung zu lösen haben, wird durch den besseren Informationsstand wesentlich reduziert. Dadurch entstehen weniger konkurrierende Zugriffe um die verfügbaren Ressourcen wie beispielsweise um die Berechnungsdienste im verteilten System.

Generell standen quantitative Optimierungen der Laufzeit der verteilten Berechnung nicht im Vordergrund der Fallstudie. Vielmehr ging es um die Untersuchung der Plausibilität des Einsatzes von *Specto* in einer verteilten und dynamischen Umgebung. Zudem wurde untersucht, wie Umweltzustände die Abläufe in einem System, das aus vielen, autonomen Agenten besteht, nachhaltig beeinflussen können.

8.4 *XRePro* – eine Fallstudie zur Deduktion auf der Basis von XML-Dokumenten

Die Fallstudien in den Abschnitten 8.2 und 8.3, in deren Mittelpunkt eine verteilte Berechnung auf der Grundlage von mobilen und autonomen *Okeanos*-Agenten steht, bilden den allgemeinen Rahmen für die Durchführung einer Untersuchung von Deduktionsmöglichkeiten[8] auf der Basis von XML-Dokumenten. Das System *XRePro* [HS01] bietet auf der Grundlage von Prolog [SS86] Deduktionsdienste für XML-basierte Datenbasen an. Hinsichtlich einer ausführlichen Erörterung der architektonischen und implementierungsrelevanten Details von *XRePro* wird auf Heumesser und Schimkat [HS01] verwiesen.
Im Mittelpunkt der Fallstudie in [HS01] steht die Evaluierung weiterer Anfrage- und Verarbeitungsmöglichkeiten von Informationen, die in *SpectoML*-Dokumenten enthalten sind. Neben den strukturierten Zugriffsverfahren (vgl. Abschnitt 6.4) stehen bei *XRePro* logische Inferenzverfahren auf der Basis von Prolog im Mittelpunkt der Betrachtung. Grundsätzlich werden dafür die *SpectoML*-Dokumente in Prolog-Terme und Fakten überführt.
Der in Abschnitt 8.3 beschriebene *Promoter-Agent* übernimmt im Rahmen der Fallstudie mit *XRePro* die Rolle eines Mediators, der zwischen den Agenten und *XRePro* vermittelnd tätig ist (vgl. Abschnitt 2.2). Die primäre Aufgabe des Promoter-Agenten ist die Generierung neuer XML-basierter Dokumente bzw. Datenbasen, welche durch die Zusammenführung der aktuellen Zustandsinformationen der jeweiligen Agenten und *Lounges* entstehen. Dafür werden zu bestimmten Zeitpunkten die aktuellen und zusammengeführten Informationen vom Promoter-Agent an *XRePro* übergeben. Durch die fortwährende Aktualisierung der Zustandsinformationen wird auf der Seite von *XRePro* eine *globale* und *virtuelle* Wissensbasis auf der Grundlage von XML-Dokumenten *inkrementell* aufgebaut:

- Die Wissensbasis von *XRePro* ist *global* hinsichtlich der enthaltenen Zustandsinformationen. Jede Zustandsinformation wird zunächst lokal generiert, d. h. ein *Okeanos*-Agent oder eine *Lounge* legen ihren aktuellen Zustand gemäß der *SpectoML* ab. Diese Zustände werden über die Verteilungsverfahren von *Specto* (vgl. Kapitel 7) und über den Promoter-Agenten an *XRePro* weitergeleitet. Schließlich entsteht so eine Wissensbasis, welche den globalen Zustand des gesamten, verteilten Systems repräsentiert.

- Die Wissensbasis wird als *virtuell* bezeichnet, da sie durch die Zusammenführung der lokal existierenden Zustandsinformationen erst entstanden ist. Die virtuelle Wissensbasis ist nicht einer zentralen Kontrollinstanz unterstellt, d. h. es können sich vielfältige Beziehungen der Daten (Zustandsinfor-

[8] *Deduktion* – „ Ableitung des Besonderen aus dem Allgemeinen“ oder „Ableitung von Schlüssen aus bereits bewiesenen Schlüssen oder Aussagen mit Hilfe logischer Schlußregeln“ [EHWM01].

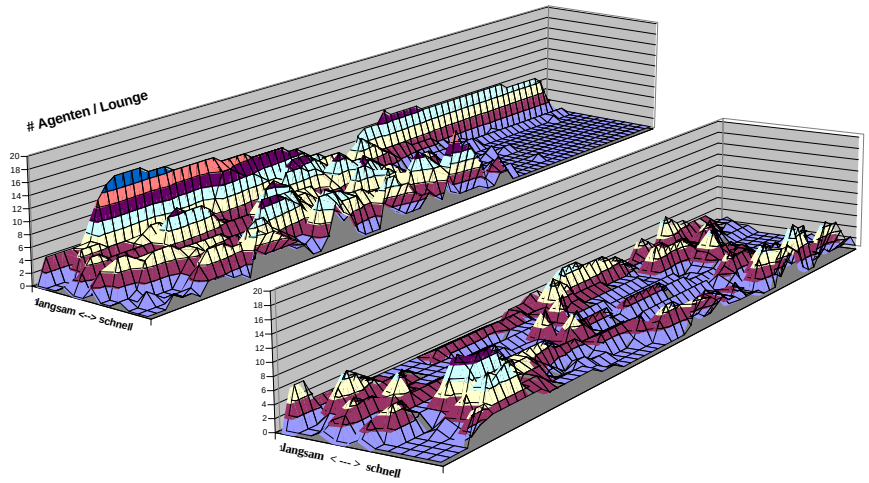


Abbildung 8.2: Grafische Visualisierung der räumlichen Verteilung von mobilen Agenten während einer verteilten Berechnung.

mationen) untereinander ergeben, ohne daß diese aus der jeweiligen lokalen Sicht ohne weiteres erkennbar sind.

- Der *inkrementelle* Aufbau der Wissensbasis bezeichnet das Vorgehen, in dem zur Laufzeit jeweils nur die neu hinzugekommenen Zustandsinformationen in die Wissensbasis von *XRePro* eingefügt werden. Der inkrementelle Aufbau der Wissensbasis garantiert die Aktualität der in Prolog repräsentierten Zustandsinformationen.

Im folgenden werden neben dem inkrementellen Aufbau einer globalen, virtuellen Wissensbasis die drei wesentlichen Ziele bei der Durchführung der Fallstudie kurz erläutert:

- *Web-basierte Zugriffsmöglichkeiten*. *XRePro* bietet eine web-basierte Kommunikationsschnittstelle an. Auf der Basis von HTTP [Wor00] ermöglicht ein eigenes Applikationsprotokoll den verteilten Zugriff auf *SpectoML*-Dokumente. Im Zusammenhang mit *XRePro* bezeichnet ein verteilter Zugriff die räumliche Trennung zwischen dem XML-Dokument – der semistrukturierten Datenbasis – und dem Promoter-Agenten, welcher die Anfrage an *XRePro* gestellt hat. Dafür wird das XML-Dokument nicht über seinen Inhalt, sondern über seine URL [BLFM98] adressiert, wie es typisch für die Organisation von Dokumenten im Web ist (vgl. auch Abschnitt 2.3.2).

  XRePro stellt eine Menge von vordefinierten Anfragen in der Form von Prolog-Regeln zur Verfügung. Über spezielle Parameter im Applikationsprotokoll werden die jeweiligen Prolog-Regeln angesprochen[9].

[9] Eine mögliche Erweiterung des Applikationsprotokolls von *XRePro* besteht darin, daß benut-

- *Automatisierte Verarbeitung*. Neben dem Aufbau einer Wissensbasis ist es entscheidend, daß das in *XRePro* generierte Wissen dem ursprünglichen System – der verteilten Berechnung mit mobilen Agenten – zur Verfügung gestellt wird. Dafür kommuniziert der Promoter-Agent über das oben erwähnte Applikationsprotokoll mit *XRePro*. Die Kommunikation mit *XRePro* basiert auf dem einfachen Anfrage/Antwort-Protokoll, d. h. jede Anfrage wird mit einer definierten Antwort quittiert. *XRePro* bestimmt sogenannte *Folgeaktionen*, wie sie bereits in Abschnitt 8.3 beschrieben sind. Der wesentliche Unterschied zwischen der Bestimmung einer Folgeaktion auf der Grundlage von objektorientierten Mechanismen (vgl. Abschnitt 8.3) und der Festlegung durch logische Inferenzverfahren – wie z. B. der Deduktion – besteht in der Ausdrucksstärke von Prolog. Generell ist der Inhalt einer Folgeaktion das Ergebnis der Abarbeitung von Regeln, die über *Wenn-Dann-Aussagen* bestimmt werden. Während bei objektorientierten Mechanismen diese Regeln nur mit größerem Aufwand nachgebildet werden können, ist in Prolog die Abarbeitung von Regeln der zentrale Ansatz.

- *Deduktionsdienste auf der Basis von Prolog*. Durch die Verwendung von Prolog soll eine globale, virtuelle Wissensbasis aufgebaut werden, die zur Analyse aktueller Systemzustände genutzt werden kann. Auf der Grundlage der Analyseinformationen sollen Vorschläge (vgl. die *Folgeaktionen* in Abschnitt 8.3) für einen optimierten Ablauf in dem autonomen und dezentralen System der verteilten Berechnung erfolgen. *XRePro* benutzt die mächtigen Verarbeitungsmöglichkeiten von Prolog, um so Anfrage- und Verarbeitungsmechanismen miteinander zu verbinden.

  Abbildung 8.2 illustriert die Analysefunktion von *XRePro*, das auf der Grundlage der globalen und virtuellen Wissensbasis mehrere Kriterien für die Bestimmung der Folgeaktionen festlegt. In Abbildung 8.2 wird die Verteilung der mobilen *Okeanos*-Agenten auf die einzelnen *Lounges* zu bestimmten Zeitpunkten der verteilten Berechnung veranschaulicht. Dabei werden die *Lounges* hinsichtlich ihrer Performanz in `langsame` und `schnelle` *Lounges* aufgeteilt. Während im oberen Diagramm in Abbildung 8.2 die Verteilung der Agenten ohne die Berücksichtigung der Folgeaktionen von *XRePro* veranschaulicht ist, werden die Vorteile bei der Miteinbeziehung der Folgeaktionen im unteren Diagramm veranschaulicht. Generell sind die jeweiligen *Lounges* wesentlich gleichmäßiger ausgelastet. Zudem reduziert der Einsatz von *XRePro* die maximale Anzahl von Agenten auf einer *Lounge*.

zerdefinierte Prolog-Regeln zur Laufzeit in *XRePro* eingebunden werden.

Teil IV

Living Documents – Proaktive Informationssysteme

Kapitel 9

Living Documents

In diesem Kapitel werden verschiedene Aspekte der Realisierung von sogenannten *Living Documents* [SK02, SKN02][1] erörtert. Der Begriff *Living Document* ist eine Abstraktion für Dokumente, die mobile, aktive und wissensbasierte Eigenschaften besitzen. *Living Documents* bilden die Grundlage zur Realisierung proaktiver Informationssysteme.
Nach der Einleitung in Abschnitt 9.1 wird die zugrunde liegende *Dokumentenarchitektur* in Abschnitt 9.2 beschrieben. Abschnitt 9.3 veranschaulicht das *Dokumentenmodell*, das sich vor allem durch die Abgeschlossenheit und die Kapselungseigenschaften eines Dokuments sowie durch die Möglichkeit der dynamischen Generierung von verschiedenen Sichten auf ein Dokument auszeichnet. Das verhaltens- und wissensbezogene *Programmiermodell* der *Living Documents* wird in Abschnitt 9.4 vorgestellt. Nach der Einordnung in die Taxonomie für Informationssysteme in Abschnitt 9.5 schließt dieses Kapitel mit einer Abgrenzung zu anderen Forschungsarbeiten im Bereich aktiver Dokumente in Abschnitt 9.6.

9.1 Einleitung

Die in diesem Kapitel vorgestellten *Living Documents* bilden die Grundlage für die Realisierung proaktiver Informationssysteme, die folgendermaßen definiert werden:

> *Proaktive Informationssysteme* – Informationssysteme, die proaktive Komponenten beinhalten und durch automatisierte Interaktionen gekennzeichnet sind. Proaktive Informationssysteme besitzen ausdrucksstarke Repräsentationsmechanismen für Informationen und Wissen. Zudem bieten sie verschiedene Möglichkeiten der Suche nach Informationen an.

[1] Siehe auch die Homepage der *Living Documents* unter http://www.living-documents.org für weitere Informationen.

Aktive und proaktive Informationssysteme differieren in der Art ihres zugrunde liegenden Dokumentenbegriffs. Konkret liegt der Unterschied zwischen aktiven und proaktiven Dokumenten, die beide in Abschnitt 3.2 definiert und ausführlich erörtert wurden. Proaktive Dokumente sind demnach spezielle aktive Dokumente des Typs C (vgl. Abschnitt 3.2.4), wobei die aktive Komponente sich durch ein antizipierendes Verhalten auszeichnet.
Die Plausibilität und Rechtfertigung für eine dokumentenzentrierte Sichtweise auf Informationssysteme – wie in diesem Kapitel vorgenommen – wird durch den hohen alltäglichen und zunehmenden Durchdringungsgrad von dokumentenzentrierten Informationssystemen wie z. B. dem Web unterstützt [Zam01, OZ99]. Dokumente im allgemeinen spielen eine zentrale Rolle in modernen Informationssystemen. Sie werden sowohl für die Spezifikation von digitalen Inhalten (z. B. HTML-Dokumenten [RHJ99] im Web) als auch für den flexiblen Datenaustausch zwischen heterogenen Softwarekomponenten (z. B. XML-basierte Auszeichnungssprachen – wie das *Simple Object Access Protocol* (SOAP) [BEK+00] – für die Datenkommunikation)[2] eingesetzt.
Die hier vorgestellten *Living Documents* setzen auf den Infrastrukturen auf, die in vorherigen Kapiteln erörtert wurden, auf. Im einzelnen sind dies das Agentenframework *Okeanos* [SBS+00] (vgl. Kapitel 5) und die XML-basierte Auszeichnungssprache *SpectoML* [SHKK00, SFK01a] (vgl. Kapitel 6):

- *Okeanos*. *Okeanos*-Agenten werden als proaktive Komponenten in den *Living Documents* verwendet. Sie fungieren als aktive Mediatoren [RW91], die jeden Zugriff auf ein Dokument verwalten. Das Prinzip der Mediation [Wie92a] und der Delegation[3] bilden die Ausgangsbasis für die Realisierung proaktiver Informationssysteme.

- *SpectoML*. Die *SpectoML* bildet die Grundlage für eine uniforme und ausdrucksstarke Repräsentation von Informationen, die für Dokumente relevant sind. Papazoglou [Pap93] erörtert im Zusammenhang mit „intelligenten“ Informationssystemen die Notwendigkeit von

  - ausdrucksstarken Repräsentationsmechanismen für Daten, Informationen und Wissen,
  - Möglichkeiten zum logischen Schließen und
  - die Unterstützung von Applikationen, die bestimmte wissensbasierte Zugriffsketten und -pfade benötigen oder erfordern.

  Diesen Forderungen Papazoglous versuchen auch proaktive Informationssysteme, die auf der Basis von *Living Documents* realisiert sind, nachzukommen[4].

[2] Siehe auch Abschnitt 4.6.2 für einen Überblick über SOAP.

[3] Vgl. Abschnitt 2.2 für eine ausführliche Darlegung des Mediator- und Delegationsprinzips.

[4] Möglichkeiten zum logischen Schließen auf der Grundlage der Informationen, die in *SpectoML*-Dokumenten enthalten sind, werden in Heumesser und Schimkat [HS01] beschrieben.

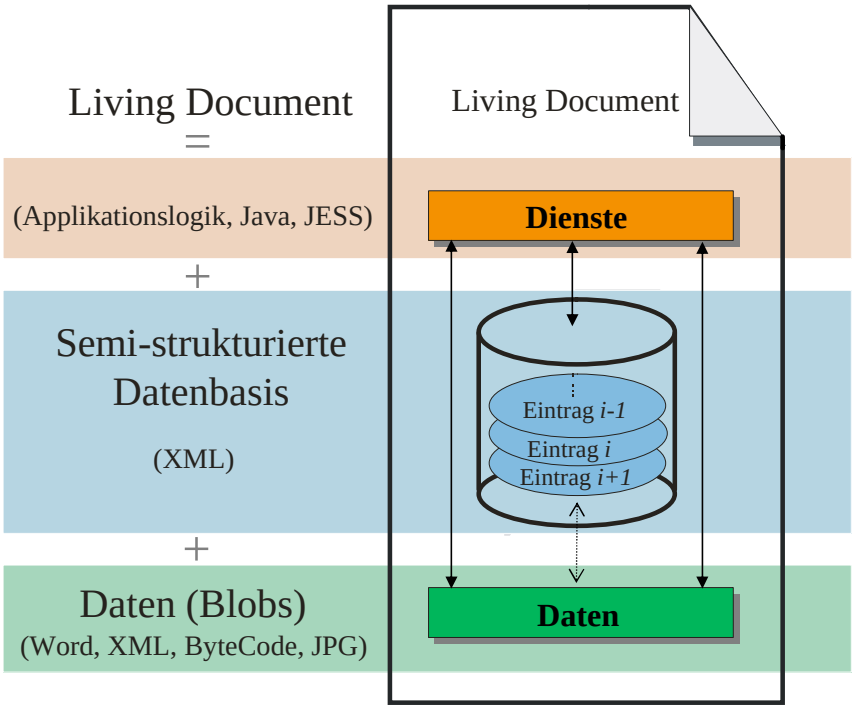


Abbildung 9.1: Schematische Übersicht der Komponenten eines *Living Documents.*

In diesem Kapitel stehen primär die Aspekte einer Zusammenführung dieser beiden Komponenten zu einem neuen Dokumentenbegriff – einem *Living Document* – im Vordergrund. Für Details der jeweiligen Komponenten sei vor allem auf die entsprechenden Kapitel 5 und 6 verwiesen. Die verschiedenen Möglichkeiten der Informationssuche in proaktiven Informationssystemen, die auf der Basis von *Living Documents* realisiert sind, werden in den Fallstudien in Kapitel 10 ausführlich erläutert.

9.2 Dokumentenarchitektur

Ein *Living Document* (*LD*) besteht im wesentlichen aus drei Teilen, wie es in Abb. 9.1 illustriert ist:

1. Code-Komponente
2. Semi-strukturierte Datenbasis zur Verwaltung von Metadaten
3. Daten oder *Binary Large Objects* (Blobs), die reale, virtuelle oder hybride Dokumente repräsentieren[5].

[5] *Reale*, *virtuelle* und *hybride* Dokumente werden in Abschnitt 3.2.1 und Abbildung 3.3 erörtert.

Formal ist ein *LD* ein Tupel:

$$LD = \langle ID, C, M, B \rangle$$

dabei ist ID eine eindeutige Nummer, die zur Identifizierung eines *LD* dient und

$$C = \{Dienst_i | i \in N\}$$
$$M = \{Metadaten_i | i \in N\}$$
$$B = \{Blobs_i | i \in N_0\}\,.$$

In den folgenden drei Abschnitten werden die *Code-Komponente*, die *semi-strukturierte Datenbasis* und der *Daten- bzw. Blob-Teil* eines *LD* näher erörtert. Jeder Abschnitt beschreibt zunächst generell geltende Eigenschaften für den jeweiligen Teil eines *LD*. Danach schließt sich jeweils eine Beschreibung der implementierungsrelevanten Details an[6]. Die gewählte Realisierung der *Living Documents* stützt sich im wesentlichen auf die Konzepte von Agenten und der XML-Technologie. Eine grundlegende Erörterung von Agenten erfolgt in Abschnitt 3.3.1. Für eine allgemeine Diskussion von passiven, aktiven und proaktiven Dokumenten wird auf Abschnitt 3.2 verwiesen.

9.2.1 Code-Komponente

Eigenschaften Die Code-Komponente beinhaltet die *Applikationslogik* des *LD*s, d. h. sie fungiert als Wissensträger für die korrekte Bearbeitung und Verwaltung des *LD*s. Die Terminologie der Applikationslogik für ein Dokument ist im Zusammenhang mit *Living Documents* gerechtfertigt, da die Code-Komponente als aktiver Wissensträger ausschließlich dem Dokument zugeordnet ist. Die Grenze zwischen traditionellen Dokumenten und Applikationen wird durch den Einsatz von *LD*s verwischt, weil Dokumente nun auch direkt Applikationslogik enthalten können.
Zudem bestimmt die Code-Komponente die *Kommunikationsschnittstelle* eines *LD*s nach außen. Sie regelt den Zugriff von anderen Softwarekomponenten und fungiert so als Mediator [Wie92a] (vgl. Abschnitt 2.2) für das gesamte *LD*. Ein Mediator erleichtert die Benutzung von *LD*s, indem er als vermittelnde Instanz zwischen den Anfragen von außen und den Datenkomponenten eines *LD*s – der semi-strukturierten Datenbasis und den Blobs – agiert. Der wesentliche Bestandteil seiner Vermittlungstätigkeit ist die Aufbereitung und Transformation der ausgetauschten Daten. Durch den *indirekten* Zugriff auf die semi-strukturierte Datenbasis und die Blobs sind *LD*s flexibel in unterschiedlichen Infomationssystemen einsetzbar[7].

[6] Die in den folgenden Abschnitten beschriebene Implementierung eines *LD* stellt eine Möglichkeit der Realisierung dar. Generell sind allerdings unterschiedliche Umsetzungen der verschiedenen Bestandteile eines *LD* möglich.

[7] Vgl. auch die Beschreibung der Eigenschaften von Mediatoren im allgemeinen wie z. B. die *dynamische Komposition von Diensten* und die *Unterstützung für autonome Datenbasen* in Abschnitt 2.2.2.

Aus der Sicht des Entwicklers legt die Code-Komponente die *Programmierschnittstelle* (API) der *LD*s fest[8]. Folgt man einer agentenbasierten Realisierung der Code-Komponenten, so ist die API durch eine nachrichtenbasierte Schnittstelle ausgezeichnet.
Die einzelnen Funktionalitäten, die ein *LD* erbringt, werden durch *Dienste* realisiert. Generell spezifiziert der Dienst ein Verhalten, das von einer Softwarekomponente implementiert wird. In Bezug auf *LD*s werden Dienste durch die Code-Komponente umgesetzt. Gemäß der formalen Definition eines *LD*s besteht die Code-Komponente aus einer Menge von Diensten. So sind die Überprüfung des Zugriffs auf ein *LD* oder das Suchen und Finden von digitalen Informationsinhalten, die in *LD*s abgelegt sind, zwei typische Beispiele für Dienste, die in der Code-Komponente verankert sind.

Implementierung Die Code-Komponente ist auf der Basis des Agentenframeworks *Okeanos* [SBS+00] realisiert worden. Dadurch entspricht ein *LD* einem *Okeanos*-Agenten[9]. Eine agentenbasierte Implementierung bringt folgende Eigenschaften für die Code-Komponente bzw. für ein *LD* mit sich:

- Dienste und Funktionalitäten eines *LD*s werden durch *Okeanos*-Agenten erbracht, die über KQML-Nachrichten miteinander kommunizieren. Die Realisierung der Code-Komponente als mobiler Agent ermöglicht einen flexiblen Einsatz in verteilten Informationssystemen, in denen *LD*s ihre Lokationen dynamisch (zur Laufzeit) ändern können.

- Die Abstraktion des Agentenorts *Lounge* (vgl. Abschnitt 5.3.2) stellt das Bindeglied zwischen einem *LD* und seiner Umgebung dar. Durch die Einbettung in die umgebenden Infrastrukturen ist ein *LD* nicht als isolierte, sondern als interagierende und kommunizierende Softwarekomponente zu betrachten. Dies ermöglicht beispielsweise die Einbeziehung umwelt-relevanter Informationen in die Entscheidungs- und Ablaufprozesse der Applikationslogik eines *LD*s.

- Eine agentenbasierte Umsetzung der Code-Komponente auf der Basis von *Okeanos* unterstützt die dynamische Aggregation von Diensten. Etwaige erforderliche Funktionalitäten werden dynamisch über den COD-Ansatz (*Code On Demand*) im Zusammenhang mit der Software-Mobilität (vgl. Abschnitt 5.1.1) nachgeladen und installiert[10]. Dies führt zu einer flexiblen und erweiterbaren Gestaltung von Diensten in *LD*s.

[8] Vgl. Abschnitt 9.4.1 für die Beschreibung der verhaltensbezogenen Programmierung von *LD*s bei der Benutzung von *Okeanos*-Agenten als Code-Komponenten.

[9] Für Details der Architektur und Eigenschaften von *Okeanos* wird auf Kapitel 5 verwiesen.

[10] Unter der Terminologie einer erforderlichen Funktionalität fällt auch ein sogenanntes Software-Update, welches Mängel in der bestehenden Software bzw. in den Code-Komponenten behebt, um den gestellten Anforderungen zu genügen. Solche Software-Updates können durch einen agentenbasierten Ansatz ebenfalls beim jeweiligen Agenten dynamisch installiert werden. Dies ist ein wichtiger Aspekt für die Wartung von Informationssystemen.

- Durch den kleinen Memory-Footprint von *Okeanos*-Agenten (vgl. Abschnitt 5.2.1) entsteht ein geringer zusätzlicher Aufwand für die Integration einer Code-Komponente in ein Dokument (Blobs).

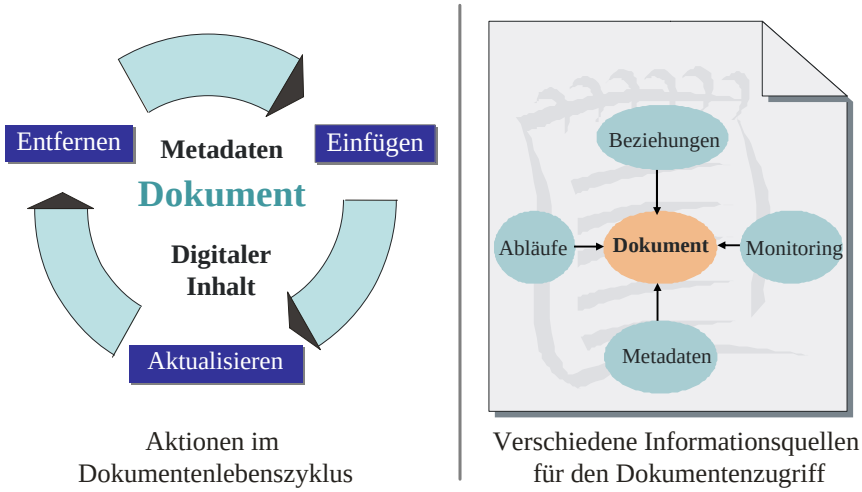
- Die regelbasierte Programmierschnittstelle in *Okeanos* (vgl. Abschnitt 5.2.2.2) ermöglicht eine deklarative Programmierung eines *LD*s. Mit Hilfe von JESS [FH99] können Teile der Applikationslogik eines *LD*s durch Regeln spezifiziert werden, die automatisch durch die in einen *Okeanos*-Agenten bzw. in ein *LD* eingebettete Inferenzmaschine verarbeitet werden. Zudem unterstützen *Okeanos*-Agenten bzw. *LD*s die dynamische Änderung und Erweiterung der JESS-Regelbasis. Regeln werden so zur Laufzeit in die Regelbasis eines *LD*s hinzugefügt. Dies ermöglicht einem *LD*, flexibel auf äußere Umstände oder erweiterte Anforderungen angemessen zu reagieren.

9.2.2 Semi-strukturierte Datenbasis

Eigenschaften Die semi-strukturierte Datenbasis dient zur Verwaltung von Informationen, die in einer Beziehung zum eigentlichen Dokument (Blobs) oder dem gesamten *LD* stehen. Generell enthält jede semi-strukturierte Datenbasis eines *LD*s eine Menge von Metadaten über das zu verwaltende Dokument. Darum wird eine semi-strukturierte Datenbasis im Zusammenhang mit *Living Documents* im folgenden als *Knowledge Respository* (KR) bezeichnet, welches dokumentenbezogenes Wissen in einer XML-basierten Datenstruktur verwaltet[11].
Einzelne Metadaten (z. B. `Eintrag i` in Abbildung 9.1) werden als *Dokumentenzustandsinformation* bezeichnet. Eine Menge von Dokumentenzustandsinformationen bildet einen sogenannten *Document State Report* (DSR), der Informationen über das zu verwaltende Dokument für einen beliebigen Zeitraum enthält. In Abbildung 9.2 wird im Zusammenhang mit dem Lebenszyklus eines Dokuments und den darin auftretenden Aktionen deutlich, daß der Lebenszyklus durch verschiedene Aktionen geprägt ist. Das *Einfügen*, *Aktualisieren* und *Entfernen* bezieht sich dabei sowohl auf den digitalen Inhalt als auch auf die Metadaten eines Dokuments, wie es in Abbildung 9.2 veranschaulicht ist. Generell besitzen Dokumente eine zeitliche Dimension: Inhalte und Metadaten enthalten Informationen oder Hinweise auf den Zeitpunkt ihrer Gültigkeit oder Erstellung.
Ein DSR enthält dokumentenbezogene Informationen, die aus *unterschiedlichen Quellen* stammen. In Abbildung 9.2 sind mehrere Informationsquellen illustriert, die in einem DSR enthalten sind:

- *Monitoring*. Zeitliche (temporale) und örtliche (spatiale) Informationen über die Dokumente werden jeweils als Dokumentenzustandsinformation in einem DSR festgehalten. Typische Fragestellungen hinsichtlich der zeitlichen und örtlichen Informationen sind u. a.

[11] Es wurde hier die englische Terminologie des *Knowledge Repositories* benutzt, um eine möglichst konsistente Verwendung zu den gewählten Begriffen in den Veröffentlichungen der *Living Documents* [SK02, SKN02] zu erreichen.



Aktionen im Dokumentenlebenszyklus

Verschiedene Informationsquellen für den Dokumentenzugriff

Abbildung 9.2: Dokumentenlebenszyklus und verschiedene Informationsquellen, die beim Suchen nach Dokumenten genutzt werden.

  - *Wann* und *wo* wurde das Dokument erzeugt?
  - *Wann* erfolgte der letzte Zugriff auf das Dokument?

- *Beziehungen*. Informationen über inhaltliche Beziehungen und Verbindungen zwischen *LD*s und ihren Umgebungen stellen einen Bestandteil eines DSR dar. So bilden beispielsweise Links in HTML-Dokumenten eine inhaltliche Verbindung zwischen zwei HTML-Dokumenten. Falls ein HTML-Dokument in ein *LD* transformiert wird, können die Links als Dokumentenzustandsinformation dienen, die im DSR festgehalten werden.

- *Abläufe*. Das Wissen um interne oder externe Abläufe ermöglicht einem *LD* auf bestimmte Nachrichten, die eine Zustandsänderung in ihrem KR bewirken, zu reagieren.

- *Metadaten*. Die Metadaten sind beispielsweise Beschreibungen über den Autor des Dokuments.

Grundsätzlich dient ein KR als eine Middleware-Komponente, die einen strukturierten Zugriff auf die einzelnen Dokumentenzustandsinformationen ermöglicht, d. h. ein KR stellt eine leichtgewichtige Datenbasis dar, die einen strukturierten Zugang zu ihren Daten – den Dokumentenzustandsinformationen – zur Verfügung stellt. Ein KR ist leichtgewichtig, da es nicht notwendigerweise an eine Datenbank, die ebenfalls strukturierte Zugriffsmöglichkeiten auf Daten unterstützt, gebunden ist[12].

[12] Vgl. den nächsten Abschnitt, in welchem die Details der Implementierung der leichtgewichtigen Datenbasis auf der Grundlage von *SpectoML*-Dokumenten erörtert wird.

Generell besitzt jedes *LD* sein eigenes KR. Der Zugriff auf ein KR erfolgt ausschließlich über die Code-Komponente des betreffenden *LD*s.

Implementierung Die semi-strukturierte Datenbasis bzw. das KR ist auf der Basis der in Kapitel 6 erörterten *SpectoML* implementiert. Generell sollte die Realisierung eines KR die prinzipiellen Eigenschaften der Code-Komponente unterstützen und umgekehrt. Sofern beispielsweise die Code-Komponente als mobiler Agent realisiert ist, sollte das KR den mobilen Einsatz eines *LD*s ermöglichen und selbst über entsprechende Eigenschaften verfügen.

- Ein DSR ist als semi-strukturiertes *SpectoML*-Dokument [SHKK00, SFK01a] abgelegt. Die Umsetzung auf der Basis von XML und *SpectoML* ermöglicht die Generierung eines DSR auf einem standardisierten Weg, der kein spezielles und proprietäres Datenformat oder bestimmte Programmier- oder Skriptsprachen erfordert.

- *SpectoML* ermöglicht die *einheitliche Repräsentierung* unterschiedlicher Dokumentenzustandsinformationen wie z. B. für das Monitoring der zeitlichen und örtlichen Informationen[13]. Die Etablierung eines allgemeinen Schemas zur Beschreibung von Zuständen wird durch die in Abschnitt 6.2.1 beschriebene DTD der *SpectoML* gewährleistet. Die Wertobjekttransformation und Kompaktifizierung der in Abschnitt 6.3 beschriebenen Eigenschaften von *SpectoML*-Dokumenten schafft die Basis für eine einheitliche Repräsentierung.

- Die Bedeutungen der Dokumentenzustandsinformationen werden explizit in einem XML-basierten Katalog abgelegt, wie in Abschnitt 2.1.2.3 erläutert. Die explizite und einheitliche Darstellung von inhaltlichen Bedeutungen unterstützt die automatisierte Verarbeitung und Interpretation von Bedeutungen durch Softwareprozesse im allgemeinen [FvHH+01] und durch *Living Documents* im besonderen.

- *SpectoML* als eine *semi-strukturierte Datenbasis* und der in Abschnitt 6.4 vorgestellte Parser als *Anfragekomponente* bilden gemeinsam ein KR. Der Zugriff auf die Dokumentenzustandsinformationen erfolgt in strukturierter und JDBC-ähnlicher Form[14].

- Die Kompaktifizierung und der Entwurf eines leichtgewichtigen Parsers ermöglichen die Realisierung eines KR, das den Anforderungen eines leichtgewichtigen *LD*s in mobilen Informationssystemen gerecht wird. Die Größe der Infrastrukturen für die Realisierung eines KR ist von Bedeutung, falls die Ressourcen – wie z. B. Speicherplatz – nur in begrenztem Maße verfügbar sind.

[13] Vgl. Abschnitt 6.3.
[14] Vgl. die Beschreibung der Eigenschaften von *SpectoML*-Dokumenten in Abschnitt 6.3.

9.2.3 Daten – Blobs

Die *Daten* bzw. *Blobs* enthalten unterschiedlichste digitalisierte Informationen, wie z. B. Office-Dokumente, Grafiken, Videos oder XML-Dokumente[15].
Im folgenden wird anhand der in Abschnitt 3.2.1 und Abbildung 3.3 diskutierten Einteilung in *reale, virtuelle* und *hybride* Dokumente die entsprechende Repräsentierung in einem *LD* vorgestellt:

- *Reale Dokumente*. Ein reales Dokument ist ein physikalisch vorhandenes digitales Dokument wie z. B. ein Office-Dokument oder eine Grafik. Es wird als Blob (Binary Large Object) im Datenteil des *LD*s repräsentiert.

- *Virtuelle Dokumente*. Virtuelle Dokumente besitzen kein reales Gegenstück, sondern entstehen durch die Aggregation einzelner Eigenschaften. So bilden beispielsweise alle Informationen über eine Person, die aus Gründen der Normalisierung in mehreren Tabellen einer relationalen Datenbank gehalten werden, ein virtuelles Dokument (siehe Abbildung 3.3). Virtuelle Dokumente werden als eine Menge von Dokumentenzustandsinformationen im KR abgelegt. In diesem Falle ist der Datenteil eines *LD*s leer.

- *Hybride Dokumente*. Hybride Dokumente bestehen aus einem realen und einem virtuellen Teil, die entsprechend den Ausführungen zu realen und virtuellen Dokumenten in *LD*s umgewandelt werden. In Abschnitt 10.1 wird in einer Fallstudie dargelegt, wie hybride Dokumente und deren reale und virtuelle Bestandteile als *LD*s repräsentiert werden.

9.2.4 Lebendige Dokumente

Living Documents werden als *lebendige* Dokumente bezeichnet, da sie folgende zwei Eigenschaften besitzen, die eine Intuition von „Lebendigkeit“ vermitteln:

1. Die Fähigkeit, innerhalb eines Computernetzwerks zu wandern und sich zu bewegen, kennzeichnet vor allem mobile und autonome Komponenten. Eine Implementierung der *Living Documents* als mobile *Okeanos*-Agenten unterstützt die Vorstellung von „lebenden“, sich bewegenden Dokumenten.

2. Ein KR als Datenbasis, in der beliebige Zustände gespeichert, aktualisiert und entfernt werden, übernimmt die Rolle eines „Gedächtnisses“ für ein *Living Document.* Ein KR wächst und schrumpft durch die Abhängigkeit von der Applikationsdomäne und dem aktuellen Umgebungskontext. Der Gesamtzustand des „Gedächtnisses“ ändert sich kontinuierlich über die Zeit hinweg. Ein menschliches Gedächtnis nimmt ebenfalls Informationen zu bestimmten Zeitpunkten auf, um dann später auf die gespeicherten Informationen (strukturiert) zuzugreifen bzw. sich daran zu erinnern.

[15] Ein XML-Dokument im Datenteil eines *LD*s ist nicht zu verwechseln mit der semi-strukturierten Datenbasis, die im vorherigen Abschnitt 9.2.2 besprochen wird.

9.3 Dokumentenmodell

Das Dokumentenmodell ist bestimmt durch die Kompaktheit und Abgeschlossenheit eines *Living Documents*. Zudem zielt das Dokumentenmodell auf die dynamische Generierung von unterschiedlichen (Teil-)Sichten auf ein und das gleiche Dokument. In den folgenden beiden Abschnitten werden die Abgeschlossenheit und die dynamische Generierung von Sichten auf ein *Living Document* näher erläutert.

9.3.1 Abgeschlossenheit

Das Dokumentenmodell der *Living Documents* ist durch die *Abgeschlossenheit* eines Dokuments gekennzeichnet. Die Abgeschlossenheit eines *Living Documents* gewährleistet, daß sowohl die relevanten Eigenschaften als auch die für die Verarbeitung notwendige Applikationslogik direkt in einem Dokument verankert sind. Die Applikationslogik bestimmt das Verhalten und die Operationen, die auf einem Dokument spezifiziert und erlaubt sind.
Hinsichtlich der Abgeschlossenheit ist das Dokumentenmodell der *Living Documents* dem Objektmodell innerhalb des objektorientierten Paradigma sehr ähnlich. Dort stehen Objekte im Mittelpunkt der Betrachtung, die aus Daten und Operationen bestehen[16]. Durch die Operationen eines Objekts werden die erlaubten Verarbeitungsprozesse auf den Objektdaten festgelegt. Allerdings ergeben sich folgende Unterschiede für das Dokumentenmodell der *Living Documents*:

- Anstelle der Spezifikation von Daten im objektorientierten Paradigma tritt die Festlegung der Wertebereiche von benutzerdefinierten Dokumentenzustandsinformationen[17]. Jede Dokumentenzustandsinformation wird uniform in einer semi-strukturierten Datenbasis abgelegt. Der Zugriff auf einzelne Einträge oder eine Menge von Einträgen in der Datenbasis erfolgt stets *indirekt* über die Initiierung einer entsprechenden Anfrage an die Datenbasis.

- Anstelle der Operationen, die auf Objekten definiert sind, treten Dienste, die auf der (internen) Datenstruktur eines *Living Documents* – der semi-strukturierten Datenbasis – operieren. Zudem regeln die Dienste die äußeren Zugriffe auf das *Living Document*.

Die Abgeschlossenheit der Dokumente hinsichtlich der Dokumentenzustandsinformationen und der dazugehörenden Dienste erweitert das Anwendungsspektrum von Dokumenten im allgemeinen. Dokumente auf der Basis von *Living Documents* werden so zu Softwareprozessen, die einen aktiven Bestandteil von Informationssystemen bilden.

[16] Vgl. auch Abschnitt 3.3.2.

[17] Im Zusammenhang mit der *SpectoML* in Kapitel 6 wird auch von der Festlegung von sogenannten *Zustandstypen* gesprochen. Ein Zustandstyp legt den Wertebereich der möglichen Zustände fest.

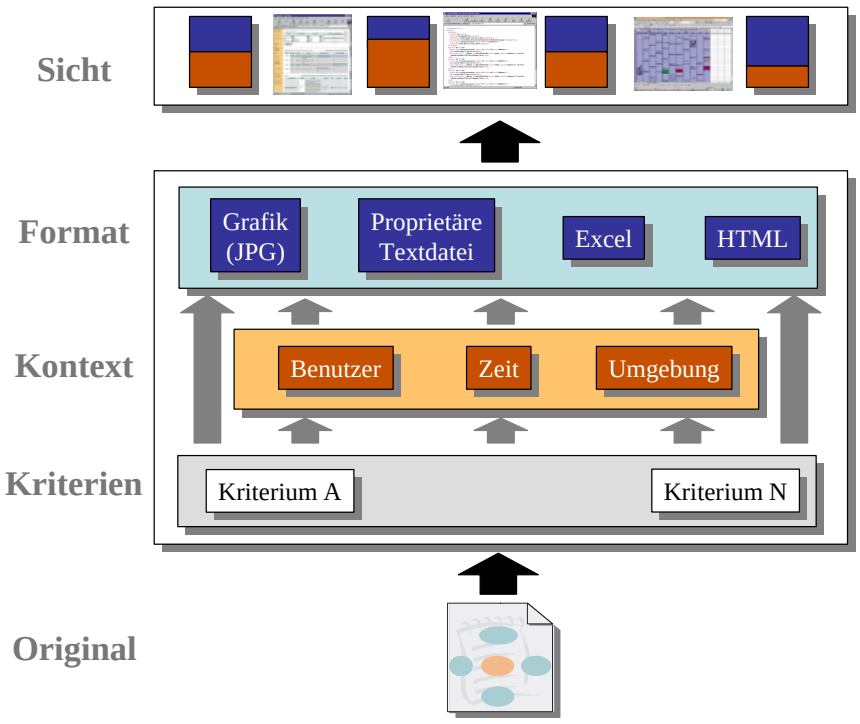


Abbildung 9.3: Unterschiedliche, dynamisch generierte Sichten auf ein *Living Document*.

9.3.2 Dynamisch generierte Sichten

Eine Sicht entspricht einer speziellen Interpretation der Daten, die in der semistrukturierten Datenbasis eines *Living Documents* enthalten sind. Eine Interpretation obliegt den Benutzern oder Applikationen, welche auf ein *Living Document* zugreifen und dieses benutzen.
So haben beispielsweise verschiedene Benutzer unterschiedliche Sichten auf ein und das gleiche *Living Document*. Wie in Abschnitt 3.2.1 über die Probleme bei der Organisation von Dokumenten in hierarchischen Dateistrukturen erläutert wird, entwerfen Benutzer die Hierarchie gemäß ihren individuellen Präferenzen und Kriterien. Um eine effektive und vor allem uniforme Organisation der Dokumente zu etablieren, sind insbesondere zwei Punkte entscheidend:

- Dokumente, die von mehreren Benutzern gemeinsam verwendet werden, sollten nicht mehrfach existieren. Es sollte eine einzige Dokumentenbasis geben, auf die alle Benutzer zugreifen.
- Dokumente sollten in Abhängigkeit des Benutzers und von dessen Präferenzen in der Lage sein, benutzerspezifische Sichten auf das eigentliche Dokument zu generieren.

Die dynamischen Anforderungen an heutige Informationssysteme und der heterogene Einsatz von Dokumenten in unterschiedlichen Applikationsdomänen erfordern die dynamische Generierung von unterschiedlichen Sichten auf Dokumente [Abi99].
Eine Sicht im Zusammenhang mit *Living Documents* ist bestimmt durch einen mehrstufigen Prozeß, der auf dem originalen Dokument ansetzt, wie in Abbildung 9.3 dargestellt. Anhand der für die Sicht spezifischen *Kriterien* werden dokumentenbezogene Informationen selektiert und an die *Kontext-* bzw. *Formatschicht* weitergereicht:

- *Kriterium*. Ein Kriterium bestimmt die Eigenschaften, welche an die Dokumentenzustandsinformationen gestellt werden. Anhand der Kriterien werden bestimmte Dokumentenzustandsinformationen im KR dynamisch ermittelt und an die höheren Schichten weitergereicht, wie es in Abbildung 9.3 veranschaulicht ist[18]. Falls keine kontextabhängigen Informationen in die Generierung der Sicht eingehen, werden die selektierten Zustandsinformationen direkt an die für die Repräsentierung verantwortliche Schicht weitergereicht.

- *Kontext*. Als Kontext werden diejenigen Informationen bezeichnet, die für die Generierung der Sicht nützlich sind bzw. sein können [SAW94]. Dies umfaßt beispielsweise zeitliche und örtliche Dokumentenzustandsinformationen, die nicht notwendigerweise in den Kriterien direkt spezifiziert wurden. Daneben können aber auch Kontextinformationen hinsichtlich der *Umgebung* oder von *Benutzerprofilen* eine Rolle spielen, wie es in Abbildung 9.3 illustriert ist. Charakteristisch für die Kontextinformationen ist ihre lokale Verfügbarkeit, d. h. für die Einbeziehung von Kontexten sind die Dokumentenzustandsinformationen maßgeblich, die (lokal) im betreffenden *Living Document* vorhanden sind.

  Ein erweiterter Ansatz zur Einbeziehung von Kontextinformationen besteht in der Möglichkeit nicht nur lokal vorhandene, sondern auch *verteilte Kontexte* bei der Sichtgenerierung zu berücksichtigen [RJM02]. Heumesser und Schimkat [HS01] zeigen wie verteilte Kontextinformationen auf der Basis von XML-Dokumenten dazu verwendet werden können, neue Sichten auf einzelne Abläufe in einem Informationssystem zu generieren.

- *Format*. Die Repräsentation oder Visualisierung einer Sicht wird durch das *Format* der Sicht bestimmt. Wie es in Abbildung 9.3 illustriert ist, kann beispielsweise eine zeitlich bestimmte Sicht in unterschiedliche Formate – wie z. B. HTML oder JPG – transformiert werden. Generell ist das Format unabhängig von dem benutzten Kontext und den spezifizierten Kriterien.

[18] Die einzelnen Kriterien werden konkret in Anfragen gemäß der Anfragesprache (vgl. Abschnitt 6.4) der *SpectoML* transformiert, welche an die semi-strukturierte Datenbasis gestellt werden.

9.4 Programmiermodell

Living Documents unterstützen die Realisierung verteilter und dynamischer Informationssysteme. Das Programmiermodell der *Living Documents*, das den grundsätzlichen Rahmen für die Programmierung von *Living Documents* vorgibt, setzt sich aus zwei Teilen zusammen:

- Die *Codekomponente* unterstützt ein verhaltensbezogenes Programmiermodell.
- Die *semi-strukturierte Datenbasis* bzw. das *Knowledge Repository* unterstützen ein wissensbezogenes Programmiermodell.

Die Aufteilung in zwei verschiedene Arten von Programmiermodellen, die in *Living Documents* flexibel kombiniert werden können, bildet die Grundlage für die Realisierung unterschiedlicher Informationssysteme. In Kapitel 10 werden anhand zweier exemplarischer Fallstudien die potentiellen Vorteile von *Living Documents* für die Entwicklung proaktiver Informationssysteme erläutert.

9.4.1 Verhaltensbezogenes Programmiermodell

Die Aktivität eines *Living Documents* steht im Mittelpunkt des verhaltensbezogenen Teils des Programmiermodells der *Living Documents.* Generell ist eine Aktivität durch die Applikationslogik des Informationssystems bestimmt. Im Zusammenhang mit *Living Documents* bezieht sich die Applikationslogik auf die korrekte Bearbeitung und Verwaltung eines realen, virtuellen oder hybriden Dokuments.
Die Aktivitäten eines *Living Documents* liegen in der Codekomponente und sind abhängig von der Art der Realisierung der Codekomponente. Falls diese als *Okeanos*-Agent implementiert wird, ist das Verhalten eines *Living Documents* maßgeblich durch die vom Agentenframework *Okeanos* unterstützten *objektorientierten* und *regelbasierten Programmiermodelle* bestimmt, die in Abschnitt 5.2.2 beschrieben sind.
Neben den Aktivitäten, die sich unmittelbar auf die Applikationslogik eines Dokuments beziehen, ist die Interaktion und Koordination mit der Umwelt und anderen *Living Documents* für das Gesamtverhalten von Bedeutung. Aktionen und Ereignisse, die außerhalb des Geltungsbereichs eines *Living Documents* stattfinden, können mittelbar Einfluß auf das Verhalten eines *Living Documents* nehmen.
Hinsichtlich der Koordinierung von *Living Documents*, die Bestandteile eines verteilten Systems sind, unterscheidet man prinzipiell zwei Fälle:

- *Lokale Koordinierung*. Unter der lokalen Koordinierung werden die Interaktionen zwischen *Living Documents* verstanden, die sich in einer gemeinsamen Ausführungsumgebung bzw. auf einer *Lounge* befinden. Das Agentenframework *Okeanos* stellt dafür nachrichtenbasierte Kommunikationsstrukturen zur Verfügung, mit denen *Living Documents* Nachrichten direkt untereinander austauschen (vgl. Abschnitt 5.3.1).

- *Globale bzw. verteilte Koordinierung*. Unter der globalen Koordinierung werden die Interaktionen zwischen *Living Documents* oder externen Systemkomponenten verstanden, die in mehreren und verteilten Ausführungsumgebungen ablaufen. Dabei müssen insbesondere zwei Aspekte berücksichtigt werden:

  – *Heterogenität*. Die Koordinierung zwischen *Living Documents* und externen Systemkomponenten wird durch die heterogene Vielfalt der Systemkomponenten erschwert. In Kapitel 7 über die Beschreibung von *Specto* – einer Infrastruktur zum Monitoring von verteilten Systemkomponenten auf der Basis von *SpectoML*-Dokumenten – wird gezeigt, wie man XML als ein Bindeglied zwischen heterogenen Systemwelten einsetzen kann. Dadurch ist eine globale Koordinierung möglich, in der *Living Documents* auf die Ereignisse und Zustände von beliebigen, externen Systemkomponenten reagieren können[19].

  – *Lose Kopplung*. Die Koordinierung zwischen verteilten *Living Documents* setzt zwingend eine inhaltliche und systemtechnische Kopplung der beteiligten *Living Documents* voraus. Einerseits kann ein *Living Document* nur auf diejenigen Ereignisse reagieren, die es kennt und verarbeiten kann, andererseits setzt es die Existenz eines Kommunikationswegs voraus, auf dem das Ereignis vom auslösenden zum empfangenden *Living Document* gelangt. Für eine skalierbare Koordinierung ist es erforderlich, die beteiligten *Living Documents* lose miteinander zu koppeln. Dabei sind die Beziehungen und Bindungen zwischen den *Living Documents* durch die Einführung logischer Indirektionen voneinander getrennt [SN92]. Eine lose Kopplung unterstützt die Realisierung von skalierfähigen, verteilten Informationssystemen. In Kapitel 10 wird anhand zweier Fallstudien der Einsatz von *Living Documents* veranschaulicht. Es wird dargestellt, wie mehrere und verteilte *Living Documents* miteinander lose verbunden sind und wie sie auf bestimmte globale Ereignisse reagieren.

9.4.2 Wissensbezogenes Programmiermodell

Generell ist ein *Knowledge Repository* (KR) eine Sammlung von Aussagen in einer bestimmten Repräsentation, die gewisse Aussagen über die modellierte Welt betreffen [LL00]. Im Zusammenhang mit einem *Living Document* beziehen sich diese Aussagen auf die Zustände und Eigenschaften von Dokumenten – die Dokumentenzustandsinformationen. Berücksichtigt man einen sehr allgemeinen Dokumentenbegriff, der sich auf reale, virtuelle und hybride Dokumente stützt, beziehen sich diese Aussagen auf unterschiedliche Arten von Informationen.

[19] Unter *beliebigen* Systemkomponenten werden diejenigen Komponenten verstanden, die über eine XML-Schnittstelle verfügen. Da dies keine prinzipielle Einschränkung darstellt, wird an dieser Stelle von beliebigen Systemkomponenten gesprochen.

Ein KR ist dadurch gekennzeichnet, daß die Menge der Aussagen bzw. der Dokumentenzustandsinformationen eine Wissensbasis bildet. Durch das Hinzufügen neuer Aussagen wird die Wissensbasis fortwährend erweitert. Während des Dokumentenlebenszyklus (vgl. Abbildung 9.2), der sich durch das ständige Hinzufügen und Entfernen von Dokumentenzustandsinformationen auszeichnet, entsteht so eine kontinuierlich aktualisierte Wissensbasis, die bestimmte Aussagen über die modellierte Welt eines Dokuments beinhaltet.
Eine Wissensbasis in einem *Living Document* kann u. a. für die folgenden zwei Aufgaben sinnvoll genutzt werden:

- *Steigerung der Softwarequalität.* Mögliches fehlerhaftes oder unvorhergesehenes Verhalten der *Living Documents* wird dadurch herausgefunden, daß man versucht, den für den Fehler verantwortlichen Dokumentenzustand im KR zu identifzieren. Schimkat et al. [SSDKK00] beschreiben ein objektorientiertes Framework und eine Infrastruktur zum Testen von Softwarekomponenten. Die Wissensbasis für die Auswertung der Testabläufe setzt sich aus einzelnen XML-basierten Testdokumenten zusammen. Jedes Testdokument ist als *SpectoML*-Dokument realisiert, wobei die Dokumentenzustandsinformationen Aussagen über den Testablauf und die beteiligten Softwarekomponenten beinhalten.

- *Analysefunktionen.* Ein KR liefert aussagekräftige Funktionen, die komplexe Abläufe in den *Living Documents* exakt beschreiben und nachvollziehbar machen. Heumesser und Schimkat [HS01] beschreiben die Auswertungsmöglichkeiten von Dokumentenzustandsinformationen auf Basis von *SpectoML*-Dokumenten. Für die Auswertung transformieren sie die Dokumentenzustandsinformationen nach Prolog [SS86], um auf die mächtigen Anfrage- und Auswertungsmechanismen von Prolog zugreifen zu können. Im folgenden sind mehrere typische Fragestellungen bei der Analyse von *Living Documents* in verteilten Informationssystemen aufgelistet:

  - *Wann* und *warum* bewegte sich ein bestimmtes *Living Document* zwischen zwei Orten?
  - *Wie* ist der globale Systemzustand eines verteilten Informationssystems, in dem die einzelnen lokalen Zustände über semi-strukturierte Datenbasen festgehalten werden ?
  - *Welche* komplexen Abläufe in einem verteilten Informationssystem sind als kritisch zu bewerten, ohne daß dies a priori zu erkennen ist[20]?

[20] In Abschnitt 6.5.3 und in [Cas01] wird *Represento* – ein System zur Visualisierung von verteilten Prozessen – beschrieben, das auf der Basis von *SpectoML*-Dokumenten komplexe Abläufe in verteilten Informationssystemen grafisch und interaktiv visualisiert. Da die Systemschnittstelle von *Represento* durch *SpectoML*-Dokumente realisiert ist, können die semi-strukturierten Datenbasen der *Living Documents* als Informationsquellen für die interaktive Visualisierung in *Represento* dienen.

Generell orientiert sich der wissensbasierte Ansatz des Programmiermodells für *Living Documents* an einem datenbanktypischen Zugriff auf Informationen und an einer netzwerkähnlichen Repräsentation und Verarbeitung von Wissen. Der Zugriff auf die Dokumentenzustandsinformationen erfolgt über die in Abschnitt 6.4 beschriebene Anfragekomponente der *SpectoML*. Die Anfragekomponente ist durch ihren JDBC-ähnlichen Zugriff auf die Datenbasis gekennzeichnet. In [HS01] illustrieren Heumesser und Schimkat, wie neben dem typischen (relationalen) Datenbankzugriff auch bestimmte Inferenztechniken im Zusammenhang mit semistrukturierten Datenbasen auf der Basis der *SpectoML* genutzt werden können.
Konzeptionell beinhaltet ein *Living Document* eine Datenstruktur, auf der bestimmte wissensbezogene Operationen definiert und implementiert werden können. Im folgenden werden anhand der Lokation der Datenstruktur bzw. der Wissensbasis und verschiedener Auswertungsmechanismen im wesentlichen drei Fälle unterschieden[21]. Hinsichtlich der Auswertung wird zwischen regelbasierten und prozeduralen Techniken unterschieden:

1. *Lokal vorhandenes Wissen auf der Basis von Regeln*. In diesem Fall ist das Wissen lokal an einem Ort (z. B. an einer Ausführungsumgebung) vorhanden und die Verarbeitung erfolgt auf der Grundlage von Regeln.

   Schimkat et al. [SBS+00] beschreiben eine Fallstudie für eine verteilte Berechnung im Bereich des Symbolischen Rechnens auf der Basis von mobilen *Okeanos*-Agenten. Das Wissen um den aktuellen Fortschritt der Berechnung steht lokal an jeder Ausführungsumgebung bzw. *Lounge* (vgl. Abschnitt 5.3.2) zur Verfügung. Jeder Agent, der eine Teilformel zu berechnen hat, ist auf die Berechnungsdienste angewiesen, die an den verteilten *Lounges* verfügbar sind. Dadurch entsteht ein konkurrierender Zugriff um die Berechnungsdienste zwischen den mobilen *Okeanos*-Agenten. Da es sich bei dem Berechnungsproblem um ein irreguläres Problem handelt, das sich nicht deterministisch in geeignete Unterberechnungen aufteilen läßt, entsteht ein unvorhersehbarer Ablaufpfad für die Berechnung. Die Applikationslogik für die Bestimmung des nächsten verfügbaren Berechungsdienstes an einer bestimmten *Lounge* wird durch JESS-Regeln und -Fakten bestimmt. Die Basis für die Entscheidung bildet das Wissen, welches lokal an der jeweiligen *Lounge* zur Verfügung steht.

2. *Zusätzliches, global vorhandenes Wissen auf der Basis von verteilten SpectoML-Dokumenten und prozeduralen Auswertungsmechanismen*. Im Hinblick auf die unter (1) beschriebene verteilte Berechnung und deren grundsätzliche Ausgangssituation steht nun zusätzliches, globales Wissen

[21] Die aufgezeigten Fälle beziehen sich primär auf mehrere Fallstudien, in deren Mittelpunkt *SpectoML*-basierte Datenbasen und *Okeanos*-Agenten stehen. Da diese beiden Komponenten die wesentlichen Bestandteile eines *Living Documents* ausmachen, stehen die vorgestellten Fälle repräsentativ auch für *Living Documents* im allgemeinen.

für die einzelnen *Okeanos*-Agenten zur Verfügung. Das lokal an jeder *Lounge* vorhandene Wissen wird via *SpectoML*-basierten Datenbasen global zusammengeführt. Dadurch entsteht eine gemeinsame, zentrale Wissensbasis, die für die verteilte Berechnung genutzt werden kann. Schimkat et al. beschreiben in [SFK01a] prozedurale Auswertungsmöglichkeiten, in denen ein spezieller *Okeanos*-Agent eine Reihe von Anfragen gemäß der Anfragesprache (vgl. Abschnitt 6.4) der *SpectoML* an die Wissenbasis stellt. Die Ergebnisse der Anfragen werden in prozeduraler Form gesammelt, aufbereitet und an die beteiligten *Lounges* und Agenten weitergegeben.

3. *Zusätzliches, global vorhandenes Wissen auf der Basis von verteilten SpectoML-Dokumenten und regelbasierten Auswertungsmechanismen.* Im Hinblick auf den Fall (2) unterscheidet sich der dritte Fall durch die regelbasierte Art und Weise der Auswertung. Die global vorhandene und XML-basierte Wissensbasis wird zu Prolog-Regeln und Fakten [SS86] transformiert[22]. Die Auswertung der Wissensbasis, die den globalen Berechnungszustand der verteilten Berechnung in [SBS+00] widerspiegelt, erfolgt in diesem Falle durch die Verwendung von Regeln, die in Prolog formuliert sind. Schimkat und Heumesser [HS01] veranschaulichen den sinnvollen Einsatz von Inferenztechniken im Zusammenhang mit Dokumentenzustandsinformationen, die in XML-basierten Datenbasen abgelegt sind.

9.5 Einordnung in die Taxonomie

Die Einordnung der *Living Documents* in die in Kapitel 3 erörterte Taxonomie für Informationssysteme ist nur bedingt möglich. Analog zu der Argumentation bei dem leichtgewichtigen Applikationsserver *Respondeo* (vgl. Kapitel 4) und dem Agentenframework *Okeanos* (vgl. Kapitel 5) stellen *Living Documents* an sich kein Informationssystem dar, sondern liefern vielmehr eine Infrastruktur für die Realisierung von Informationssystemen.
Allerdings ist der konzeptionelle Unterschied zwischen einer Infrastruktur und einem Informationssystem nicht mehr so offensichtlich, wie er bei *Respondeo* oder *Okeanos* noch gewesen ist. Denn jedes *Living Document* stellt bereits eine *abgeschlossene Einheit* dar, die – isoliert betrachtet – ein Mikro-Informationssystem bildet. In dessen Mittelpunkt steht ein reales, virtuelles oder hybrides Dokument, welches von einem *Living Document* verwaltet wird.
Die in dieser Arbeit gewählte Realisierung der *Living Documents* basiert einerseits auf dem Agentenframework *Okeanos* für die Gestaltung der Code-Komponente und andererseits auf der *SpectoML* für die Entwicklung einer semi-strukturierten

[22] Das Informationssystem *XRePro* [HS01], das für die Transformation zuständig ist, bietet eine Web-Schnittstelle, über welche die einzelnen *SpectoML*-basierten Datenbasen auf einfache Art und Weise innerhalb eines verteilten Systems zusammengeführt werden.

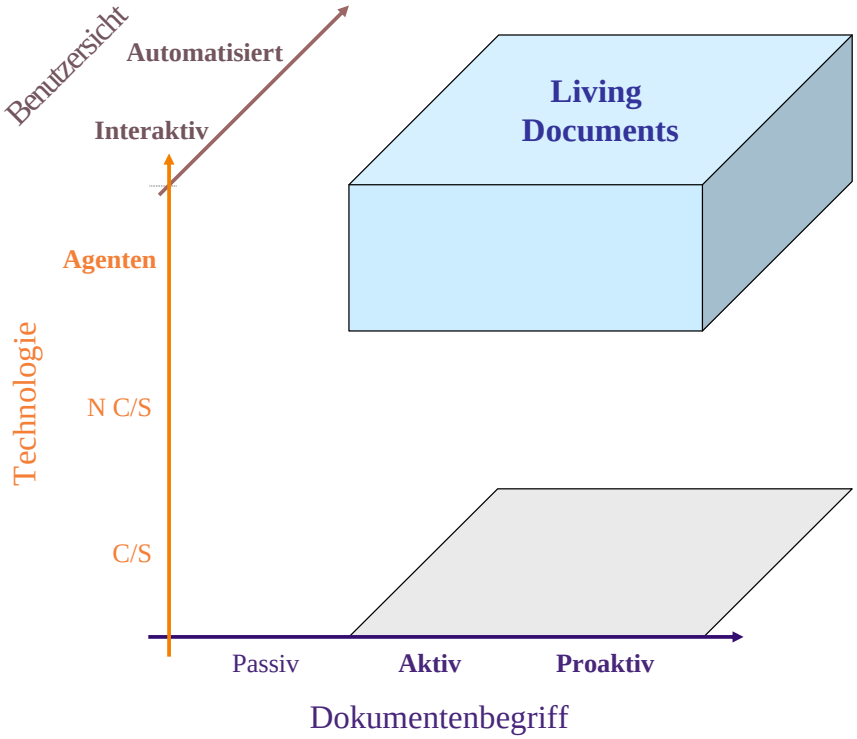


Abbildung 9.4: Einordnung der *Living Documents* in die Taxonomie.

Datenbasis. Daher orientiert sich die in Abbildung 9.4 vorgenommene Einordnung der *Living Documents* an der Klassifizierung des Agentenframeworks *Okeanos*, die bereits in Abschnitt 5.4 erfolgt ist. Bei der Klassifizierung der *Living Documents* fällt auf, daß passive Dokumente nicht Bestandteil von Informationssystemen auf der Basis von *Living Documents* sind. Die Existenz der Code-Komponente in einem *Living Document* bedingt immer eine Form der Aktivität hinsichtlich des gesamten Dokuments. Für eine genauere Beschreibung der Einordnung wird explizit auf Abschnitt 5.4 verwiesen.

9.6 Abgrenzung zu anderen Forschungsarbeiten

In diesem Abschnitt werden einige Forschungsarbeiten beschrieben, die im engeren Sinne mit *Living Documents* und den zugrunde liegenden Konzepten verwandt sind. All diesen Arbeiten ist das Ziel der Generierung eines aktiven Dokumentenbegriffs (vgl. Abschnitt 3.2.3) gemeinsam, welcher die Basis für neuartige Informationssysteme bildet. Die im folgenden beschriebenen Projekte unterscheiden sich allerdings hinsichtlich der Anwendungsdomänen, der Berücksichtigung von mobilen Eigenschaften von Dokumenten und der Bestimmung von aktivem Verhalten.

9.6.1 Placeless

Placeless [DELS99b, EL99, DELS99a] ist ein Dokumentenverwaltungssystem, das in der Forschungsabteilung von Xerox PARC in den USA entwickelt wird. Die primäre Anwendungsdomäne von *Placeless* sind sogenannte Desktop-Applikationen für die flexible Verwaltung von Dokumenten. Im Mittelpunkt einer solchen Anwendung steht der Benutzer mit seinen proprietären Präferenzen hinsichtlich der Organisation und Verwaltung seiner Dokumente. Das Ziel von *Placeless* besteht in der Bereitstellung einer Dokumenteninfrastruktur, die einerseits auf der Benutzerebene individuelle Gestaltungsräume für die Organisation von Dokumenten bietet und andererseits auf der darunterliegenden Systemebene ein einheitliches Management der Dokumente ermöglicht.
Die Basis für *Placeless* bildet ein Dokumentenbegriff, der primär auf Eigenschaften beruht. Eigenschaften in *Placeless* bestehen aus einer Menge von Name-Wert-Paaren, die in keiner definierten Ordnung bzw. Reihenfolge einem Dokument zugeordnet sind. Der Zugriff auf und die Suche nach Dokumenten erfolgt ausschließlich über ihre Eigenschaften. Jeder Benutzer eines Dokuments kann seine eigenen Eigenschaften definieren, d. h. er kann nicht nur spezielle Werte für bereits existente Eigenschaften eines Dokuments belegen, sondern auch neue Name-Wert-Paare einführen. Ähnlich der Generierung von dynamischen Sichten auf ein *Living Document* (vgl. Abschnitt 9.3.2) bilden Dokumente in *Placeless* die Basis für individuelle, unterschiedliche und komplexe Sichten auf ein Dokument. In dieser Hinsicht vereint das Eigenschaftskonzept in *Placeless* unterschiedliche Dokumentenbenutzer.
Aktive Eigenschaften in *Placeless* sind Eigenschaften, die aktive Komponenten beinhalten. In dieser Hinsicht umfaßt der Begriff der Eigenschaft in *Placeless* sowohl statische Ausprägungen von Attributen (z. B. den Namen und den Autor eines Dokuments) als auch aktives Verhalten, das über Code-Fragmente (z. B. in der Programmiersprache Java) realisiert ist.
Die Unterschiede zwischen Dokumenten in *Placeless* und den in dieser Arbeit vorgestellten *Living Documents* bestehen vor allem in den folgenden Punkten:

- *Kapselung*. *Placeless* setzt eine mächtige Ausführungsumgebung für den Zugriff auf die Dokumente voraus. Darüber hinaus sind die Eigenschaften eines Dokuments vom tatsächlichen Dokumenteninhalt getrennt. Um auf den Dokumenteninhalt zuzugreifen, sind mehrere Verarbeitungsprozesse und Systemkomponenten notwendig, die im Widerspruch zu den Kapselungseigenschaften (vgl. Abschnitt 9.3.1) von *Living Documents* stehen.

- *Spezifikation von Eigenschaften*. Während Eigenschaften in *Placeless* als simple Name-Wert-Paare abgelegt sind, definieren sich Eigenschaften in *Living Documents* als Dokumentenzustandsinformationen, die gemäß der *SpectoML* (vgl. Abschnitt 6.2.1) aufgebaut sind. Die DTD für die Beschreibung der Dokumentenzustandsinformationen ist sehr mächtig, erweiterbar

und bietet die Möglichkeit, komplexe Eigenschaften und Beziehungen der Dokumente einheitlich abzulegen.

- *Mobilität.* Die logische Mobilität bzw. Software-Mobilität [CPV97] (vgl. Abschnitt 5.1) von Dokumenten spielt in *Placeless* nur eine untergeordnete Rolle. Im Vordergrund stehen bei *Placeless* Desktop-Applikationen, die von fixen und statisch vorgegebenen Lokationen der Dokumente ausgehen. Im Gegensatz dazu unterstützen *Living Documents* die Mobilität von Dokumenten in zweierlei Hinsicht:

  – Durch die Realisierung der Code-Komponente als mobiler Agent (*Okeanos*-Agent) sind *Living Documents* in der Lage, ihre Örtlichkeiten dynamisch zu wechseln.

  – Die Eigenschaften und Folgen einer expliziten Berücksichtigung der Software-Mobilität – wie z. B. die Änderung der Örtlichkeit oder die Berücksichtigung neuer Umgebungsparameter nach einer Migration zwischen zwei Ausführungsumgebungen – werden einheitlich bestimmt. In *Living Documents* findet keine prinzipielle Unterscheidung zwischen statisch festgelegten (z. B. der Name eines Dokuments) und dynamisch ermittelten (z. B. aktueller Ort des Dokuments) Eigenschaften statt, sondern alle Eigenschaften werden uniform in der semistrukturierten Datenbasis abgelegt.

- *Automatisierte Interaktionen.* Ein Problem bei der beliebigen Bestimmung und Zuweisung von Eigenschaften zu Dokumenten in *Placeless* ist die fehlende, gemeinsame Basis für die Festlegung der inhaltlichen Bedeutungen der zugewiesenen Eigenschaften. So ist jeder Benutzer lediglich in der Lage, die ihm a priori bekannten Eigenschaften anzufragen oder zu verwenden. Die explizite Darstellung der Bedeutungen der Dokumentenzustandsinformationen in den *Living Documents* (vgl. Abschnitt 6.2.2) bietet dagegen die Möglichkeit, die Semantik der Eigenschaften zur Laufzeit zu ermitteln. Dies ist eine Anforderung an Informationssysteme, die sich durch automatisierte Interaktionen auszeichnen (vgl. Abschnitt 2.1.2.3 über automatisierte Arbeitsformen).

9.6.2 Intensionale Dokumente

Ein intensionales Dokument (IntDoc) [SMP00, Wad99, Bro98] ist ein Dokument, in das aktive Komponenten eingebettet sind. Ein IntDoc ist immer im Zusammenhang mit seiner Umgebung zu betrachten. Die Umgebung ist durch sich fortwährend ändernde Gegebenheiten, die auf die Verarbeitung eines Dokuments Einfluß nehmen können, charakterisiert. Wenn man den Zielen der intensionalen Programmierung [PP95] folgt, kann eine Änderung in der Umwelt – dem sogenannten Umweltkontext – das Verhalten und die Benutzung eines Dokuments beeinflussen.

Ein IntDoc erweitert den traditionellen, statisch bestimmten Dokumentenbegriff, in dessen Mittelpunkt Dokumente lediglich als passive Container von digitalen Inhalten fungieren. Durch die Einführung von IntDocs sollen Dokumente in die Lage versetzt werden, in Abhängigkeit vom aktuellen Kontext unterschiedliche Versionen des Dokuments bereitzustellen. Durch diese Art der Versionierung von Dokumenten besteht die Möglichkeit, den Dokumenteninhalt adäquater an die aktuellen Gegebenheiten anzupassen und zu präsentieren.
Für die Speicherung und die Suche nach Kontexten wird in IntDocs ein sogenanntes „Kontextgedächtnis“ (Context Memory) benutzt. Aus konzeptioneller Hinsicht entspricht dies der semi-strukturierten Datenbasis eines *Living Documents*. Dort können ebenfalls beliebige Kontextinformationen, die für das betreffende Dokument von Bedeutung sind, einheitlich abgelegt und wiedergefunden werden.
Im Gegensatz zu *Living Documents* spielen die Aspekte der Mobilität und einer automatisierten Form der Interaktion zwischen Dokumenten bei IntDocs keine Rolle. Im Vordergrund der IntDocs stehen Applikationen, in denen aktive Dokumente der Typen A und B (vgl. Abschnitt 3.2.3) auf bestimmte Ereignisse in der Umwelt reagieren und eine entsprechende Version des Dokuments dynamisch generieren. So unterstützt beispielsweise die *Intensional Markup Language* [Wad99] die Berücksichtigung unterschiedlicher Kontexte für die dynamische Darstellung von Dokumenten im Web.

9.6.3 Adlets

Adlets [CZ01] sind Abstraktionen für die Spezifizierung von Metadaten. *Adlets* und die dazu gehörigen Infrastrukturen (z. B. ein Kommunikationsserver) werden im Rahmen eines Forschungsprojekts an der Universität von Pittsburgh in den USA entwickelt.
Der Dokumentenbegriff, der den *Adlets* zugrunde liegt, ist durch eine Trennung zwischen den Metadaten und dem Dokumenteninhalt gekennzeichnet. Die Metadaten werden durch *Adlets* repräsentiert. Ein *Adlet* besteht u. a. aus einer Referenz auf das eigentliche Dokument, einem Typ und aus mehreren Datenstrukturen zur Beschreibung der Eigenschaften eines Dokuments.
Ein *Adlet* ist in der Lage, sich in einem Netzwerk zu bewegen und mit anderen *Adlets* zu kommunizieren. Ein solches Netzwerk setzt sich aus den Systemkomponenten der *Adlet*-Infrastruktur – wie z. B. einem Kommunikationsserver – zusammen. Der Kommunikationsserver ist für das Versenden und Empfangen von *Adlets* zuständig und muß auf jedem Netzwerkknoten vorhanden sein. Generell sind diese Beschreibungen von Dokumenten in der Lage, nach bestimmten Informationen, die in anderen *Adlets* vorhanden sind, zu suchen. Zudem sind mehrere Operationen auf einem *Adlet* definiert, die den Zusammenschluß oder die Kooperation zwischen *Adlets* ermöglichen.
Das Wandern und die Mobilität von Metadaten, wie sie in *Adlets* realisiert sind, ist keine prinzipielle Einschränkung für die *Living Documents*. Im Zusammenhang mit *Living Documents* stellt ein *Adlet* ein spezielles *Living Document* dar, das aus-

schließlich aus einer Code-Komponente und einer semi-strukturierten Datenbasis für die Speicherung der Metadaten besteht. Der Daten- bzw. Blob-Teil des *Living Documents* ist in diesem Falle leer.

Kapitel 10

Fallstudien für *Living Documents*

In diesem Kapitel werden zwei Fallstudien erörtert, welche den Einsatz von *Living Documents* für die Realisierung von verteilten Informationssystemen auf der Basis des Client/Server-Prinzips (vgl. Abschnitt 2.3.1) und der netzwerkbasierten Organisation von Daten (vgl. Abschnitt 2.3.2) weiter untersuchen. Die vorgestellten Fallstudien beruhen auf einem Informationssystem, das als typischer Repräsentant mehrstufiger Client/Server-Informationssysteme betrachtet werden kann:

- In Abschnitt 10.1 wird ein mehrstufiges Client/Server-Informationssystem namens *PaperBase*, das aus einem Thin-Client, einem Applikationsserver sowie einer relationalen Datenbank besteht, beschrieben. Die *PaperBase* als Repräsentant datenbankzentrierter Informationssysteme verwaltet die eigentlichen Daten und Informationen in Datenbanken bzw. Datenbankmanagement-Systemen. Neben einer allgemeinen Beschreibung der Funktionen, Eigenschaften und Architektur steht im Abschnitt 10.1.3 die Transformation der *PaperBase* von einem mehrstufigen Client/Server-Informationssystem zu einem verteilten Informationssystem *PaperBaseLD* [SK02] auf der Grundlage von *Living Documents* im Vordergrund. Dabei werden die Daten und Informationen, die ursprünglich in der Datenbank gehalten wurden, zu *Living Documents* transformiert. Nach der Einführung der *Living Documents* entsteht ein verteiltes Informationssystem, in dem die Existenz einer Datenbank nicht mehr zwingend erforderlich ist.

  Anhand der Fallstudie in Abschnitt 10.1 werden primär folgende Ziele verfolgt:

  - Durchführung einer Machbarkeitsstudie für den Einsatz von *Living Documents* in verteilten Informationssystemen.
  - Transformation von Informationssystemen mit einem passiven Dokumentenbegriff zu aktiven Informationssystemen, in deren Mittelpunkt aktive Dokumente stehen.
  - Untersuchung verschiedener Möglichkeiten der Informationssuche in Informationssystemen, die auf *Living Documents* basieren.

- *Living Hypertext.* In Abschnitt 10.2 wird das netzwerkbasierte Informationssystem *Living Hypertext* [SKN02] erörtert. Der grundsätzliche Aufbau der Architektur des *Living Hypertextes* basiert auf dem Informationssystem *PaperBaseLD*, das im Abschnitt 10.1 vorgestellt wird. Generell schließt ein *Living Hypertext* die Brücke zwischen datenbank- und web-basierten Informationssystemen. Die Grundlage hierfür bildet ein verallgemeinerter, proaktiver Dokumentenbegriff, der Dokumenten sowohl daten- als auch verhaltensbezogene Eigenschaften zuordnet. So kapseln *Living Documents* unterschiedliche Informationsquellen auf eine uniforme Art und Weise. Die Fähigkeit, verschiedene Sichten auf ein *Living Document* dynamisch zu generieren (vgl. Abschnitt 9.3.2), ermöglicht die Verwendung eines Dokuments in verschiedenen – datenbank- und web-basierten – Anwendungsszenarien.

  Anhand der Fallstudie in Abschnitt 10.1 werden primär folgende Ziele verfolgt:

  - Generierung eines *Living Hypertext*, der die grundsätzlichen Eigenschaften eines traditionellen Hypertextes wie z. B. dem Web übernimmt. Analog zum Web sind die Autonomie der Datenbasen und deren dezentrale Organisation in einem Netzwerk von Dokumenten (vgl. Abschnitt 2.3.2) wichtige Eigenschaften eines *Living Hypertextes.* Der wesentliche Unterschied besteht in der Ablösung der primär passiven Dokumente im Web durch die Transformation in aktive und proaktive Dokumente, die sich in den *Living Documents* widerspiegeln[1].
  - Bereitstellung verschiedener interaktiver und automatisierter Formen der Informationssuche. Eine datenbank-, agenten- und vor allem eine *navigationsbasierte Informationssuche* unterstreichen die flexiblen Einsatzmöglichkeiten von *Living Documents.*

- Im Abschnitt 10.3 erfolgt eine Einordnung der Informationssysteme *PaperBaseLD* und *Living Hypertext* in die im Kapitel 3 vorgestellte Taxonomie.

10.1 *PaperBase* – ein typisches, mehrstufiges Web-Informationssystem

PaperBase ist ein verteiltes Informationssystem zur Verwaltung von individuellen, virtuellen Arbeitsräumen im Web[2]. Ein virtueller Arbeitsraum besteht aus einer Menge von digitalen Dokumenten, die ein Benutzer in individuellen und hierarchischen Strukturen anordnet. Der Zugriff und die Verwaltung auf den Arbeitsraum

[1] Der Fokus liegt auf denjenigen Dokumenten im Web, die z. B. durch die Web-Suchmaschinen erfaßt werden. Bei diesen Dokumenten handelt es sich primär um *passive Dokumente*(vgl. Abschnitt 3.2.2).

[2] Siehe auch die Homepage der *PaperBase* unter http://www-sr.informatik.uni-tuebingen.de/~schimkat/pb für weitere Informationen.

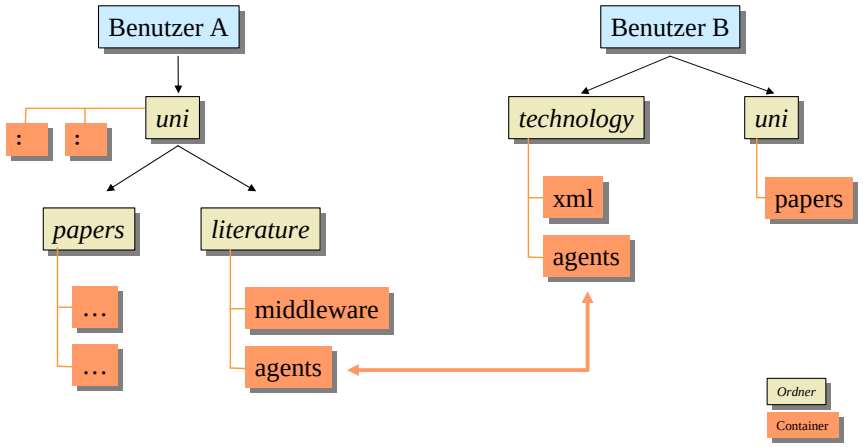


Abbildung 10.1: Individuelle Organisation virtueller Arbeitsräume im Informationssystem *PaperBase*.

erfolgt ausschließlich über das Web. Ein Arbeitsraum wird als virtuell bezeichnet, wenn dieser nicht an reale vorhandene Örtlichkeiten gebunden ist[3].
Ein weiteres Ziel bei der Realisierung der *PaperBase* ist die gemeinsame Nutzung über das Web von Teilen eines virtuellen Arbeitsraums mit anderen Benutzern. Dadurch bilden sich virtuelle Dokumentenorganisationen, die sich überschneiden und unabhängig von der aktuellen Örtlichkeit den Benutzern zur Verfügung stehen. In der folgenden Auflistung werden die Begriffe *Container*, *Dokument* und *Ordner*, die Bestandteil der spezifischen Terminologie von *PaperBase* sind, beschrieben:

- Ein *Container* innerhalb von *PaperBase* enthält eine benutzerdefinierte Menge von Dokumenten. Jeder Container besitzt eine Reihe von Attributen, wie z. B. einen Typ, eine Liste von Stichwörtern, einen Autor oder einen Titel. Jeder Container kann zwischen Benutzern im Web gemeinsam verwendet werden. Generell stellt ein Container die grundsätzliche Einheit dar, in der Information und Wissen zwischen Benutzern geteilt bzw. gemeinsam verwendet werden kann.

- Ein *Dokument* besteht aus einem digitalen Inhalt und wird logisch jeweils einem Container zugeordnet. Jedes Dokument enthält ein Attribut *note*, in dem das Dokument kurz beschrieben ist.

- Ein *Ordner* ist – ähnlich wie ein Container – eine Abstraktion für die Gruppierung von Dokumenten. Jeder Ordner ist ein Bestandteil einer baumarti-

[3] Bei der Realisierung von *PaperBase* stand die Entwicklung einer komplexen, mehrdimensionalen grafischen Benutzerschnittstelle *nicht* im Vordergrund. Folglich wurde die Realisierung einer wirklichkeitsnahen Benutzerschnittstelle, die dem Benutzer den Übergang zwischen virtueller und realer Welt erleichtern soll (*virtual reality*), nicht weiter verfolgt.

gen Struktur, in der alle Ordner eines Benutzer enthalten sind. In der Abbildung 10.1 sind die individuellen Hierarchien von zwei Benutzern dargestellt. Jeder Knoten in dieser Hierarchie entspricht einem Ordner, der jeweils eine Menge von Containern enthält. `Benutzer A` und `B` teilen sich den Container `agents`, wobei der `Benutzer A` diesen Container in seiner Ordnerhierarchie unter `uni:literature` und `Benutzer B` unter `technology` abgelegt hat[4].

Die primären Operationen, die in einem virtuellen Arbeitsraum in *PaperBase* zur Verfügung stehen, sind das Hinzufügen und Entfernen von Dokumenten aus Containern, die Freigabe von bestimmten Containern für die gemeinsame Nutzung mit weiteren Benutzern, eine Versionsverwaltung der Dokumente und die Organisation der Container und Ordner.
Im folgenden werden die primären Ziele beim Entwurf der *PaperBase* zusammengefaßt:

1. *Web-basierter Zugriff.* Durch den web-basierten Zugriff ist die Benutzung von *PaperBase* unabhängig von den aktuellen Örtlichkeiten der Benutzer[5]. Der Zugriff auf *PaperBase* sowie dessen Diensten und Dokumenten erfolgt ausschließlich über den in Kapitel 4 vorgestellten leichtgewichtigen Applikationsserver *Respondeo* und dessen Web-Schnittstelle. Die Verwaltung, das Hinzufügen oder Entfernen von Dokumenten, die Freigabe von Containern für den gemeinsamen Zugriff sowie die Suche nach Dokumenten oder Containern erfolgt ausschließlich über die web-basierte Schnittstelle von *PaperBase.*

2. *Individuelle Benutzerhierachien.* Der Aufbau gemeinsamer Arbeitsräume über das Web unterstützt Benutzeraktivitäten, die sich durch ein hohes Maß an Kollaboration auszeichnen. So wird beispielsweise ein virtueller Arbeitsraum bzw. dessen gemeinsam benutzter Container zu einer Daten-, Dokumenten- oder Wissensbasis für ein bestimmtes Thema. Die relevanten Inhalte bezüglich eines Themas spiegeln sich in den digitalen Dokumenten wider, die in den entsprechenden Containern organisiert sind.

   Ein virtueller Arbeitsraum, der sich durch web-basierte Zugriffe (vgl. 1) und die Gestaltungsmöglichkeit individueller und doch gemeinsamer Arbeitsräume auszeichnet, unterstützt Benutzer, die zwar gemeinsame Aktivitäten verfolgen, aber räumlich voneinander getrennt sind. Außerdem kann der Aspekt der räumlichen Trennung durch stark ausgeprägte mobile Eigenschaften der Benutzer (z. B. häufiges Wechseln der aktuellen Örtlichkeiten) verstärkt werden.

[4] Die Ordnerhierarchie `uni:literature` bezeichnet einen Pfad durch die benutzerdefinierte Hierarchie von Ordnern, die mit dem Wurzelelement `uni` beginnt.

[5] Die einzige Restriktion für den Zugriff auf *PaperBase* ist ein Zugang bzw. eine Verbindung zum Web.

3. *Push-basierte Sekundärdienste*. Neben den oben angeführten Diensten – wie z. B. das Hinzufügen von Dokumenten in Containern – gibt es auch sogenannte push-basierte Sekundärdienste, die in automatisierter Form mit den Benutzern interagieren. Ein Sekundärdienst stellt eine wichtige Dienstleistung für den Benutzer zur Verfügung, die aber nicht essentiell notwendig für den korrekten Ablauf eines Informationssystems ist. In *PaperBase* wurde ein spezieller E-Mail-Dienst integriert, der Benutzer über die Veränderungen in Teilen der virtuellen Arbeitsräume informiert[6]. Bei einem push-basierten Informationsfluß übernimmt der Benutzer lediglich die Rolle des Empfängers einer Anfrage. Der E-Mail-Dienst spielt dabei den Initiator, der für das Auslösen der Anfrage bzw. des Informationsflusses verantwortlich ist.

4. *Einfache Benutzbarkeit*. Beim Entwurf der *PaperBase* wurde auf ein komplexes Datenmodell verzichtet, das sich durch zahlreiche Attribute für die Spezifikation von Dokumenten, Containern und Ordnern auszeichnet. Generell ist zwischen komplexen Datenmodellen und einer einfachen Erfassungsmöglichkeit abzuwägen. Ein komplexes Datenmodell für Dokumente in *PaperBase* würde sich durch zahlreiche Attribute auszeichnen, welche die Basis für mächtige Anfragen bilden. Die Erfassungsmöglichkeit bezeichnet den Vorgang, in dem Dokumente in *PaperBase* eingeführt werden. Im allgemeinen erschwert eine große Anzahl an Dokumentenattributen die Erfassung, da für jedes Attribut bestimmte Werte zur Verfügung gestellt werden sollten.

   Beim Entwurf der *PaperBase* wurde dem Prinzip der Einfachheit gefolgt, d. h. in kürzester Zeit sollte ein Benutzer in der Lage sein, Dokumente in seinen virtuellen Arbeitsraum hinzuzufügen oder zu entfernen. Diese Vorgehensweise wird dadurch erleichtert, daß der Benutzer nur eine begrenzte Anzahl von Daten und Informationen bei der Erfassung der Dokumente an das Informationssystem weiterreichen muß.

10.1.1 Architektur und Eigenschaften

Als mehrstufiges Client/Server-Informationssystem, auf das über Web-Schnittstellen zugegriffen wird, ist *PaperBase* ein typischer Vertreter für mehrstufige Client/Server-Informationssysteme im allgemeinen[7]. Für die Architektur von *PaperBase* sind folgende Eigenschaften kennzeichnend (siehe Abbildung 10.2):

- Der Benutzer tritt als Initiator von Anfragen und Informationsflüssen auf, in dem er über den Web-Client (Thin-Client) auf den Applikationsserver

[6] Vgl. auch den Abschnitt 4.4.2 über die Beschreibung des E-Mail-Dienstes in *Respondeo*.

[7] Vgl. auch Abschnitt 2.3.1 für eine ausführliche Diskussion von Client/Server-Informationssystemen.

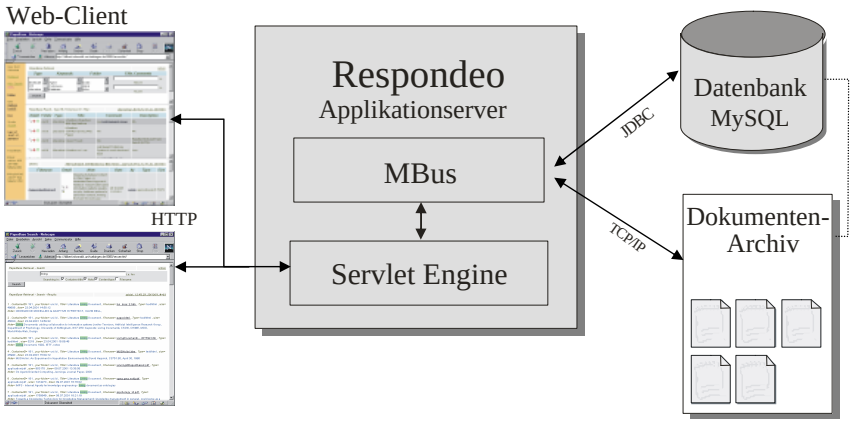


Abbildung 10.2: Systemüberblick der *PaperBase*.

Respondeo zugreift. Für die Kommunikation zwischen den Web-Clients und *Respondeo* wird HTTP [Wor00] benutzt. Der Zugriff von *Respondeo* und dessen Applikationsobjekten auf die relationale Datenbank erfolgt über den speziellen Datenbankdienst von *Respondeo*, der auf der Basis von JDBC [Sun02b] realisiert ist[8]. Die Benutzerschnittstelle auf der Seite des Web-Clients basiert auf HTML [RHJ99] und wird für jede Anfrage an *PaperBase* dynamisch aufgebaut. Der dynamische Aufbau der Benutzeroberfläche orientiert sich dabei u. a. am aktuellen Zustand der Dokumente, Ordner und Container.

- Dokumente in *PaperBase* und in vielen der heutigen mehrstufigen Client/-Server-Informationssystemen sind von einem *passiven Dokumentenbegriff* geprägt[9]. Passive Dokumente besitzen keine eigenen Aktivitäten, sondern fungieren ausschließlich als digitale Datencontainer. Die Dokumenteninhalte, die in *PaperBase* verwaltet und organisiert werden, können von unterschiedlichen Typen sein, z. B. Office-Dokumente, Grafikdateien oder serialisierte Java-Klassen. Grundsätzlich gibt es keine Beschränkung auf bestimmte Typen von Dokumenteninhalten in *PaperBase*.

- Im Mittelpunkt der Dokumentenorganisation in *PaperBase* stehen *hybride Dokumente* (vgl. Abbildung 3.3 auf der Seite 51)[10]. Hybride Dokumente nehmen eine Trennung zwischen den Eigenschaften und dem eigentlichen Dokumenteninhalt vor. Für diese Trennung wurde in *PaperBase* ein soge-

[8] Vgl. Abschnitte 4.4.1 und 4.3.1 für eine Beschreibung des Datenbankdienstes von *Respondeo*.

[9] Vgl. Abschnitt 3.2 für eine detaillierte Erörterung des passiven Dokumentenbegriffs, dessen Eigenschaften und der damit verbundenen potentiellen Probleme für die Realisierung von Informationssystemen.

[10] Vgl. Abschnitt 3.2.1.2 für eine Einführung und Erläuterung verschiedener Arten von Dokumenten – reale, virtuelle und hybride Dokumente.

nanntes *digitales Archiv* (vgl. Abbildung 10.2) entworfen, in dem die Dokumente strukturiert abgelegt sind. Die Eigenschaften eines Dokuments werden über vordefinierte Attribute in der relationalen Datenbank verwaltet[11]. Durch dieses Vorgehen wird lediglich ein Verweis auf das eigentliche Dokument in der Datenbank verwaltet. Ein Zugriff auf ein bestimmtes Dokument erfordert dabei stets einen mehrstufigen Prozeß, in dem im ersten Schritt die Datenbank kontaktiert wird, um die entsprechenden Informationen über die Lokation des Dokuments im digitalen Archiv zu erfahren. Im nächsten Schritt wird das betreffende Dokument direkt aus dem digitalen Archiv geholt.

Im Anhang B sind einige der grafischen Benutzeroberflächen des Web-Clients der *PaperBase* abgebildet.

10.1.2 Implementierung

Die Applikationslogik und die Systemkomponente *Respondeo*[12] sind in Java implementiert. Hinsichtlich der Datenbasis wurde ein relationales Datenmodell entworfen und in der relationalen Datenbank *MySQL* [MyS02b] umgesetzt.
Zum aktuellen Zeitpunkt sind ca. 1450 verschiedene Dokumente in ca. 130 Containern in *PaperBase* enthalten. Das digitale Archiv hat eine Größe von ca. 850 MB und die aktuelle Anzahl der Benutzer beträgt 13.

10.1.3 Entwurf der *PaperBase* mit *Living Documents*

Die erweiterte Realisierung des Informationssystems *PaperBase* auf der Basis von *Living Documents* wird im folgenden mit *PaperBaseLD* [SK02] bezeichnet. Ein Systemüberblick von *PaperBaseLD* ist in Abbildung 10.3 dargestellt. Im Vordergrund von *PaperBaseLD* steht ein Entwurf, in dem die Aktivität von den bisherigen Kommunikationsendpunkten (Web-Client, Applikationsserver und Datenbank) zu den Dokumenten verlagert wird.
Prinzipiell wird durch die Aktivierung der Dokumente die Verwendung und der Einsatz einer Datenbank und eines Applikationsservers obsolet. Die Applikationslogik und die damit verbundene Aktivität ist in den eigentlichen Informationsträgern, den Dokumenten, verankert. Durch den Wegfall zentraler Kontroll- und Koordinierungsinstanzen – wie beispielsweise dem Applikationsserver – übernehmen verteilte und lose gekoppelte Informationsinfrastrukturen (wie z. B. das Notifizierungssystem *Siena* [CRW01, CRW00] in Abbildung 10.3) eine Vermittlungsfunktion für den verteilten Verbund von *Living Documents*. Eine

[11] Die Attribute von *PaperBase* sind in einem relationalen Datenmodell abgebildet, welches in normalisierter Form in einer MySQL-Datenbank umgesetzt ist.

[12] Für eine ausführliche Erörterung von *Respondeo* ist auf Kapitel 4 und auf Schimkat et al. [SMKK00] verwiesen.

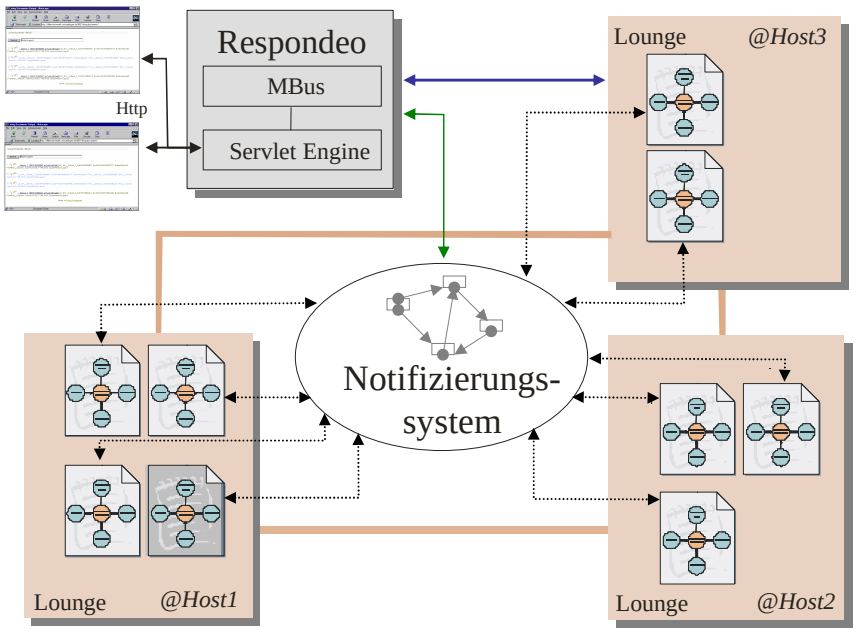


Abbildung 10.3: Systemüberblick der *PaperBaseLD*.

Vermittlungsfunktion bezeichnet dabei die Möglichkeit der Interaktion und Kommunikation zwischen autonomen Prozessen. Die Etablierung von *Living Documents* als autonome Informationsträger in *PaperBaseLD* erfordert eine Informationsinfrastrukur oder Middleware, die in der Lage ist, geeignete Kooperations- und Koordinierungsstrukturen zur Verfügung zu stellen.

Das erweiterte Informationssystem *PaperBaseLD* basiert auf den in Kapitel 9 vorgestellten *Living Documents*. Die Implementierung der *Living Documents* für die Realisierung von *PaperBaseLD* stützt sich dabei auf dem Agentenframework *Okeanos* [SBS+00] und der XML-basierten Auszeichnungssprache *SpectoML* [SHKK00, SFK01a], die in den Kapiteln 5 bzw. 6 detailliert erörtert werden.

Im folgenden Abschnitt 10.1.3.1 werden die erforderlichen Schritte für die Transformation der (passiven) Dokumente in *PaperBase* zu *Living Documents* erörtert. Danach werden in Abschnitt 10.1.3.2 die grundlegenden Eigenschaften von *PaperBaseLD* beschrieben. Anschließend werden in den Abschnitten 10.1.3.3 und 10.1.3.4 verschiedene Möglichkeiten der Informationssuche in *PaperBaseLD* aufgezeigt, die durch den Einsatz von *Living Documents* ermöglicht werden.

10.1.3.1 Transformation der Dokumente zu *Living Documents*

Die Transformation der passiven Dokumente in *PaperBase* zu *Living Documents* ist ein mehrstufiger Prozeß, dessen einzelne Schritte in Abbildung 10.4 veranschaulicht sind. Generell muß für jeden Teil eines *Living Documents* (vgl. Ab-

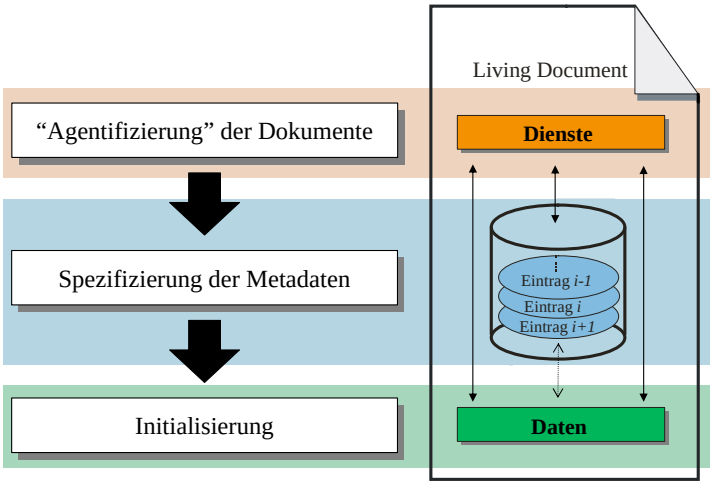


Abbildung 10.4: Transformation von Dokumenten zu *Living Documents*.

schnitt 9.2)– die Code-Komponente, die semi-strukturierte Datenbasis und die Daten der eigentlichen Dokumenteninhalte – eine geeigente Repräsentierung bestimmt werden.

1. *Agentifizierung*. Die Agentifizierung bezeichnet den Prozeß, in dem ein Dokument zu einem Agenten transformiert wird. Im Falle der *PaperBaseLD* wurden dafür verschiedene Agentenklassen entworfen, die unterschiedliche Typen von *Living Documents* repräsentieren.

   Generell wird in dieser Phase der Transformation die *Code-Komponente* (vgl. 9.2.1) eines *Living Documents* bestimmt. Diese ist abhängig von der gewählten Implementierung, d. h. von der jeweiligen Programmiersprache.

2. *Spezifizierung der Metadaten*. In dieser Phase werden die Metadaten der *Living Documents* bestimmt. Dafür werden die relevanten Typen von Dokumentenzustandsinformationen festgelegt und gemäß der DTD (vgl. Listing 6.1 auf der Seite 132) spezifiziert. Nach der Initialisierungsphase werden dynamisch entsprechende Instanzen der jeweiligen Typen von Dokumentenzustandsinformationen erzeugt und in der *semi-strukturierten Datenbasis* des *Living Documents* abgelegt[13].

3. *Initialisierung*. Der *Blob-Teil* eines *Living Documents* wird mit dem entsprechenden Dokument bzw. dessen Dokumenteninhalten gefüllt. Ein weiterer

[13] Vgl. Kapitel 6 für eine detaillierte Beschreibung der Syntax und Semantik der *SpectoML*. Abschnitt 9.2.2 beschreibt die Zusammenhänge zwischen der semi-strukturierten Datenbasis eines *Living Documents* und der *SpectoML*, die beiden wesentlichen Bestandteile der in dieser Arbeit gewählte Form der Implementierung der Datenbasis.

Bestandteil der Initialisierungsphase sind realisierungsabhängige Registrierungen. Im Falle einer agentenbasierten Realisierung der *Living Documents* (vgl. Abschnitt 9.2) sind verschiedene Registrierungsprozesse bei den Ausführungsumgebungen (*Lounges*) notwendig.

10.1.3.2 Eigenschaften von *PaperBaseLD*

Im folgenden werden zwei wesentliche Eigenschaften von *PaperBaseLD* beschrieben, die für die Realisierung des Informationssystems von Bedeutung sind.
Im nächsten Abschnitt wird speziell die Transformation der Datenbankattribute zu Dokumentenzustandsinformationen erläutert, die ein wichtiger Bestandteil des in Abschnitt 10.1.3.1 Punkt 2 erwähnten Transformationsprozesses sind. Im darauf folgenden Abschnitt werden einige der architektonischen Eigenschaften von *PaperBaseLD* beschrieben. Im speziellen werden die lose Kopplung der *Living Documents* erörtert, welche die Voraussetzung für ein verteiltes Informationssystem bilden, das sich aus vielen verteilten *Living Documents* zusammensetzt.

Transformation der Datenbankattribute Das relationale Datenmodell, das *PaperBase* zugrunde liegt, wird auf eine Menge von Dokumentenzustandsinformationen der *Living Documents* abgebildet. Diese Abbildung ist ein Bestandteil der Spezifizierung der Metadaten (siehe Abschnitt 10.1.3.1). Diese Transformation der Datenbankattribute auf Dokumentenzustandsinformationen ist notwendig, da aufgrund der Abgeschlossenheit (vgl. Abschnitt 9.3.1) der *Living Documents* jedes Dokumentenattribut explizit in der semi-strukturierten Datenbasis abgelegt sein sollte[14]. Die auf den ersten Blick redundante Datenhaltung der Attribute in jedem Dokument ist notwendig, da es keine zentrale Komponente für die Dokumentenorganisation mehr gibt. Jedes *Living Document* ist in sich abgeschlossen. Der neu geschaffene Verbund von *Living Documents* zeichnet sich u. a. dadurch aus, daß die einzelnen Dokumente nun verteilt und autonom organisiert sind.
Jedes Attribut des relationalen Schemas von *PaperBase* wird auf ein Tripel der folgenden Form abgebildet:

$$Datenbankattribut = \langle schema, name, wert \rangle$$

- `schema` stellt eine eindeutige Beschreibung für das benutzte relationale Schema dar.
- `name` ist ein String, der sich aus dem Tabellen- und Spaltennamen des Attributs zusammensetzt.

[14] Eine Dokumentenzustandsinformation entspricht einem konkreten Eintrag in der semi-strukturierten Datenbasis eines *Living Documents* (vgl. Abschnitt 9.2.2). Die grundsätzlichen Eigenschaften von Zustandsinformationen, die auf der *SpectoML* basieren, werden in Kapitel 6 beschrieben.

- `wert` ist der Wert des Attributs in der Datenbank.

Das beschriebene Tripel wird als ein bestimmter Typ von Dokumentenzustandsinformation spezifiziert. Zur Laufzeit bzw. zur Initialisierungsphase des *Living Documents* werden dann verschiedene Instanzen dieses Typs als Dokumentenzustandsinformation in der semi-strukturierten Datenbasis der *Living Documents* abgelegt. Die Anzahl der generierten Instanzen ist direkt abhängig von den Datenbankattributen, die für das jeweilige Dokument relevant sind.

Lose Kopplung der *Living Documents* *PaperBaseLD* zeichnet sich dadurch aus, daß die *Living Documents* verteilt organisiert sind und es keine zentrale Komponente für die Verwaltung der Dokumente gibt. In Abbildung 10.3 sind 9 *Living Documents* auf 3 *Lounges* (`Host1`, `Host2` und `Host3`) verteilt[15].
Siena [CRW01, CRW00][16] wird als nachrichtenbasiertes Notifizierungssystem in *PaperBaseLD* eingesetzt. *Siena* basiert auf dem sogenannten „publish/subscribe" Kommunikationsparadigma. Jedes *Living Document* publiziert (*publish*) Informationen in *Siena*. Außerdem meldet ein *Living Document* gleichzeitig sein Interesse (*subscribe*) für den Empfang von spezifischen Notifizierungen an. Durch dieses Vorgehen sind die Sender und Empfänger von Nachrichten bzw. Notifizierungen nur lose miteinander gekoppelt. Es ist die Aufgabe von *Siena* auf der Basis der publizierten Nachrichteninhalte zu entscheiden, welche Empfänger, die zuvor ihr Interesse für spezielle Nachrichten signalisiert haben, als tatsächliche Adressaten der Nachricht in Frage kommen.
Die Grundlage für die lose Kopplung bilden die Nachrichten und Notifizierungen und nicht syntaktische Beschreibungen der Kommunikationsssschnittstellen der beteiligten Sender und Empfänger, wie es typisch für viele Client/Server-Informationssysteme ist, die konzeptionell auf dem entfernten Methodenaufruf (*Remote Procedure Call* [BN84]) aufsetzen[17].
In technischer Hinsicht ist *Siena* eine Infrastruktur, die ihrerseits dezentral – wie in Abbildung 10.3 angedeutet – realisiert ist. Generell übernimmt *Siena* in *PaperBaseLD* die Rolle zentraler Komponenten – wie z. B. des Applikationsservers *Respondeo* – in *PaperBase*[18].

[15] Eine *Lounge* ist eine Abstraktion im *Okeanos*-Framework für eine Ausführungsumgebung, welche die Grundlage für die Existenz verschiedener *Okeanos*-Agenten bzw. *Living Documents* darstellt.

[16] Für Details des Entwurfs und der Implementierung wird auch auf die Homepage von *Siena* unter http://www.cs.colorado.edu/~carzanig/siena/index.html verwiesen.

[17] Siehe auch Abschnitt 2.3.1 auf der Seite 28ff über Beispiele für Client/Server-Informationssysteme.

[18] *PaperBaseLD* ist nicht notwendigerweise an *Siena* als Notifizierungssystem gebunden. Es sind auch andere Infrastrukturen einsetzbar, die eine lose Kopplung der verteilten *Living Documents* in *PaperBaseLD* unterstützen. In einem frühen Entwicklungsstadium von *PaperBaseLD* wurde das nachrichtenbasierte Notifizierungssystem *Mitto*, das ebenfalls im Rahmen dieser Arbeit entstanden ist, eingesetzt.

10.1.3.3 Datenbankzentrierte Suche

Eine datenbankzentrierte Suche bezeichnet den Prozeß, in dem der Benutzer eine Informationsanfrage an das Informationssystem bzw. an die Datenbank initiiert. Die Spezifikation einer solchen Anfrage basiert letztlich auf den Attributen, die im relationalen Schema der Datenbank a priori definiert wurden. Ein Ziel beim Entwurf von *PaperBaseLD* war die Bereitstellung der gleichen Funktionalität hinsichtlich der Suchmöglichkeiten von Dokumenten wie beim mehrstufig entworfenen Client/Server-Informationssystem *PaperBase*. Dabei ist zu beachten, daß die einzelnen Informationsquellen (*Living Documents*) in einem lose und verteilten Verbund organisiert sind.
Im folgenden wird der grundsätzliche Ablauf einer datenbankzentrierten Suche in *PaperBaseLD* beschrieben:

- Der Benutzer initiiert eine Anfrage über den Web-Client, welcher die Anfrage an *Respondeo* weiterreicht[19]. Danach wird die Anfrage direkt in *Siena* publiziert, wie in Abbildung 10.3 veranschaulicht ist. Zudem wird *Siena* der Empfänger für die Ergebnisse der aktuellen Anfrage mitgeteilt.

- *Siena* entscheidet auf der Grundlage der existierenden Registrierungen, welche *Living Documents* über die Nachricht mit der aktuellen Benutzeranfrage informiert werden sollten.

- Jedes der notifizierten *Living Documents* generiert auf der Grundlage seiner aktuellen Metadaten (Dokumentenzustandsinformationen) in der semistrukturierten Datenbasis und des Dokumenteninhalts in seinem Datenteil ein entsprechendes Teilergebnis. Die Anfrage an die Datenbasis des *Living Documents* erfolgt dabei über die in Abschnitt 6.4 beschriebene Anfragekomponente bzw. -sprache der *SpectoML*. Das Teilergebnis stellt eine aktuelle und dynamisch generierte Sicht (vgl. Abschnitt 9.3.2) auf die in der semi-strukturierten Datenbasis enthaltenen Informationen dar. Abschließend publiziert jedes *Living Document* sein generiertes (Teil-)Ergebnis in *Siena*.

- Der Empfänger für die Ergebnisse sammelt die eintreffenden Notifizierungen bzw. Teilergebnisse der einzelnen *Living Documents* ein und leitet sie schließlich an den Benutzer weiter.

10.1.3.4 Agentenbasierte Suche

Neben der Möglichkeit einer datenbankzentrierten Suche, die im vorherigen Abschnitt beschrieben ist, unterstützt *PaperBaseLD* einen weiteren Ansatz der Informationssuche. Dieser Ansatz ist typisch für Informationssysteme, die auf

[19] *Respondeo* fungiert in *PaperBaseLD* lediglich als Mittler zwischen einer web-basierten Benutzerschnittstelle und der Anbindung an *Siena*. Die eigentliche Applikations- bzw. Dokumentenlogik liegt nun bei den *Living Documents*.

(mobilen) Agenten basieren und wird im folgenden mit *agentenbasierter Suche* [GKP+01] bezeichnet.
Bei der agentenbasierten Suche initiiert der Benutzer – analog zur datenbankzentrierten Suche in Abschnitt 10.1.3.3 – über den Web-Client eine Anfrage an das Informationssystem *PaperBaseLD*. Der Applikationsserver leitet die Benutzeranfrage direkt an eine der verfügbaren *Lounges* weiter. In Abbildung 10.3 erfolgt die Weiterleitung der Anfrage von *Respondeo* an die *Lounge* `Host3`, das durch den blauen Pfeil verdeutlicht wird. Die *Lounge* `Host3` übernimmt eine Sonderrolle hinsichtlich der Bearbeitung der aktuellen Anfrage. Es ist zu beachten, daß diese Sonderrolle von jeder *Lounge* übernommen werden kann.
Auf der *Lounge* `Host3` wird ein spezielles *Living Document* – ein `LDsearch` – erzeugt. `LDsearch` enthält alle Informationen, die für die Beschreibung der aktuellen Suchanfrage notwendig sind. Der Datenteil von `LDsearch` ist leer, da die Suchanfrage in der semi-strukturierten Datenbasis abgelegt ist. Es ist die Aufgabe von `LDsearch`, alle verfügbaren, verteilten *Lounges* und *Living Documents* in *PaperBaseLD* zu ermitteln und jedes *Living Document* bezüglich der aktuellen Benutzeranfrage zu kontaktieren. Dafür migriert das `LDsearch` auf die jeweiligen *Lounges* und befragt die dortigen *Living Documents*.
Für die Migrationen zwischen *Lounges* und den Kommunikationsprozessen der *Living Documents* werden die Infrastrukturen benutzt, die von *Okeanos* zur Verfügung gestellt werden wie z. B. KQML (vgl. Abschnitt 5.3.1) als Nachrichtenformat für die Kommunikation zwischen *Living Documents*.
Der Algorithmus für die Beantwortung der Anfrage und die Migration von `LDsearch` als mobiler Agent zwischen den *Lounges* ist im folgenden beschrieben[20]:

1. Ermittle eine *Lounge* `A` in *PaperBaseLD*, die noch nicht im Zusammenhang mit der aktuellen Suchanfrage kontaktiert wurde.

   (a) Falls alle *Lounges*, die zum aktuellen Zeitpunkt in *PaperBaseLD* zur Verfügung stehen, besucht wurden, schicke das Gesamtergebnis zum Benutzer zurück. `LDsearch` „mutiert“ nach dem Versenden zu einem normalen *Living Document*, welches die gesammelten Informationen für künftige Anfragen nutzbar macht und bereitstellt[21].

   (b) Falls eine *Lounge* `A` ermittelt werden kann, die noch nicht besucht wurde, fahre mit Schritt 2 fort.

2. Versuche auf die ermittelte *Lounge* `A` zu migrieren.

[20] Es ist zu beachten, daß verschiedene Algorithmen für die agentenbasierte Suche realisierbar sind. Im einzelnen orientiert sich ein bestimmtes Vorgehen für die Implementierung einer solchen Suche an den jeweiligen Anforderungen, die an das Informationssystem gestellt werden.

[21] In dieser Hinsicht fungiert ein `LDsearch` als eine Art *Cache* oder Zwischenspeicher, der bestimmte Informationen, die in der Umgebung des verteilten Informationssystems ermittelt wurden, bereithält.

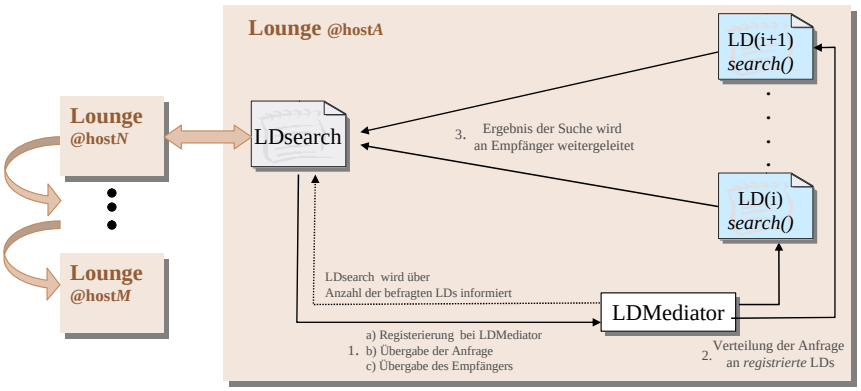


Abbildung 10.5: Agentenbasierte Suche in *PaperBaseLD*.

(a) Falls *Lounge* A verfügbar bzw. erreichbar ist, migriere zu *Lounge* A und fahre mit Schritt 3 fort.

(b) Falls *Lounge* A nicht erreichbar ist, d. h. es besteht zur Zeit keine Möglichkeit dorthin zu migrieren (z. B. ist die Netzwerkverbindung unterbrochen), fahre mit Schritt 1 fort und streiche *Lounge* A aus der Liste der zu besuchenden *Lounges*.

3. `LDsearch` registriert sich als *Living Document* beim `LDMediator` der *Lounge* A. In der Abbildung 10.5 ist der generelle, mehrstufige Ablauf der agentenbasierten Suche hinsichtlich einer *Lounge* veranschaulicht.

(a) `LDMediator` ist ein spezieller Mediator[22], der für die Verwaltung und Organisation der *Living Documents* auf einer *Lounge* verantwortlich ist. Die Einführung eines Mediatoren, in diesem Fall eines sogenannten `LDMediators`, unterstützt auf jeder *Lounge* ein flexibles Management der *Living Documents*, da jeder `LDMediator` spezifische Funktionalitäten zur Verfügung stellen kann. So ist beispielsweise jede Suchanfrage (`LDsearch`) von den jeweiligen *Living Documents* logisch entkoppelt. Es ist die Aufgabe des `LDMediators`, zwischen diesen zu vermitteln und etwaige erforderliche Datentransformationen vorzunehmen. Diese Funktion des `LDMediators` entspricht der Rolle eines Applikationsservers im Zusammenhang mit mehrstufigen Client/Server-Informationssystemen (vgl. Abschnitt 2.3.1).

(b) Die eigentliche Suchanfrage wird von `LDsearch` an `LDMediator` übergeben. Dabei wird der Empfänger für die potentiellen Ergebnis-

[22] Siehe die ausführliche Diskussion von Mediatoren im Zusammenhang mit der Realisierung von Informationssystemen im Abschnitt 2.2.

se der übergebenen Suchanfrage ebenfalls `LDMediator` mitgeteilt[23]. Die Ausführung der Suchanfragen erfolgt parallel, da *Okeanos* prinzipiell jedem *Living Document* einen eigenen Kontrollfluß (*thread of control*) zuweist[24].

(c) `LDMediator` leitet die Anfrage an die aktuell registrierten *Living Documents* (`LD(i)` und `LD(i+1)` in Abbildung 10.5) auf der lokalen *Lounge* weiter.

(d) Im letzten Schritt werden die Ergebnisse auf der *Lounge* `A` direkt vom jeweiligen *LD* an den Empfänger gesendet (vgl. Abbildung 10.5).

4. Streiche die aktuell besuchte *Lounge* `A` aus der Liste der zu besuchenden *Lounges* und fahre mit Schritt 1 fort.

Die agentenbasierte Suche in *PaperBaseLD* ist von Vorteil, wenn man in bezug auf die verteilte Umgebung – den Verbund der *Living Documents*, die über mehrere *Lounges* verteilt sind – nur sehr vage Annahmen über die Verfügbarkeit der Komponenten im Netzwerk treffen kann. Durch die Migrationsfähigkeit einer Suchanfrage via `LDsearch` und den damit verbundenen *lokalen* Interaktionen mit den jeweiligen *Living Documents* ist die Informationssuche unabhängig von temporären „Ausfällen“ einzelner entfernter *Lounges*. In dynamischen Umgebungen ist die Annahme einer permanenten Verfügbarkeit aller beteiligten Komponenten, wie sie typisch für viele Informationssysteme ist, nicht unbedingt haltbar.
Ein potentieller Nachteil, der für die Nutzung einer agentenbasierten Suche in Erwägung gezogen werden muß, ist die Performanz des gesamten Ablaufs im Vergleich zu einem zentralen Ansatz wie in *PaperBase*. Durch die Migrationen von `LDsearch` sind vielfältige und zusätzliche Interaktionen mit den jeweiligen Systemkomponenten – z. B. `LDMediator`en – nötig, welche die Antwortzeiten hinsichtlich der gestellten Suchanfrage verzögern können.
Generell muß im Einzelfall anhand der Anforderungen des Informationssystems abgewogen werden, welche Art von Informationssuche präferiert wird. In dieser Fallstudie geht es primär um die generelle Machbarkeit und Untersuchung verschiedener Methoden für die Informationssuche in verteilten Informationssystemen, die mit *Living Documents* realisiert wurden. Das in den beiden letzten Abschnitten beschriebene datenbanken- und agentenbasierte Suchen nach Informationen zeigt die prinzipiellen Möglichkeiten der *Living Documents* in einem verteilten Informationssystem.

[23] Die Rolle des Empfängers der Ergebnisse muß nicht notwendigerweise von `LDsearch` ausgeübt werden, sondern kann von beliebigen Komponenten übernommen werden.

[24] Falls die aktuelle Anzahl *Living Documents* auf einer *Lounge* eine bestimmte Grenze überschreitet, besteht die Gefahr, daß zu viele nebenläufige Kontrollflüsse gleichzeitig aktiv sind. Dies kann zu Engpässen von Ressourcen im Betriebssystem führen. Um dies zu verhindern, bietet *Okeanos* die Möglichkeit, daß bestimmte Kontrollflüsse von mehreren *Living Documents* gemeinsam benutzt werden können.

10.2 *Living Hypertext* – ein dynamisches, netzwerkbasiertes Informationssystem

Die Definition eines *Living Hypertextes* orientiert sich an der allgemeinen Definition eines Hypertextes von Smith und Weiss [SW88][25] und stellt zusätzliche Eigenschaften, die auf dem Einsatz von *Living Documents* beruhen, für netzwerkbasierte Informationssysteme in den Vordergrund der Betrachtung:

> Ein *Living Hypertext* [SKN02] ist ein *netzwerkbasiertes* Informationssystem, in welchem die Daten in einem Netzwerk von Knoten organisiert sind. Die einzelnen Knoten sind durch Links miteinander verbunden, die *dynamisch* generiert und *automatisiert* verwaltet werden. Die Knoten in einem *Living Hypertext* werden durch *Living Documents* repräsentiert.

Demnach verbindet das Konzept eines *Living Hypertextes* die Eigenschaften einer netzwerkartigen, verteilten Organisation von Informationsinhalten mit einem aktiven und proaktiven Dokumentenbegriff, der typisch für *Living Documents* ist[26]:

- *Netzwerkbasierte Organisation.* Eine netzwerkbasierte Organisation von Informationsinhalten, die durch *Living Documents* repräsentiert werden, unterstützt autonome und dezentrale Strukturen, die lose in einem verteilten Verbund – einem *Living Hypertext* – organisiert sind[27].
- *Dynamik.* Die Dynamik eines *Living Hypertextes* bezieht sich auf die aktiven und proaktiven Eigenschaften von *Living Documents.* So sind Anfragen im Zusammenhang mit einem *Living Hypertext* gemäß dem Client/Server-Prinzip möglich, in denen einerseits menschliche Benutzer und andererseits Softwareprozesse (digitale Benutzer) als Initiatoren von Anfragen in Erscheinung treten[28]. Die Dokumentenzustandsinformationen in der semistrukturierten Datenbasis eines *Living Documents* repräsentieren den *aktuellen* Zustand, in dem sich ein Dokument befindet.
- *Umwelt- und Kontextinformationen.* Da jeder Knoten im Netzwerk durch ein *Living Document* repräsentiert wird, besteht die Möglichkeit, verschiedene, zusätzliche Informationsinhalte (Kontextinformationen) in den *Living*

[25] Die netzwerkbasierten Informationssysteme bilden die konzeptionelle Grundlage für die Realisierung eines *Living Hypertextes* (siehe Abschnitt 2.3.2.1).

[26] Für eine ausführliche Erörterung verschiedener Dokumentenbegriffe – passive, aktive und proaktive Dokumente – wird auf den Abschnitt 3.2 verwiesen.

[27] Die netzwerkartige Organisation von Informationsinhalten ist typisch für das Web und wird ausführlich in Abschnitt 2.3.2 beschrieben. Zudem werden die *lokale Autonomie der Datenbasen* – der HTML-Seiten – und die *dezentrale Systemarchitektur* als zwei wesentliche Erfolgsfaktoren für das Web ausgemacht.

[28] Für eine Einführung in die Terminologie verschiedener Benutzertypen – von „menschlichen“ und „digitalen“ Benutzern – und deren unterschiedlichen Interaktionsformen wird auf Abschnitt 2.1.1 verwiesen.

Hypertext einzubinden und für die Suche nach Informationen zur Verfügung zu stellen [FSK02]. Kontextinformationen werden in der semi-strukturierten Datenbasis eines *Living Documents* als Dokumentenzustandsinformation abgelegt[29]. Diese Zusatzinformationen können den Suchprozeß bzw. die Qualität der Suchergebnisse verbessern, da sie als weitere Klassifizierungsmerkmale in den Gesamtprozeß der Suche eingehen.

Nach der generellen Einordnung in Abschnitt 10.2.1 eines *Living Hypertextes* in die bestehenden Forschungsansätze auf dem Gebiet der automatisierten Informationssuche im Web werden in Abschnitt 10.2.2 die Architektur und die Implementierung eines *Living Hypertextes* auf der Grundlage von *Living Documents* erläutert. Das im vorherigen Abschnitt 10.1 vorgestellte Informationssystem *PaperBaseLD* bildet die Basis für den in dieser Fallstudie implementierten *Living Hypertext.* In Abschnitt 10.2.3 werden die einzelnen Abläufe bei einer Informationssuche eines Benutzers beschrieben. Nach dieser interaktiven Benutzersicht (vgl. Abschnitt 3.1.1) auf einen *Living Hypertext* werden im darauffolgenden Abschnitt 10.2.4 die Aspekte einer automatisierten Benutzersicht (vgl. Abschnitt 3.1.2) auf einen *Living Hypertext* erörtert[30]. Abschließend wird im Abschnitt 10.2.5 ein Mechanismus für die automatische Erstellung von Themenkomplexen in einem *Living Hypertext* erläutert. Dafür wurde ein bestehendes Softwarewerkzeug, das ursprünglich für das Web entworfen wurde, hinsichtlich der von den *Living Documents* gestellten Anforderungen angepaßt und anschließend in den *Living Hypertext* integriert. Die automatische Erstellung von Themenkomplexen auf der Grundlage bestehender Web-Technologien unterstreicht die konzeptionelle Mächtigkeit eines *Living Hypertextes.* Dieser verbindet die Eigenschaften klassischer, netzwerkartiger Informationssysteme mit den Vorteilen von *Living Documents.*

10.2.1 Motivation

Netzwerkbasierte Informationssysteme – wie z. B. das Web – zeichnen sich durch ihre autonomen und verteilten Datenbasen aus[31]. Für die Durchführung einer Informationssuche im Web wird typischerweise ein (zentraler) Dokumentenindex, der häufig Kopien der real existierenden Dokumente bzw. Datenbasen enthält, aufgebaut. Dafür werden die Dokumente im Web heruntergeladen und in den Index überführt. Die von Benutzern initiierten Suchanfragen werden auf der Grundlage des erstellten Indexes und nicht auf der tatsächlich existierenden, verteilten Sammlung von Dokumenten beantwortet.
Es zeichnen sich zwei Trends im Zusammenhang mit der Suche nach Informationen und Dokumenten im Web ab:

[29] Siehe auch die Beschreibung verschiedener Sichten auf ein Dokument in Abschnitt 9.3.2. Die Grundlage für die dynamische Generierung dieser Sichten bilden die Dokumentenzustandsinformationen in der Datenbasis des *Living Documents.*

[30] Eine ausführliche Erläuterung der verschiedenen Aspekte einer interaktiven und automatisierten Benutzersicht erfolgt in Abschnitt 3.1.

[31] Vgl. Abschnitt 2.3.2.

- Für die Beantwortung von Suchanfragen eines Benutzers werden nicht nur die eigentlichen Dokumenteninhalte, sondern auch die darin enthaltenen Verweise (Links)[32] in den Suchprozeß eingebunden. Durch die Miteinbeziehung des Graphen, der durch die Links zwischen HTML-Dokumenten gebildet wird, verbessert sich die Qualität der Informationssuche im Web, wie Brin und Page [BP98b] für die Web-Suchmaschine *Google* demonstrieren. Generell stellen Links eine Art von *Zusatzinformationen* dar, die bei der Suche nach Informationen bessere bzw. genauere Ergebnisse liefern können.
- Eng verbunden mit der Suche nach bestimmten Dokumenten im Web ist die *Klassifizierung* von Dokumenten in entsprechende Themenkomplexe. Anstelle der interaktiven Initiierung von Anfragen durch den Benutzer tritt dabei ein *navigierender Zugriff* auf die a priori erstellten Themenkomplexe im Web. Im Mittelpunkt der thematischen Klassifizierung steht weniger die Ermittlung desjenigen Ergebnisses, das zu einer gegebenen Benutzeranfrage die „besten“ Dokumente enthält (klassische Web-Suchanfrage), als vielmehr die Bestimmung des eigentlichen Themas, das der gestellten Anfrage zugrunde liegt. Nachdem der entsprechende Themenkomplex auf der Basis der aktuellen Suchanfrage ermittelt worden ist, werden dem Benutzer die inhaltlich relevanten Dokumente in einem sogenannten *Verzeichnis* präsentiert [CDG+98]. Nun hat der Benutzer die Möglichkeit durch den ad hoc bestimmten Themenkomplex zu navigieren, indem er den Links im Verzeichnis folgt. Die Grundlage für die Bestimmung von Themenkomplexen im Web bildet wiederum die Analyse des Web-Graphen [Law00, KL01, Kle99], der durch die Links zwischen Dokumenten im Web aufgebaut wird.

Ein *Living Hypertext*, der analog zum Web auf autonomen und netzwerkartig organisierten Datenbasen aufbaut, verfügt über ähnliche Möglichkeiten wie die zuvor beschriebene interaktive Suche nach Informationen oder wie die Generierung von Themenkomplexen im Web. In Abschnitt 10.2.3 werden die Abläufe bei einer interaktiven Suche nach Informationen durch einen Benutzer genau geschildert. Die automatisierte Generierung von Themenkomplexen in einem *Living Hypertext* nach dem Vorbild des Web [Kle99] wird in Abschnitt 10.2.5 erörtert.
Außerdem sind auf der Basis aktiver oder proaktiver Dokumente, die durch *Living Documents* repräsentiert werden, vielfältige und erweiterte Anwendungsszenarien in einem netzwerkbasierten Informationssystem – wie einem *Living Hypertext* – realisierbar. Der Einsatz von aktiven und proaktiven Dokumenten zielt auf die potentielle Schwachstelle im Web, in dessen Mittelpunkt passive Dokumente stehen, ab und bietet die Möglichkeit, mehrere Prozesse bei der Organisation der Datenbasen und bei der Suche nach Informationen zu automatisieren.
In dieser Hinsicht vereint ein *Living Hypertext* die Eigenschaften eines netzwerkbasierten Informationssystems wie dem Web mit den Vorteilen mehrstufiger

[32] Im Zusammenhang mit dem Web werden Verweise zwischen einzelnen HTML-Dokumenten als *Links* bezeichnet. Ein Autor bringt mittels eines Links eine bestimmte inhaltliche Ähnlichkeit zwischen HTML-Dokumenten zum Ausdruck.

Client/Server-Informationssysteme. So entsteht also ein verteilter Verbund von autonomen und netzwerkartig organisierten Datenbasen, die sich durch automatisierte Interaktionen auszeichnen. In der folgenden Auflistung werden einige der erweiterten konzeptionellen und funktionalen Möglichkeiten zur Realisierung eines *Living Hypertextes* auf der Basis von *Living Documents* angeführt[33]:

- Das Wissen in den semi-strukturierten Datenbasen der *Living Documents* wird für die Verwaltung unterschiedlicher Kontextinformationen benutzt. Analog zu der zuvor beschriebenen Berücksichtigung der Linkstrukturen für die Informationssuche im Web dienen diese (zusätzlichen) Kontextinformationen dazu, den Suchprozeß innerhalb eines *Living Hypertext*s zu verbessern.

- Neben der Suche nach Informationen, die in den (eigentlichen) Dokumenten enthalten sind, ermöglicht ein *Living Hypertext* eine strukturierte Suche nach bestimmten Kontexten. Diese Art der Suche erfolgt dabei in interaktiver oder automatisierter Form, d. h. sowohl menschliche als auch digitale Benutzer (wie z. B. andere *Living Documents*) sind in der Lage, die semi-strukturierten Datenbasen hinsichtlich der Kontextinformationen abzufragen[34]. Die Ergebnisse sind abhängig vom Inhalt der Datenbasen und repräsentieren stets den *aktuellen* Zustand des Dokuments.

- Innerhalb eines *Living Hypertextes* sind verschiedene Arten von Suchprozessen möglich. So wird in Abhängigkeit der Betrachtungsweise auf ein *Living Document* die entsprechende Sicht aktiv. Bereits in den Abschnitten 10.1.3.3 und 10.1.3.4 wurde eine datenbanken- und agentenbasierte Suche in einem verteilten und lose gekoppelten Verbund von *Living Documents* erörtert. In diesem Kapitel wird eine weitere Art der Suche nach Informationen im Zusammenhang mit *Living Documents* besprochen. Die sogenannte *Hypertext-Sicht* ermöglicht einen navigierenden Zugriff auf den verteilten und netzwerkartigen Verbund von *Living Documents*[35].

10.2.2 Architektur und Implementierung

Die Architektur des in dieser Fallstudie zugrunde liegenden *Living Hypertextes* basiert auf dem Informationssystem *PaperBaseLD*, das bereits im Mittelpunkt der

[33] Es wird an dieser Stelle darauf hingewiesen, daß existierende Werkzeuge und Algorithmen für die Suche nach Informationen im Web auch in einem *Living Hypertext*, der auf der Basis von *Living Documents* realisiert ist, einsetzbar sind. So wird in Abschnitt 10.2.5 die Anbindung eines Werkzeugs zur automatisierten Erstellung von Themenkomplexen in einem *Living Hypertext* beschrieben. Dieses Werkzeug ist sowohl im Web als auch in einem *Living Hypertext* einsetzbar.

[34] Vgl. Abbildung 10.8 der Darstellung der web-basierten Benutzerschnittstelle für die Abfrage von Informationen in der semi-strukturierten Datenbasis eine *Living Documents*.

[35] In der Abbildung 10.6 wird der Zusammenhang zwischen einer datenbanken-, agenten- und hypertext-basierten Suche illustriert.

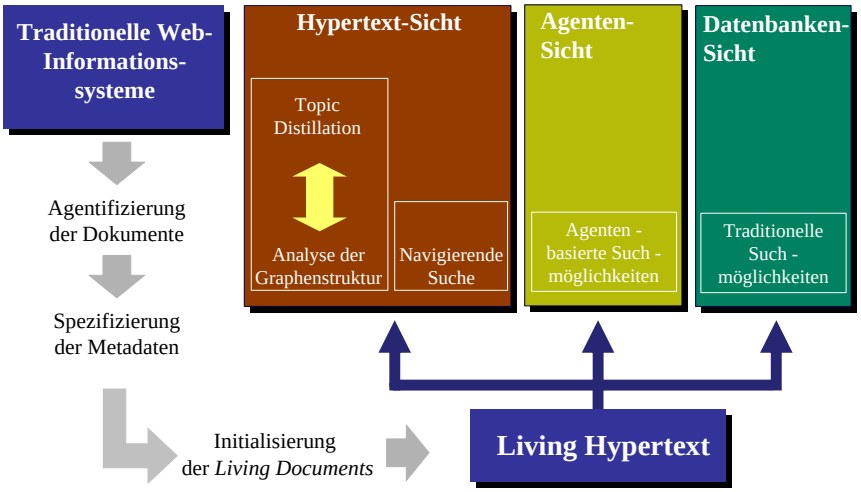


Abbildung 10.6: Transformation eines Informationssystems in ein *Living Hypertext*.

vorherigen Fallstudie in Abschnitt 10.1 stand. Demnach steht weiterhin die Anwendungsdomäne des Informationssystems *PaperBase* im Vordergrund. Die Implementierung der *Living Documents* folgt einer agentenbasierten Realisierung auf der Basis von *Okeanos*-Agenten (vgl. Kapitel 5). Die semi-strukturierten Datenbasen der *Living Documents* innerhalb des *Living Hypertextes* sind als *SpectoML*-Dokumente (vgl. Kapitel 6) realisiert. Anfragen an die Datenbasis werden gemäß der Anfragesprache von *SpectoML* (vgl. Abschnitt 6.4) gestellt.
Jedes *Living Document* im *Living Hypertext* ist über ein Notifizierungssystem (*Siena*) lose miteinander verbunden (vgl. Abschnitt 10.1.3.2). Über das Notifizierungssystem werden Systemzustände, Änderungen des Umfelds und Mitteilungen von *Living Documents* kommuniziert. Im Gegensatz zu den *Living Documents* im Informationssystem *PaperBaseLD* tritt jedes *Living Document* innerhalb eines *Living Hypertextes* als Empfänger *und* Sender von Nachrichten und Mitteilungen in Erscheinung. Es ist erforderlich, daß Veränderungen in den Zuständen oder Dokumenteninhalten im lose und verteilt organisierten Verbund der *Living Documents* – dem *Living Hypertext* – bekannt gemacht werden[36]. So kann jedes *Living Document* selbständig entscheiden, wie es auf solche Veränderungen reagiert.
Für die in Abbildung 10.6 dargestellte *Hypertext-Sicht* auf einen *Living Hypertext* ist der Verbindungsgrad der *Living Documents*, wie sie innerhalb von *PaperBaseLD* organisiert sind, nicht ausreichend. Der Verbindungsgrad ist ein Maß für die Anzahl der logischen Verbindungen, die zwischen den (verteilten) *Living Documents* bestehen. Besitzen beispielsweise zwei Dokumente ähnliche Dokumenteninhalte, sind sie logisch miteinander verbunden. Die aus der Transformation der Datenbankattribute der *PaperBase* entstehenden Dokumentenzustands-

[36] Vgl. Abschnitt 10.2.4 über die *automatisierte Linkvalidierung*.

informationen drücken bereits eine große Ähnlichkeit der jeweiligen Dokumente aus. Zwischen Dokumenten, die ursprünglich in einem gemeinsamen Container in *PaperBase* abgelegt sind, besteht offensichtlich eine inhaltliche Gemeinsamkeit. So hatte beispielsweise der *PaperBase*-Benutzer als Erfasser der Dokumente die Intention, diese im gleichen *PaperBase*-Container zu organisieren, weil die betreffenden Dokumenteninhalte die jeweiligen Themen des *PaperBase*-Containers adressieren.
Die Voraussetzung zur Realisierung einer Hypertext-Sicht ist die Möglichkeit eines navigierenden Zugriffs auf den *Living Hypertext*. Bei der Navigation durch einen Hypertext folgt der Benutzer den Verbindungen (Links), die zwischen den Dokumenten bestehen. Unter der ausschließlichen Berücksichtigung der zuvor beschriebenen Verbindungen, die sich durch die ursprüngliche Transformation der Datenbankattribute in *PaperBase* ergeben, ist ein navigationsbasierter Zugriff auf den *Living Hypertext* nur bedingt sinnvoll nutzbar. Ein Benutzer würde stets nur in denjenigen virtuellen Arbeitsräumen navigieren, die bereits vom Erfasser in *PaperBase* durch die Festlegung der Container vorbestimmt waren.
Deswegen wurde für den navigationsbasierten Zugriff auf den *Living Hypertext* ein Mechanismus bestimmt, der den Verbindungsgrad zwischen den Dokumenten erhöht. Dadurch erhält ein Benutzer erweiterte Wahlmöglichkeiten, durch den *Living Hypertext* zu navigieren.
So wird jedem *Living Document* ein übergeordneter *Auftrag* zugewiesen. Ein *Living Document* versucht seinem Auftrag fortwährend nachzukommen. Ein Auftrag besteht beispielsweise darin, Dokumente mit ähnlichen Dokumenteninhalten zu finden. Dafür wird ein simples Ähnlichkeitsmaß definiert, daß zwei Dokumente als ähnlich bezeichnet, falls die Dokumenteninhalte oder deren semi-strukturierten Datenbasen sich bezüglich bestimmter Sachthemen überschneiden[37].
Aus der Sicht eines *Living Document* bestehen grundsätzlich zwei Möglichkeiten, ähnliche Dokumente innerhalb des *Living Hypertext* ohne benutzerbedingte „Hilfen“ zu finden:

- *Reaktiver Ansatz*. Auf der Grundlage der propagierten Zustandsänderungen der *Living Documents* über das Notifizierungssystem besteht die Möglichkeit, bestimmte Ereignisse zu selektieren und bei Eintritt dieser Ereignisse entsprechend zu reagieren.

  Falls z. B. der Dokumenteninhalt eines bestimmten *Living Documents* sich dahingehend ändert, daß er nun auch Informationen über den Themenkomplex „mobile Agenten“ enthält, wird dies vom betreffenden *Living Document* publiziert. Andere *Living Documents*, die ähnliche Inhalte besitzen, sind nun in der Lage, eine logische Verbindung zum publizierenden Dokument aufzubauen. In der Datenbasis der anderen *Living Documents* wird

[37] Es wird darauf hingewiesen, daß einem *Living Document* auch beliebig andere Aufträge erteilt werden können. Zudem ist eine Bestimmung für die Ähnlichkeit zwischen Dokumenten nicht an das zuvor erwähnte Ähnlichkeitsmaß gebunden.

nun ein Vermerk bzw. eine Dokumentenzustandsinformation hinterlegt, die besagt, daß diese Dokumente ähnliche Inhalte besitzen.

- *Proaktiver Ansatz.* Auf der Basis des jeweiligen *Auftrags* ist ein *Living Document* in der Lage, selbst aktiv zu werden, indem es mit anderen *Living Documents* kommuniziert und interagiert[38]. Dafür werden entsprechende Anfragen zwischen den *Living Documents* initiiert. Solche Anfragen betreffen sowohl den Dokumenteninhalt als auch die Informationen in den jeweiligen semi-strukturierten Datenbasen der *Living Documents.* Eine Anfrage an eine Datenbasis erfolgt stets gemäß der Anfragesprache der *SpectoML.* Generell handelt es sich dabei um automatisierte Anfragen und Interaktionen, in denen ein (menschlicher) Benutzer nicht involviert ist.

  Falls ein *Living Document* durch diese proaktive Vorgehensweise ähnliche Dokumente im *Living Hypertext* findet, werden – analog zum reaktiven Ansatz – entsprechende logische Verbindungen zwischen den beteiligten *Living Documents* etabliert.

In Abbildung 10.6 ist die Transformation eines Web-Informationssystems – wie z. B. *PaperBase* – in einen *Living Hypertext* veranschaulicht. Die für die Transformation notwendigen Schritte sind bereits in Abschnitt 10.1.3.1 beschrieben. Abbildung 10.6 verdeutlicht die vielfältigen Möglichkeiten der Suche nach Informationen in einem *Living Hypertext.* Neben der bereits in den Abschnitten 10.1.3.3 und 10.1.3.4 beschriebenen *datenbanken-* und *agentenbasierten Suche* unterstützt ein *Living Hypertext* auch eine Vorgehensweise bei der Suche nach Informationen, die typisch für Hypertexte im allgemeinen ist. So wird im folgenden Abschnitt 10.2.3 eine interaktive Suche in einem *Living Hypertext* aus der Sicht eines Benutzers genauer beschrieben. Abschnitt 10.2.5 gibt einen Überblick über die automatisierte Erstellung von Themenkomplexen durch die Analyse der Graphenstruktur in einem *Living Hypertext.*

10.2.3 Typischer Anwendungsfall für interaktive Benutzersichten

In diesem Abschnitt wird ein typischer Ablauf bei der interaktiven Suche nach Informationen und Dokumenten in einem *Living Hypertext* beschrieben. Im Mittelpunkt der Betrachtung steht eine *interaktive Benutzersicht* auf einen *Living Hypertext,* die sich u. a. durch eine *unmittelbare Visualisierung* und *Ausführung* der initiierten Anfragen des Benutzers auszeichnet (vgl. Abschnitt 3.1.1).
Der Ablauf der interaktiven Suche in einem *Living Hypertext* orientiert sich sehr stark an dem für das Web typischen Anwendungsfall. Im Vordergrund steht ein navigierender Zugriff auf die Informationssinhalte. Allerdings bildet die Initiierung einer Anfrage den Ausgangspunkt für die Suche eines Benutzers nach Informationen im *Living Hypertext.* Anschließend werden die Ergebnisse der Suchanfrage als „Einstiegspunkte“ in den *Living Hypertext* genutzt. Danach greift der Benutzer

[38] Vgl. Abschnitt 3.2.4 über eine ausführliche Erörterung von *proaktiven Dokumenten*.

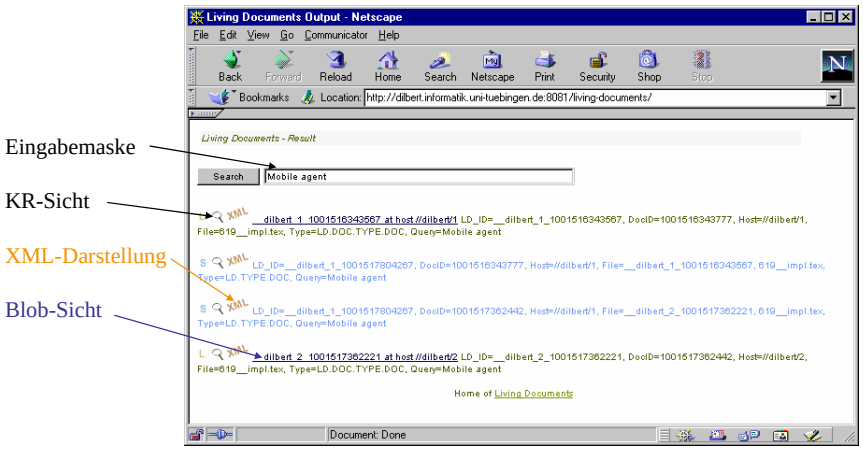


Abbildung 10.7: Die Initiierung einer Suchanfrage als Einstiegspunkt in den *Living Hypertext.*

primär navigierend auf den *Living Hypertext* zu, indem er den Links, die durch die *Living Documents* dynamisch aufgebaut und präsentiert werden, folgt[39].
Im folgenden sind die wesentlichen Schritte einer interaktiven Suche in einem *Living Hypertext* überblicksartig geschildert. Generell bietet der an dieser Stelle beschriebene *Living Hypertext* eine „Web-Schnittstelle“, die auf der Basis von *HTML* [RHJ99] als Dokumentensprache und *HTTP* [Wor00] als Kommunikationsprotokoll die bestehenden Web-Infrastrukturen (z. B. den Browser) nutzbar macht:

1. *Einstiegspunkt.* Der Benutzer bestimmt die Begriffe, nach denen er suchen möchte, und teilt diese dem *Living Hypertext* über ein Formular mit. Das Eingabeformular ist ein Bestandteil der Web-Schnittstelle des *Living Hypertextes.* In Abbildung 10.7 sucht ein Benutzer Dokumente bzw. *Living Documents,* die im Zusammenhang mit dem Begriff „Mobile Agent“ stehen.

2. *Präsentation der Ergebnisse.* Die im *Living Hypertext* gefunden *Living Documents,* welche für die spezifizierte Benutzeranfrage („Mobile Agent“) relevant sind, werden dem Benutzer in einer Auflistung präsentiert. Jedes *Living Document* bietet dem Benutzer verschiedene Sichten (vgl. Abschnitt 9.3.2) an, wie es in Abbildung 10.7 dargestellt ist:

   - *KR-Sicht.* Die *Knowledge Repository-Sicht* (KR-Sicht) liefert eine exklusive Sicht auf die Dokumentenzustandsinformationen, die in der semi-strukturierten Datenbasis – im Knowledge Repository – eines *Living Documents* enthalten sind.

[39] Der grundsätzliche Aufbau des beschriebenen *Living Hypertextes* orientiert sich an der Architektur des Informationssystems *PaperBaseLD* (vgl. Abbildung 10.3), das bereits auf der Basis von *Living Documents* realisiert ist. Abschnitt 10.2.2 erläutert weitere Details der Architektur und der Implementierung eines *Living Hypertextes.*

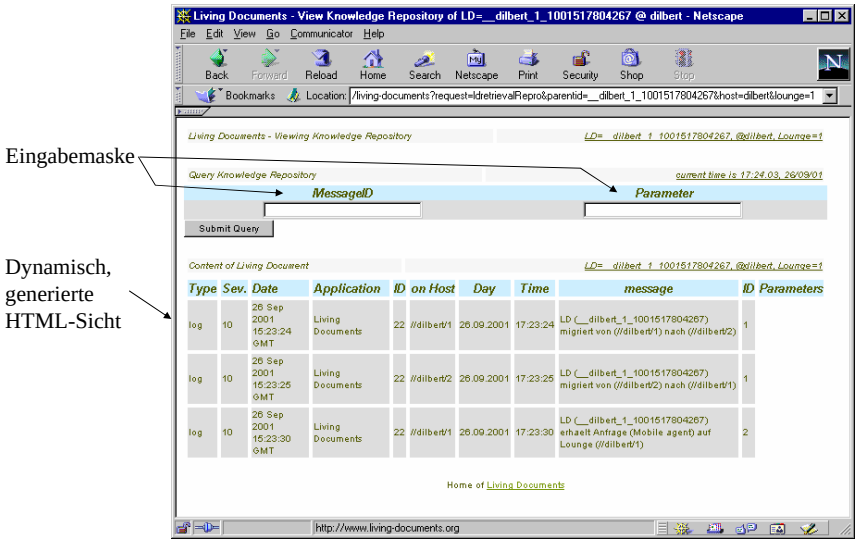


Abbildung 10.8: Interaktive Benutzersicht auf die semi-strukturierte Datenbasis eines *Living Documents.*

- *XML-Darstellung*. Diese Sicht ist eine Spezialisierung der zuvor angeführten KR-Sicht. Sie liefert den Inhalt der semi-strukturierten Datenbasis als XML- bzw. *SpectoML*-Dokument zurück[40].
- *Blob-Sicht*. In der Blob-Sicht werden die (eigentlichen) Dokumenteninhalte in der Daten- bzw. Blob-Komponente eines *Living Documents* visualisiert[41].

3. *Navigation*. Nach der Präsentation der Ergebnisse (vgl. Schritt 2) entscheidet der Benutzer sich für ein bestimmtes *Living Document.* Dabei hat er zusätzlich die Wahl zwischen den verschiedenen, dynamisch generierten Sichten des *Living Documents.*

4. Wie in Abbildung 10.8 dargestellt, hat sich der Benutzer für die *KR-Sicht* eines *Living Documents* entschieden. Im folgenden sind einige Aspekte der web-basierten Benutzerschnittstelle hinsichtlich der *KR-Sicht* geschildert:

   - Die *KR-Sicht* bietet dem Benutzer die Möglichkeit, beliebige Dokumentenzustands- bzw. Kontextinformationen in bezug auf das betreffende *Living Document* abzufragen[42]. So kann es für den Benut-

[40] Eine Darstellung der semi-strukturierten Datenbasis als XML-Dokument ist in Abbildung B.3 im Anhang dargestellt.

[41] Die Blob-Komponente eines *Living Documents* kann unterschiedliche Typen von Dokumenteninhalten besitzen, wie es in Abschnitt 9.2.3 erläutert wird.

[42] Vgl. Abschnitt 10.2.1.

zer von Interesse sein, zu ermitteln, welche anderen Dokumente aus der Sicht des betreffenden *Living Documents* ähnliche Inhalte besitzen. Dabei dient das *Living Document* bzw. seine semi-strukturierte Datenbasis als Informationsquelle für den Benutzer.

- Die web-basierte Benutzerschnittstelle im Zusammenhang mit der *KR-Sicht* stellt ein Formular zur Verfügung, in dem die gewünschten Suchbegriffe eingegeben werden. Auf der Basis der spezifizierten Begriffe wird eine Anfrage gemäß der Anfragesprache der *SpectoML* (vgl. Abschnitt 6.4) an die semi-strukturierte Datenbasis des betreffenden *Living Documents* gestellt[43]. Danach werden die gefundenen Dokumentenzustandsinformationen entsprechend durch die Benutzerschnittstelle visualisiert. In Abbildung 10.8 ist eine tabellenartige Visualisierung der Ergebnisse dargestellt.

Im Vordergrund der Beschreibung der interaktiven Benutzersicht auf einen *Living Hypertext* stehen menschliche Benutzer und deren Aktionen. Allerdings werden auch automatisierte Interaktionen mit digitalen Benutzern unterstützt[44]. Im folgenden Abschnitt 10.2.4 stehen automatisierte Interaktionen und Arbeitsformen im Mittelpunkt.

10.2.4 Linkkonzepte und automatisierte Linkvalidierung

Links sind im Zusammenhang mit der Suche nach Informationen in einem netzwerkbasierten Informationssystem von großer Bedeutung (vgl. Abschnitt 10.2.1). Grundsätzlich stellt ein Link eine Informationsquelle dar, die zusätzliche Aussagen über Informationsinhalte im Netzwerk trifft[45]. In netzwerkbasierten Informationssystemen – wie z. B. einem *Living Hypertext* oder dem Web – werden die Informationen, die durch Links zur Verfügung gestellt werden, für die Steigerung der Qualität des Suchprozesses in dem Informationssystem genutzt [BP98b, Kle99].
Die Linkvalidierung beschreibt den konsistenten Zustand eines Links zwischen Dokumenten, d. h. die durch den Link ausgedrückte Verbindung zwischen Dokumenten existiert und ist zum aktuellen Zeitpunkt gültig. Im Zusammenhang mit der Linkvalidierung spielen sowohl *inhaltliche* – die Art der Verbindung zwischen Dokumenten – und *dynamische* Aspekte – die Gültigkeit der Verbindung zu einem gegebenen Zeitpunkt – eine wichtige Rolle.
Das Problem der Linkvalidierung in einem netzwerkbasierten Informationssystem liegt in der Autonomie und Verteiltheit der beteiligten Datenbasen. So ziehen bei-

[43] Es wird darauf hingewiesen, daß die *Living Documents* in einem verteilten Verbund lose organisiert sind. Durch die Interaktion mittels eines Browsers bleiben die Details der realen Anordnung der *Living Documents* dem Benutzer verborgen.

[44] In Abschnitt 2.1.2.3 werden detailliert *interaktive* und *automatisierte Arbeitsformen* erörtert.

[45] Auch im Zusammenhang mit mehrstufigen Client/Server-Informationssystemen, die auf *hybriden Dokumenten* (vgl. Abbildung 3.3 in Abschnitt 3.2.1.2) basieren spielen Links bzw. Verweise eine wichtige Rolle.

spielsweise inhaltliche Änderungen eines Dokuments sehr häufig auch Anpassungen der Linkstrukturen auf das betreffende Dokument nach sich. Aufgrund der Autonomie, Verteiltheit und der häufig sehr großen Anzahl der Dokumente in einem netzwerkbasierten Informationssystem wie z. B. dem Web sind Validierungsprozesse erforderlich, die durch automatisierte Abläufe gekennzeichnet sind.
Im Gegensatz zu den passiven Dokumenten, die im Web zum Einsatz kommen, besteht ein *Living Hypertext* aus *Living Documents* , die sich durch aktive oder proaktive Dokumente (vgl. Abschnitt 3.2) auszeichnen. Aktive und proaktive Dokumente bilden die Basis für automatisierte Interaktionen und Abläufe in einem netzwerkbasierten Informationssystem wie einem *Living Hypertext*, ohne daß dabei die Autonomie und die Verteiltheit der beteiligten Dokumente vernachlässigt wird.
Im folgenden werden anhand mehrerer Eigenschaften von Links – *Dynamik*, *Aggregation* und *Multi-Dimensionalität* – die Umsetzung und die Realisierung des Linkkonzepts in einem *Living Hypertext* beschrieben:

Dynamik Die Gültigkeit eines Links im Web ist oft nur von temporärer Dauer, da das Ziel bzw. die Zielkomponente eines Links in den meisten Fällen von der Existenz der Quelle und deren Link nicht unterrichtet ist. Falls sich der Dokumenteninhalt oder die Existenz der Zielkomponente eines Links ändern, wird die Quelle im Regelfall davon nicht unterrichtet. Dies führt schließlich zu ungültigen bzw. invaliden Links (*dangling links*) im Informationssystem. Die Autonomie der Dokumente im Web bedingt die Unabhängigkeit der Verwaltung und Organisation der Dokumente und kann so schnell zu ungültigen Links führen[46].

Living Hypertext: Der proaktive Dokumentenbegriff, der *Living Documents* und *Living Hypertext*en zugrunde liegt, ermöglicht die Gestaltung von komplexen und automatisierten Informationsflüssen. Ein Beispiel für einen derartigen automatisierten Informationsfluß ist die Kommunikation von proaktiven, verlinkten Dokumenten mit dem Ziel der Etablierung von konsistenten und validen Links untereinander. So sind diese Dokumente in der Lage, die inhaltlichen und dynamischen Aspekte der Linkvalidierung selbst und folglich in automatisierter Form zu übernehmen:

- Beispielsweise sendet das *Living Document*, das einen Link auf ein anderes *Living Document* enthält, eine Anfrage, ob dieser Link zum aktuellen Zeitpunkt gültig ist. Das betreffende, empfangende *Living Document* leitet diese Anfrage an seine semi-strukturierte Datenbasis weiter, auf deren Basis ein Ergebnis bezüglich der Konsistenzanfrage berechnet wird[47].

[46] Eine ausführliche Beschreibung von netzwerkbasierten Informationssystemen, deren Autonomie sowie deren dezentrale Orgnisationstrukturen erfolgt in Abschnitt 2.3.2 erörtert).

[47] Anfragen an eine semi-strukturierte Datenbasis eines *Living Documents* erfolgen stets gemäß der Anfragesprache von *SpectoML* (siehe Abschnitt 6.4).

- Eine weitere Möglichkeit, die Konsistenz von Links zwischen proaktiven Dokumenten wie den *Living Documents* zu garantieren, besteht darin, daß die Änderungen in den Dokumenteninhalten vom jeweiligen *Living Document* über Notifizierungssysteme (z. B. *Siena*) publiziert werden. Jedes *Living Document*, das einen Link bezüglich des sendenden *Living Documents* und dessen Dokumenteninhalten besitzt, ist nun in der Lage, selbständig zu entscheiden, ob der betreffende Link unverändert gültig ist.

Aggregation Links ergeben sich nicht nur aus einer einzigen, bestimmten Eigenschaft, die Dokumente miteinander verbindet, sondern können das Ergebnis einer komplexen Berechnung sein. Als Ergebnis dieser dynamischen Berechnung wird entschieden, ob ein Link – eine inhaltliche Verbindung – zwischen Dokumenten besteht. Eine solche aggregierte Sichtweise auf Links entscheidet zum aktuellen Zeitpunkt und unter Berücksichtigung mehrerer Umstände, ob ein Link zwischen Dokumenten besteht. Dies steht im Gegensatz zu einer rein statischen Festlegung von Links, wie sie häufig im Web Verwendung findet.

Living Hypertext: Die Voraussetzung für solche komplexen Berechnungen bildet die Existenz einer Datenbasis, die Anfragen in strukturierter Form ermöglicht. Anfragen an die semi-strukturierte Datenbasis eines *Living Documents* erfolgen stets gemäß der Anfragesprache der *SpectoML* (vgl. Abschnitt 6.4) und werden jeweils durch die Code-Komponente (vgl. Abschnitt 9.2.1) geleitet. Die Code-Komponente eines *Living Documents* fungiert in diesem Zusammenhang als Mediator (vgl. Abschnitt 2.2), der für die Ausführung der Anfragen und für etwaige Transformationen der Ergebnisse verantwortlich ist[48].

Multi-Dimensionalität Die Verbindung zwischen Dokumenten, die durch einen Link zum Ausdruck gebracht wird, kann sich durch unterschiedliche Eigenschaften auszeichnen. So kann sich ein Link zwischen Dokumenten aufgrund von *zeitlichen*, *inhaltlichen* oder *örtlichen* Zusammenhängen ergeben [HG98, AC93]. Jeder dieser Zusammenhänge spezifiziert eine unterschiedliche Dimension, die durch einen Link bestimmt wird.

Living Hypertext: Unterschiedliche Dimensionen hinsichtlich der Spezifizierung von Links werden im Zusammenhang mit *Living Documents* jeweils als eigene Typen von Dokumentenzustandsinformationen gemäß der

[48] Grundsätzlich wäre auch eine direkte Abfrage der jeweiligen Datenbasen der *Living Documents* möglich. Dies entspräche allerdings dem Vorgehen, wie es typischerweise in 2-stufigen Client/-Server-Informationssystemen praktiziert wird. Aus Gründen der Wartung, der Flexibilität und der Entkopplung der beteiligten *Living Documents* werden Anfragen und Berechnungen ausschließlich über den Mediator (den Code-Komponenten) abgewickelt. Eine ausführliche und allgemeine Erörterung dieser Aspekte erfolgt im Zusammenhang mit mehrstufigen Client/Server-Informationssystemen im Abschnitt 2.3.1.

SpectoML definiert. Ein Zustandstyp in der *SpectoML* (vgl. Abschnitt 6.2) legt den gültigen Wertebereich für den Link fest. Die Semantik des Links wird über den *Katalog-Ansatz* der *SpectoML* (vgl. Abschnitt 6.2.2) bestimmt und als XML-Dokument abgelegt. Durch diese Vorgehensweise können beliebige Arten oder Typen von Links durch ein *Living Document* repräsentiert werden.

10.2.5 Techniken zur netzwerkbasierten Informationssuche

10.2.5.1 Einleitung

Bei der Suche nach Informationen, die in Datenbanken verwaltet werden, wird oft davon ausgegangen, daß der Benutzer mit der Semantik des zugrunde liegenden Schemas vertraut ist. So sollte der Benutzer über die existierenden Attribute, die Beziehungen und die Abhängigkeiten der Attribute gewisse Kenntnisse besitzen. Bei der *datenbank-* und *agentenbasierten Suche* bilden diese Kenntnisse des Benutzers die Basis für einen erfolgreichen Suchprozeß (vgl. Abschnitte 10.1.3.3 und 10.1.3.4).
In diesem Abschnitt wird eine weitere Möglichkeit der Informationssuche – die sogenannte *Topic Distillation* – hinsichtlich der *Living Documents* bzw. eines *Living Hypertextes* beschrieben. *Topic Distillation* ist ein Prozeß mit dem Ziel, möglichst qualitativ hochwertige Dokumente hinsichtlich eines bestimmten Themas zu finden [BH98]. Dabei geht es nicht darum alle existierenden Dokumente, die für eine gestellte Anfrage relevant sein könnten, zu indexieren, nach ihnen zu suchen oder sie zu klassifizieren, wie es typisch für datenbankzentrierte Informationssysteme ist. Vielmehr geht es bei *Topic Distillation* darum, nur diejenigen Dokumente in den Suchprozeß einzubinden, die wertvolle Informationen für das gesuchte Thema liefern[49]. Das Thema einer gestellten Anfrage ergibt sich in netzwerkbasierten Informationssystemen – wie z. B. dem Web oder einem *Living Hypertext* – aus den folgenden drei Informationsquellen:

- Die *Schlüsselwörter*, die ein Benutzer im Web bei der Suche nach Informationen spezifiziert.
- Die *Inhalte der Dokumente*, die als Ergebnis zurückgeliefert werden.
- Die *Struktur der Links* zwischen den Dokumenten.

In dynamischen und veränderlichen netzwerkbasierten Informationssystemen wie dem Web oder einem *Living Hypertext* treten folgende grundsätzliche Probleme auf, welche den Prozeß der Informationssuche erschweren:

[49] Siehe auch Abschnitt 2.1.2.3 im Zusammenhang mit der Diskussion *verschiedener Arbeitsformen in Informationssystemen* bezüglich einer überblicksartigen Beschreibung von *Topic Distillation*.

1. Eine komplette Vorverarbeitung von Dokumenten mit ihren Inhalten und Linkstrukturen ist im allgemeinen nicht möglich, da die Anzahl der zu berücksichtigenden Dokumente zu groß ist[50]. Ein Aufbau eines solchen Indexes, welcher die gesamte Dokumentensammlung repräsentiert, ist nur bedingt sinnvoll einsetzbar. Die Autonomie und die dezentrale Organisation der Datenbasen – beispielsweise der HTML-Dokumente im Web – erschweren die Generierung eines globalen und stets aktuellen Indexes (vgl. Abschnitt 3.2.2 über *passive Dokumente*).

   In den Abschnitten 10.2.5.2 und 10.2.5.3 werden die automatisierte Generierung von Themenkomplexen in einem *Living Hypertext* erörtert. Die Basis für den Generierungsprozeß bildet nicht ein Dokumentenindex aus den oben genannten Gründen. Vielmehr wird ein dynamischer Ansatz für die Erstellung der Themen verfolgt, der zur Laufzeit die Linkstrukturen zwischen den Dokumenten und deren Dokumenteninhalte analysiert. Allerdings ist die Informationssuche und der Aufbau der Themenkomplexe weniger performant als ein datenbanktypisches Vorgehen, das sich durch den Aufbau eines zentralen Dokumentenindexes auszeichnet. Die Erfassungs-, Analyse- und die Evaluations-Komponenten (vgl. Abschnitt 10.2.5.2 bzw. Abbildung 10.10), die einen wesentlichen Bestandteil des dynamischen und automatisierten Ansatzes darstellen, erfordern stets zusätzliche Berechnungen im Vergleich zum statischen Aufbau eines zentralen Dokumentenindexes.

2. Die Struktur eines netzwerkbasierten Informationssystems – wie z. B. eines Hypertextes – unterliegt fortwährenden Änderungen. Die Links und die damit ausgedrückten inhaltlichen Verbindungen zwischen einzelnen Dokumenten ändern sich häufig (vgl. Abschnitt 10.2.4 über die *Dynamik von Linkstrukturen*).

   Die Transformation von passiven Dokumenten – wie sie typischerweise im Web zum Einsatz kommen – zu aktiven und proaktiven *Living Documents* bildet die Grundlage für ein konsistentes Management der Links in dynamischen und netzwerkbasierten Informationssystemen (vgl. Abschnitt 10.2.4 über die *Dynamik, Aggregation* und *Multi-Dimensionalität* von Links).

Die Basis für die Bestimmung von Themenkomplexen bilden die Schlüsselwörter einer Anfrage, die Dokumenteninhalte und die Linkstrukturen zwischen den Dokumenten, wie es bereits zuvor erwähnt wurde. Insbesondere die Analyse der Linkstrukturen im Web ist ein Forschungsgebiet in der Informatik, das in den letzten Jahren zunehmend an Bedeutung gewinnt [Law00, KL01, CDG+98, BP98b]. Für die in den folgenden Abschnitten erläuterte automatisierte Generierung von Themenkomplexen bildet der Algorithmus gemäß Kleinberg [Kle99] die Basis für die Analyse der Linkstruktur in einem *Living Hypertext* . Kleinberg entwickelte diesen

[50] So enthält beispielsweise die Suchmaschine *Google* nach eigenen Angaben 3,083,324,652 Dokumente im November 2002.

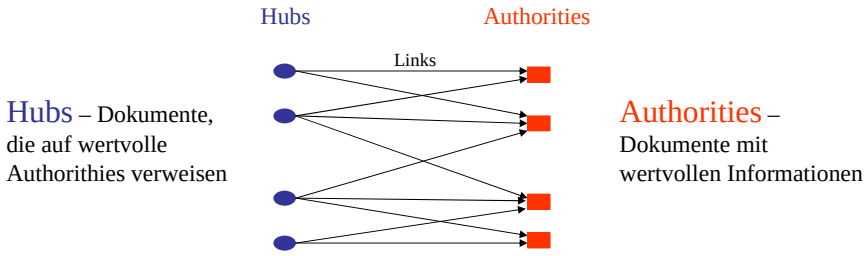


Abbildung 10.9: Analyse der Linkstrukturem in einem Hypertext gemäß Kleinberg [Kle99].

Algorithmus ursprünglich für die Analyse der Linkstrukturen im Web. Die konzeptionellen Gemeinsamkeiten zwischen dem Web und einem *Living Hypertext* – *Dokumente* und *Links* sind zentrale Bestandteile des jeweiligen netzwerkbasierten Informationssystems – ermöglichen die Übertragung und die Anwendung des Algorithmus auch im Umfeld eines *Living Hypertextes* (siehe Abschnitt 10.2.5.2). Kleinberg führt im wesentlichen zwei Abstraktionen, die für die Klassifizierung von Dokumenten wichtig sind, ein (siehe Abbildung 10.9):

- *Hubs* sind Dokumente bzw. HTML-Seiten, die eine Reihe von HTTP-Links zu *Authorities* beinhalten.
- *Authorities* sind Dokumente in einem netzwerkbasierten Informationssystem, die wertvolle Informationen zu einem bestimmten Thema besitzen.

Das Ziel des Kleinberg-Algorithmus ist nun, möglichst wertvolle *Hubs*, die auf besonders informative *Authorities* verweisen, zu finden. Für die Details des Algorithmus wird auf Kleinberg [Kle99] verwiesen, da diese nicht im primären Fokus dieser Arbeit liegen. Einige algorithmische Erweiterungen wurden im Rahmen des *Clever-Projekts* von IBM[51] durchgeführt. Weitere Erweiterungsmöglichkeiten finden sich bei Bharat und Henzinger [BH98]. Diejenigen Anpassungen und Erweiterungen, welche im Rahmen der Einbindung und Integration in einen *Living Hypertext* vollzogen wurden, sind in [Nes01, SKN02] beschrieben.

10.2.5.2 Automatisierte Generierung von Themenkomplexen

Smider [Nes01, SKN02] ist eine Softwarekomponente, die für die automatisierte Generierung von Themenkomplexen im Web entworfen wurde. Ein Themenkomplex bezeichnet dabei eine Menge von HTML-Dokumenten, die inhaltlich verwandt zueinander sind und hinsichtlich bestimmter Schlüsselwörter das gleiche,

[51] Siehe auch Homepage des *Clever-Projekts* unter http://www.almaden.ibm.com/cs/k53/clever.html.

den Schlüsselwörtern zugrunde liegende Thema adressieren[52].
Die automatisierte Generierung beschreibt den dynamischen Prozeß, welcher der Erstellung der Themenkomplexe zugrunde liegt. Auf der Basis von Schlüsselwörtern „simuliert" *Smider* das interaktive Verhalten eines Benutzers im Web, der auf der Suche nach Informationen und interessanten Dokumenten ist. *Smider* tritt an die Stelle des Benutzers, indem er in automatisierter Form nach den gewünschten Dokumenten im Web sucht.
Bevor die einzelnen Bestandteile von *Smider* erläutert werden, wird im folgenden der typische Ablauf einer Informationssuche im Web aus der Sicht eines Benutzers beschrieben. Diese Vorgänge dienen als Grundlage zur automatisierten Informationssuche von *Smider*:

1. Der Benutzer bestimmt die seiner Meinung nach relevanten Schlüsselwörter bzw. Suchbegriffe und kontaktiert eine Suchmaschine seiner Wahl (z. B. *Google* [Goo02] oder *AltaVista* [Alt02]).

2. Auf der Basis der erhaltenen Ergebnisse entscheidet der Benutzer, welche Dokumente für ihn relevant sein könnten und navigiert zu dem entsprechenden Dokument, indem er dem Link, der von der Suchmaschine ermittelt wurde, folgt. Sofern das kontaktierte Dokument nicht seinen Anforderungen entspricht, hat er mehrere Wahlmöglichkeiten:

   (a) *Standardfall*. Der Benutzer begibt sich auf das ursprüngliche Ergebnisdokument der Suchmaschine und wählt ein anderes Dokument aus der Liste aus.

   (b) *Verfeinerung der Suche*. Der Benutzer begibt sich zu der gleichen Suchmaschine und verfeinert seine Suche, indem er zusätzliche Schlüsselwörter bereitstellt.

   (c) *Einbeziehung weiterer Suchmaschinen*. Der Benutzer kontaktiert eine andere Suchmaschine mit den gleichen oder bereits angepaßten Schlüsselwörtern.

   (d) *Navigation und Themenverschiebung*. Der Benutzer folgt den Links, die auf einem Ergebnisdokument enthalten sind und gelangt so zu weiteren Dokumenten. Dabei kann es zu einer Themenverschiebung kommen, bei welcher der Benutzer an Dokumenten und deren Inhalten interessiert ist, die in der ursprünglichen Suche nicht notwendigerweise enthalten waren.

   Im Regelfall sind die unter 2a-2d beschriebenen Wahlmöglichkeiten des Benutzers jeweils ein Bestandteil eines sich wiederholenden Vorgangs.

[52] Sogenannte *Web-Verzeichnisse* enthalten Informationen und Dokumente, die hinsichtlich eines bestimmten Themas zusammengestellt werden. Während Web-Verzeichnisse häufig statisch festgelegt und manuell gepflegt werden, generiert *Smider* – basierend auf einer Menge von Schlüsselwörtern – den dazugehörenden Themenkomplex dynamisch.

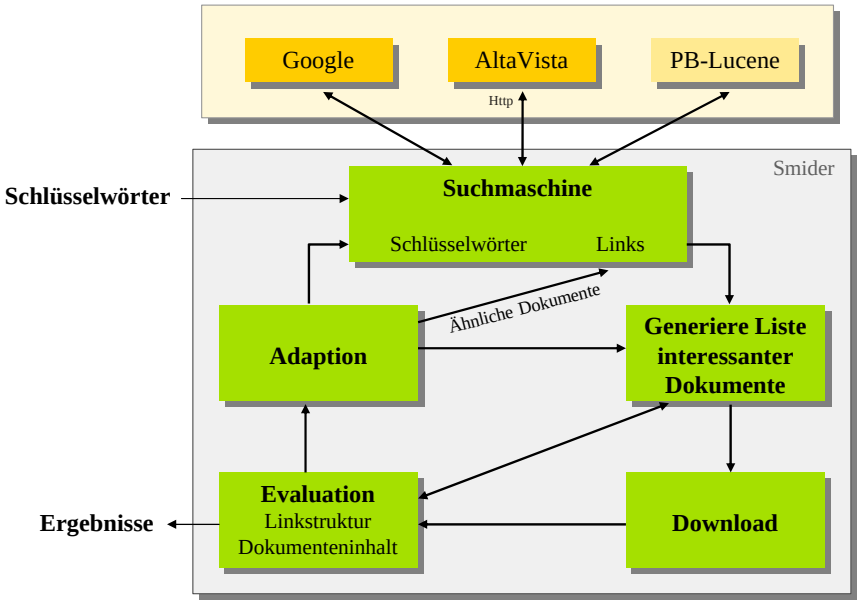


Abbildung 10.10: Systemüberblick von *Smider*.

In Abbildung 10.10 sind die einzelnen Komponenten von *Smider* und ihre Beziehungen untereinander illustriert. Grundsätzlich ermittelt *Smider* auf der Grundlage einer Menge von Schlüsselwörtern (*keywords*) die entsprechenden Ergebnisdokumente, die im Web verfügbar sind[53]. Die Ermittlung der Ergebnisse ist ein kontinuierlicher Prozeß. So werden die Ergebnisse bis zur Terminierung des Algorithmus (vgl. Abschnitt 10.2.5.1) ständig an die aktuellen Berechnungen angepaßt.
Im folgenden werden die einzelnen Komponenten *Smiders* gemäß Abbildung 10.10 näher erläutert:

Suchmaschine Diese Komponente *Smiders* nimmt die vom Benutzer bereitgestellten Schlüsselwörter und kontaktiert eine konfigurierbare Menge von Web-Suchmaschinen (z. B. *Google*, *AltaVista*). In *Smider* werden die Ergebnisse der jeweiligen Web-Suchmaschine gewichtet. So werden beispielsweise Suchergebnisse von *Google* höher gewichtet als die von anderen Suchmaschinen[54]. Die Ergebnisse der Suchmaschinen dienen als sogenannte *Startpunkte* für *Smider*.

Liste interessanter Dokumente In dieser Komponente *Smiders* wird auf der Ba-

[53] Im Abschnitt 10.2.5.3 wird die Integration von *Smider* in den *Living Hypertext* erläutert, so daß die Ergebnisdokumente sowohl aus dem Web als auch aus dem *Living Hypertext* stammen können.

[54] `PB-Lucene` in Abbildung 10.10 ist eine Suchmaschine, die im Rahmen dieser Fallstudie auf der Basis von *Lucene* [Jak02] entwickelt wurde. `PB-Lucene` enthält ausschließlich Informationen über die *Living Documents*, die im *Living Hypertext* organisiert sind. In Abschnitt 10.2.5.3 wird `PB-Lucene` genauer beschrieben.

sis der zuvor erhaltenen Ergebnisse der Suchmaschinen eine Liste von interessanten Dokumenten bestimmt. Ein Dokument wird als interessant bezeichnet, wenn aufgrund des Dokumenteninhalts oder durch die Analyse der Linkstruktur festgestellt wird, daß das betreffende Dokument hinsichtlich der vom Benutzer spezifizierten Schlüsselwörter relevant ist.

Download Die Download-Komponente dient zum Herunterladen von Dokumenten, damit deren Inhalte und Linkstrukturen analysiert werden können.

Evaluation In dieser Komponente werden die heruntergeladenen Dokumente hinsichtlich ihres Dokumenteninhalts und ihrer Linkstrukturen analysiert. Die Ergebnisse dieser Analyse gehen sowohl in die eigentlichen Ergebnisse (siehe Abbildung 10.10) als auch in die Komponente für die Generierung der interessanten Dokumente ein.

Die Evaluations-Komponente bildet den Kern für die Berechnung und Analyse der durch die Links in einem Hypertext aufgebauten Graphenstruktur. Der Algorithmus für die Evaluation basiert auf dem von Kleinberg [Kle99] vorgestellten Verfahren (vgl. Abbildung 10.9).

Adaption Ein Ziel bei der Adaption ist es, den Suchprozeß zielgerichtet und effektiv zu gestalten. So sollten nur die tatsächlich als relevant betrachteten Dokumente heruntergeladen und in die Berechnung einbezogen werden. Außerdem ermöglicht die Adaptions-Komponente eine erweiterte Betrachtung der relevanten Dokumente (vgl. die *Themenverschiebung* unter Punkt 2d).

Hinsichtlich der Anpassung des Suchprozesses bestehen grundsätzlich folgende Möglichkeiten:

- Verschiedene Suchmaschinen bieten eine Funktionalität, die zu einem gegebenen Dokument im Web eine Liste von *ähnlichen* Dokumenten zurückliefert[55]. Diese Funktion nutzt *Smider*, um weitere relevante Dokumente zu erfassen und der Evaluations-Komponente zur Verfügung zu stellen.
- Während der Evaluation und Adaption sind weitere Schlüsselwörter, die für die aktuelle Suche relevant sein könnten, ermittelt worden[56]. In einem separaten Schritt werden die Suchmaschinen auf der Basis dieser „neuen“ Schlüsselwörter kontaktiert. Die Ergebnisse der Anfrage werden wiederum in den Evaluationsprozeß *Smider*s einbezogen.

[55] Der Algorithmus und das Maß für die Bestimmung der Ähnlichkeit ist abhängig von der Suchmaschine und variiert je nach Suchmaschine. Für die Suchmaschine *Google* bestimmt beispielsweise der Eingabestring `related:<url>` alle ähnlichen Dokumente hinsichtlich einer gegebenen URL.

[56] Die tatsächliche Relevanz der zusätzlich ermittelten Schlüsselwörter wird erst nach nochmaligen Durchlaufen der Evaluations-Komponente festgestellt. In diesem Schritt geht es zunächst primär darum, das den Schlüsselwörtern zugrundliegende Thema zu ermitteln. Dafür werden weitere Schlüsselwörter gesucht, die als *Ergänzung* zu den vom Benutzer angegebenen betrachtet werden können.

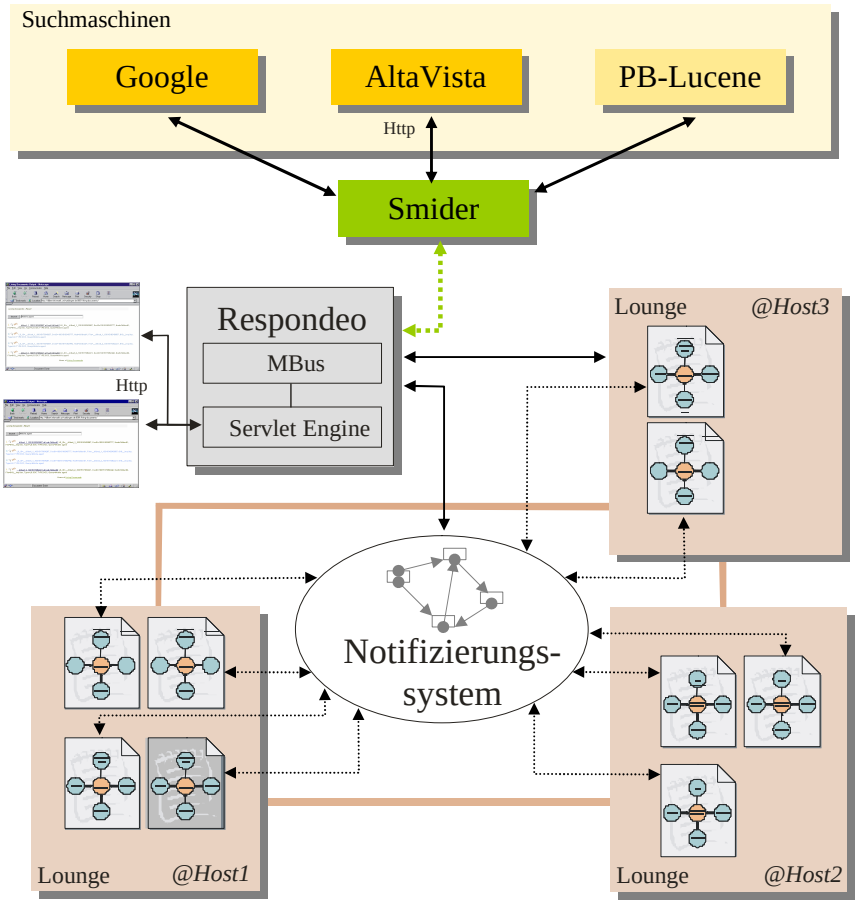


Abbildung 10.11: Systemüberblick des *Living Hypertextes* mit der Integration von *Smider.*

10.2.5.3 Topic Distillation im *Living Hypertext*

Die Informationssuche gemäß der *Topic Distillation* erweitert die Palette der verfügbaren Suchmöglichkeiten in einem *Living Hypertext.* Neben einer datenbank- und agentenbasierten Suche nach Informationen steht nun ein weiterer Suchmechanismus (vgl. *Hypertext-Sicht* in Abbildung 10.6) in einem *Living Hypertext* zur Verfügung, der sich durch die im vorherigen Abschnitt beschriebenen, automatisierten Abläufe auszeichnet. Zudem verbindet er traditionelle Web-Infrastrukturen – wie z. B. HTML-Dokumente und Web-Suchmaschinen, die auf einem passiven Dokumentenbegriff basieren – mit dem Konzept der *Living Documents*, denen aktive oder proaktive Dokumente zugrunde liegen.

Ein *Living Document* ist in der Lage, unterschiedliche Rollen zu übernehmen. Im Zusammenhang mit der Flexibilität eines *Living Documents* wird in Abschnitt 9.3.2 auch von einer *Sicht* auf ein *Living Document* gesprochen. In einer web-

ähnlichen Umgebung wie einem *Living Hypertext* kann ein *Living Document* HTML-basierte Repräsentationen dynamisch generieren. Auf der konzeptionellen Ebene findet so eine Annäherung der Informationsinhalte im *Living Hypertext* – den *Living Documents* – und denen im Web – den HTML-Dokumenten – statt.
Im Hinblick auf die datenbankzentrierten Ursprünge der Fallstudie über die Informationssysteme *PaperBase* und *PaperBaseLD* wird deutlich, daß in dem hier beschriebenen *Living Hypertext* eine Annäherung der Informationsinhalte von Datenbanken und von Dokumenten im Web stattfindet. Die Infrastrukturen beider „Welten“ sind nun einheitlich nutzbar. Schließlich entsteht so ein dezentrales, netzwerkartiges Informationssystem, das sich durch automatisierte Abläufe und durch eine einheitliche Behandlung von passiven und proaktiven Dokumenten auszeichnet.
Die eingehenden Anfragen von *Smider* an die *Living Documents* werden, wie es in Abbildung 10.11 dargestellt ist, stets über die Web-Schnittstelle des Applikationsservers *Respondeo* (vgl. Kapitel 4) ausgeführt. Dies betrifft sowohl Anfragen hinsichtlich des Dokumenteninhalts als auch Anfragen an eine semi-strukturierte Datenbasis eines *Living Documents*. Beide Informationsquellen – der Dokumenteninhalt und die semi-strukturierte Datenbasis – bilden die Grundlage für die dynamische Bestimmung der Themenkomplexe hinsichtlich der gestellten Benutzeranfrage.
Gemäß Abbildung 10.11 ist eine weitere spezielle Suchmaschine `PB-Lucene` auf der Basis von *Lucene* [Jak02] in *Smider* eingebunden. `PB-Lucene` verwaltet ausschließlich Informationen über die *Living Documents* in einem *Living Hypertext*. Diese Informationen repräsentieren einen Zustand des *Living Hypertextes*, der sich zu einem bestimmten Zeitpunkt in der Vergangenheit ergeben hat. Aufgrund der dynamischen Eigenschaften von *Living Documents* kann es zu fortwährenden Veränderungen in den Dokumenteninhalten und in den semi-strukturierten Datenbasen kommen. In dieser Hinsicht hat `PB-Lucene` nicht den Anspruch, den aktuellen Zustand des *Living Hypertextes* zu repräsentieren. Es geht lediglich darum, potentiell wertvolle Informationen über Inhalte und Linkstrukturen für *Smider* zur Verfügung zu stellen. Sie dienen vorwiegend als Startpunkte für Anfragen von *Smider* an den *Living Hypertext*. Danach werden die dynamischen Auswertungsmechanismen von *Smider* für die Bestimmung der relevanten Dokumente herangezogen. `PB-Lucene` bietet – analog zu den Suchmaschinen im Web – die Funktionalität, zu einem gegebenen Dokument, eine Menge von ähnlichen Dokumenten zurückzuliefern. Dies ist insbesondere von Bedeutung für die Adaptions-Komponente von *Smider* (vgl. Abbildung 10.10 in Abschnitt 10.2.5.2).

10.3 Einordnung in die Taxonomie

Die in Kapitel 3 erörterte Taxonomie klassifiziert Informationssysteme hinsichtlich der *Benutzersicht* (vgl. Abschnitt 3.1), des zugrunde liegenden *Dokumentenbegriffs* (vgl. Abschnitt 3.2) und der verwendeten *Technologie* (vgl. Abschnitt 3.3). In Ab-

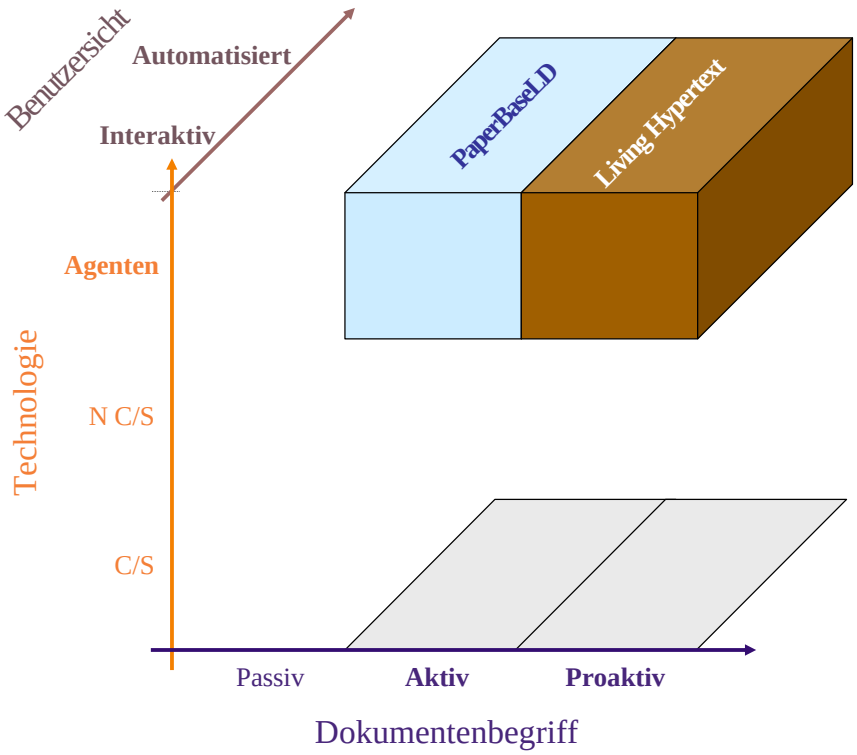


Abbildung 10.12: Einordnung der Informationssysteme *PaperBaseLD* und *Living Hypertext*.

bildung 10.12 sind die Einordnungen der Informationssysteme *PaperBaseLD* und *Living Hypertext* in die Taxonomie veranschaulicht. Beide Informationssysteme bilden hinsichtlich der Einordnung einen Quader, der sich über die Merkmale einer *interaktiven* und *automatisierten* Benutzersicht erstreckt (vgl. Abschnitte 3.1.1 und 3.1.2). Der wesentliche Unterschied zwischen *PaperBaseLD* und *Living Hypertext* liegt in dem verwendeten Dokumentenbegriff. Während in *PaperBaseLD* vorwiegend *aktive Dokumente* (vgl. Abschnitt 3.2.3) zum Einsatz kommen, stehen im *Living Hypertext proaktive Dokumente* (vgl. Abschnitt 3.2.4) im Mittelpunkt. Hinsichtlich der Technologie stehen bei beiden Informationssystemen *agentenbasierte* Realisierungen der Softwarekomponenten im Vordergrund (vgl. Abschnitt 3.3.1)[57]. Anschließend werden spezifische Eigenschaften des jeweiligen Informationssystems genauer erörtert:

Benutzersicht Sowohl *PaperBaseLD* als auch *Living Hypertext* unterstützen

[57] Der Einsatz des Applikationsservers *Respondeo* in beiden Informationssystemen als Vertreter von mehrstufigen Client/Server-Informationssystemen (vgl. Abschnitt 2.3) spielt bei der Einordnung in die Taxonomie nur eine untergeordnete Rolle und wird deswegen im folgenden nicht näher betrachtet.

interaktive und automatisierte Benutzersichten.

- *PaperBaseLD.* Bei der *datenbankbasierten Suche* (vgl. Abschnitt 10.1.3.3) initiiert der Benutzer eine Anfrage, die über das Notifizierungssystem *Siena* an alle *Living Documents* weitergeleitet wird. Die Ergebnisse derjenigen *Living Documents*, welche auf die gestellte Anfrage reagiert haben, werden eingesammelt und dem Benutzer direkt durch die web-basierte Benutzerschnittstelle präsentiert. Bei der *agentenbasierte Suche* (vgl. Abschnitt 10.1.3.4) wird für die initiierte Benutzeranfrage ein spezielles *Living Document* (`LDsearch`) generiert, das für die Bearbeitung der Anfrage verantwortlich ist. Das erzeugte `LDsearch` interagiert in automatisierter Form mit den verteilten *Living Documents*. Nach erfolgreicher Verarbeitung der Anfrage „mutiert" das `LDsearch` zu einem normalen *Living Document*, das seine gesammelten Informationen für künftige Anfragen an das Informationssystem zur Verfügung stellt.
- *Living Hypertext.* Generell unterstützt der im Abschnitt 10.2 erörterte *Living Hypertext* die gleichen Anfragemöglichkeiten wie das Informationssystem *PaperBaseLD*. Außerdem bietet es einen *navigierenden Zugriff* (vgl. Abschnitt 10.2.3) auf die verteilten Informationen in den *Living Documents* an, wie es typisch für Hypertexte ist. Die navigationsbasierte Suche in einem *Living Hypertext* wird weiterhin in eine interaktive und automatisierte Benutzersicht unterteilt:
  - Bei der interaktiven, navigationsbasierten Suche steht der Benutzer im Mittelpunkt, der durch die web-basierte Benutzerschnittstelle den Links in einem *Living Hypertext* folgt. Dabei stößt er Anfragen interaktiv an, die ihn im *Living Hypertext* zwischen den einzelnen *Living Documents* und deren Inhalten navigieren lassen (vgl. Abschnitt 10.2.3). Die Ergebnisse seiner Anfragen werden unmittelbar in der Benutzerschnittstelle visualisiert.
  - Bei der automatisierten, navigationsbasierten Suche stehen anstelle von Benutzern vielmehr Softwareprozesse wie z. B. *Smider* im Vordergrund der Betrachtung. In Abschnitt 10.2.4 wird *Smider* dazu benutzt, in automatisierter Form mit den *Living Documents* zu interagieren und die für den Benutzer relevanten Dokumente in entsprechenden Themenkomplexen zusammenzustellen.

Dokumentenbegriff Während das Informationssystem *PaperBaseLD* einen aktiven Dokumentenbegriff unterstützt, ist ein *Living Hypertext* durch proaktive Dokumente gekennzeichnet[58]. Eine ausführliche Erörterung von aktiven und proaktiven Dokumenten findet in Abschnitt 3.2 statt.

[58] In Abschnitt 3.2.4 werden proaktive Dokumente als spezielle aktive Dokumente des Typs C, die zusätzliche Eigenschaften besitzen, definiert. Die verschiedenen Typen von aktiven Dokumenten werden in Abbildung 3.5 auf Seite 58 veranschaulicht.

- *PaperBaseLD.* Aktive Dokumente des *Typs C* zeichnen sich durch die Objektivierung von Dokumenten aus. Dadurch fungieren Dokumente nicht mehr nur als passive Datencontainer mit einfachen Textinhalten, sondern übernehmen Aufgaben, die typisch für aktive Objekte sind. So initiieren aktive Dokumente in *PaperBaseLD* komplexe Anfragen an die semi-strukturierten Datenbasen oder generieren dynamisch eine HTML-Sicht bezüglich ihrer Datenbasis.
- *Living Hypertext.* Proaktive Dokumente im *Living Hypertext* zeichnen sich im Unterschied zu den aktiven Dokumenten des Typs C innerhalb von *PaperBaseLD* dadurch aus, daß sie einen sogenannten *Auftrag* besitzen (vgl. Abschnitt 10.2.2). Diesem Auftrag versuchen sie fortwährend und kontinuierlich nachzukommen, indem sie selbständig Aktionen initiieren. Eine wichtige Aktion ist dabei die (automatisierte) Interaktion mit anderen *Living Documents*, um festzustellen, ob diese möglicherweise für die Erfüllung ihres *Auftrags* relevante Informationen besitzen.

  Durch den aktiven und proaktiven Dokumentenbegriff, der sich in den *Living Documents* bzw. in einem *Living Hypertext* widerspiegelt, werden die Interaktions-, Kommunikations- und Suchprozesse flexibler gestaltet[59]. So sind proaktive Dokumente in der Lage, selbständig nach Informationen zu suchen oder etwaige Suchmaschinen über die Veränderungen in ihrer Datenbasis oder ihres Dokumenteninhalts (Blob) *aktiv* zu informieren. Anstelle der (zentralen) Dokumentenindexe bestehender Suchmaschinen treten nun eine Vielzahl aktiver Prozesse, die sich über Veränderungen in den Dokumenteninhalten oder deren semistrukturierten Datenbasen gegenseitig informieren.

Technologie Die agentenbasierte Realisierung der *Living Documents* (vgl. Kapitel 9) führt zu der Einordnung von *PaperBaseLD* und *Living Hypertext* in die Taxonomie, wie es in Abbildung 10.12 dargestellt ist. Die wesentlichen Merkmale des mehrstufigen Client/Server-Informationssystems *PaperBase*, auf dessen Basis die erweiterten und agentenbasierten Informationssysteme *PaperBaseLD* und *Living Hypertext* realisiert sind, gehen nicht mehr direkt in die Taxonomie ein.

[59] Vgl. die unterschiedlichen Sichten auf einen *Living Hypertext*: *Hypertext-*, *Datenbank-* und *Agentensicht* in Abbildung 10.6.

Teil V

Zusammenfassung der Ergebnisse mit einem Ausblick

Kapitel 11

Zusammenfassung

Im Rahmen der vorliegenden Dissertation wurden Techniken und Aspekte zur Realisierung von proaktiven Informationssystemen thematisiert. Auf der Grundlage dreier wesentlicher Säulen von Informationssystemen – der *Benutzersicht*, dem *Dokumentenbegriff* und der verwendeten *Technologie* – wurde eine Taxonomie erstellt. Nach einer ausführlichen Einführung und Abgrenzung von mehrstufigen Client/Server-Informationssystemen und netzwerkbasierten Informationssystemen wurden die Eigenschaften der entwickelten Taxonomie sowie deren inhaltliche Einordnung im Vergleich zu anderen Forschungsarbeiten erörtert. Im Mittelpunkt der Taxonomie steht eine ausgiebige Betrachtung und Diskussion passiver, aktiver und proaktiver Dokumentenbegriffe. Generell wurden die verschiedenen Infrastrukturen und Komponenten, die im Rahmen dieser Arbeit entworfen und entwickelt wurden, anhand der jeweiligen Benutzersichten, des zugrunde liegenden Dokumentenbegriffs sowie der eingesetzten Technologie diskutiert.

Im einzelnen wurden im ersten Teil dieser Arbeit ein leichtgewichtiger Applikationsserver, ein Agentenframework für den Entwurf von mobilen Agenten sowie eine spezielle Auszeichnungssprache auf der Basis von XML für die systematische und einheitliche Beschreibung von Systemzuständen erörtert. Im zweiten Teil wurden verschiedene Aspekte der realisierten Komponenten anhand mehrerer Fallstudien evaluiert. Die Fallstudien im zweiten Teil sind in unterschiedlichen Fachgebieten – konkret in den Bereichen des Monitorings in verteilten Systemen, des Managements von XML-basierten Testdokumenten und einer verteilten Berechnung auf der Basis von mobilen Agenten im Symbolischen Rechnen – durchgeführt worden. Die grundsätzlichen Problemstellungen, die in den Fallstudien untersucht wurden, spielen zudem eine wichtige Rolle bei der Realisierung proaktiver Informationssysteme. Deswegen können die Ergebnisse der Fallstudien auf die Domäne der proaktiven Informationssysteme übertragen werden, ohne daß dies für alle Aspekte im einzelnen vollzogen wurde.

Den zentralen Kern zur Realisierung von proaktiven Informationssystemen bildet schließlich der dritte Teil der Arbeit, in dessen Mittelpunkt das Konzept der sogenannten *Living Documents* steht. Hier werden die Techniken der entwickelten

Infrastrukturen und Komponenten, die im ersten Teil diskutiert wurden, vor dem Hintergrund der Realisierung proaktiver Informationssysteme zusammengeführt.
Ein *Living Document* beruht auf einem proaktiven Dokumentenbegriff, in dessen Zentrum eine semi-strukturierte Datenbasis zur automatisierten Verwaltung von Kontextinformationen und ein agentenbasierter Mediator für die Steuerung der Applikationslogik eines Dokuments stehen. Der aktive und proaktive Dokumentenbegriff führt zu einer Objektivierung von Dokumenten im allgemeinen, was eine konzeptionelle Annäherung von Dokumenten an objektorientierte Paradigmen mit sich bringt. *Living Documents* stellen verschiedene Programmiermodelle zur Realisierung von Informationssystemen zur Verfügung. Die Informations- und Kontrollflüsse in Informationssystemen, die auf *Living Documents* basieren, zeichnen sich durch zahlreiche automatisierte Interaktionsformen aus. Im Gegensatz zu klassischen Client/Server-Informationssystemen , in denen vorwiegend der Benutzer Informationsanfragen anstößt, übernehmen in proaktiven Informationssystemen die Dokumente – die *Living Documents* – selbständig die Rolle des Initiators von Informations- und Kontrollflüssen.
Anhand von zwei Fallstudien wurde der Einsatz und die Plausibilität des Konzepts der *Living Documents* veranschaulicht. Im Rahmen der ersten Fallstudie wurde ein bestehendes, datenbankzentriertes, mehrstufiges Client/Server-Informationssystem, das als ein typischer Repräsentant für heutige, web-basierte Informationssysteme anzusehen ist, in ein aus *Living Documents* bestehendes netzwerkbasiertes Informationssystem transformiert. Dafür wurden die passiven Dokumente des ursprünglichen Client/Server-Informationssystems jeweils in entsprechende *Living Documents* umgewandelt.
In der zweiten Fallstudie stand die Einbettung des netzwerkbasierten Informationssystems *Living Hypertext* in die Infrastrukturen des Webs im Zentrum der Betrachtung. Die Flexibilität der *Living Documents* ermöglicht die Realisierung verschiedener Formen der Informationssuche. Neben einer datenbank- und agentenbasierten Suche nach Informationen wurde zudem eine navigationsbasierte Suchmöglichkeit in einem *Living Hypertext* erörtert. Außerdem wurde im Rahmen dieser Fallstudie veranschaulicht, wie die zusätzliche Berücksichtigung von Umwelt- oder Kontextinformationen, die in den semi-strukturierten Datenbasen der *Living Documents* abgelegt sind, in den allgemeinen Suchprozeß eingebunden und für die dynamische, automatisierte Erstellung von Themenkomplexen benutzt werden kann.
Im Rahmen dieser Arbeit sind folgende Publikationen erschienen (in chronologischer Reihenfolge):

1. SCHIMKAT, R.-D., W. KÜCHLIN und R. KRAUTTER: *An Object-Oriented Framework for Rapid Client-Side Integration of Information Management Systems*. South African Computer Journal, (24):244–248, November 1999.

2. SCHIMKAT, R.-D., S. MÜLLER, W. KÜCHLIN und R. KRAUTTER: *A Lightweight, Message-Oriented Application Server for the WWW*. In: CAR-

ROLL, J., E. DAMIANI, H. HADDAD und D. OPPENHEIM (Herausgeber): *Proceedings of the 15th ACM Symposium on Applied Computing (SAC 2000)*, Seiten 934–941, Como, Italy, März 2000. ACM Press.

3. SCHIMKAT, R.-D., W. BLOCHINGER, C. SINZ, M. FRIEDRICH und W. KÜCHLIN: *A Service-Based Agent Framework for Distributed Symbolic Computation*. In: BUBAK, M., R. WILLIAMS, H. AFSARMANESH und B. HERTZBERGER (Herausgeber): *Proceedings of the 8th International Conference on High Performance Computing and Networking Europe (HPCN'00)*, Band 1823 der Reihe *Lecture Notes in Computer Science*, Seiten 644–656, Amsterdam, Netherlands, Mai 2000. Springer.

4. SCHIMKAT, R.-D., G. NUSSER und D. BÜHLER: *Scalability and Interoperability in Service-Centric Architectures for the Web*. In: TJOA, A M., R.R. WAGNER und A. AL-ZOBAIDIE (Herausgeber): *Proceedings of the 11th International Workshop on Database and Expert Systems Applications (DEXA 2000)*, Seiten 51–57, London, England, September 2000. IEEE Computer Society Press.

5. SCHIMKAT, R.-D., M. HÄUSSER, W. KÜCHLIN und R. KRAUTTER: *Web Application Middleware to Support XML-Based Monitoring in Distributed Systems*. In: DEBNATH, N. (Herausgeber): *Proceedings of 13th International Conference on Computer and Applications in Industry and Engineering (CAINE 2000)*, Seiten 203–207, Hawaii, USA, November 2000. International Society for Computers and Their Applications.

6. SCHIMKAT, R.-D. und W. KÜCHLIN: *Aspekte skalierbarer Infrastrukturen für die Integration verteilter Anwendungsarchitekturen*. In: KALETTA, D. und E. EDELHOFF (Herausgeber): *Proceedings 14. GI-Tagung Anwendungs- und System-Management im Zeichen von Multimedia und E-Business*, Tübingen, Germany, April 2001. Zentrum für Datenverarbeitung.

7. SCHIMKAT, R.-D., M. SCHMIDT-DANNERT, W. KÜCHLIN und R. KRAUTTER: *Tempto - An Object-Oriented Test Framework and Test Management Infrastructure Based On Fine-Grained XML-Documents*. In: *NetObjectDays 2000 - Object-Oriented Software Systems*, Seiten 274–287, Erfurt, Germany, Oktober 2000. Net.ObjectDays-Forum.

8. SCHIMKAT, R.-D., G. NUSSER und D. BÜHLER: *Scalability and Interoperability in Service-Centric Architectures for the Web*. In: TJOA, A M., R.R. WAGNER und A. AL-ZOBAIDIE (Herausgeber): *Proceedings of the 11th International Workshop on Database and Expert Systems Applications (DEXA 2000)*, Seiten 51–57, London, England, September 2000. IEEE Computer Society Press.

9. NUSSER, G. und R.-D. SCHIMKAT: *Rapid Application Development of Middleware Components by Using XML*. In: *Proceedings of the 12th IEEE*

Intl. Workshop on Rapid System Prototyping (RSP 2001), Seiten 116–121, Monterey CA, USA, June 2001. IEEE Computer Society Press.

10. SCHIMKAT, R.-D., M. FRIEDRICH und W. KÜCHLIN: *On Maintaining Code Mobility*. In: *Workshop on Software Engineering and Mobility, co-located with International Conference on Software Engineering 2001 (ICSE 2001)*, Toronto, Canada, Mai 2001.

11. SCHIMKAT, R.-D., M. FRIEDRICH und W. KÜCHLIN: *Deploying Distributed State Information in Mobile Agents Systems*. In: BATINI, C., F. GIUNCHIGLIA, P. GIORGINI und M. MECELLA (Herausgeber): *Proceedings 9th International Conference on Cooperative Information Systems (CoopIS 2001)*, Band 2172, der Reihe *Lecture Notes in Computer Science*, Seiten 80–94, Trento, Italy, September 2001. Springer.

12. MÜLLER, S. und R.-D. SCHIMKAT: *A General, Web-Enabled Model Retrieval Approach*. In: WANG, Y., S. PATEL und R.H. JOHNSTON (Herausgeber): *Proceedings of the 7th International Conference on Object-Oriented Information Systems (OOIS 2001)*, Seiten 487–496, Calgary, Canada, August 2001. Springer.

13. HEUMESSER, B.D. und R.-D. SCHIMKAT: *Deduction on XML Documents: A Case Study*. In: *Proceedings of the 14th International Conference of Applications of Prolog (INAP 2001) - Stream Content Management*, Seiten 20–29, Tokyo, Japan, November 2001. Prolog Association of Japan.

14. MÜLLER, S., R.-D. SCHIMKAT und R. MÜLLER: *Query Language for Structural Retrieval of Deep Web Information*. In: *2nd International Workshop on Web Dynamics (WebDyn 2002)*, Honolulu, Hawaii, USA, Mai 2002.

15. SCHIMKAT, R.-D. und W. KÜCHLIN: *Living Documents – Micro Servers for Documents*. In: CHAUDHRI, A.B., R. UNLAND, C. DJERABA und W. LINDNER (Herausgeber): *XML-Based Data Management and Multimedia Engineering – EDBT 2002 Workshops*, Band 2490 der Reihe *Lecture Notes in Computer Science*, Prague, Czech Republic, März 2002. Springer. Revised Papers.

16. SCHIMKAT, R.-D., W. KÜCHLIN und F. NESTEL: *Living Hypertext – Web Retrieval Techniques for Traditional Database-Centric Information*. In: UNGER, H., T. BÖHME und A. MIKLER (Herausgeber): *Proceedings of Second International Workshop on Innovative Internet Computing Systems (I2CS)*, Band 2346 der Reihe *Lecture Notes in Computer Science*, Seiten 1–14, Kühlungsborn, Germany, Juni 2002. Springer.

17. FRIEDRICH, M., R.-D. SCHIMKAT und W. KÜCHLIN: *Information Retrieval in Distributed Environments Based on Context-Aware, Proactive Documents*. In: ADAMCZAK, W. und A. NASE (Herausgeber): *Gaining Insight*

from Research Information – Proceedings of 6th International Conference on Current Research Information Systems (CRIS 2002), Seiten 153–158, Kassel, Germany, August 2002. University Press.

18. HEUMESSER, B.D., D. SEIPEL, R.-D. SCHIMKAT und U. Güntzer *A Web-Information System for Retrieving and Reasoning about XML-Based Mathematical Knowledge*. In: *Proceedings of International Conference on Electronic Information and Communication in Mathematics - How to find and post your mathematics in the web*, der Reihe *Springer Lecture Notes for Artificial Intelligence*. Akzeptiert zur Publikation. Voraussichtliche Veröffentlichung im Dezember 2002.

Kapitel 12

Ausblick

Nachfolgend werden exemplarisch einige weiterführende Themen angesprochen, deren vertiefte Betrachtung für zukünftige Untersuchungen im Zusammenhang mit proaktiven Informationssystemen vielversprechende Ansätze liefert. Im einzelnen werden zusätzliche Aspekte der *Implementierung* (siehe Abschnitt 12.1) und *Formalisierung* (siehe Abschnitt 12.2) proaktiver Informationssysteme sowie *weitergehende empirische Studien* (siehe Abschnitt 12.3) für den Einsatz von *Living Documents* besprochen.

12.1 Anbindung weiterer Infrastrukturen

Hinsichtlich einer Erweiterung implementierungsrelevanter Aspekte bei der Realisierung proaktiver Informationssysteme können zusätzliche Kommunikationsprotokolle für *Living Documents* zur Verfügung gestellt werden. Die agentenbasierte Implementierung der Code-Komponente bietet auf der Grundlage von KQML [FLM97] eine Schnittstelle für *Living Documents* an. In den Fallstudien in Kapitel 10 wurde die Anbindung proaktiver Informationssysteme an das Web auf der Basis von HTTP [Wor00] beschrieben. Aus Implementierungssicht stellt der Applikationsserver *Respondeo* einen speziellen Dienst zur Transformation der Anfragen von HTTP nach KQML zur Verfügung.

Die Anbindung zusätzlicher Infrastrukturen zur Kommunikation – wie beispielsweise das *Network File System (NFS)* [Inc89] oder das *Internet Message Access Protocol (IMAP4)* [Cri96] – bildet den Rahmen für erweiterte Einsatzmöglichkeiten proaktiver Informationssysteme. Mit der Einbeziehung von NFS oder IMAP4 wird die klassische Dokumentenwelt, die von passiven Dokumenten in Dateisystemen geprägt ist, mit einer dynamischen und aktiven Dokumenten- und Informationswelt verbunden. Dadurch ergibt sich für den Benutzer ein uniformer Zugriff sowohl auf passive als auch auf aktive Dokumente.

12.2 Formalisierung

Die Basis für die Realisierung proaktiver Informationssysteme bilden *Living Documents*, die durch agentenbasierte Verfahren ihr Verhalten bestimmen. *Living Documents* treffen proaktiv Entscheidungen über Informationsabläufe auf der Grundlage ihres lokal vorhandenen Wissens in den semi-strukturierten Datenbasen. Letztere enthalten nicht nur spezifische Informationen über das Dokument, sondern auch aktuelle Informationen über dessen Umwelt. Die vielfältigen und dynamischen Interaktionen zwischen *Living Documents* und ihrer Umwelt sind maßgebliche Einflußfaktoren für das Gesamtverhalten eines proaktiven Informationssystems.
Das *Co-Field Modell* [LMZ02] ist ein Framework zur Modellierung von Multiagentensystemen. Es besteht aus verteilten Datenstrukturen, die – ähnlich wie das Monitoring-System *Specto* – Abstraktionen für Agenten zur Wahrnehmung ihrer Umwelt bieten. Auf der Basis von Differentialgleichungen werden verhaltensrelevante Aspekte der autonomen Agenten sowie ihre Auswirkungen auf das gesamte System formal beschrieben [Par98, Par01]. Ein derartiger formaler Rahmen bietet mit Hilfe von Simulationsverfahren die Möglichkeit, in Abhängigkeit vom Verhalten einzelner autonomer Agenten – wie *Living Documents* – das Gesamtverhalten eines Informationssystems zu prognostizieren und zu analysieren.

12.3 Deep Web

Der Begriff „Deep Web“ bezieht sich auf diejenigen Dokumente im Web, die „tief“ in Online-Datenbanken abgelegt sind und von traditionellen Suchmaschinen nicht erfaßt werden. Im Gegensatz dazu bilden die von den Agenten-Crawlern erfaßbaren HTML-Dokumente das sogenannte „Surface Web“. Der überwiegende Teil der im Web verfügbaren Dokumente wird in Datenbanken abgelegt [Ber01], was die grundsätzliche Bedeutung des Deep Web als Dokumentenbasis für Informationssysteme unterstreicht. Die generellen Probleme traditioneller Suchmaschinen für eine systematische Einbeziehung der Dokumente im Deep Web finden ihren Ursprung in einem passiven Dokumentenbegriff, in dem Dokumente lediglich als Datencontainer betrachtet werden[1]. In zukünftigen Arbeiten können – basierend auf den Fallstudien zu *PaperBaseLD* und *Living Hypertext* in Kapitel 10 – weitere Untersuchungen der *Living Documents* im Zusammenhang mit dem Deep Web und innovativen Web-Infrastrukturen erfolgen. Ein Ziel kann es dabei sein, den passiven durch einen aktivierten Dokumentenbegriff zu ersetzen, damit Dokumente des Deep Web den Benutzern in ähnlicher Qualität zur Verfügung stehen wie Dokumente des Surface Web (vgl. Kapitel 10). Bei diesen Arbeiten sollten weniger implementierungsrelevante Aspekte der *Living Documents* als vielmehr empirische Untersuchungen im Vordergrund der Forschungstätigkeit stehen.

[1] Vgl. Abschnitt 3.2.1.2 für eine ausführliche Erörterung der Problematik eines passiven Dokumentenbegriffs im Zusammenhang mit Datenbanken und Dateisystemen.

Anhang A

Abbildungen der Infrastrukturen

```
# Beispielhafte Spezifikation von Applikationsobjekten

# XML-basiertes Monitoring
#   von T-Systems Komponenten
Debis1_Cer=capi.djmonitor.EventDatabase
    ,archivePath=C:\\public\\logArchive\\Debis1
    ,logFilePrefix=T
    ,logFileSuffix=log
    ,maxEvents=1000

# Einfacher Dienst zur Anbindung einer ODBC Datenbanken
DefaultSearchObjectDB=api.djenterprise.appserv.logic.SearchObject
    ,dbURL=jdbc:ODBC:test
    ,dbUser=scott
    ,dbPwd=tiger
```

Listing A.1: Konfiguration und Parametrisierung der Applikationsobjekte *Respondeo*s.

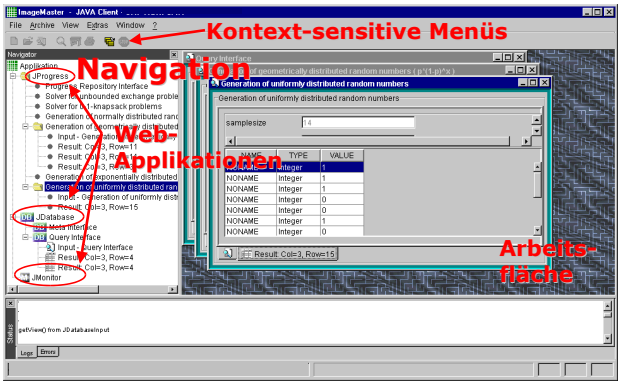


Abbildung A.1: Grafische Benutzeroberfläche des *ImageMasters* basierend auf dem *JIma*-Framework.

```
/**
   Jeder Nachricht besteht aus einem Header und einem Body.
   Der Header hat fixe Attribute und flexible
   erweiterbare Attribute. Diese sind ueber Name-Value-Paare
   definierbar. Fuer einfache Nachrichten reicht es
   beispielweise aus, ueber Name-Value-Paare zu
   kommunizieren, ohne ein Body-Objekt mitzuverschicken.
   <br>
   Der Nachrichtenfilter ist ebenfalls im Header integriert. Der
   Nachrichtenfilter besteht aus einer Liste von Filtern.
   Jeder Filter wirdü ber einen Namen eindeutig referenziert
   und angesprochen.

   @author Ralf Schimkat
   @version Dezember 1998

*/ public class MessageHeader implements java.io.Serializable{
      /** Quelle der Nachricht */
      public String originator;
      /** Ziel der Nachricht */
      public String target;
      /** name der Nachricht */
      public String messageName;
      /** Nachrichtenfilter der Nachricht */
      public Hashtable filter;
      /** Name-Value-Paare */
      public NameValuePair[] variableHeader;
      /** Client Hostname */
      public String clientHost;
      /** Konstruktor
      @param orginator Quelle der Nachricht
      @param target Ziel der Nachricht
      @param messageName Name der Nachricht
      @param i Anzahl der Name-Value-Paare
      */
      public MessageHeader(String originator,
                String target, String messageName, int i) {
         this.originator = originator;
         this.target = target;
         this.messageName = messageName;
         this.filter = null;
         if (i>=0)
            variableHeader = new NameValuePair[i];
         else
            variableHeader = null;
         clientHost = "NA";
      }

      /** Setzen des Nachrichtenfilters
      @param filter Liste von einzelnen Nachrichtenfilter
      */
      public void setFilter(Hashtable filter) {
         this.filter = filter;
      }
}
```

Listing A.2: Kompletter Aufbau der Metadaten in der Klasse `MessageHeader`.

```
56 /** Rueckgabe der
       Liste von Nachrichtenfilter
58     @return Hashtable Liste von Nachrichtenfilter
*/
60
  public Hashtable getFilter() {
62         return(filter);
  }
64
  /** @return String-Darstellung des Headers (=> XML-Like) */
66   public String toString() {
     StringBuffer sb = new StringBuffer("<MessageHeader>\n");
68   sb.append("<originator>");
     sb.append(originator);
70   sb.append("</originator>\n");
     sb.append("<target>");
72   sb.append(target);
     sb.append("</target>\n");
74   sb.append("<messagename>");
     sb.append(messageName);
76   sb.append("</messagename>\n");
     sb.append("<filter>\n");
78   if (filter != null) {
       for (Enumeration enum=filter.keys(); enum.hasMoreElements();){
80       sb.append("<key>" + enum.nextElement() + "</key>\n");
       }
82   }
     sb.append("</filter>\n");
84   for (int i=0; i<variableHeader.length; i++) {
         sb.append("<variable>\n");
86       sb.append("<name>"+variableHeader[i].name+"</name>\n");
         sb.append("<value>"+variableHeader[i].value+"</value>\n");
88       sb.append("</variable>\n");
     }
90   sb.append("</MessageHeader>\n");
     return sb.toString();
92 }
}
```

Listing A.3: Kompletter Aufbau der Metadaten in der Klasse `MessageHeader` (Teil 2).

```
<?xml version="1.0"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="spectoml">
    <xs:annotation>
      <xs:documentation> !DOCTYPE spectoml; </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="preamble" />
        <xs:choice minOccurs="0" maxOccurs="unbounded">
          <xs:element ref="log" />
          <xs:element ref="warn" />
          <xs:element ref="error" />
        </xs:choice>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="preamble">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="version" />
        <xs:element ref="timeZone" minOccurs="0" />
        <xs:element ref="datePattern" minOccurs="0" />
        <xs:element ref="messageType" minOccurs="0" />
        <xs:element ref="source" minOccurs="0" />
        <xs:element ref="startDate" minOccurs="0" />
        <xs:element ref="info" minOccurs="0" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="version" type="xs:string">
    <xs:annotation>
      <xs:documentation> ?xml version="1.0"? </xs:documentation>
    </xs:annotation>
  </xs:element>
  <xs:element name="timeZone" />
  <xs:element name="datePattern" />
  <xs:element name="messageType" />
  <xs:element name="startDate" />
  <xs:element name="info" />
```

Listing A.4: Eine Übersetzung der *SpectoML*-DTD in eine XML-Schema Spezifikation.

```
42    <xs:element name="log">
        <xs:complexType>
44        <xs:sequence>
            <xs:element ref="source" minOccurs="0" />
46          <xs:element ref="date" />
            <xs:element ref="msg" />
48          <xs:element ref="param" minOccurs="0" maxOccurs="unbounded" />
          </xs:sequence>
50        <xs:attribute name="sev" type="xs:string" />
          <xs:attribute name="id" type="xs:ID" />
52        <xs:attribute name="fatherID" type="xs:IDREF" />
        </xs:complexType>
54    </xs:element>
      <xs:element name="warn">
56      <xs:complexType>
          <xs:sequence>
58          <xs:element ref="source" minOccurs="0" />
            <xs:element ref="date" />
60          <xs:element ref="msg" />
            <xs:element ref="param" minOccurs="0" maxOccurs="unbounded" />
62        </xs:sequence>
          <xs:attribute name="sev" type="xs:string" />
64        <xs:attribute name="id" type="xs:ID" />
          <xs:attribute name="fatherID" type="xs:IDREF" />
66      </xs:complexType>
      </xs:element>
68    <xs:element name="error">
        <xs:complexType>
70        <xs:sequence>
            <xs:element ref="source" minOccurs="0" />
72          <xs:element ref="date" />
            <xs:element ref="msg" />
74          <xs:element ref="param" minOccurs="0" maxOccurs="unbounded" />
          </xs:sequence>
76        <xs:attribute name="sev" type="xs:string" />
          <xs:attribute name="id" type="xs:ID" />
78        <xs:attribute name="fatherID" type="xs:IDREF" />
        </xs:complexType>
80    </xs:element>
```

Listing A.5: XML Schema der *SpectoML* (Teil 2).

```
  <xs:element name="source">
    <xs:complexType>
      <xs:attribute name="app" type="xs:string" use="required" />
      <xs:attribute name="appID" type="xs:string" />
      <xs:attribute name="class" type="xs:string" />
      <xs:attribute name="IP" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>
  <xs:element name="date" />
  <xs:element name="msg">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute name="id" type="xs:integer" />
          <xs:attribute name="para1" type="xs:string" />
          <xs:attribute name="para2" type="xs:string" />
          <xs:attribute name="para3" type="xs:string" />
          <xs:attribute name="para4" type="xs:string" />
          <xs:attribute name="para5" type="xs:string" />
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="param">
    <xs:complexType>
      <xs:attribute name="type" type="xs:string" />
      <xs:attribute name="name" type="xs:string" />
      <xs:attribute name="value" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Listing A.6: XML Schema der *SpectoML* (Teil 3).

```
void doParse() : {
  Token t;
}{
  "SELECT" ( <WHITESPACE> perfSelect() )
      ("," <WHITESPACE> perfSelect())* <WHITESPACE>
  "FROM" (<WHITESPACE> perfFrom())
      ("," <WHITESPACE> perfFrom())* <WHITESPACE>
  (";" |
      ("WHERE" <WHITESPACE> perfWhere()
        ( "::" <WHITESPACE> perfWhere())* <WHITESPACE> ))
  (";"|("ATTRIBUTES" <WHITESPACE> perfAttributes()
      <WHITESPACE> (perfAttributes() <WHITESPACE>)* ";") )
}

void perfAttributes() : {
  Token t;
}{
  t = "BLOCKED"  {
   pc.blockSelectedIDs = new boolean[]{true, true, true};
  }
 | t = "UNBLOCKED"  {
   pc.blockSelectedIDs = new boolean[]{false, false, false};
  }
 | t = <WORD> {
   throw new ParseException("Unknown Attribute:"
      + String.valueOf(t));
 }
}

void getHostNames() : {
   Token op3;
} {
   "," <WHITESPACE> op3= <WORD> {
      hostNames.addElement(String.valueOf(op3));
   }
}
```

Listing A.7: Die Grammatik des Parsers für *SpectoML*-Dokumente.

```
String expression() : {
  Token t;
  String cont;
  String expr ="";
  String res = "";
} {
    t = <WORD>
    {
      return t.image;
    }
  | t = <STRING>
    {
      System.out.println("String␣:␣" +t.image);
      return t.image;
    }
  | "("
      t = <WORD>
      ( <WHITESPACE> expr = expression() { res = res + "␣" + expr; })*
     ")"
    {
      return t.image + "␣" + res;
    }
  | "#" t = <WORD> "," cont = readString(t.image)
    {
      return cont;
    }
}


TOKEN: {
    <WORD: (<CHARACTERWO>)+>
  | <CHARACTERWO: <ALPHA>|<NUMERIC>|<SPECIALWO>>
  | <CHARACTERW: <ALPHA>|<NUMERIC>|<SPECIALW>>
  | <CHARACTERDEFAULT: <ALPHA>|<NUMERIC>>
  | <SPECIALWO: ["<" , ">" , "=" , "+" , "—" , "*"
     , "/" , "&" , "^" , "~" , "_" , "@" , "$" , "%"
     , "." , "!" , "?"]>
  | <SPECIALW: <SPECIALWO> | ":">
  | <ALPHA: ["a"-"z"] | ["A"-"Z"]>
  | <NUMERIC:  ["0"-"9"]>
  | <WHITESPACE: (<WHITE>)+>
  | <WHITE: ["␣", "\t", "\r", "\n"]>
  | <STRING: "\"" (<STRINGCHAR>)* "\"">
  | <STRINGCHAR: <CHARACTERW>|<WHITE>|"("|")">
  | <WS: ("+" (<CHARACTERW>)*)| ("—" (<CHARACTERW>)*)
       | ("+"<STRING>) | ("—"<STRING>)>
  | <ATTLIST: ("a" "a" "a")|("b" "b" "b")>
}
```

Listing A.8: Die Grammatik des Parsers für *SpectoML*-Dokumente (Teil 2).

Anhang B

Abbildungen der *Living Documents*

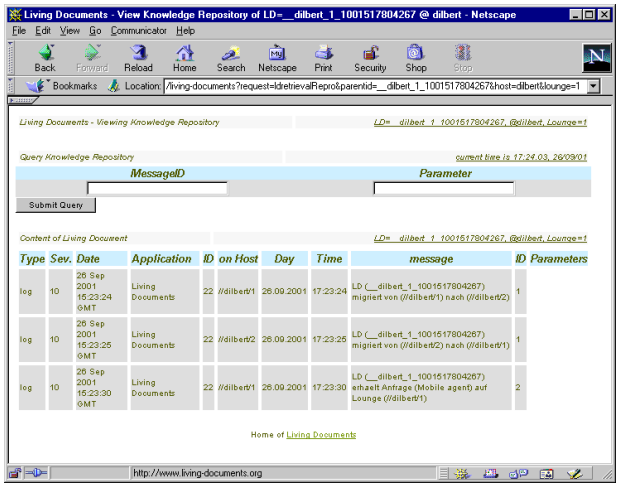


Abbildung B.1: Web-Benutzerschnittstelle zur Anfrage der semi-strukturierten Datenbasis eines *Living Documents*.

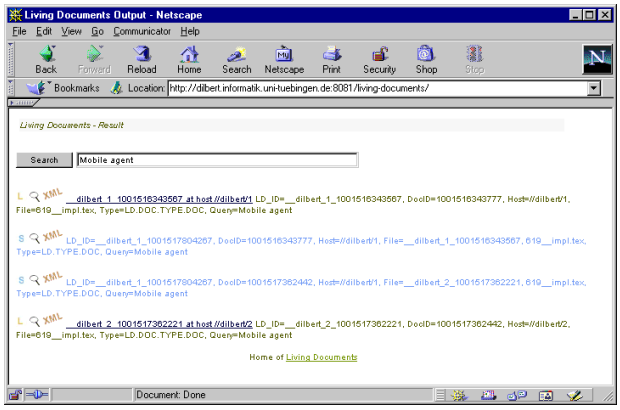


Abbildung B.2: Visualisierung des Ergebnisses einer Anfrage an ein proaktives Informationssystem, das auf *Living Documents* basiert.

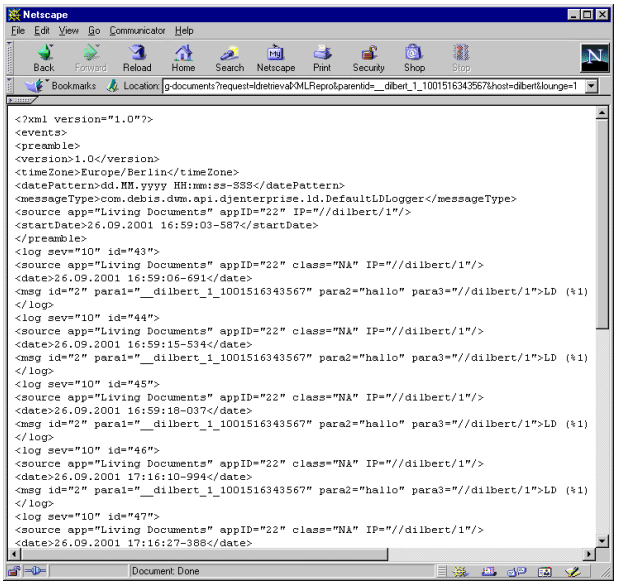


Abbildung B.3: Darstellung der Datenbasis als XML im Browser.

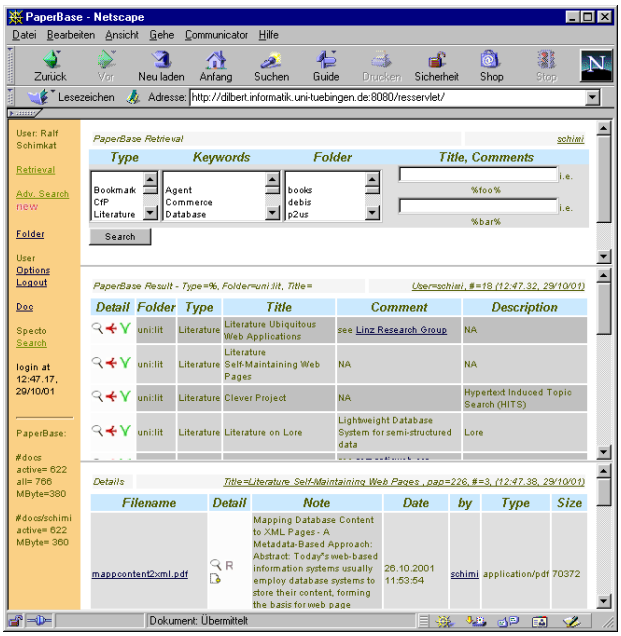


Abbildung B.4: Grundsätzlicher Aufbau der Benutzerschnittstelle des Web-Clients der *PaperBase*.

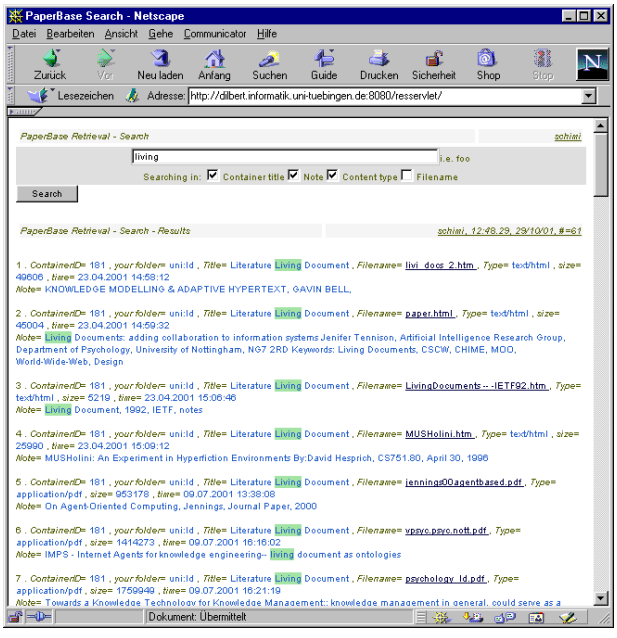


Abbildung B.5: Web-Client der *PaperBase* für die Suche nach Containern und Dokumenten.

Literaturverzeichnis

[Abi97] ABITEBOUL, S.: *Querying Semi-Structured Data*. In: AFRATI, F.N. und P. KOLAITIS (Herausgeber): *Proceedings of the 6th International Conference on Database Theory – (ICDT)*, Band 1186 der Reihe *Lecture Notes in Computer Science*, Seiten 1–18. Springer, Januar 1997.

[Abi99] ABITEBOUL, S.: *On Views and XML*. In: *Proceedings of the 18th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, Seiten 1–19, Mai 1999.

[AC93] AGOSTI, M. und F. CRESTANI: *A Methodology for the Automatic Construction of a Hypertext for Information Retrieval*. In: *Proceedings of the 1993 ACM Symposium on Applied Computing (SAC'93)*, Seiten 745–753, Indianapolis, USA, 1993. ACM.

[AGB+02] AUSTIN, D., W.W. GRAINGER, A. BARBIR, C. FERRIS und S. GARG: *Web Services Architecture Requirements – W3C Working Draft 19 August 2002*. World Wide Web Consortium (W3C), http://www.w3.org/TR/2002/WD-wsa-reqs-20020819, August 2002.

[AGH00] ARNOLD, K., J. GOSLING und D. HOLMES: *The Java Programming Language*. Addison-Wesley, Reading, Massachusetts, 3. Auflage, Juni 2000.

[ALB96] ABDU, HASINA, HANAN LUYTFIYYA und MICHAEL A. BAUER: *Investigating Monitoring Configurations*. In: *Proceedings of the 1996 ACM Symposium on Applied Computing*, Seiten 366–373, February 1996.

[Alt02] ALTAVISTA, http://www.altavista.com: *AltaVista – Homepage*, Juli 2002.

[AM00] AGOSTI, M. und M. MELUCCI: *Information Retrieval on the Web*. In: AGOSTI, M., F. CRESTANI und G. PASI (Herausgeber): *Lectures on Information Retrieval of the Third European Summer-School*,

ESSIR 2000, Revised Lectures, Band 1980, Seiten 242–285, Varenna, Italy, September 2000. Springer LNCS.

[BCV99] BOMPANI, L., P. CIANCARINI und F. VITALI: *Active Documents in XML*. ACM SigWeb Newsletter, 8(1):27–32, 1999.

[Bec95] BECKER, P.: *A Framework for Providing and Using Algorithms and Algorithmic Meta Knowledge on the Internet*. In: RAM, S. und M. JARKE (Herausgeber): *Proceedings of the 5th Annual Workshop on Information Technologies & Systems (WITS'95)*, Nummer 95–15 in *Aachener Informatik-Berichte*, Seiten 2–11, 1995.

[BEK+00] BOX, D., D. EHNEBUSKE, G. KAKIVAYA, A. LAYMAN, N. MENDELSOHN, H.F. NIELSEN, S. THATTE und D. WINER: *Simple Object Access Protocol (SOAP) 1.1 – W3C Note 08 May 2000*. World Wide Web Consortium (W3C), http://www.w3.org/TR/SOAP, Mai 2000. http://www.w3.org/TR/2000/NOTE-SOAP-20000508.

[Ber96] BERNSTEIN, P.: *Middleware: A Model for Distributed System Services*. Communications of the ACM, 39(2):86–98, Jul 1996.

[Ber01] BERGMAN, M.K.: *The Deep Web: Surfacing Hidden Value*. The Journal of Electronic Publishing, 7(1), August 2001. http://www.press.umich.edu/jep/07-01/bergman.html.

[BG02] BECK, K. und E. GAMMA: *JUnit 3.2*, 2002. Verfügbar unter http://www.xprogramming.com/software.html.

[BGK+97] BÄUMER, D., G. GRYCZAN, R. KNOLL, C. LILIENTHAL, D. RIEHLE und H. ZÜLLIGHOVEN: *Frameworks Development for Large Systems*. Communications of the ACM, 40(10), Oktober 1997.

[BH98] BHARAT, K. und M.R. HENZINGER: *Improved Algorithms for Topic Distillation in a Hyperlinked Environment*. In: *Proceedings of the 21st annual International ACM SIGIR Conference Research and Development in Information Retrieval*, Seiten 104–111, Melbourne, Australia, 1998.

[BHH+02] BELLWOOD, T., L. CLEMENTAND D. EHNEBUSKEAND A. HATELY, M. HONDO, Y.L. HUSBAND, K. JANUSZEWSKI, S. LEE, B. MCKEE, J. MUNTER und C. VON RIEGEN: *UDDI Version 3.0 – Published Specification*. UDDI.org, http://www.uddi.org/pubs/uddi-v3.00-published-20020719.htm, Juli 2002.

[BJ94] BECK, K. und R. JOHNSON: *Patterns Generate Architectures*. In: *Proceedings of European Conference for Object-Oriented Programming (ECOOP'94)*, Seiten 139–149. Springer-Verlag, 1994.

[BK01] BÜHLER, D. und W. KÜCHLIN: *Flexible Similarity Assessment for XML Documents Based on XQL and Java Reflection*. In: MONOSTORI, L., J. VÄNCZA und M. ALI (Herausgeber): *Proceedings of the 14th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems IEA/AIE 2001*, Band 2070 der Reihe *LNCS*, Seiten 175–186, Budapest, Hungary, Juni 2001. Springer.

[BL99] BERNERS-LEE, T.: *Weaving the Web*. Harper, San Francisco, März 1999.

[BLFM98] BERNERS-LEE, T., R. FIELDING und L. MASINTER: *Uniform Ressource Identifiers (URI): Generic Syntax*. The Internet Engineering Task Force: Network Working Group, http://www.ietf.org/rfc/rfc2396.txt, August 1998.

[BLHL01] BERNERS-LEE, T., J. HENDLER und O. LASSILA: *The Semantic Web - A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities*. Scientific American, 284(5):34–43, 2001.

[BM01] BIRON, P.V. und A. MALHOTRA: *XML Schema Part 2: Datatypes – W3C Recommendation May 2001*. World Wide Web Consortium (W3C), http://www.w3.org/TR/xmlschema-2/, Mai 2001.

[BN84] BIRELL, A.D. und B.J. NELSON: *Implementing Remote Procedure Calls*. ACM Transactions on Computer Systems, 2(1):39–59, Februar 1984.

[Boo94] BOOCH, GRADY: *Object-Oriented Analysis and Design With Applications*. Addison Wesley Publishing Company, Reading Massachusetts, 1994.

[BP89] BIGGERSTAFF, T.J. und A.J. PERLIS: *Software Reusability: Concepts and Models*, Band 1. ACM Press, New York, 1989.

[BP98a] BALDI, M. und G.P. PICCO: *Evaluating the Tradeoffs of Mobile Code Design Paradigms in Network Management Applications*. In: *Proceedings of the 20th International Conference on Software Engineering*, Seiten 146–155. ACM Pres, 1998.

[BP98b] BRIN, S. und L. PAGE: *The anatomy of a large-scale hypertextual Web search engine*. Computer Networks and ISDN Systems, 30(1–7):107–117, 1998.

[BPSMM00] BRAY, T., J. PAOLI, C.M. SPERBERG-MCQUEEN und E. MALER: *Extensible Markup Language (XML) 1.0 (Second Edition)*.

World Wide Web Consortium (W3C), http://www.w3.org/TR/REC-xml, Oktober 2000.

[Bra97] BRADSHAW, J.M. (Herausgeber): *Software Agents*. AAAI Press / The MIT Press, 1997.

[Bro98] BROWN, G.: *Intensional HTML 2: A practical approach*. Master Thesis, Department of Computer Science, University of Victoria, Canada, 1998.

[Bro99] BRODIE, M.: *Que Sera, Sera: The Coincidental Confluence of Economics, Business, And Collaborative Computing*. In: *Proceedings of 15th Internationel Conference on Data Engineering (ICDE'99)*, Seiten 2–3, Sidney, Australia, 1999.

[Bus45] BUSH, V.: *As We May Think*. The Atlantic Monthly, 176(1):101–108, Juli 1945.

[Cas01] CASSAR, B.: *Interaktive Visualisierung von verteilten Prozessen*. Diplomarbeit, Universität Tübingen, Juli 2001.

[CCF+01] CHAMBERLIN, D., J. CLARK, D. FLORESCU, J. ROBIE, J. SIMEON und M. STEFANESCU: *XQuery 1.0: An XML Query Language*. Available at http://www.w3.org/TR/xquery, Juni 2001.

[CCMW01] CHRISTENSEN, E., F. CURBERA, G. MEREDITH und S. WEERAWARANA: *Web Services Description Language (WSDL) 1.1 – W3C Note*. World Wide Web Consortium (W3C), http://www.w3.org/TR/wsdl, März 2001. http://www.w3.org/TR/2001/NOTE-wsdl-20010315.

[CD99] CLARK, J. und S. DEROSE: *XML Path Language (XPath) 1.0*. World Wide Web Consortium (W3C), http://www.w3.org/TR/xpath, November 1999.

[CDG+98] CHAKRABARTI, S., B. DOM, D. GIBSON, J. KLEINBERG, P. RAGHAVAN und S. RAJAGOPALAN: *Automatic Ressource list compilation by analyzing hyperlink structure and associated text*. In: *Proceedings of the 7th International World Wide Web Conference*, 1998.

[CH01] CABEZA, D. und M. HERMENEGILDO: *Programming in XPCE/Prolog*. Available at http://www.swi.psy.uva.nl/projects/xpce/, Februar 2001.

[Che75] CHEN, P.P.: *The Entity-Relationship Model: Toward a Unified View of Data*. In: *Proceedings of the 1th Conference on Very Large Databases*, Seite pp.173. Morgan Kaufman (Los Altos CA), 1975.

[Che76] CHEN, P.: *The entity-relationship model: Toward a unified view of data*. ACM Transactions on Database Systems (TODS), 1(1):9–36, 1976.

[Cod70] CODD, E. F.: *A Relational Model of Data for Large Shared Data Banks*. Communications of the ACM, 13(6):377–387, 1970.

[CPV97] CARZANIGA, A., G.P. PICCO und G. VIGNA: *Designing Distributed Applications with Mobile Code Paradigms*. In: *Proceedings of the 19th International Conference on Software Engineering*, Seiten 22–32, Mai 1997.

[CRF00] CHAMBERLIN, D., J. ROBIE und D. FLORESCU: *Quilt: An XML Query Language for Heterogeneous Data Sources*. In: *Proceedings of the Conference on Web Databases (WebDB)*, LNCS. Springer-Verlag, 2000.

[Cri96] CRISPIN, M.: *Internet Message Access Protocol version 4rev1*. Network Working Group, http://www.isi.edu/in-notes/rfc2060.txt, Dezember 1996.

[CRW00] CARZANIGA, A., D.S. ROSENBLUM und ALEXANDER L. WOLF: *Achieving Scalability and Expressiveness in an Internet-Scale Event Notification Service*. In: *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, Seiten 219–227, Portland OR, USA, Juli 2000.

[CRW01] CARZANIGA, A., D.S. ROSENBLUM und A.L. WOLF: *Design and Evaluation of a Wide-Area Event Notification Service*. ACM Transactions on Computer Systems, 19(3):332–383, August 2001.

[CS93] COMER, D.E. und D.L. STEVENS: *Internetworking with TCP/IP. Volume III: Client-Server Programming and Applications*, Band III. Prentice Hall, 1993.

[CvdBD99] CHAKRABARTI, S., M. VAN DEN BERG und B. DOM: *Focused Crawling: A New Approach to Topic-Specific Web Ressource Discovery*. In: *8th World Wide Web Conference*, Toronto, May 1999.

[CW85] CARDELLI, C. und P. WEGNER: *On Understanding Types, Data Abstraction, and Polymorphism*. ACM Computing Surveys, 17(4):471–522, 1985.

[CW00] CHAUDHURI, S. und G. WEIKUM: *Rethinking Database System Architecture: Towards a Self-Tuning RISC-style Database System*. In: ABBADI, A.EL, M.L. BRODIE, S. CHAKRAVARTHY, U. DAYAL, N. KAMEL, G. SCHLAGETER und K.-Y. WHANG (Herausgeber):

Proceedings of the 26th International Conference on Very Large Databases, Seiten 1–10, Cairo, Egypt, 2000. Morgan Kaufmann.

[CZ01] CHANG, S.-K. und T. ZNATI: *Adlet: An Active Document Abstraction for Multimedia Information Fusion*. IEEE Transactions on Knowledge and Data Engineering, 13(1):112–123, 2001.

[Dat95] DATE, C.J.: *An Introduction to Database Systems*. Addison-Wesley, Reading, MA, USA, 1995. 6th Ed.

[DD93] DATE, C.J. und H. DARWEN: *A Guide to The SQL Standard*. Addison-Wesley, 3. Auflage, 1993.

[DELS99a] DOURISH, P., W.K. EDWARDS, A. LAMARCA und M. SALISBURY: *Presto: An Experimental Architecture for Fluid Interactive Document Spaces*. ACM Transactions on Computer-Human Interaction (TOCHI), 6(2):131–161, 1999.

[DELS99b] DOURISH, P., W.K. EDWARDS, A. LAMARCA und M. SALISBURY: *Using properties for uniform interaction in the Presto document system*. In: *Proceedings of the 12th annual ACM symposium on User interface software and technology*, Asheville, North Carolina, USA, 1999.

[DFF+98] DEUTSCH, A., M. FERNANDEZ, D. FLORESCU, A. LEVY und D. SUCIU: *XML-QL: A Query Language for XML*. Available at http://www.w3.org/TR/NOTE-xml-ql/, August 1998.

[DHW95] DAYAL, U., E.N. HANSON und J. WIDOM: *Active Database Systems*. In: KIM, W. (Herausgeber): *Modern Database Systems: The Object Model, Interoperability, and Beyond*, Seiten 434–456. ACM Press and Addison-Wesley, 1995.

[DMNS97] DEMEYER, S., T.D. MEIJLER, O. NIERSTRASZ und P. STEYAERT: *Design Guidelines for Tailorable Frameworks*. Communications of the ACM, 40(10):60–64, Oktober 1997.

[DMO01] DEROSE, S., E. MALER und D. ORCHARD: *XML Linking Language (XLink) 1.0*. World Wide Web Consortium (W3C), http://www.w3.org/TR/xlink, Juni 2001.

[Dud96] DUDENREDAKTION: *Duden – Die deutsche Rechtschreibung*. Band 1. Dudenverlag, 21 Auflage, 1996.

[EC94] EHRLICH, K. und D. CASH: *Turning information into knowledge: Information finding as a collaborative activity*. In: *Proceedings of Digital Libraries (DL'94)*, Seiten 119–125, Texas, USA, Juni 1994.

[EHWM01] EICKHOFF, B., A. HALLER-WOLF und D. MANG: *Der Duden – Das Fremwörterbuch*. Dudenverlag, 2001.

[EL99] EDWARDS, W.K. und A. LAMARCA: *Balancing Generality and Specificity in Document Management Systems*. In: *Proceedings of the 7th IFIP Conference on Human-Computer Interaction (Interact'99)*, Edinburgh, Scotland, August 1999.

[Fed93] FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATIONS, http://www.itl.nist.gov/fipspubs/fip127-2.htm: *Announcing the Standard for Database Language SQL*, 1993.

[FH99] FRIEDMAN-HILL, E.J.: *Jess, The Java Expert System Shell*. Available at the URL: http://herzberg.ca.sandia.gov/jess/, 1999.

[FHvH+00] FENSEL, D., I. HORROCKS, F. VAN HARMELEN, S. DECKER, M. ERDMANN und M. KLEIN: *OIL in a Nutshell*. In: *European Knowledge Acquisition Conference (EKAW-2000)*, Seiten 38–45. Springer LNAI, Oktober 2000.

[Fla01] FLANAGAN, D.: *JavaScript: The Definitive Guide*. O'Reilly and Associates, 2001.

[FLM97] FINN, T., Y. LABROU und J. MAYFIELD: *KQML as an Agent Communication Language*. In: BRADSHAW, J.M. (Herausgeber): *Software Agents*, Seiten 291–316. MIT Press, 1997.

[For82] FORGY, C.L.: *Rete: A Fast Algorithm for the Many Pattern/ Many Object Pattern Match Problem*. Artificial Intelligence, 19:17–32, 1982.

[Fow99] FOWLER, M.: *Refactoring – Improving the Design of Existing Code*. Addison Wesley, 1999.

[FPV98] FUGETTA, A., G.P. PICCO und G. VIGNA: *Understanding Code Mobility*. IEEE Transactions on Software Engineering, 24(5):342–361, Mai 1998.

[Fri99] FRIEDRICH, M.: *Entwurf und Implementierung eines Systems für mobile Agenten*. Diplomarbeit, Universität Tübingen, September 1999.

[FS97] FAYAD, M. und D. SCHMIDT: *Object-Oriented Application Frameworks*. Communications of the ACM, 40(10), Oktober 1997.

[FSK02] FRIEDRICH, M., R.-D. SCHIMKAT und W. KÜCHLIN: *Information Retrieval in Distributed Environments Based on Context-Aware,*

Proactive Documents. In: ADAMCZAK, W. und A. NASE (Herausgeber): *Gaining Insight from Research Information – Proceedings of 6th International Conference on Current Research Information Systems (CRIS 2002)*, Seiten 153–158, Kassel, Germany, August 2002. University Press.

[Fuh00] FUHR, N.: *Models in Information Retrieval*. In: AGOSTI, M., F. CRESTANI und G. PASI (Herausgeber): *Lectures on Information Retrieval of the Third European Summer-School, ESSIR 2000, Revised Lectures*, Band 1980, Seiten 21–50, Varenna, Italy, September 2000. Springer LNCS.

[FvHH+01] FENSEL, D., F. VAN HARMELEN, I. HORROCKS, D.L. MCGUINNESS und P.F. PATEL-SCHNEIDER: *OIL: An Ontology Infrastructure for the Semantic Web*. IEEE Intelligent Systems, 16(2):38–45, März 2001.

[FZ94] FORMAN, G.H. und J. ZAHORJAN: *The Challenges of Mobile Computing*. IEEE Computer, 27(4):38–47, 1994.

[GK94] GENESERETH, M.R. und S.P. KETCHPEL: *Software Agents*. Communications of the ACM, 37(7), 1994.

[GKP+01] GRAY, R.S., D. KOTZ, R.A. PETERSON, J. BARTON, D.A. CHACON, P. GERKEN, M.O. HOFMANN, J. BRADSHAW, M.R. BREEDY, R.JEFFERS und N. SURI: *Mobile-Agent versus Client/Server Performance: Scalability in an Information-Retrieval Task*. In: PICCO, G.P. (Herausgeber): *Proceedings of 5th International Conference on Mobile Agents*, Band 2240 der Reihe *Lecture Notes in Computer Science*. Springer, Dezember 2001.

[Goo02] GOOGLE, http://www.google.com: *Google – Homepage*, Juli 2002.

[GR83] GOLDBERG, A. und D. ROBSON: *Smalltalk-80: The Language and Its Implementation*. Addison-Wesley, Reading, Massachusetts, 1983.

[Gru93] GRUBER, T.R.: *A Translation Approach to Portable Ontology Specifications*. Knowledge Acquisition, 5:199–220, 1993.

[Häu99] HÄUSSER, M.: *XML-based Monitoring in Distributed Systems*. Diplomarbeit, Universität Tübingen, Dezember 1999.

[Hen01] HENDLER, JAMES: *Agents and the Semantic Web*. IEEE Intelligent Systems, 16(2):30–37, März 2001.

[Hew77] HEWITT, C.: *Viewing Control Structures as Patterns of Passing Messages*. Artificial Intelligence, 8(3):323–364, 1977.

[Hew02] HEWLETT PACKARD CORP., http://www.e-speak.net/: *e-speak.net - Homepage*, Mai 2002.

[HG98] HAAS, S.W. und E.S. GRAMS: *Page and link classifications: connecting diverse Ressources*. In: *Proceedings of the third ACM Conference on Digital libraries*, Seiten 99–107. ACM Press, 1998.

[HM00] HEINRICH, E. und H. MAURER: *Active Documents: Concept, Implementation, and Applications*. Journal of Universal Computer Science, 6(12):1197–2002, 2000.

[HS01] HEUMESSER, B.D. und R.-D. SCHIMKAT: *Deduction on XML Documents: A Case Study*. In: *Proceedings of the 14th International Conference of Applications of Prolog (INAP 2001) - Stream Content Management*, Seiten 20–29, Tokyo, Japan, November 2001. Prolog Association of Japan.

[HSSG02] HEUMESSER, B.D., D. SEIPEL, R.-D. SCHIMKAT und U. GÜNTZER: *A Web-Information System for Retrieving and Reasoning about XML-Based Mathematical Knowledge*. In: *Proceedings of International Conference on Electronic Information and Communication in Mathematics (EIC) - How to find and post your mathematics in the web*, Peking, China, Oktober 2002. Accepted for publication.

[Inc89] INC., SUN MICROSYSTEMS: *NFS: Network File System Protocol Specification – RFC 1094*. Network Working Group, http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc1094.html, März 1989.

[ISO86] ISO, http://www.iso.ch/cate/d16387.html: *Standard Generalized Markup Language (SGML)*, 1986.

[Jak02] THE JAKARTA APACHE PROJECT, http://jakarta.apache.org/lucene/docs/index.html: *Jakarta Lucene – Homepage*, Mai 2002.

[Jav02] JAVA COMMUNITY PROCESS, http://java.sun.com/jcp/: *Java Rule Engine API Specification – Version 1.3*, Juli 2002. JSP-94.

[Jen00] JENNINGS, N.R.: *On agent-based software engineering*. Artificial Intelligence, 177(2):277–296, 2000.

[Jen01] JENNINGS, N.R.: *An Agent-Based Approach For Building Complex Software Systems - Why Agent-Oriented Approaches Are Well Suited For Developing Complex, Distributed Systems*. Communications of the ACM, 44(4):35–41, 2001.

[JF88] JOHNSON, R. und B. FOOTE: *Designing Reusable Classes*. Object-Oriented Programming, 1(2):22–35, 1988.

[JK01] JOSEPH, S. und T. KAWAMURA: *Why Autonomy Makes the Agent*. In: LIU, J., Y.Y. TANG, N. ZHONG und P. WANG (Herausgeber): *Series in Machine Perception and Artificial Intelligence - Agent Engineering*, Band 43. World Scientific, Juni 2001.

[JLSU87] JOYCE, J., G. LOMOW, K. SLIND und B. UNGER: *Monitoring Distributed Systems*. ACM Transactions on Computer Systems, 5(2):121–150, Februar 1987.

[Joh97] JOHNSON, R.E.: *Frameworks=(Components+Patterns): How frameworks compare to other object-oriented reuse techniques* . Communications of the ACM, 40(10):39–42, Oktober 1997.

[JR91] JOHNSON, R. und V. RUSSO: *Reusing Object-Oriented Design*. Technischer Bericht 91-1996, University of Illinois, 1991.

[Kal02] KALETTA, D.: *Die digitale Bibliothek und das semantische Netz - Ist die digitale Bibliothek mehr/anders als ein semantisches Netz?* Berliner Bibliothekswissenschaftliches Kolloquium, Januar 2002. Verfügbar unter http://www.ib.hu-berlin.de/bbk/kaletta/kaletta.pdf.

[Kar00] KARP, A.: *E-speak Explained*. Technischer Bericht HPL-2000-101 20000807, Hewlett-Packard Labs Technical Reports, 2000.

[KL01] KLEINBERG, J. und S. LAWRENCE: *The Structure of the Web*. Science, 294:1849ff, November 2001.

[Kle99] KLEINBERG, J.M.: *Authoritative sources in a hyperlinked environment*. Journal of the ACM, 46(5):604–632, 1999.

[Kle01] KLENSIN, J.: *Simple Mail Transfer Protocol – RFC 2821*. Network Working Group, http://rfc.sunsite.dk/rfc/rfc2821.html, April 2001.

[KR97] KHARE, R. und A. RIFKIN: *XML: A Door to Automated Web Applications*. IEEE Internet Computing, 1(4):78–87, 1997.

[Law00] LAWRENCE, S.: *Context in Web Search*. IEEE Data Engineering Bulletin, 23(3):25–32, 2000.

[Lev94] LEVY, D.M.: *Fixed or Fluid? Document Stalability and New Media*. In: *Proceedings of ACM European Conference on Hypertext (ECHT'94)*, Seiten 24–31, Edinburgh, Scotland, Sep 1994. ACM Press.

[Lia99] LIANG, S.: *Java Native Interface: Programmer's Guide and Specification*. The Java Series. Addison Wesley, Mai 1999.

[Lic60] LICKLIDER, J.C.R.: *Man-Computer Symbiosis*. Transaction Human Factoring Engineering, 1(1):4–11, März 1960.

[Lie86] LIEBERMAN, H.: *Using Prototypical Objects to Implement Shared Behavior in Object-Oriented Systems*. SIGPLAN Notices, 21(11):214–223, 1986.

[LL00] LEVESQUE, H.J. und G. LAKEMEYER: *The logic of knowledge bases*. MIT Press, Cambridge, Massachusetts, 2000.

[LM95] LEVY, D.M. und C.C. MARSHALL: *Going Digital: A Look at Assumptions Underlying Digital Libraries*. Communications of the ACM, 38(4):77–84, April 1995.

[LMZ02] LEONARDI, L., M. MAMEI und F. ZAMBONELLI: *Co-Fields: Towards a Unifying Model for Swarm Intelligence*. Technischer Bericht DISMI-UNIMO-3-2002, University of Modena and Reggio Emilia, 2002.

[Mar90] MARSHALL, C.C.: *A multi-tiered approach to hypertext integration: Negotiating standards for a heterogeneous application environment*. In: MOLINE, J., D. BENIGNI und J. BARONAS (Herausgeber): *Proceedings of the Hypertext Standardization Workshop*, Seiten 167–177, Gaithersburg, Md, USA, Januar 1990.

[Mar98] MARSHALL, C.C.: *Making metadata: a study of metadata creation for a mixed physical-digital collection*. In: *Proceedings of the third ACM Conference on Digital libraries*, Seiten 162–171. ACM Press, 1998.

[Mey97] MEYER, B.: *Object-Oriented Software Construction*, Band Second Edition. Prentice Hall, 1997.

[Mey98] MEYER, B.: *The Future of Object Technology*. IEEE Computer, 31(1):140–141, Januar 1998.

[MG01] MCCRAY, A.T. und M.E. GALLAGHER: *Principles for digital library development*. Communications of the ACM, 44(5):48–54, Mai 2001.

[MH99] MATENA, V. und M. HAPNER: *Enterprise JavaBeans Specification*. Sun Microsystems Inc., 1999. Revision 1.1.

[Mic02] MICROSOFT CORP., http://www.microsoft.com/net/: *.net – Homepage*, Mai 2002.

[MM97] MOWBRAY, T.J. und R.C. MALVEAU: *CORBA Design Patterns*. John Wiley and Sons, 1997.

[Mor81] MORAN, T.P.: *The Command Language Grammar. A Representation for the User Interface of Interactive Computer Systems*. International Journal of Man-Machine Studies, 15(1):3–50, 1981.

[MS01] MÜLLER, S. und R.-D. SCHIMKAT: *A General, Web-Enabled Model Retrieval Approach.* In: WANG, Y., S. PATEL und R.H. JOHNSTON (Herausgeber): *Proceedings of the 7th International Conference on Object-Oriented Information Systems (OOIS 2001)*, Seiten 487–497, Calgary, Canada, August 2001. Springer.

[MSM02] MÜLLER, S., R.-D. SCHIMKAT und R. MÜLLER: *Query Language for Structural Retrieval of Deep Web Information*. In: *2nd International Workshop on Web Dynamics (WebDyn 2002)*, Honolulu, Hawaii, USA, Mai 2002.

[Mül01] MÜLLER, S.: *FIRESTORM – FIrst a REtrieval SysTem for Operations Research Models*, 2001. Verfügbar unter http://firestorm.informatik.uni-tuebingen.de.

[MyS02a] MYSQL: *MySQL – the world's most popular Open Source Database, Homepage http://www.mysql.com*, September 2002.

[MyS02b] MYSQL AB, http://www.mysql.com: *MySQL – Open Source Database*, 2002.

[Nes01] NESTEL, F.: *Smider – Automatic resource compiling on the Web*. http://frank.spieleck.de/metasuch, Februar 2001.

[Neu94] NEUMAN, B.C.: *Scale in Distributed Systems*. In: CASAVANT, T.L. und M. SINGHAL (Herausgeber): *Readings in Distributed Computing Systems*, Seiten 463–489, Los Alamitos, CA, USA, 1994. IEEE Computer Society.

[NN99] NWANA, H. S. und D. T. NDUMU: *A Perspective on Software Agents Research*. The Knowledge Engineering Review, 14(2):1–18, 1999.

[Nor99] NORMAN, D.A.: *The Invisible Computer: Why Good Products Can Fail, the Personal Computer is So Complex, and Information Appliances are the Solution*. MIT Press, 1999.

[Now98] NOWACK, P.: *Frameworks - Representations and Perspectives*. In: *Workshop on Language Support for Design Patterns and Object-Oriented Frameworks, (ECOOP '97)*, 1998.

[NS01] NUSSER, G. und R.-D. SCHIMKAT: *Rapid Application Development of Middleware Components by Using XML*. In: *Proceedings of the 12th IEEE International Workshop on Rapid System Prototyping (RSP 2001)*, Seiten 116–121, Monterey CA, USA, June 2001. IEEE Computer Society Press.

[Nwa95] NWANA, H.S.: *Software Agents: An Overview*. Knowledge Engineering Review, 11(2):205–244, 1995.

[Obj00a] OBJECT MANAGEMENT GOUP (OMG), http://www.omg.org/: *The Common Object Request Broker: Architecture and Specification*, August 2000. Revision 2.4.1.

[Obj00b] OBJECT MANAGEMENT GROUP: *Unified Modeling Language - UML*, 2000. Available at http://www.omg.org/technology/documents/formal/unified_modeling_language.htm.

[OZ99] OMICINI, A. und F. ZAMBONELLI: *Coordination for Internet Application Development*. Journal of Autonomous Agents and Multi-Agent Systems, 1(3):251–269, September 1999.

[Pap93] PAPAZOGLOU, M.P.: *On the duality of distributed database and distributed AI systems*. In: *Proceedings of the 2nd International Conference on Information and Knowledge Management*, Seiten 1–10, November 1993.

[Par72] PARNAS, D.L.: *On the criteria to be used in decomposing systems into modules*. Communications of the ACM, 15(12):1053–1058, Dezember 1972.

[Par97] PARUNAK, H.: *Go to the Ant: Engineering Principles from Natural Multi-Agent Systems*. Annals of Operations Research - Special Issue on Artificial Intelligence and Management Science, 75:69–101, 1997.

[Par98] PARUNAK, H.: *A Dynamical Systems Perspective on Agent-Based Going Concerns*. In: *Proceedings of International Workshop on Multi-Agent Systems*, Dedham, MA, USA, Oktober 1998.

[Par01] PARUNAK, H.: *Entropy and Self-Organization in Multi Agent Systems*. In: *Proceedings of Proceedings of Autonomous Agents*, Montreal, Canada, Mai 2001.

[PP95] PLAICE, J. und J. PACUET: *Introduction to Intensional Programming*. In: *Intensional Programming I*, Seiten 1–14, Singapore, 1995. World-Scientific.

[Pre94] PREE, W.: *Design Patterns for Object-Oriented Software Development*. Addison-Wesley, Reading, Massachusetts, 1994.

[RCF00] ROBIE, J., D. CHAMBERLIN und D. FLORESCU: *Quilt: an XML Query Language*. Available at http://www.almaden.ibm.com/cs/people/chamberlin/quilt_euro.html, März 2000.

[Ree00] REESE, G.: *Database Programming with JDBC and Java*. O'Reilly and Associates, November 2000.

[RHJ99] RAGGET, D., A. LE HORS und I. JACOBS: *Hypertext Markup Language (HTML 4.01)*. World Wide Web Consortium (W3C), http://-www.w3.org/TR/html4, Dezember 1999.

[RJM02] ROMAN, G.-C., C. JULIEN und A.L. MURPHY: *A Declarative Approach to Agent-Centered Context-Aware Computing in Ad Hoc Wireless Environments*. In: *1st International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS)*, Orlando, USA, Mai 2002.

[RW91] RISCH, T. und G. WIEDERHOLD: *Building Adaptive Applications using Active Mediators*. In: KARAGIANNIS, D. (Herausgeber): *Database and Expert Systems Applications (DEXA'91)*, Seiten 508–513, Berlin, Germany, 1991. Springer-Verlag.

[SAW94] SCHILIT, B., N. ADAMS und R. WANT: *Context-Aware Computing Applications*. In: *IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, USA, 1994.

[Say01] SAYERS, C.: *XML Document Agents*. Technischer Bericht HPL-2001-2888, Hewlett-Packard Labs Technical Reports, November 2001.

[SBS+00] SCHIMKAT, R.-D., W. BLOCHINGER, C. SINZ, M. FRIEDRICH und W. KÜCHLIN: *A Service-Based Agent Framework for Distributed Symbolic Computation*. In: BUBAK, M., R. WILLIAMS, H. AFSARMANESH und B. HERTZBERGER (Herausgeber): *Proceedings of the 8th International Conference on High Performance Computing and Networking Europe (HPCN'00)*, Band 1823 der Reihe *Lecture Notes in Computer Science*, Seiten 644–656, Amsterdam, Netherlands, Mai 2000. Springer.

[Sch00] SCHROEDER, S.: *Entwurf und Implementierung eines objektorientierten Frameworks für verteiltes Systemmangement*. Diplomarbeit, Universität Tübingen, Januar 2000.

[SD00] SCHMIDT-DANNERT, M.: *An Object-Oriented Framework and Test Management Infrastructure based on Fine-Grained XML-Documents*. Diplomarbeit, Universität Tübingen, Mai 2000.

[SEM01] *The Semantic Web Community Portal*. http://www.semanticweb.org/index.html, 2001.

[SFK01a] SCHIMKAT, R.-D., M. FRIEDRICH und W. KÜCHLIN: *Deploying Distributed State Information in Mobile Agents Systems*. In: BATINI, C., F. GIUNCHIGLIA, P. GIORGINI und M. MECELLA (Herausgeber): *Proceedings 9th International Conference on Cooperative Information Systems (CoopIS 2001)*, Band 2172 der Reihe *Lecture Notes in Computer Science*, Seiten 80–94, Trento, Italy, September 2001. Springer.

[SFK01b] SCHIMKAT, R.-D., M. FRIEDRICH und W. KÜCHLIN: *On Maintaining Code Mobility*. In: *Workshop on Software Engineering and Mobility, co-located with International Conference on Software Engineering (ICSE 2001)*, Toronto, Canada, Mai 2001.

[SHKK00] SCHIMKAT, R.-D., M. HÄUSSER, W. KÜCHLIN und R. KRAUTTER: *Web Application Middleware to Support XML-Based Monitoring in Distributed Systems*. In: DEBNATH, N. (Herausgeber): *Proceedings of 13th International Conference on Computer and Applications in Industry and Engineering (CAINE 2000)*, Seiten 203–207, Hawaii, USA, November 2000. International Society for Computers and Their Applications.

[Shn89] SHNEIDERMAN, B.: *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley, Reading, MA, 1989.

[SK01] SCHIMKAT, R.-D. und W. KÜCHLIN: *Aspekte skalierbarer Infrastrukturen für die Integration verteilter Anwendungsarchitekturen*. In: KALETTA, D. und E. EDELHOFF (Herausgeber): *Proceedings 14. GI-Tagung Anwendungs- und System-Management im Zeichen von Multimedia und E-Business*, Tübingen, Germany, April 2001. ZDV.

[SK02] SCHIMKAT, R.-D. und W. KÜCHLIN: *Living Documents - Micro Servers for Documents*. In: CHAUDHRI, A.B., R. UNLAND, C. DJERABA und W. LINDNER (Herausgeber): *XML-Based Data Management and Multimedia Engineering – EDBT 2002 Workshops*, Band 2490 der Reihe *Lecture Notes in Computer Science*, Prague, Czech Republic, März 2002. Springer. Revised Papers.

[SKK99] SCHIMKAT, R.-D., W. KÜCHLIN und R. KRAUTTER: *An Object-Oriented Framework for Rapid Client-side Integration of Information Management Systems*. South African Computer Journal, (24):244–248, November 1999.

[SKN02] SCHIMKAT, R.-D., W. KÜCHLIN und F. NESTEL: *Living Hypertext - Web Retrieval Techniques for Traditional Database-Centric Information*. In: UNGER, H., T. BÖHME und A. MIKLER (Herausgeber):

Proceedings of Second International Workshop on Innovative Internet Computing Systems (I2CS), Band 2346 der Reihe *Lecture Notes in Computer Science*, Seiten 1–14, Kühlungsborn, Germany, Juni 2002. Springer.

[SL90] SHETH, A.P. und J.A. LARSON: *Federated database systems for managing distributed, heterogeneous, and autonomous databases*. ACM Computing Surveys, 22(3):183–236, 1990.

[SMKK00] SCHIMKAT, R.-D., S. MÜLLER, W. KÜCHLIN und R. KRAUTTER: *A Lightweight, Message-Oriented Application Server for the WWW*. In: CARROLL, J., E. DAMIANI, H. HADDAD und D. OPPENHEIM (Herausgeber): *Proceedings of the 15th ACM Symposium on Applied Computing (SAC 2000)*, Seiten 934–941, Como, Italy, März 2000. ACM Press.

[SMP00] SCHRAEFEL, M.C., B. MANCILLA und J. PLAICE: *Intensional Hypertext*. In: GERGATSOULIS, M. und P. RONDOGIANNIS (Herausgeber): *Intensional Programming II*, Seiten 40–54, Singapore, 2000. World-Scientific.

[SN92] SULLIVAN, K. J. und D. NOTKIN: *Reconciling environment integration and software evolution*. ACM Transactions on Software Engineering and Methodology, 1(3):229–268, Oktober 1992.

[SNB00] SCHIMKAT, R.-D., G. NUSSER und D. BÜHLER: *Scalability and Interoperability in Service-Centric Architectures for the Web*. In: TJOA, A M., R.R. WAGNER und A. AL-ZOBAIDIE (Herausgeber): *Proceedings of the 11th International Workshop on Database and Expert Systems Applications (DEXA 2000)*, Seiten 51–57, London, England, September 2000. IEEE Computer Society.

[Sno88] SNODGRASS, R.T.: *Relational Approach to Monitoring Complex Systems*. ACM Transactions on Computer Systems, 6(2):157–196, 1988.

[SS86] STERLING, L. und E. SHAPIRO: *The Art of Prolog*. The MIT Press, Cambridge, MA, 1986.

[SS01] SCHANTZ, R.E. und D.C. SCHMIDT: *Middleware for Distributed Systems: Evolving the Common Structure for Network-centric Applications*. Encyclopedia of Software Engineering. Wiley and Sons, Dec 2001.

[SSDKK00] SCHIMKAT, R.-D., M. SCHMIDT-DANNERT, W. KÜCHLIN und R. KRAUTTER: *Tempto - An Object-Oriented Test Framework*

and Test Management Infrastructure Based On Fine-Grained XML-Documents. In: *NetObjectDays 2000 - Object-Oriented Software Systems*, Seiten 274–287, Erfurt, Germany, Oktober 2000. Net.ObjectDays-Forum.

[Str88] STROUSTRUP, B.: *What is Object-Oriented Programming?* IEEE Software, 5(3):10–20, 1988.

[Str97] STROUSTRUP, BJARNE: *The C++ Programming Language*. Addison-Wesley, Reading, Massachusetts, Third Auflage, 1997.

[Sun] SUN MICROSYSTEMS INC., http://www.sun.com/jini/: *JINI Network Technology*.

[Sun01] SUN MICROSYSTEMS INC., http://java.sun.com/products/jdk/1.3/docs/guide/rmi/spec/rmiTOC.doc.html: *Remote Method Invocation Specification*, 2001.

[Sun02a] SUN MICROSYSTEMS INC., http://www.java.sun.com/j2ee: *J2EE - Java EnterpriseBeans*, 2002.

[Sun02b] SUN MICROSYSTEMS INC., http://java.sun.com/products/jdbc/: *JDBC - Java Database Connectivity*, 2002.

[SW88] SMITH, J. und S. WEISS: *An Overview of Hypertext*. Communications of the ACM, Seiten 816–819, Juli 1988.

[TBMM01] THOMPSON, H.S., D. BEECH, M. MALONEY und N. MENDELSOHN: *XML Schema Part 1: Structures – W3C Recommendation May 2001*. World Wide Web Consortium (W3C), http://www.w3.org/TR/xmlschema-1/, Mai 2001.

[Ten00] TENNENHOUSE, D.: *Proactive Computing*. Communications of the ACM, 43(5):43–50, Mai 2000.

[THE02] *THEOI Project – A Guide to Greek Gods, Spirits and Monsters*. http://www.theoi.com/index.htm, 2002.

[TL01] THAI, T. und H. LAM: *.NET Framework Essentials*. O'Reilly, 2001.

[Use02] USERLAND SOFTWARE INC., http://www.xml-rpc.com: *XML-RPC*, 2002.

[Vin02] VINOSKI, S.: *Where is Middleware?* IEEE Internet Computing, 6(2):83–85, April 2002.

[Wad99] WADGE, W.W.: *Intensional Markup Language*. In: KROPF, P., G. BABIN, J. PLAICE und H. UNGER (Herausgeber): *3rd International Workshop on Distributed COmmunities on the Web*, Band

1830 der Reihe *LNCS*, Seiten 82–89, Quebec City, Canada, 1999. Springer.

[Wal99] WALDO, J.: *The Jini architecture for network-centric computing*. Communications of the ACM, 44(3), Juli 1999.

[Wan93] WANDMACHER, J.: *Software-Ergonomie*. de Gruyter, Berlin, New York, 1993.

[Weg97] WEGNER, P.: *Why Interaction is more Powerful than Algorithms*. Communications of the ACM, 40(5):80–91, 1997.

[Wei91] WEISER, M.: *The computer for the 21st century*. Scientific American, 265(3):94–104, September 1991.

[Wei99] WEISS, G.: *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, Cambridge, Massachusetts, 1999.

[Wei00] WEIKUM, G.: *The Web in 2010: Challenges and Opportunities for Database Research*. In: WILHELM, R. (Herausgeber): *Conference at the Occasion of Dagstuhl's 10th Anniversary: Informatics - 10 Years Back, 10 Years Ahead*, Band 2000, Seiten 1–23, Saarbrücken, Germany, August 2000. Springer LNCS.

[WG97] WIEDERHOLD, G. und M. GENESERETH: *The Conceptual Basis for Mediation Services*. IEEE Intelligent Systems, 12(5):38–47, September 1997.

[Wie92a] WIEDERHOLD, G.: *Mediators in the Architecture of Future Information Systems*. IEEE Computer, 25(3):38–49, März 1992.

[Wie92b] WIEDERHOLD, G.: *The Roles of Artificial Intelligence in Information Systems*. Journal of Intelligent Information Systems, 11(1):35–56, 1992.

[Wie95] WIEDERHOLD, G.: *Digital libraries, value, and productivity*. Communications of the ACM, 38(4):85–96, April 1995.

[Wie01] WIEDERHOLD, G:: *Obtaining Precision When Integrating Information*. In: *Proceedings of the International Conference on Enterprise Information Systems (ICEIS 2001)*, Seiten 9–22, September 2001.

[WJ95] WOOLDRIDGE, M. und N. JENNINGS: *Intelligent Agents: Theory and Practice*. The Knowledge Engineering Review, 10(2):115–152, 1995.

[Woo97] WOOLDRIDGE, M.: *Agent-Based Software Engineering*. Proceedings of IEE Software Engineering, 144(1):26–37, 1997.

[Wor00] WORLD WIDE WEB CONSORTIUM (W3C), http://www.w3.org/Protocols: *Hypertext Transfer Protocol (HTTP)*, November 2000.

[WRD+90] WIEDERHOLD, G., T. RISCH, P. RATHMANN L. DEMICHIEL, S. CHAUDRY, B.S. LEE, K.H. LAW, T.BARSALOU und D. QUASS: *A Mediator Architecture for Abstract Data Access*. Technischer Bericht STAN-CS-90-1301, Stanford Computer Science Department, 1990.

[WWWK94] WALDO, J., G. WYANT, A. WOLLRATH und S. KENDALL: *A Note on Distributed Computing*. Technical Report TR-94-29, SUN Microsystems Laboratories, Nov 1994.

[YC79] YOURDON, E. und L.L. CONSTANTINE: *Structured Design: Fundamentals of a dicipline of computer program and systems design*. Prentice Hall International, Englewood Cliffs, NJ, 1979.

[Zam01] ZAMBONELLI, F.: *From Design to Intention: Signs of a Revolution*. In: OMICINI, A. und M. VIROLI (Herausgeber): *Proceedings of 2nd Italian Workshop on Objects and Agents - (WOA2001)*, Modena, Italy, September 2001. Pitagora Editrice Bologna.

Lebens- und Bildungsgang

Seit Oktober 2002	**Software-Architekt bei der Fa. *Feilmeier & Junker (FJA) AG*, Zürich, Schweiz.**
Juni 1998 bis August 2002	**Wissenschaftlicher Mitarbeiter an der Universität Tübingen.** Wilhelm-Schickard-Institut für Informatik, Arbeitsbereich *Symbolisches Rechnen*, Prof. Dr. W. Küchlin.
Oktober 1990 bis April 1998	**Studium der Informatik mit Nebenfach Betriebswirtschaftslehre an der Eberhard-Karls-Universität Tübingen.**
Oktober 1994 bis September 1995	**Auslandsstudium an der University of Washington, Seattle, USA.**
1982 bis Mai 1990	**Allgemeine Hochschulreife am Nellenburg-Gymnasium Stockach.**