Flexible Einbindung von Webapplikationen in Webinformationssysteme

Diplomarbeit am Fachbereich Informatik,
Arbeitsbereich Softwaretechnik der
Universität Hamburg

Von Wolfgang Riese
Matr.-Nr.: 4627477

2003-01-16

Erstbetreuerin:	Prof. Dr. Christiane Floyd
Zweitbetreuer:	Prof. Dr. Horst Oberquelle

Erklärung:
Ich versichere hiermit, diese Arbeit selbständig und unter ausschließlicher Zuhilfenahme der in der Arbeit aufgeführten Hilfsmittel erstellt zu haben.

Hamburg, den 16.01.2003

Diplomarbeit am

Fachbereich Informatik
Universität Hamburg
Vogt-Kölln-Straße 30
22527 Hamburg

Vorgelegt von

Wolfgang Riese
Framheinstr. 26
22083 Hamburg

Betreut durch

Prof. Dr. Christiane Floyd (Erstbetreuerin)
Universität Hamburg
Fachbereich Informatik
Arbeitsbereich Softwaretechnik

Prof. Dr. Horst Oberquelle (Zweitbetreuer)
Universität Hamburg
Fachbereich Informatik
Arbeitsbereich Angewandte und Sozialorientierte Informatik

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Diese Arbeit beschäftigt sich mit softwaretechnischen Fragen im Zusammenhang mit dem Internet und im besonderen dem darauf basierenden World Wide Web (WWW oder einfach Web).

Das Internet, welches aus dem bereits Ende 1969 in Betrieb gegangenen ARPANET entstanden ist, ist von Anfang an ein Medium für kooperative Arbeit und Informationsaustausch gewesen (siehe [ISOC 2000]). Anfangs noch auf den Dokumentenaustausch durch FTP und seit 1972 E-Mail beschränkt, entwickelte sich das Internet durch die Einführung des Webs (siehe. [Cailliau 1995]) zu Beginn der 90er Jahre des vorigen Jahrhunderts zu dem „Prototypen einer globalen Informationsinfrastruktur“ (siehe [ISOC 2000]). Der Einfluss des Webs beschränkt sich dabei nicht nur auf die Welt der Computerkommunikation, sondern durch die immer stärkere Verbreitung von E-Kommerz und Informationsbeschaffung über das Web auch auf die gesamte Gesellschaft. Das Internet und im besonderen das darauf basierende Web bieten die Möglichkeit, weltweit auf einfachste Weise Informationen zu verbreiten/erlangen und unabhängig von der geographischen Position zusammen zu arbeiten (siehe [ISOC 2000]).

Um den Benutzern einen einfachen Zugang zu den Möglichkeiten des Webs zu bieten, entwickelten sich im Laufe der Zeit spezielle Einstiegsseiten und Suchmaschinen. Diese Webinformationssysteme wurden im Laufe der Zeit immer umfangreicher und entwickelten sich zu so genannten Portalen. Die Verwendung des Begriffs Portal für diese Art von Webinformationssysteme bürgerte sich um 1998 ein (siehe [SchumacherSchwickert 1999, S. 4]).

Das Thema meiner Arbeit entstand während der Entwicklung eines Prototypen eines Lebenslageninformationssystems (siehe [BleekFloyd 2000]), in der Scriptsprache PHP, am Arbeitsbereich SWT im Jahr 2000. Bei der Erstellung dieses Prototypen kam die Frage auf, wie zu einer Lebenslage aufgefundene kleine Applikationen von Drittanbietern in dem umgebenden Lebenslageninformationssystem eingebunden werden könnten. Zugleich ergab sich das Problem, in wie weit ein Mechanismus gefunden werden könnte, der dies so universell wie möglich lösen würde. Dadurch würde man keine Insellösung erhalten, sondern einen Mechanismus, der es den Applikationsanbietern ermöglichen würde, ihre Applikationen bei möglichst vielen verschiedenen Informationssystemen gleichzeitig einsetzen zu können. Für die Entwicklung des Prototypen wurde dies damals zunächst vernachlässigt.

Im Jahr 2001 griffen W.-G. Bleek und ich das Thema wieder auf, als er mich bei der Themenfindung für meine Diplomarbeit betreute. In diesem Umfeld ermöglichte er es mir im Rahmen eines Beratungsvertrages der Universität mit der Firma SNetLine, vorübergehend bei dieser zu arbeiten. In diesem Projekt sollte ich dort ursprünglich an der Entwicklung eines Lebenslageninformationssystems im hamburg.de Kontext beteiligt sein. Leider ist es dazu auf Grund von Restrukturierungsmaßnahmen bei der Firma SNetLine und dem auslaufenden Beratervertrag der Universität mit SNetLine nicht mehr gekommen.

Dies ändert nichts an der Wichtigkeit der Fragestellung nach der flexiblen Einbindung von Webapplikationen in Webinformationssysteme. Mit dieser Thematik beschäftigt sich unter anderem zur Zeit ein Java Community Process (vgl. [JCP 168]) unter dem Begriff „Portlets“.

Ähnliche Ziele durch Benutzung von Web Services werden zur Zeit durch die „.net“ Strategie von Microsoft (vgl. [Microsoft 2002a]) und die „Sun One“ Strategie von Sun (vgl. [Sun 2002])

sowie Standardisierungsbemühungen im Bereich der Web Services durch das World Wide Web Consortiums (vgl. [W3C 2002]) auf diesem Gebiet verfolgt.

1.2 Das Ziel der Arbeit

In meiner Diplomarbeit beschäftige ich mich mit der Softwareentwicklung von Webapplikationen, die in verschiedenen Kontexten verwendet werden können. Diese Webapplikationen sollen so konstruiert sein, dass sie sowohl in Webinformationssystemen (Portalen) als auch als eigenständige Webapplikation eingesetzt werden können (siehe Abbildung 1). Das bedeutet, in Webinformationssystemen müssen solche Webapplikationen möglichst frei konfigurier- und einsetzbar sein, nicht an eine Bildschirmposition gebunden sein und mit einem möglichst minimalen Benutzerinterface auskommen, und Eingaben in einer Webapplikation im Webinformationssystem dürfen nicht zu Veränderungen in parallel laufenden Webapplikationen führen. Im Gegensatz dazu wird eine eigenständige Webapplikation, die nicht innerhalb eines Webinformationssystems abläuft, ein komplexeres Benutzerinterface anbieten können. Dies werde ich am Beispiel kleinerer Webapplikationen untersuchen.

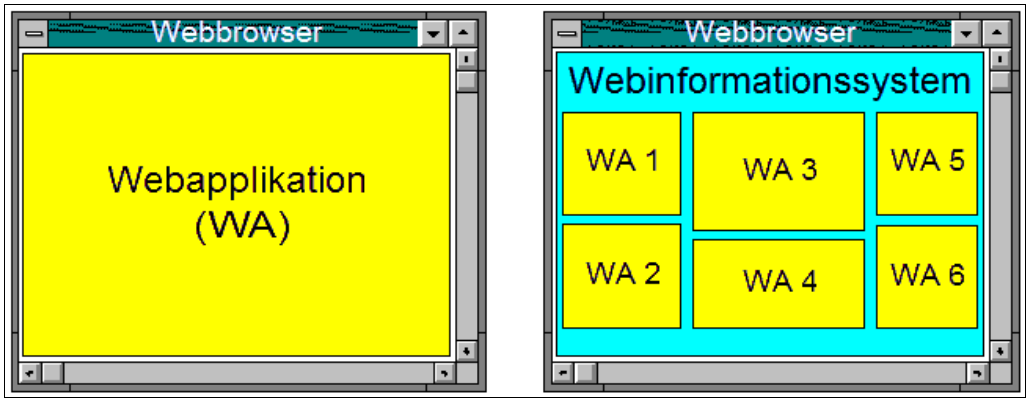

Abbildung 1 - Webapplikationen in verschiedenen Kontexten

Es geht darum, heraus zu finden, welche Eigenschaften eine Webapplikation erfüllen muss, damit sie ohne größere Probleme in möglichst vielen Kontexten eingesetzt werden kann.

Wie konstruiert man also eine Webapplikation, die sowohl als eigenständige Webapplikation als auch in ein beliebiges Webinformationssystem eingebundene Webapplikation funktioniert? Zusätzlich sollen dafür keine unnötigen Veränderungen am Quelltext vorzunehmen sein. In diesem Zusammenhang ist es notwendig, ein Modell zu finden, in dem auf das Zusammenspiel von Webserver, Webinformationsystem und Applikationsserver eingegangen wird. Es ist festzustellen, welchen Einfluss diese Kombination auf die Konstruktion einer Webapplikation hat. Ich werde dies beispielhaft an der Kombination Apache Webserver und Tomcat als Applikationsserver für Java Anwendungen untersuchen.

Eine solche Webapplikation muss, sowohl in ihren Dialogstrukturen als auch in ihrem Kontrollfluss, damit umgehen können, Daten direkt vom Webbrowser eines Benutzers, oder indirekt durch ein Webinformationssystem zu erhalten. Sie muss in der Lage sein, ihren Kontrollfluss und ihre URLs dynamisch der jeweiligen Situation anzupassen.

Eine weitere Anforderung ist, dass die Webapplikation in der Lage sein muss, sich bei der Ausgabe ihrer Daten auf verschiedene Situationen einzustellen. So sollte die Ausgabe ihrer Daten und ihrer Dialogstrukturen nicht nur funktionieren, wenn sie als einzige Applikation auf ein Ausgabefenster zugreift, sondern auch dann noch, wenn sie sich die Ausgabefläche mit anderen Webanwendungen teilen muss und die Kommunikation über ein Webinformationssystem erfolgt und nicht direkt über den Applikationsserver.

Es ist daher zu überlegen, wie die Aufteilung in Interaktionskomponente (IAK) und Funktionskomponente (FK) (siehe [Züllighoven 1998, S. 236ff]) genau vorzunehmen ist, bzw. ob diese Trennung ausreichend ist (vgl. [Lipert et al. 2001]) und ob eine Webapplikation selbst für die verschiedenen Kontexte jeweils eine IAK zur Verfügung stellt, oder ob man evtl. in dieser Situation die genaue Darstellung dem umgebenden Webinformationssystem überlässt und nur grobe Vorgaben zum Layout und die Formularfeldnamen mit übermittelt (z.B. mit XML).

Im Kontext dieser Arbeit untersuche ich, in wie weit sich Webapplikationen, die für den Einsatz in Webinformationssystemen gedacht sind, in unterschiedlichen Kategorien einordnen und unterscheiden lassen.

Das Ziel meiner Arbeit wird es sein, herauszuarbeiten, wie die Darstellung der Inhalte, die Dialoge und der Kontrollfluss innerhalb einer solchen Webapplikation anzulegen sind, damit sie die oben aufgestellten Forderungen erfüllen.

1.3 Gliederung der Arbeit

Im zweiten Kapitel werde ich die für diese Arbeit wichtigen Grundbegriffe „Webinformationssystem“, „Webapplikation“ und „Service“ klären. Ich werde den Zusammenhang von Webinformationssystemen und Portalen darlegen und anhand der Beispiele „hamburg.de“ und „My.Netscape“ den derzeitigen Stand des Einsatzes von Webapplikationen in Webinformationssystemen und Portalen zeigen.

Auf die Motivation zur flexiblen Einbindung von Webapplikationen in Webinformationssysteme gehe ich im dritten Kapitel ein. Dazu werde ich zunächst den Versuch einer Klassifikation von Webapplikationen vornehmen und daran anschließend auf grundlegende Probleme im Design von Webapplikationen eingehen.

Dann werde ich die Interessen und Probleme der Betreiber von Webinformationssystemen und Webapplikationen in diesem Kontext untersuchen.

Zum Abschluss dieses Kapitels werde ich weitere technische Probleme untersuchen, die sich aus den verschiedenen zur Verfügung stehenden Technologien ergeben.

Bei der Untersuchung existierender Ansätze zur Einbindung von Webapplikationen nimmt die im vierten Kapitel untersuchte Web Services Architektur eine wichtige Position ein. Diese spielt unter anderem in Microsofts „.net“ (vgl. [Microsoft 2002a]) und Suns „Sun One“ (vgl. [Sun 2002]) Strategien eine zentrale Rolle.

Ich werde das Web Services Architekturmodell vorstellen und einen Schwerpunkt auf das Kommunikationsmodell desselben legen. Abschließed werde ich darlegen, in welchem Zusammenhang mein Architekturmodell zu den der Web Services steht.

Im fünften Kapitel erfolgt eine Einordnung der Architektur der Web Services und meines Architekturmodells im Vergleich zu dem am Arbeitsbereich im Rahmen des WAM Ansatzes (vgl.

[Züllighoven 1998]) erarbeiteten Konzeptes des Fachlichen (Domain) Services (vgl. [OttoSchuler 2000] und [Lipert et al. 2001]).

Dann werde ich im sechsten Kapitel auf mein Architekturmodell eingehen und zunächst die Anforderungen an dieses darlegen. Des Weiteren werde ich auf die von mir gewählten Kommunikationstechnologien und Implementationssprache eingehen und die Funktionalität des implementierten WebAp Frameworks vorstellen.

Im siebten Kapitel werde ich das WebAp Framework selbst aus softwaretechnischer Sicht erläutern. Ich werde die wichtigsten Klassen und Schnittstellen Definitionen darlegen, verwendete Entwurfsmuster (vgl. [Gamma et al. 1996]) erwähnen und Diagramme nach der Unified Modeling Language Specification 1.4 (vgl. [UML1.4 2001]) verwenden. Abschließend werde ich das von mir implementierte Beispiel Webinformationssystem und dessen Webapplikationen vorstellen.

Im letzten Kapitel werde ich alle vorhergehenden Kapitel zusammenfassen sowie diskutieren, welche Themen in diesem Zusammenhang evtl. noch von Interesse sein könnten, um sie zukünftig noch genauer zu untersuchen. Dies gilt auch für mögliche Erweiterungen des WebAp Frameworks.

1.4 Danksagung

Mein Dank geht an alle diejenigen, die mir bei der Erstellung dieser Arbeit mit Rat und Tat zur Seite standen.

Bedanken möchten ich mich zunächst bei den Betreuern der Schriftlichen Arbeit, Frau Prof. Dr. Christiane Floyd für die Erstbetreuung und Herrn Prof. Dr. Horst Oberquelle für die Zweitbetreuung.

Mein besonderer Dank für die direkte Betreuung am Arbeitsbereich SWT gilt Wolf-Gideon Bleek, der mir immer für Diskussionen, Literaturhinweise und Feedback zur Verfügung gestanden hat.

Bei meiner Mutter bedanke ich mich für das Korrekturlesen dieser Arbeit.

2 Webinformationssysteme – Portale

In diesem Kapitel werde ich anhand von verschiedenen Beispielen auf den derzeitigen Stand des Einsatzes von Webapplikationen in Webinformationssystemen (Portalen) eingehen. Dazu werde ich zunächst die in diesem Zusammenhang wichtigen Begriffe Webapplikation und Service klären. Danach erläutere ich, was ich unter Webinformationssystemen und Portalen verstehe. Des Weiteren werde ich untersuchen, welche Arten von Webapplikationen in Webinformationssystemen zum Einsatz kommen. Dies werde ich anhand der Systeme „My Netscape“ und „hamburg.de“ tun. Anschließend werde ich kurz auf Lebenslageninformationssysteme eingehen. Zunächst aber die drei Begriffsdefinitionen, beginnend mit der Frage: „Was sind Webapplikationen?“

2.1 Begriffsdefinition Webapplikation

Die erste sich anbietende Definition ergibt sich aus dem Begriff selbst. Demnach handelt es sich um ein Anwendungsprogramm für oder im World Wide Web.

In der Literatur gibt es keine einheitliche Begriffsdefinition der Webapplikation. So gibt es beispielsweise viele Server Produkte am Markt, die damit werben, eine Plattform für „webapplications, web-applications oder web based applications“ zu sein (z.B. BeaWeblogic, Tomcat, usw.), aber eine genaue Definition, was eine Webapplikation ist, findet man kaum.

In „JAVA Servlet Programming” findet sich folgende Definition:

> “While a web page merely displays static content and lets the user navigate through that content, a web application provides a more interactive experience. A web application can be as simple as a keyword search on a document archive or as complex as an electronic storefront.”
> (siehe [HunterCrawford 1998, S. 2])

Dieser kurzen Definition, die nur die Interaktion als Kriterium der Unterscheidung zwischen Webseiten und Webapplikation betrachtet, steht beispielsweise die Definition in der FAQ des Applikationsserver Tomcat ([Tomcat 3.2.2]) gegenüber. In ihr wird eine „web-application“ beschrieben als eine Sammlung von Resourcen, die über einen speziellen Uniform Resource Identifier (URI) als Prefix angesprochen werden kann. Solch eine Sammlung von Resourcen kann nach der dortigen Definition unter anderem aus JSPs, servlets, HTML-Dateien, Bildern und ähnlichem bestehen. Diese Definition bezieht sich darauf, welche Bestandteile eine Webapplikation in einem Applikationsserver basierend auf dem J2EE von SUN wie z.B. der Tomcat beinhalten kann. Des Weiteren kann man daraus erkennen, dass in diesem Zusammenhang die Zuordnung der Bestandteile einer Webapplikation anhand eines bestimmten URI Prefixes erfolgt.

Die nachfolgenden zwei Definitionen hingegen haben einen anderen Betrachtungswinkel:

> „...the scope of Web-based applications has grown enormously, now encompassing four general kinds of Web-based systems: Intranets, to support internal work, Web-presence sites that are marketing tools designed to reach costumers outside the firm, electronic commerce systems that support consumer interaction, such as online shopping, and a blend of internal and external systems to support business-to-business communication, commonly called extranets.“
> (siehe [Isakowitz et al. 1998, S. 78])

Diese Definition betrachtet Webapplikationen nur nach ihrem Einsatzgebiet und unterscheidet in diesem Zusammenhang zwischen vier Typen:

1. Intranets – zur Arbeitsunterstützung innerhalb einer Organisation.
2. Web Sites zu Repräsentationszwecken.
3. Applikationen für Elektronischen Kommerz (z.B. online shopping).
4. Extranets – Systeme zur Interaktion zwischen Organistationen.

In der letzten von mir untersuchten Definition wird jede Art von Software, die in irgend einer Form das Web verwendet oder von seinem Vorhandensein abhängig ist, als Webapplikation bezeichnet. In diesem Zusammenhang wird darauf hingewiesen, dass die meisten speziell für das Web entwickelten Applikationen als Schwerpunkt die Präsentation von Inhalt besitzen.

> „We define a Web application as any software application that depends on the Web for its correct execution. Obviously, software explicitly designed for delivery over the Web falls under this definition — for example, Web sites or Web-based journals. These kinds of software are characterized by a strong notion of content.
> We also include software that uses the Web infrastructure for its execution. ...“ (siehe [GellersenGaedke 1999, S. 61])

Basierend auf den vorhergehenden Definitionen werde ich im folgenden unter einer Webapplikation eine Softwareanwendung verstehen, die ihr Frontend im Web hat. Dies schließt auch so kleine Anwendungen wie dynamisch erzeugte Börsenkurse, Wetterberichte oder Horoskope ein, wie sie vielfach in Webinformationssystemen vorkommen. Eine Webapplikation muss nicht ursprünglich für das Web entwickelt worden sein, kann also erst nachträglich ein Webinterface erhalten haben. Jedoch betrachte ich Sammlungen von einfachen statischen Webseiten ohne jede Interaktion nicht als Webapplikationen. Eine Webapplikation zeichnet sich in meinen Augen dadurch aus, dass ein Mindestmaß an Programmlogik zum Einsatz kommt. Damit meine ich nicht das einfache Ausliefern einer HTML-Seite durch die Programmlogik des Webservers oder deren Darstellung durch den Webbrowser. Dagegen ist die oben vorgestellte Einschränkung auf ein bestimmtes URI Prefix für alle Teile einer Webapplikation zu einschränkend, da diese durchaus auf mehrere Systeme mit unterschiedlichen URIs verteilt sein könnte. Schließlich ist die Verteilung und Aufhebung von Systemgrenzen eine Eigenschaft des Webs.

2.2 Begriffsdefinition Service

Der Begriff des Service an sich wird im Deutschen zumeist als Dienst bzw. Dienstleistung bezeichnet und besagt, dass es sich dabei um jede Art von Tätigkeit handelt, welcher sich jemand zum Nutzen eines Anderen unterzieht (vgl. [Pierer 1858]), genauer gesagt :

> Eine Dienstleistung ist ein ökonomisches Gut, welches der Befriedigung menschlicher Bedürfnisse dient. Sie ist aber im Unterschied zu Sachgütern (Waren) nicht lagerfähig und ihre Produktion und ihr Verbrauch fallen zeitlich zusammen.
> (vgl. [Pierer 1858], [Mayer 1987])

In [Blank 2001, S. 9] und in [OttoSchuler 2000, S. 42] sind zusätzlich aus verschiedenen Quellen die folgenden Eigenschaften eines Service heraus gearbeitet worden:

- „Immaterialität (Überwiegende)[1,2]
- Zweck: Befriedigung eines menschlichen Bedürfnisses[2]
- Integration des externen Faktors[1]
- Gleichzeitigkeit von Produktion und Konsum (Uno-Actu-Prinzip)[1,2]
- Bedarfsdeckung erfolgt durch Leistung[1]
- Mehrstufige Leistungsproduktion[1]
- Kundenorientierte Dienstleistungen können nach der Leistungsvereinbarung an individualisierte Bedürfnisse angepasst werden[1]
- Übertragbarkeit der Aufgabe[2]“

Der Begriff des Service wird in verschiedenen Kontexten durchaus unterschiedlich interpretiert. So lässt es sich auch in dieser Arbeit nicht ganz vermeiden, zwischen einem allgemeinen Service Begriff und einem speziellen im softwaretechnischen Sinne zu unterscheiden.

In den folgenden Kapiteln wird der Service Begriff im Zusammenhang mit Webinformationssystemen und deren Leistungsumfang benutzt. Der Service Begriff steht dann für eine Leistung, die für einen Benutzer des Systems erbracht wird. Dies kann beispielsweise ein Wetterbericht, ein Börsenbericht, Webmail oder ähnliches sein. Dabei geht es nicht um das technisch zugrunde liegende System, sondern um den Nutzen, der dem menschlichen Benutzer des Systems erbracht wird. Er deckt sich dabei mit den oben beschrieben Definitionen.

Der Begriff des Service wird aber in den Kapiteln 4, 5, 6 und 7 im Zusammenhang mit Web Services, Fachlichen (Domain) Services und meinem Entwurf speziell im softwaretechnischen Sinne gebraucht. Dies schränkt die obigen Eigenschaften um die folgende Definition ein: In diesem Kontext verstehe ich unter einem Service ein Programm X, welches Aufträge eines Konsumenten (Programm Y) erledigt auf Grundlage seiner Spezifikation. Dieses findet auf der Ebene von Methoden Aufrufen zwischen verschiedenen Objekten der jeweiligen Programme statt, ohne dass ein direkter menschlicher Eingriff erfolgt. In der Softwaretechnik wird dies beim Softwareentwurf als „Design by Contract“ (vgl. [MeyerB 1997, S. 331ff]) oder als „Vertragsmodell“ (vgl. [Züllighoven 1998, S. 44ff]) bezeichnet:

> „Das *Vertragsmodell* betrachtet die Benutzt-Beziehung zwischen Klassen als ein Verhältnis von Leistungsanbietern und Klienten, das durch einen formalen Vertrag geregelt ist.
>
> Der Vertrag legt fest, welche Vorbedingungen ein Klient erbringen muss, damit der Anbieter seine Leistungen erfüllt.
>
> Verträge sind beim Anbieter beschrieben und bestehen aus Vor- und Nachbedingungen sowie Invarianten.“
> (siehe [Züllighoven 1998, S 45])

1 [Blank 2001], S. 9
2 [OttoSchuler 2000], S. 42

2.3 Begriffsdefinition Webinformationssystem und Portal

Ein Webinformationsystem (engl.: web[-based] information system - WIS) ist zunächst ganz allgemein formuliert ein System im World Wide *Web*, in dem sich Besucher desselben Informationen besorgen können. Dies können ganz allgemeine Informationen sein oder nur ausgewählte zu einem speziellen Themenschwerpunkt. Vielfach findet man auch synonym den Begriff des Portals für Webinformationssysteme.

Portal kommt vom lat. porta und bedeutet nichts weiter als Tor, Tür bzw. steht für den Eingang ganz allgemein (vgl. [Duden 1999]). Ein bekannter Portalbau ist beispielsweise das Porta Nigra in Trier.

In diesem Kontext ist darunter ein Eingangstor zum World Wide Web zu verstehen. Ein Portal ist daher von seiner Bedeutung her streng genommen eine spezielle Unterart vom Webinformationssystem mit einer speziellen inhaltlichen Zielsetzung:

> „*Portal* Einstiegsseite und erster Anlaufpunkt, über den der Zugang zu weiteren Web Sites erfolgt. Informationen und Dienstleistungen werden einem offenen oder geschlossenen Userkreis zugänglich gemacht."
> (vgl. [TDNE 2001])

Zur allgemeinen Definition von Webinformationssystemen findet sich unter anderem die folgende Aussage, die auf den integrativen Charakter von Webinformationssystemen abzielt:

> „WISs can integrate processes or systems within a single interface, and allow access over a local intranet or global Internet network."
> (siehe [Isakowitz et al. 1998, S. 80])

Eine andere Definition versucht, zusätzlich über die möglichen Ausprägungen eines Webinformationssystems, diesen Begriff zu erfassen:

> „A WIS is an Information System providing facilities to access complex data and interactive services through the Web. E-Business applications, Intranet systems, CRM[3] and supply chain applications are examples of WIS."
> (siehe [Gnaho 2001, S. 105])

Besonders im Vergleich dieser Definition von Webinformationssystemen mit den folgenden Definitionen für verschiedene Typen von Portalen wird die enge Verwandtschaft der beiden Begriffe deutlich.

Neben diesem allgemeinen Portalbegriff gibt es noch mehrere spezielle, die sich nach der inhaltlichen Ausrichtung des jeweiligen speziellen Typs orientieren. Diese Zuordnung geschieht grob zwischen horizontalen Portalen mit einem unspezifischen und vertikalen Portalen mit einem Zielgruppen spezifischen Inhaltsangebot sowie der Unterteilung, ob dieses einem offenen oder geschlossenen Personenkreis zugänglich ist.

> „Das *vertikale* Portal ist ein Internetangebot, das sich an eine klar umrissene Zielgruppe wendet. Es konzentriert sich auf ein Thema und geht eher in die Tiefe als in die Breite, wie sich etwa www.fahrschule.de an alle Fahrlehrer und Fahrschüler wendet. Vertikale Portale

3 CRM – Customer Relationship Management

sind vor allem im B2B-Bereich wichtig. Im Gegensatz dazu richtet sich ein *horizontales* Portal an eine sehr breite Nutzerschicht mit allgemeinen Interessen, wie etwa der Onlinedienst und die Suchmaschine www.web.de. Ein *Business*-Portal ist ein auf eine Branche spezialisiertes vertikales Portal, welches ausschließlich geschäftlichen Zwecken dient, wie etwa ein Internetangebot für die Holz verarbeitende Industrie, über das man zu den einzelnen Anbietern oder zu einem digitalen Marktplatz kommt."
(vgl. [TDNE 2001])

In [SchumacherSchwickert 1999] z.B. wird schließlich zwischen vier verschiedenen Typen von Portalen unterschieden für eine Einordnung in vertikal, horizontal, offen oder geschlossen siehe Tabelle 1:

- *„Consumer-Portal*e
  sind hochfrequentierte Web-Einstiegsseiten im Internet. Sie bieten Nutzern, die sich von der Informationsflut des Netzes überfordert fühlen, eine kostenlose Einstiegs- und Orientierungshilfe. Consumer-Portale aggregieren Informtationen und Funktionen des Internets unter 'einem Dach'. ...
- *Enterprise-Information-Portal*e
  sind die Einstiegsseiten für Webseiten eines Unternehmens. Sie unterstützen Interessen der Nutzer (Endkunden und Mitarbeiter), um unternehmensspezifische Informationen zu finden. ... Eine Sonderform des EIP stellt das Branchenportal dar. Ein Branchenportal unterstützt die Nutzer bei der Suche nach fachspezifischen Informationen nicht nur über einzelne Unternehmen, sondern auch über eine gesamte Branche oder Berufsgruppe. ...
- *Extranet-Portal*
  ... Das wesentliche Konzept und die Basis-Elemente sind mit denen der Internet- und Intranet-Portale identisch. Die Zielgruppe des Extranet-Portals eines Unternehmens bilden die aktiv und potentiell kooperierenden Geschäftspartner in der erweiterten Wertschöpfungskette des Unternehmens. Das Extranet-Portal berücksichtigt die speziellen Interessen der Kooperatoren mit besonderen Informationen und Services wie z.B. Bestell- und Liefervorgänge (Logistik). ...
- *Intranet-Portal*
  ... Ziel eines Intranet-Portals ist es, dem Mitarbeiter ein ähnliches Selbstbedienungserlebnis bei unternehmensinternen Aktivitäten zu vermitteln wie dem Nutzer bei einem öffentlichen Consumer-Portal. Dem Mitarbeiter werden durch das Intranet-Portal ein konsistenter Blick auf das Unternehmen, leistungsstarke Suchmöglichkeiten, ein direkter Zugriff auf Unternehmensapplikationen, Verweise auf wichtige Informationen und ein persönlich angepaßter Zugriff auf die Intranet-Inhalte geboten. ..."
  ([SchumacherSchwickert 1999, S. 8-9])

Nutzerkreis	offen		geschlossen	
Netzwerk	Internet		Extranet	Intranet
E-Business-Segment	E-Kommerz		E-Integration	E-Workflow
Bezeichnung in der Literatur	Consumer Portal	Enterprise-Information-Portal	Extranet-Portal	Intranet-Portal
	Internet-Portal		Extranet-Portal	Intranet-Portal
	E-Kommerz-Portal		E-Integrations-Portal	E-Workflow-Portal
	B2C[4]-Portal		B2B[5]-Portal	B2E[6]-Portal
Orientierung	kunden-spezifisch	unternehmens-spezifisch	kooperativ	unternehmens-intern
	horizontal	vertikal		

Tabelle 1 - Gliederung und Bezeichnung von Portalen vgl. [SchumacherSchwickert 1999, Abb. 3, S. 8]

Basierend auf den obigen Definitionen verstehe ich unter einem Webinformationssystem eine Web Site, auf der ein Besucher verschiedene integrierte Serviceleistungen, Informationen und Nachrichten unter einer Oberfläche erhalten kann. Dies erfolgt durch die Integration verschiedener *Webapplikationen* unter eine einheitliche Oberfläche.

Der zur Zeit im Web vorherrschende Typ von Webinformationssystem ist das Portal. Auf Grund dessen und der im vorherigen beschriebenen unterschiedlichen Typen von Portalen ist es im allgemeinen Sprachgebrauch inzwischen üblich, die Begriffe Portal und Webinformationssystem synonym zu benutzen bzw. nur von Portalen zu sprechen. Obwohl das Webinformationssystem eigentlich das allgemeinere Konzept der beiden ist und ein Portal ein Webinformationssystem ist mit der Zielsetzung, dem Benutzer den Mehrwert zu bieten sowie einen Einstieg in das Internet bzw. einen bestimmten Themenkomplex zu bieten und zu erleichtern, zumeist vor einem kommerziellen Hintergrund.

In dieser Arbeit wird aus diesem Grund zwischen Webinformationssystemen und Portalen unterschieden und nur von Portalen gesprochen, wenn es tatsächlich um Internet Portale als Einstiegspunkt in das Internet oder einen speziellen Themenkomplex geht, bzw. eine Quelle von Portalen spricht.

Ein weiterer Punkt von Interesse, der hauptsächlich im Zusammenhang mit Portalen genannt wird, ist, zwischen personalisierbaren und nicht personalisierbaren Angeboten zu unterscheiden. Bei personalisierbaren Angeboten hat der Benutzer die zusätzliche Möglichkeit, nach dem er sich beim Webinformationssystem oder der jeweiligen Webapplikation persönlich angemeldet hat, die Angebote für sich anzupassen bzw. Zusatzleistungen in Anspruch zu nehmen, was bei nicht personalisierbaren Angeboten nicht möglich ist.

Dies kann Zusatzdienste wie Mail, Kalender oder eigene Homepage umfassen. Bei einigen Anbietern ist es möglich, sich eine eigene Startseite zu erstellen. Auf dieser kann man, je nach persönlichem Interesse, Informations- und Dienstleistungsangebote platzieren, die man aus einer Auswahl des gesamten Systems heraussuchen kann:

> „... hat der Nutzer von Portalen im engeren Sinne (personalisierbare Portale) Einfluß auf die ihm präsentierten Inhalte. Er kann hier entweder seine Präferenzen angeben und eine Seite

4 B2C – Business to Consumer
5 B2B – Business to Business
6 B2E – Business to Employee

> automatisch erstellen lassen, oder seine persönliche Seite selbst durch die individuelle Auswahl von Services zusammenstellen.“
> (siehe [SchumacherSchwickert 1999, S. 7])

Zur Wichtigkeit der Personalisierbarkeit von Portalen wird weiter ausgeführt:

> „Nach Aussagen von Portalanbietern kommen Nutzer, die ihr Portal personalisiert haben, fünfmal häufiger wieder und verbringen zehnmal mehr Zeit auf dem Web-Portal.“
> (siehe [SchumacherSchwickert 1999, S. 24])

Daraus lässt sich ersehen, dass es für Entwickler von Webinformationssystemen und Webapplikationen wichtig ist, möglichst einen Mechanismus zur Personalisierung ihrer Systeme anzubieten, um diese für ihre Kunden und Geschäftspartner attraktiv zu machen.

Dies ist vor allem für Entwickler und Betreiber von Webinformationssystemen ein Problem, die Webapplikationen von vielen verschiedenen Fremdanbietern einbinden wollen, da es keinen allgemeinen Standard der benötigten Angaben und deren Benennung gibt.

Ein weiteres Problem in diesem Zusammenhang ist die Frage, welche Daten an Partner weitergegeben werden dürfen und in welcher Form? Was akzeptieren Benutzer und welche Datenschutzbestimmungen sind zu beachten? Derzeitige Lösungen, die sich mit diesen Problemen beschäftigen, sind beispielsweise das bereits existierende „.net Passport“ System (vgl. [Microsoft 2002b]) von Microsoft und die vorerst nur als Spezifikation existierende Lösung der Liberty Alliance (siehe [LibertyAlliance 2002]).

Da eine genaue Untersuchung dieses Themenkomplexes und der beiden Systeme bezüglich ihrer Funktionalität auf die Akzeptanz bei Internetnutzern und im Hinblick auf europäische und internationale Datenschutzrichtlinien den Rahmen der vorliegenden Arbeit gesprengt hätten, bietet die vorliegende Arbeit keine tiefer gehende Lösung dieses Problems in den konstruktiven Kapiteln an.

2.4 Beispiele

Im folgenden werde ich nun die beiden Portale „My Netscape“ und „hamburg.de“ als zwei unterschiedliche Beispiele genauer untersuchen. Diese beiden unterscheiden sich vor allem darin, dass es sich bei „My Netscape“ um ein personalisierbares Portal mit vielen kleinen personalisierbaren Webapplikationen handelt und bei „hamburg.de“ um ein so gut wie gar nicht personalisierbares Portal, das seine Services direkt in die Seiten einbindet.

Dazu kommt, dass es sich bei „My Netscape“ um ein horizontales Portal handelt, welches versucht, möglichst alle Themengebiete abzudecken, die einen Benutzer interessieren könnten. Bei „hamburg.de“ handelt es sich dagegen eher um eine Mischung aus einem vertikalen und horizontalen Portal, welches sich primär mit der Stadt und der Region Hamburg beschäftigt, in diesem Zusammenhang aber versucht, ähnlich viel anzubieten, wie es ein horizontales Portal tut.

In [SchumacherSchwickert 1999] wird auf Grund einer statistischen Erhebung eine Liste mit muss und kann Elementen für Portale erstellt (siehe Tabelle 2), die von den beiden Beispielen weitestgehend erfüllt wird (vgl [SchumacherSchwickert 1999, S. 27-31]).

Diese Tabelle gibt einen Überblick über weit verbreitete Arten von Webapplikationen und bietet einen ersten Anhaltspunkt darüber, welche Funktionalitäten Webapplikationen haben, die von dem im praktischen Teil dieser Arbeit erstellten Framework unterstützt werden sollen.

Muss-Elemente	Kann-Elemente
• Suchdienste • E-Mail • Nachrichten • Börsenkurse • Portfolioverwaltung • Diskussionsforen und Newsgroups • Linksammlung und Quick-Link • Online Shopping • Downloads	• Adressbuchverwaltung • Auskunftdienste • Chat und Konferenzen • Homepage • Horoskope • Instant Messaging • Internet-Telefonie • Landkarten/Routenplanung • Lexika • Link- und Lesezeichen-Sammlung • Online-Backups • SMS und Pager • Spielen • Sportergebnisse • Terminkalender • TV-Programme • Veranstaltungskalender • Währungsrechner und Wechselkurse • Wetterbericht / Reisewetter • Web-Fax

Tabelle 2 - Muss- und Kann-Elemente von Portalen nach [Schumacher 1999, S. 27-31]

2.4.1 Personalisierbares Portal „My Netscape“

Bei „My Netscape“ handelt es sich um die persönliche Startseite des englisch sprachigen Portals von Netscape. Dieses richtet sich zunächst an alle Benutzer des World Wide Web, und es ist ein Portal mit recht ausgeprägten Möglichkeiten zur Personalisierung. So ermöglicht es dem Benutzer, auf seiner persönlichen Startseite verschiedene kleine Webapplikationen einzurichten und nach eigenen Vorlieben innerhalb von drei Spalten zu positionieren. Die Palette reicht von Webapplikationen für die aktuellen Börsenkurse über Wetterinformationen und Horoskope bis zu einfachen Kurznachrichten. Des Weiteren kann das Web durchsucht werden (vgl. [SchumacherSchwickert 1999, S. 38-39]).

Zusätzlich gibt es die größeren Webapplikationen Webmail und Calendar, die von der Startseite aus aufgerufen werden können. Dies geschieht unter der Webapplikation „My Services“, in der auch der Status von Webmail und Calendar angezeigt wird.

Einige der kleinen Webapplikationen lassen eine weitere Personalisierung zu, so dass z.B. bei der Wetterapplikation der Wetterbericht für bestimmte Orte ausgewählt werden kann, oder beim Horoskop die angezeigten Sternzeichen. Ein Beispiel wie eine „My Netscape“ Seite aussehen kann, findet sich in Abbildung 2.

Das Layout des Portals teilt den Bildschirm in drei Spalten, in denen Webapplikationen angezeigt werden. Diese bestehen aus zwei schmäleren an den beiden Rändern und einer breiteren in der Mitte. Oben und unten befinden sich jeweils über die gesamte Breite eine Zeile, die oben Statusinformationen ,eine Suchfunktion über das Web und Button zur Personalisierung enthält und unten die Möglichkeit bietet, direkt Webapplikationen zu der jeweils darüber befindlichen Spalte hinzuzufügen.

Die Webapplikationen werden dabei so angezeigt, dass der Eindruck vermittelt wird, dass diese wie unabhängige Applikationen in einem Fenstersystem ausgeführt werden.

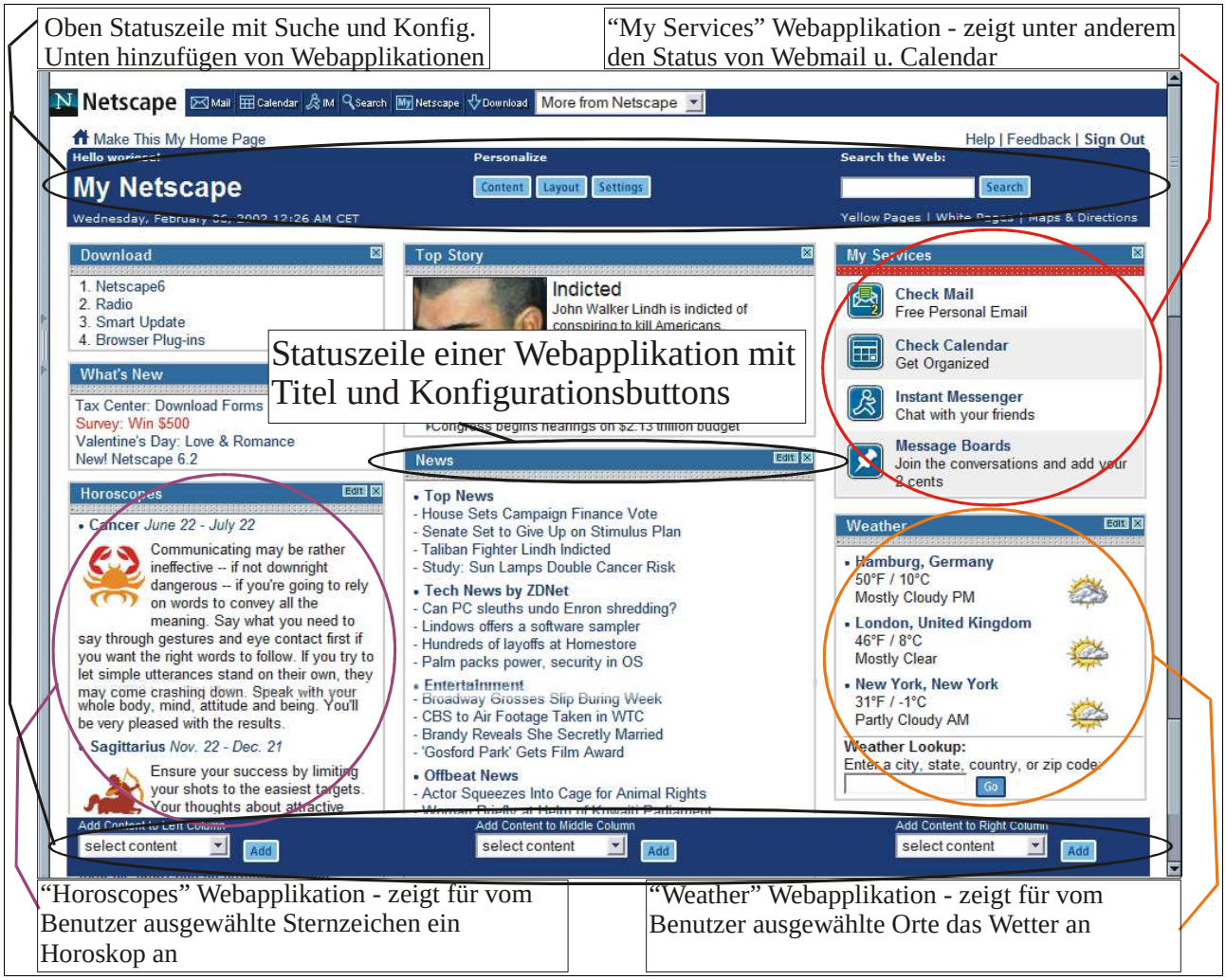

Abbildung 2 - zeigt eine personalisierte Startseite des englisch sprachigen „My Netscape“ Portals mit personalisierten Webapplikationen. Jede Webapplikation hat einen Rahmen mit Statuszeile.

2.4.2 Nichtpersonalisierbares Portal „hamburg.de“

Bei „hamburg.de“ handelt es sich um das Portal der Freien und Hansestadt Hamburg und fällt daher noch zusätzlich unter die Gruppe der Stadtinformationssysteme. Dies hat zur Folge, dass die primäre Zielgruppe des Portals zunächst einmal all jene umfasst, die in dieser Stadt leben, bzw. sich für diese Stadt interessieren. Sei es nun um in Hamburg Urlaub zu machen, vielleicht dort hin umzuziehen oder einfach nur neugierig auf Hamburg zu sein. Daraus resultiert ein zum Großteil anderes Inhaltsangebot als es „My Netscape“ besitzt. Dieses befasst sich daher zentral mit allem, was die Stadt und Region Hamburg betrifft.

Das Portal bietet zur Zeit Personalisierung in Form eines Webmail Angebotes und privater Homepages. Außerdem bietet es einen online Homepageeditor. Es verfügt über ein Repertoire an Webapplikationen, für die keine weitere Personalisierung benötigt wird. Hier gibt es unter anderem einen Stadtplan von Hamburg, in dem nach Straßen gesucht werden kann, das Bürgerinformationssystem DiBIS mit Informationen zu den Behörden und der Verwaltung der Stadt Hamburg, und im Vorfeld der Bürgerschaftswahl 2001 gab es eine Webapplikation, die es den Bürgern der Stadt ermöglichte, ihre Biefwahlunterlagen anzufordern. Außerdem gibt es eine „Jobbörse“ und eine Suchfunktion über den gesamten Inhalt des Portals. Als Beispiel zum jetzigen Stand des Portals sei hier die Startseite vom 24.11.2001 abgebildet (siehe Abbildung 3).

Das Design des Portals „hamburg.de“ setzt ebenfalls wie „My.Netscape“ auf ein drei Spalten Layout. Diese haben jede für sich eine unterschiedliche Bedeutung. Die linke Spalte bietet eine Navigation über den Funktionsumfang des Portals. In der großen mittleren Spalte wird die jeweils ausgewählte Webapplikation dargestellt. Die rechte beinhaltet den Aufruf von Webmail, der Verwaltungsoptionen der persönlichen Angaben, der Verwaltung der eigenen Homepage und Links zu den Web Sites von Partnerunternehemen.

Für die Zukunft war ursprünglich eine stärkere Personalisierung des Portals geplant. Auf Grund von Restrukturierungsmaßnahmen der Betreibergesellschaften und wegen der schwieriger gewordenen Situation in der „New Economy“ wurde dieses Projekt jedoch vorerst, nach meiner Kenntnis, gestoppt. Für einen ersten Schritt in diese Richtung sollte die Einführung eines Lebenslagen-informationssystems sorgen. Dies war in mehreren Ausbaustufen geplant und sollte anfangs ohne Personalisierung auskommen. Erst in einer späteren Version sollte es schließlich personalisierbar werden.

Ein Lebenslageninformationssystem stellt seinem Benutzer Informationen zu Verfügung, um ihn bei einer bestimmten für ihn zutreffenden Lebenslage zu unterstützen. Unter einer Lebenslage versteht sich folgendes:

> „Unter einer Lebenslage verstehen wir die konkrete Situation eines Anwenders oder einer Gruppe von Anwendern (z.B. Ehepaar). Die Lebenslage definiert sich durch ein Vorhaben oder ein Ereignis im Leben des Anwenders (z.B. Geburt eines Kindes). ... Lebenslagen sind grundsätzlich langlebige Vorhaben im Leben einer oder mehrerer Personen. Allerdings kann die Bedeutung von langlebig in vielen Fällen unterschiedlich interpretiert werden. Einerseits ist das Verständnis z.B. für die Planung einer Hochzeit bei unterschiedlichen Personenkreisen und Lebenssituationen unterschiedlich (von Monate im voraus bis einige Wochen vorher).“
> (siehe [BleekFloyd 2000, S. 5-6])

Abbildung 3 - zeigt die Startseite von „hamburg.de“, links befindet sich die Navigation über dem Portal in der Mitte wird der ausgewählte Service angezeigt rechts befinden sich die Anmeldungen für Webmail und Homepage, sowie Links zu Partnerunternehmen

Um für beispielsweise verschiedene Konfessionen bei Heirat oder Geburt jeweils passende Lebenslagen anbieten zu können, gibt es verschiedene Varianten solcher Lebenslage. Zum anderen bietet ein Lebenslageninformationssystem die Möglichkeit, die jeweils ausgewählte Lebenslage an die persönliche Situation anzupassen. Dies geschieht durch den Einsatz einer so genannten Vorgangsmappe:

> „Eine *Vorgangsmappe* ist die Zusammenfassung eines Transportbehälters, in dem die Materialien zur Bearbeitung einer kooperativen Aufgabe enthalten sind, und eines Prozessmusters. Mit dieser Einheit wird den Beteiligten am kooperativen Arbeitsprozess ein Arbeitsgegenstand zur Verfügung gestellt, mit dem die Konsistenz eines Bearbeitungsvorganges überwacht werden kann.“
> (siehe [Gryczan 1996, S. 189])

> „Eine Lebenslage wird mit dem fachlichen Konzept einer Vorgangsmappe unterstützt. Diese Vorgangsmappe gehört einer Person, der sie zugeordnet wird. Diese kann anderen Personen einen Einblick in die Mappe gewähren und/oder ihnen den Umgang mit der Mappe erlauben. Die Vorgangsmappe führt eine Historie über die Arbeit mit ihr,... Zentrale Funktion der Vorgangsmappe ist die Verwaltung des Vorgangs mit Hilfe einer Checkliste. ...“
> (siehe [BleekFloyd 2000, S. 6])

Ein Lebenslageninformationssystem kann eine personalisierte Webapplikation sein, die es ihrem Benutzer ermöglicht, seine Lebenslage zu managen. Dazu stellt sie ihm eine Auswahl an Lebens-

lagen und deren Varianten zur Verfügung. Diese können weiter an die Situation des Benutzers angepasst werden. Durch die Personalisierung kann der Benutzer die im Laufe der Verwendung zusammengetragenen Informationen jederzeit wieder einsehen und damit überblicken, was er schon erledigt hat. In nicht personalisierten Versionen von Lebenslageninformationssystemen fehlen diese nützlichen Möglichkeiten leider, so dass nur allgemeinere Informationen geboten werden können.

In diesem Kapitel habe ich die fachlichen Konzepte Webapplikation, Service, Webinformationssystem und Portal im Zusammenhang mit dieser Arbeit untersucht und definiert. Im Zusammenhang mit Webapplikationen habe ich erste Anhaltspunkte gesammelt, welche in horizontalen Portalen zum Einsatz kommen. Außerdem habe ich exemplarisch die Portale „My Netscape“ und „hamburg.de“ vorgestellt.

Im nächsten Kapitel werde ich auf die Probleme eingehen, die im Zusammenhang mit der Einbindung von Webapplikationen in Webinformationssysteme auftreten. Dabei werde ich technische und organisatorische Probleme erläutern sowie auf das Problem der zwischen den Partnern verteilten verschiedenen fachlichen Kompetenzen eingehen.

3 Flexible Einbindung von Webapplikationen in Webinformationssystemen

In diesem Kapitel beschäftige ich mich mit der grundlegenden Motivation zur flexiblen Einbindung von Webapplikationen in Webinformationssysteme und den daraus resultierenden Problemen. Um diese besser zu verstehen, werde ich zuerst untersuchen, welche Arten von Webapplikationen es gibt und anschließend auf die grundlegende Problematik des Designs von Webapplikationen eingehen. Dann werde ich auf die Interessen der Betreiber und Anbieter von Webinformationssystemen und Webapplikationen eingehen, sowie die fachlichen, technischen und organisatorischen Probleme, die sich beim Einbinden von Webapplikationen in Webinformationssystemen ergeben, untersuchen. Abschließend werde ich die durch die zugrunde liegenden Server Architekturen entstehenden technischen Probleme darlegen.

3.1 Versuch der Klassifikation von Webapplikationen

In der Literatur findet sich beispielsweise die folgende Klassifikation von Webapplikationen, die folgende Unterscheidung trifft:

> „... These applications can be distinguished into two basic categories, *Web Hypermedia Applications* and *Web Software Applications*.
> A *Web Hypermedia Application* is the structuring of an information space in concepts of nodes (chunks of information), links (relations among nodes), anchors, access structures and the delivery of this structure over the Web. ... a collection of Web pages.
>
> A *Web Software Application* is any 'classic' software application that depends on the Web, or uses the Web infrastructure, for correct execution. ... This category includes, among others, legacy information systems accessible through online gateways via a Web browser, such as databases (DBs), booking systems, knowledgebases, numerical software, etc.“
> (siehe [Christodulou 2001, S. 170-171])

Bei dieser Klassifikation wird zwischen Applikationen unterschieden, die Texte strukturiert im Web anbieten, worunter hier hauptsächlich einfache Webseiten bzw. Sammlungen von diesen verstanden werden und solchen, die das Web zur Ausführung benötigen bzw. ihre Ergebnisse im Web zur Verfügung stellen.

In der Einleitung der Arbeit habe ich die Frage aufgeworfen, ob Webapplikationen danach zu unterscheiden sind, ob ihr Inhalt unabhängig vom Kontext der Umgebung ist, in der sie eingesetzt werden. Bei genauerer Untersuchung stellt sich heraus, dass im Grunde genommen alle Webapplikationen in der einen oder anderen Form davon abhängig sein können, in welcher Umgebung sie benutzt werden. Ob das nun eine Webapplikation beispielsweise in einem Stadtinformationssystem wie „hamburg.de“ ist, die einen Wetterbericht für Hamburg anzeigt, oder einen Aktienticker für die Hamburger Börse.

Meine nun folgende Klassifikation richtet sich mehr an den Unterschieden aus, die sich bei der Problematik der flexiblen Einbindung von Webapplikationen in Webinformationssystemen ergeben.

Als kleinster mir bekannter Typ von Webapplikationen, was Interaktion und Anzahl der verschiedenen Zustände der Webapplikation betrifft, ergibt sich aus meiner Sicht der Typ Newsticker, Börsenbericht, Wetterbericht und kleinere Info-Applikationen. Bei „my Netscape“ z.B. gibt es eine Applikation, die Information über neu eingegangene E-Mails gibt. Hierbei handelt es sich um Webapplikationen, die hauptsächlich kurze Textinformationen mit evtl. kleinen Piktogrammen darstellen. Zusätzlich enthalten sie oft Links auf eigenständige externe Webseiten bzw. Webapplikationen. Falls Personalisierung zur Verfügung steht, ist meist eine Konfiguration über das jeweilige Webinformationssystem möglich, in der der Benutzer Inhalte zur Darstellung aus- und abwählen bzw. filtern kann. Ansonsten ermöglichen sie keine weitere Interaktion. Diese Art von Webapplikationen werden als „channel“ bezeichnet, wie z.B. in [McLaughlin 2000, S. 401-414]. Dies bezieht sich darauf, dass der „channel“ ursprünglich als „push“ Technologie (vgl. [Whatis 2002]) entwickelt worden ist. Das Ziel bei der „push“ Technologie ist es, dem Benutzer Webinhalte, wie Nachrichten bei einem Nachrichtensender („channel“) im Radio oder Fernsehen, sobald sie zur Verfügung stehen, automatisch zuzusenden („push“), im Gegensatz zum normalen Abfragen („pull“) durch den Webbrowser des Benutzers. Ein Format für channel Webapplikationen ist das Rich Site Summery (RSS) Format von Netscape [RSS0.91 1999]. Es handelt sich um eine XML-Anwendung, für die eine entsprechende „public[7]“ DTD von Netscape vorliegt. Bei diesem Typ liegt bereits eine gute Trennung von Layout, Applikationslogik und Inhalten vor, da beim RSS Format nur Inhalte ohne die zugrunde liegende Applikationslogik und ohne konkretes Layout gekapselt übertragen werden.

Der zweite Typ von Webapplikation ermöglicht ein größeres Maß an Interaktivität, als der eben genannte „channel“ und umfasst speziell für das Web entwickelte Applikationen. Darunter fallen z.B. business to business (B2B) und business to consumer (B2C) Anwendungen oder Lebenslagen-informationssysteme. Dieser häufig scriptbasierte Typ von Webapplikation besitzt oft keine Trennung von Applikationslogik, Inhalten und Layout. So beinhalten die Scripte bunt gemischt HTML-Code zur Layouterzeugung und Scriptcode mit Programmlogik und Datenbankabfragen. Dieser Scriptcode beinhaltet zum Teil zusätzlich noch HTML-Code zur weiteren dynamischen Layouterzeugung.

Beim letzten von mir identifizierten Typ handelt es sich um eigentlich vom Web unabhängige Anwendungen, die zusätzlich noch ein Webinterface erhalten haben, um so von überall im Web zugänglich zu sein, ob nun für den eingeschränkten Personenkreis von Angestellten einer Firma oder die Allgemeinheit der Netzbenutzer. Dies können beispielsweise Buchungs- oder Lagerverwaltungssysteme sein, für die man nachträglich ein Webinterface angebunden hat. Dieser Typ hat die besten Voraussetzungen für eine Trennung von Applikationslogik und Layout, da er ja auf bereits bestehende „klassische“ Applikationen aufbaut. Diese besitzen entweder bereits von vorn herein eine Trennung von Oberfläche (Layout) und Logik, oder aber spätestens für die Webanbindung hätte man eine entsprechende Schnittstelle einführen können.

Abgesehen von diesen eher die inhaltliche und technische Komplexität betreffenden Kriterien zur Kategorisierung, bietet sich noch die inhaltliche Ausrichtung der Webapplikation für eine solche an.

7 Es wird bei der Einbindung von DTDs in XML-Dokumente zwischen „public“ und „system“ unterschieden. DTDs vom Typ „public“ sind für den freien und uneingeschränkten Gebrauch veröffentlicht und besitzen zur eindeutigen Identifikation neben einer URI einen eindeutigen Namen. DTDs vom Typ „system“ sind dagegen für den internen Gebrauch gedacht und werden nur über ihre URI identifiziert. (siehe [EcksteinCasabianca 2001, S. 16])

In „Internet Portals in Europe“ [Goldman 1999, S. 17] werden beispielsweise verschiedene Kategorien von Funktionalität genannt, die von horizontalen Portalen abgedeckt werden bzw. selektiv von vertikalen. Diese sind: „*Kontext*, *Inhalt*, *Kommerz*, *Kommunikation* und *Gemeinschaften*.“ In „Analyse und objektorientierter Entwurf eines integrierten Portalsystems für das Wissensmanagement“ [Wegner 2002, S. 54-55] werden diese noch folgendermaßen interpretiert:

- „*Kontext:*
  Gemeint sind inhaltsstrukturierende Funktionalitäten wie Suchfunktionen über den Inhalt, bzw. Inhaltsverzeichnisse und Navigationselemente.
- *Inhalt*:
  Es sind Artikel, Nachrichten und ähnliches gemeint, die im Webinformationssystem einsehbar sind.
- *Kommerz*:
  Darunter fallen alle E-Kommerz Angebote, sowie Online-Banking.
- *Kommunikation*:
  Alles an Kommunikationsangeboten, von E-Mail bis hin zu Chat Systemen.
- *Gemeinschaften:*
  Hierbei handelt es sich um alle Aktivitäten rund um Diskussionsforen, Chats, Hompages usw.“

Ein weiterer wichtiger Aspekt bei der Klassifikation und Einbindung von Webapplikationen ist der Sicherheitsaspekt, der durch die Übertragung von personenbezogenen Daten, vor allem bei E-Kommerz, auftritt. In diesem Zusammenhang wird im allgemeinen auf technischer Seite das HTTPS (HTTP Secure Protokoll) verwendet. Bei der flexiblen Einbindung in ein Webinformationssystem ist zu beachten, dass die Daten durch das Webinformationssystem an die Webapplikation weitergereicht werden und dadurch ein besonderes Vertrauensverhältnis zwischen den Geschäftspartnern und Kunden entsteht.

Das Ziel meines WebAp Frameworks ist es, für all diese verschiedenen Typen von Webapplikationen eine einheitliche Schnittstelle und Kommunikationsform zu finden, die es ermöglicht, unabhängig von der inhaltlichen Ausrichtung und dem technischen Hintergrund der ursprünglichen Webapplikation, diese in ein Webinformationssystem zu integrieren.

Im Folgenden werden die Probleme untersucht, die beim herkömmlichen Design von Webapplikationen und deren späteren Modifikation auftreten.

3.2 Grundlegende Probleme im Design von Webapplikationen

Diese haben ihre Ursache hauptsächlich in der Vermischung von fachlich motivierter Anwendungslogik und Layout, sowie den daraus resultierenden Inhalten beim Entwurf und der Implementation von Webapplikationen (vgl. Bsp. Listing 1).

```
<table border='1'>
 <tr><td> </td>
  <td background='#356326' >
    Temperatur
  </td>
 </tr>
<%
   ResultSet Rs = Stmnt.executeQuery("SELECT orte.Ort, orte.OrtLink, orte.IsImg, orte.Temp
FROM orte, user WHERE user.OrteID = orte.ID ORDER BY orte.Ort");
  ResultSetMetaData RsMe = Rs.getMetaData();
  int cols = RsMe.getColumnCount();
  while(Rs.next()) {
%>
 <tr>
  <td background='#34ff23' >
    <% if(Rs.getString(3).compareTo("true") == 0) { %>
      <a href='<%= Rs.getString(2) %>' ><img src='/img/ort/<%= Rs.getString(1) %>.gif' ></a>
    <% } else { %>
      <a href ='<%= Rs.getString(2) %>'><%= Rs.getString(1) %></a>
    <% } %>
  </td>
  <td background='#235678' >
   <%= Rs.getString(4) %>&deg;C
  </td>
 </tr>
<%
  }
%>
</table>
```

Listing 1 – Beispiel für die Vermischung von Layout, Inhalt und Logik bei vielen Webapplikationen.
Layout ist in Standard, **Inhalte in fett und unterstrichen** *und Logik in kursiv gesetzt.*

Das Beispiel in Listing 1 verdeutlicht dieses Problem. Es zeigt die Vermischung von HTML-Layoutelementen zur Erzeugung einer Tabelle, Programmlogik zum Auslesen einer Datenbank und Einfügen von Inhalten in die zu erzeugende Webseite, sowie die zum Inhalt gehörenden Elemente „Temperatur und °C“.

In der Literatur findet sich zu diesen Problemen unter anderem die folgende Aussage:

> „Key requirements for successful web sites are support for change (e.g., change of layout requirements) and scalability. ... Layout independence will be a basic requirement for web sites of the future that will have to provide services to a large variety of different devices, components and interfaces. ...
> ...the problem of separating the business logic from the layout and content in dynamic web services has not received much attention yet.
> A web based service is concerned with the following three issues.
>
> **Business Logic.** We define the *buiseness logic* of a web site as the functionality that is necessary for providing the dynamic interaction and services to the users.
>
> **Content.** We define the *content* of a web site as the 'real' information offered by it (e.g., price lists, titles, names, etc.).
>
> **Layout.** *Layout* denotes the formatting information applied to the content to display it.„
> (siehe [KererKirda 2001, S. 135-137])

Wie sich im Laufe der Arbeit zeigte, ist zumindest eine deutliche Trennung nötig zwischen der Programmlogik zur Erzeugung von Inhalten auf der einen und dem Layout auf der anderen Seite, um Webapplikation für verschiedene Kontexte anpassen und einsetzen zu können. Eine hundertprozentige Trennung von statischen Inhalten und der Programmlogik zur Erzeugung von

dynamischen Inhalten ist in der Praxis nicht zu erreichen, kann aber auf ein Mindestmaß reduziert werden, in dem die dynamischen Inhalte an einer zentralen Stelle generiert werden und dann nur noch an den entsprechenden Stellen durch Variablen integriert werden müssen. Außerdem lässt es sich nicht vermeiden, Variablen und URL-Adressen für die Aufrufe von Folgezuständen in den Inhalten zu integrieren.

3.3 Die Interessen und Probleme der Beteiligten

Die Interessen auf der Seite des Betreibers des Webinformationssystems ergeben sich daraus, dass er auf seinem System möglichst viele Serviceleistungen anbieten möchte, um möglichst viele Besucher an sich zu binden. Dies wird er zu erreichen versuchen, in dem er qualitativ hochwertige Serviceleistungen anbietet. Diese wird er jedoch zumeist nicht selber entwickeln wollen, da er sich sonst erst das Wissen der jeweiligen Fachgebiete bei entsprechendem Zeit- und Kostenaufwand aneignen und schließlich noch implementieren müsste, zumal es im Web bereits Angebote zu beinahe jedem erdenklichen Gebiet gibt.

Auf der Seite des Anbieters einer Webapplikation ergibt sich eine ähnliche Situation. Dieser möchte umgekehrt seine Webapplikation aus Kosten und Effektivitätsgründen in anderen Web Sites einsetzen und seinen Service so einem größeren Nutzerkreis zugängig machen, als ihm dies nur mit seiner eigenen Web Site möglich wäre. Am Beispiel einer fiktiven Versicherungs-Webapplikation zeigt Abbildung 4 wie so etwas im Idealfall aussehen kann. Auf der linken Seite sieht man die Webapplikation in der Web Site des Versicherungsanbieters und rechts in einem Webinformationssystem integriert und an dessen Layout angepasst.

Abbildung 4 - Zeigt eine fiktive Versicherungs-Webapplikation als eigenständige Webseite und integriert in ein Webinformationssystem

Daraus resultiert das Problem, wie diese Integration technisch und organisatorisch zu realisieren ist und wie das Ergebnis dieser aussehen soll.

Der Betreiber des Webinformationssystems sucht dafür ein System, welches es ihm ermöglicht, verschiedene Webapplikationen von beliebigen Anbietern so einfach wie möglich in sein System zu integrieren und dabei soweit an sein Layout anzupassen, dass der Benutzer eine Zusammengehörigkeit zwischen Webinformationssystem und Webapplikation erkennt. Zusätzlich muss es möglich sein, verschiedene Webapplikationen gleichzeitig nebeneinander einzusetzen.

Der Anbieter der Webapplikation benötigt seinerseits ein System, das es ihm erlaubt, seine Webapplikation bei geringem Aufwand in verschiedenen Webinformationssystemen und seiner

eigenen Web Site einzusetzen. Dabei muss es möglich sein, das Layout flexibel an die jeweiligen Gegebenheiten anzupassen, ohne dafür jedes Mal Veränderungen an der Webapplikation selbst vornehmen zu müssen.

Die technisch einfachste Lösung ist die textuelle oder graphische Verlinkung des Angebots aus dem Webinformationssystem heraus. Dies bietet zwar den Vorteil, dass weder am Webinformationssystem noch an der Webapplikation etwas angepasst werden muss, aber der Benutzer verlässt dabei das Webinformationssystem. Dazu wird entweder ein neues Fenster geöffnet, oder das mit dem Webinformationssystem wird durch die Webapplikation ersetzt.

Dies hat zwei gravierende Nachteile. Erstens, der Benutzer kehrt möglicherweise nicht wieder zum Webinformationssystem zurück und zweitens, er befindet sich in einem für ihn fremden System eines neuen und erst einmal fremden Ansprechpartners, den er nicht mehr mit dem Webinformationssystem verbindet. Dies könnte auch Auswirkungen auf das zu Stande kommen einer Servicebeziehung haben:

> „Ob der Kunde eine Servicebeziehung eingeht und aufrecht erhält, hängt vor allem von seinem Vertrauen in den Dienstleister ab (als Person oder bzw. und Organisation) und von der Gesamtheit der jeweiligen Randbedingungen wie persönliche Vorerfahrungen oder Abhängigkeit von Dritten bzw. von äußeren Bedingungen...“
> (siehe [Klischewski 2000, S. 2-3])

Dazu kommt, dass eine Personalisierung vom Webinformationssystem und dessen Daten der Webapplikation nicht zur Verfügung stehen. Aus technischer Sicht ergibt sich das Problem, dass der Service nicht verfügbar ist, wenn der Anbieter der Webapplikation Probleme mit seinem Web-Auftritt hat. Diese Lösung hat den Vorteil, dass keiner der Beteiligten fachliches oder technisches Know-how seines Systems preisgeben muss.

Als zweite Lösung bietet sich an, die Webapplikation wieder nur aus dem Webinformationssystem heraus zu verlinken. Der Anbieter der Webapplikation stellt in diesem Ansatz eine an das Layout des Webinformationssystems angepasste Webapplikation zur Verfügung. Dies hat den Vorteil, dass der Benutzer des Service, der vom Webinformationssystem kommt, nicht unbedingt mitbekommt, dass er das Webinformationssystem verlassen hat und sich bei einem fremden Anbieter befindet. Es ist dabei eine Verbundenheit der beiden Anbieter durch die einheitliche Optik ersichtlich.

Allerdings ist der Aufwand zur Anpassung auf Seiten des Anbieters der Webapplikation größer als bei der ersten Lösung. Schließlich muss er das gesamte Layout der Webapplikation an das Webinformationssystem anpassen. Außerdem ergibt sich für ihn das technisch und organisatorische Problem, die Webapplikation in verschiedenen Varianten zur Verfügung stellen zu müssen, einmal für die eigene Web Site und für jedes Webinformationssystem, in dem er seine Webapplikation anbietet. Dafür benötigt er zusätzlich Informationen zum Layout des jeweiligen Webinformationssystems.

Diese Anpassung stellt sich schwer dar, wenn keine Trennung von Layout, Inhalt und Logik existiert, da die Webapplikation in diesem Fall jedes Mal komplett neu implementiert werden muss. Ansonsten muss zumindest der Teil, der das Layout enthält, so oft implementiert werden, wie die Webapplikation in verschiedenen Kontexten eingesetzt wird.

Keiner der beiden muss bei dieser Lösung fachliches Know-how seines Systems preisgeben, und die technische Verfügbarkeit des Service stellt sich genau so wie bei der ersten Lösung dar.

Die dritte Lösung, die sich ergibt, ist die vollständige Integration der Webapplikation ins Webinformationssystem, indem sie direkt in diesem implementiert wird. Hierbei entsteht dem Betreiber des

Webinformationssystems das technische und organisatorische Problem, dass er bzw. der Anbieter der Webapplikation, diese im Rahmen des Webinformationssystems mehr oder weniger komplett neu programmieren muss. Dies hängt zum einen davon ab, ob die ursprünglich verwendete Programmiersprache auch vom System des Betreibers des Webinformationssystems unterstützt wird und ob bereits eine Trennung von Layout, Inhalt und Logik bestand. Sollte dies nicht der Fall sein, ist eine einfache Anpassung der existierenden Webapplikation meist nicht möglich, da auf Grund der daraus resultierenden Unübersichtlichkeit des Codes die Gefahr groß ist, unbeabsichtigt die Logik zu verändern oder zu ändernde Layoutelemente zu übersehen. Außerdem müssen Pfade für Datenbanken und für Aufrufe von Folgezuständen an das neue System angepasst werden.

All dies kann nur geschehen, wenn einer der beiden dem anderen tiefe Einblicke in sein System gewährt und so einen Teil seines fachlichen Wissens preisgibt.
Das ganze hat allerdings den Vorteil, dass der Besucher des Webinformationssystems dieses nicht verlässt, um den Service zu nutzen. Außerdem hat er es augenscheinlich immer noch mit dem selben Ansprechpartner zu tun. Dies dürfte das Vertrauen in den angebotenen Service erhöhen. Die technische Verfügbarkeit des Service erhöht sich in soweit, als dass dieser sobald und solange verfügbar ist, wie das eigentliche Webinformationssystem. Bei dieser Lösung ist noch eine Variante denkbar, wo noch zu verwendende Datenbanken der Webapplikation im System des Anbieters der Webapplikation verbleiben und nur der eigentliche Code Teil integriert wird. In diesem Fall ist die Verfügbarkeit nur noch auf die von den Datenbanken unabhängigen Teile zu beziehen, und für die von den Datenbanken abhängigen Teile gilt wie bei den vorherigen Lösungen, dass sie von der Verfügbarkeit des Systems des Anbieters der Webapplikation abhängig sind.

Die vierte Lösung hat zum Ziel, die Webapplikation weiterhin komplett im System des Anbieters zu belassen, sie aber trotzdem in das Webinformationssystem zu integrieren, so dass sie nicht nur über einen Link erreichbar ist. Dabei ergibt sich der Vorteil, dass weder der Betreiber des Webinformationssystems noch der Anbieter der Webapplikation viele Informationen an den jeweils anderen über das eigene System geben muss. Beide müssen lediglich eine gemeinsame Schnittstelle finden.

Der Anbieter der Webapplikation und der Betreiber des Webinformationssystems müssen lediglich Informationen darüber austauschen, welche Daten zur Personalisierung der jeweiligen Webapplikation zu übertragen sind und sich darüber einigen, wie die Webapplikation vom Layout her in das System eingepasst werden soll.

Die wichtigste Aufgabe der Schnittstelle zwischen Webinformationssystem und Webapplikation ist es, den Austausch der Inhalte der verschiedenen Zustände der Webapplikation mit dem Webinformationssystem zu ermöglichen. Bei dieser Übertragung müssen zusätzlich Variablendefinitionen und Adressen für Aufrufe von Folgezuständen übermittelt werden. Außerdem muss die Schnittstelle es ermöglichen, technisch unterschiedlich realisierte Systeme miteinander zu verbinden. Dabei müssen die Sessions der beiden Systeme für das jeweils andere transparent sein. Des Weiteren soll es die Schnittstelle dem Betreiber des Webinformationssystems möglich machen, gleichzeitig verschiedene Webapplikationen von unterschiedlichen Anbietern zu integrieren und es genauso dem Anbieter der Webapplikation gestatten, seine Webapplikation in verschiedenen Kontexten ohne weitere Änderungen einsetzen zu können, immer vorausgesetzt, dass alle diese einheitliche Schnittstelle verwenden.

Bei dieser Lösung besteht ebenfalls der Vorteil der Integration der Webapplikation in das Webinformationssystem und dass der Besucher es nicht verlassen muss, um die Webapplikation zu nutzen. So hat der Nutzer auch bei dieser Lösung den Eindruck, es mit demselben Ansprechpartner zu tun zu haben, was sich positiv auf das Vertrauensverhältnis auswirken dürfte.

Die technische Verfügbarkeit der Webapplikation ist aber wie in der ersten Lösung wieder davon abhängig, dass nicht nur das System des Webinformationssystems sondern auch das System des Anbieters der Webapplikation verfügbar ist.

Um die im folgenden und in Kapitel 3.4 aufgeführten Probleme bei der Einbindung von Webapplikationen in Webinformationssysteme so gering wie möglich zu halten und ein sinnvolles Maß an Integration zu erlangen, habe ich mich bei meinem in den Kapiteln 6 und 7 beschriebenen WebAp Framework für eine Lösung entschieden, wie ich sie als vierte Variante beschrieben habe.

Festzuhaltende Probleme bei der Einbindung von Webapplikationen:

- Es gibt vielfach Vermischungen von Layoutelementen und Anwendungslogik. Daraus resultiert zudem das Fehlen einer eindeutigen Trennung von Inhalten und Oberflächenstrukturelementen. Dies hat zur Folge, dass das Anpassen von Anwendungen an andere Layouts erschwert wird, da die Gefahr besteht, versehentlich Anwendungslogik und Inhalte zu verändern.
- Bei der Integration von Webapplikationen in Webinformationssysteme besteht außerdem das grundsätzliche Problem der Anpassung des Kontrollflusses der Applikation an das Webinformationssystem bzw. des Webinformationssystems an die integrierten Webapplikationen.
- Es besteht das Problem, zu viel eigenes Know-how an den Geschäftspartner preisgeben zu müssen und sich selbst so auf lange Sicht überflüssig bzw. Konkurrenz zu machen.
- Andersherum gesehen muss der Geschäftspartner evtl. Zeit opfern, um sich mit Dingen zu beschäftigen, die für sein eigenes Aufgabenfeld eigentlich nicht relevant sind.

3.4 Weitere technische Ursachen des Problems

Weitere technische Ursachen sind in den den Webinformationssystemen und Webapplikationen zu Grunde liegenden Architekturen begründet. Diese verwenden dabei verschiedene Kombinationen der im folgenden beschriebenen Server, sowie einen Webbrowser zum Abrufen und Anzeigen der HTML-Webseiten (siehe Abbildung 5):

- *Webserver* dienen zur Kommunikation mit dem Webbrowser des Anwendungsbenutzers. Sie liefern die eigentlichen HTML-Webseiten an diesen aus. Außerdem reichen sie die vom Webbrowser übertragenen Eingaben des Benutzers an WCMS oder Applikationsserver weiter. Die von ihnen ausgelieferten Webseiten liegen entweder statisch vor, oder werden dynamisch durch ein WCMS oder von einem Applikationsserver generiert. Auf Grund der Zustandslosigkeit des ihrer Kommunikation zugrunde liegenden HTTP-Protokolls, ist jeder Seitenaufruf für einen Webserver ein neues Ereignis, das von allen vorherigen und nachfolgenden vollkommen unabhängig ist.
- *Applikationsserver* besitzen ein Sessionmanagement, um aufeinander abfolgende zusammengehörige Anfragen zu identifizieren. Auf diese Weise sind sie nicht wie die Webserver der Zustandslosigkeit unterworfen. Sie besitzen außerdem zumeist eine Datenbankanbindung. Ihre Hauptaufgabe besteht darin, durch die in ihnen ablaufenden Programme dynamisch Webseiten zu erzeugen.

- *Web Content Management Systeme (WCMS)* sorgen ebenfalls für eine dynamische Seitengenerierung z.B. über Templates und besitzen ebenfalls eine Verwaltung von Sessions mit dem Webbrowser des Endanwenders. Auch WCMS besitzen eine Datenbankanbindung und können mit den Applikationservern, auf denen Webapplikationen ausgeführt werden, kommunizieren.
- *Webbrowser* stellen die vom Webserver ausgelieferte HTML-Seiten dar und sind für die direkte Interaktion mit dem Benutzer zuständig. Sie übertragen Eingaben, die vom Benutzer in Formularen gemacht worden sind nach Bestätigung des Benutzers an Webserver.

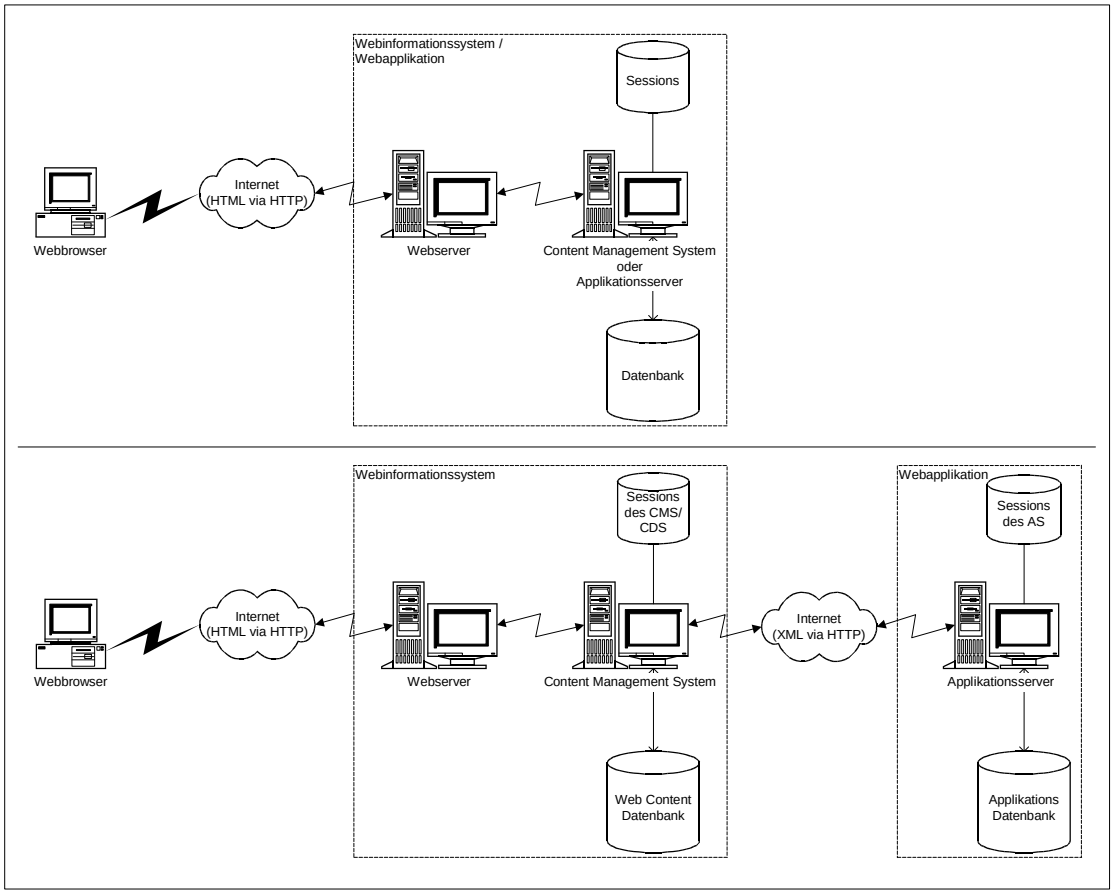

Abbildung 5 - Konkurrierende Architekturmodelle
Oben: einzelne Webapplikation betrieben in einem Applikationsserver bzw. CMS
Unten: Webinformationssystem in einem CMS mit Webapplikationen in Applikationsservern

Die daraus entstehenden Probleme resultieren aus der Tatsache, dass diese Systeme über teilweise konkurrierende Modelle ihres Einsatzkontextes verfügen.

Webserver, wie z.B. der von mir eingesetzte Apache, sind dafür gedacht, Anfragen von Webbrowsern aus dem Web zu beantworten, in dem sie lokale Dateien an diese versenden. Durch Erweiterungsmodule sind sie in der Lage, Anfragen auf bestimmte Dateien an Scriptinterpreter mit Datenbankanbindung weiterzuleiten und so einfache dynamische Seiten zu erzeugen. Ihr Spezialgebiet ist das performante Ausliefern von Webseiten, durch Caching der statischen Webseiten im Arbeitsspeicher des Rechners, auf dem der Webserver läuft. Ein Webserver kennt keinen Zustand,

der über das Bearbeiten einer einzelnen Anfrage hinausgeht. Für ihn ist jede Anfrage ein für sich unabhängiger Vorgang, der nicht von vorhergehenden Anfragen beeinflusst wird.

Web Content Management Systeme wiederum sind dazu gedacht, große Datenbestände aus Datenbanken möglichst dynamisch über zu füllende Templates als Webseiten an einen Webserver auszuliefern. Dazu bieten sie die Möglichkeit des Sessionmanagements. Hierdurch führen sie ein Zustandskonzept ein, das es ihnen ermöglicht, Anfragen durch einen bestimmten Benutzer diesem zuzuordnen und jede Anfrage im Kontext der bereits in dieser Session vorangegangenen und in der Datenbank gespeicherten Informationen zu behandeln. Dadurch wird eine „Personalisierung“ von Web Seiten für den jeweiligen Benutzer möglich. Außerdem verfügen sie über Skript-Sprachen zum Programmieren der Templates, mit denen sich in begrenztem Rahmen Webapplikationen realisieren lassen.

Die Aufgabe eines Applikationsserver wiederum ist es, eine Umgebung für Programme zu schaffen, in der diese ablaufen können und aus der heraus diese in der Lage sind, Anfragen aus dem Internet zu empfangen und darauf zu antworten. Er kann hierfür entweder mit einem existierenden Webserver kooperieren oder diesen ganz ersetzen. Ich setze als Applikationsserver einen Tomcat Server ein, der eine Umgebung für Java Programme implementiert. In dieser Umgebung kann mit Servlets und Java Server Pages (JSPs) auf Anfragen von Webbrowsern reagiert und Webseiten generiert werden. Außerdem können von Servlets und JSPs neue Java Prozesse gestartet werden, die nicht sofort nach dem Abarbeiten einer einzelnen Anfrage beendet werden und bei einem weiteren Aufruf wieder zur Verfügung stehen. Der Applikationsserver verfügt über ein Sessionmanagement und die Möglichkeit, auf Datenbanken zuzugreifen. Dadurch konkurriert er in seinem Funktionsumfang mit den Web Content Management Systemen. Diese sind jedoch auf ihrem Spezialgebiet dem Applikationsserver, was die Komfortabilität der Seitenerstellung und dem Datenbankmanagement betrifft, überlegen.

Auf der Grundlage des im vorherigen und diesem Kapitel erarbeiteten Verständnisses von Webinformationssystemen und Webapplikationen ist es das Ziel dieser Arbeit, ein Architekturmodell zu erarbeiten, welches es ermöglicht, Inhalte von einer Webapplikation an ein Webinformationssystem zu übertragen. Als erster Anhaltspunkt dient dabei das RSS-Dokumentenformat. Diese Inhalte sollten nach Möglichkeit keine Layoutinformationen enthalten, mit Ausnahme von notwendigen Strukturinformationen, wie z.B. Absatzmarkierungen. Des Weiteren muss dieses Dokumentenformat es erlauben, Variablendefinitionen und Aufrufadressen von Folgezuständen zu übermitteln. Das Dokumentenformat soll dabei unabhängig von den verwendeten Programmiersprachen sein, so dass Webinformationssystem und Webapplikation in verschiedenen Sprachen realisiert sein können. Aus diesen übertragenen Dokumenten muss durch gezielte Transformationen HTML mit Layoutinformationen erzeugt werden können. Dieses Dokumentenformat werde ich als DTD für XML-Dokumente entwerfen.

Die Webapplikationen müssen eine Schnittstelle zum Web anbieten, um Eingabedaten zu erhalten und das den aktuellen Zustand wiedergebende Dokument versenden zu können. Diese Schnittstelle wird durch den Aufruf von URIs mit Übergabeparametern angesprochen und liefert dann ein XML-Dokument mit den entsprechenden Eigenschaften zurück.

Zur Integration der Webapplikation in ein Webinformationssystem bzw. der Web Site des Anbieters derselben, werde ich ein Java Framework entwerfen, welches es ermöglichen wird, das Dokument, das den aktuellen Zustand einer Webapplikation repräsentiert, nach HTML zu transformieren und mit den Layoutinformationen für das jeweilige System zu versorgen. Dieses wird außerdem dafür sorgen, dass Eingaben eines Benutzers an die richtige Webapplikation weitergeleitet werden und der entsprechende Folgezustand der Webapplikation aufgerufen wird. Wenn mehrere Webapplikationen gleichzeitig durch das Framework repräsentiert werden, an einer

eine Eingabe vorgenommen und ein Folgezustand aufgerufen wird, dann trägt es dafür Sorge, dass die jeweiligen Zustände der anderen Webapplikationen im Framework, die von dieser unabhängig sind, erhalten bleiben.

Im nächsten Kapitel werde ich die derzeit aktuelle Web Services Architektur im Zusammenhang mit dem Problem der flexiblen Einbindung von Webapplikationen untersuchen. Dabei werde ich mich im besonderen um das Kommunikationsprotokoll SOAP kümmern und die von mir gewählte Lösung in diesem Kontext positionieren.

4 Web Services

In diesem Kapitel werde ich auf den Begriff des Web Service genauer eingehen. Der Begriff des Web Service steht für ein 5-Schichten Architekturmodell zur Entwicklung von verteilten Anwendungen und beschreibt deren Kommunikation und Integration. In diesem Zusammenhang wurden bereits verschiedene Protokolle und Dienste entwickelt, unter anderem das Simple Object Access Protokoll (SOAP). Bei diesem handelt es sich um eine Implementation des im Web Service Modell vorgesehenen Paketdienstes. Eine andere Implementation ist z.B. das XML-RPC Protokoll.
Ich werde im folgenden die dem Modell zugrunde liegende Definition eines Web Services untersuchen und zeigen, wie sich mein Webapplikations-Ansatz im Vergleich hierzu einordnen lässt. Zunächst folgt eine Klärung des Begriffs Web Service.

4.1 Was ist ein Web Service?

Der Begriff des Web Service hat sich in den vergangenen 3 Jahren im Umfeld des World Wide *Web* (WWW) gebildet. Derzeit befasst sich das World Wide Web Consortium (W3C) mit einer genauen Definition der Web Services Architektur(vgl. [W3CWS 2002]) und deren Komponenten, wie z.B. SOAP.

Um zu klären, was ein Web Service ist, werde ich im folgenden einige Definitionen des Begriffs vorstellen, die diesen aus unterschiedlichen Perspektiven betrachten und anhand dieser eine allgemeine Definition des Begriffs Web Service ableiten.

Die erste Definition fasst sehr allgemein jede Art von Applikation, die über ein Netzwerk mit verschiedenen Protokollen ansprechbar ist, als Web Service auf:

> „Falls man eine Applikation über ein Netzwerk durch die Benutzung einer Kombination von Protokollen wie HTTP, XML, SMTP oder Jabber ansprechen kann, dann ist es ein Web Service.“
> (siehe [Snell et al. 2002, S. 1])

Die nächste Definition versucht zunächst den Begriff des Web Services ebenfalls in nur einem Satz zu fassen, wobei aber gezielt auf Softwaretechnische Konzepte wie lose Kopplung, Wiederverwendbarkeit usw. verwiesen wird:

> „Loosely coupled, reusable software components that semantically encapsulate discrete functionality and are distributed and programmatically accessible over standard Internet protocols.“
> (siehe [SleeperRobins 2001, S. 2])

Außerdem werden einige der Forderungen anschließend weiter ausgeführt. So wird im Zusammenhang mit „semantically encapsulated discrete functionality“ gefordert, dass ein Web Service seine Ein- und Ausgabeparameter so spezifiziert, dass andere Software diese erfragen kann und daraufhin in der Lage sein soll, den Web Service zu nutzen. Aus der Forderung „programmatically accessible“ wird weitergehend gefolgert, dass ein Web Service, im Gegensatz zu Web Sites und Desktop-Anwedungen, nicht zur direkten Interaktion mit dem Menschen gedacht ist und daher kein grafisches Benutzerinterface besitzt, sondern nur mit anderer Software auf der Ebene

von Programmaufrufen interagiert. Abschließend wird auf den sich ergebenden Vorteil der Nutzung von „standart Interntet protocolls“ hingewiesen. Da bereits weit verbreitete Protokolle wie das HTTP verwendet werden, ist die Infrastruktur bereits überall vorhanden.

In der folgenden Definition heißt es jedoch dieser widersprechend:

> „A Web service consumer can be a human user accessing the service through a desktop or wireless browser, it could be an application program, or it could be another Web service.“
> (vgl. [Manes 2001])

Es werden schließlich folgende 5 Punkte aufgeführt, die einen Web Service ausmachen und auf die zumindest theoretische Plattformunabhängigkeit hingewiesen: „

> 1. A Web service is accessible over the Web.
> 2. A Web service exposes an XML interface.
> 3. A Web service is registered and can be located through a Web service registry.
> 4. Web services communicate using XML messages over standard Web protocols.
> 5. Web services support loosely-coupled connections between systems.
>
> Perhaps what is most intriguing about Web services is that it doesn't matter what technologies are used to build Web services. Because Web services communicate over standard Web protocols using XML interfaces and XML messages, all Web services environments can interoperate--at least in theory.“
> (vgl. [Manes 2001])

Aus den vorangegangenen Definitionen ergeben sich die folgenden fünf Punkte, die nach meiner Meinung die grundlegenden Anforderungen an einen Web Service darstellen:

1. Es handelt sich um ausführbaren Programmcode
2. Dieser ist über ein Netzwerk (Web) mit einem Standardprotokoll ansprechbar (HTTP, SMTP, Jabber usw.) und die Daten werden in XML übertragen
3. Auf Grund der Übertragung der Daten in XML und der Verwendung von Standard Protokollen ist eine plattformübergreifende Verwendung von Web Services möglich
4. Durch seine Schnittstelle soll eine lose Koppelung des Programms möglich sein und dessen Wiederverwendbarkeit sichergestellt werden
5. Web Services können über Directory Services auffindbar sein, und es sollte eine manuelle oder automatische Einbindung des gefundenen Web Services in andere Systeme möglich sein

Punkt 5 ist bewusst vorsichtig formuliert, da es je nach Komplexität des Web Service fraglich ist, ob eine automatische Einbindung immer möglich sein wird. Dieses ist besonders dann der Fall, wenn nicht nur einfache Remote Procedure Calls durchgeführt werden, sondern komplette elektronische Dokumente als Nachricht („Message“) übertragen werden. Beides wird von der SOAP-Spezifikation unterstützt (vgl. [SOAP 2000]). Weiterhin ist es in der Fachliteratur umstritten,

ob wirklich jeder Betreiber von Web Services es wünscht, seinen Service über einen Directory Service allgemein und weltweit verfügbar zu machen. Dabei spielen mehrere Punkte eine Rolle. Zum einen ist es die Frage der bisher ungeklärten Abrechnung von Web Services gegenüber einem Kunden, zum anderen werden Web Services zum Teil innerbetrieblich eingesetzt, um verschiedene Systeme innerhalb einer Firma zu verbinden. Hier ist es nicht erwünscht, dass ein Web Service außerhalb dieser bekannt ist, hier sind nur lokale Lösungen von Interesse.

Bei der Verwendung eines Directory Service ergibt sich eine Architektur wie in Abbildung 6.

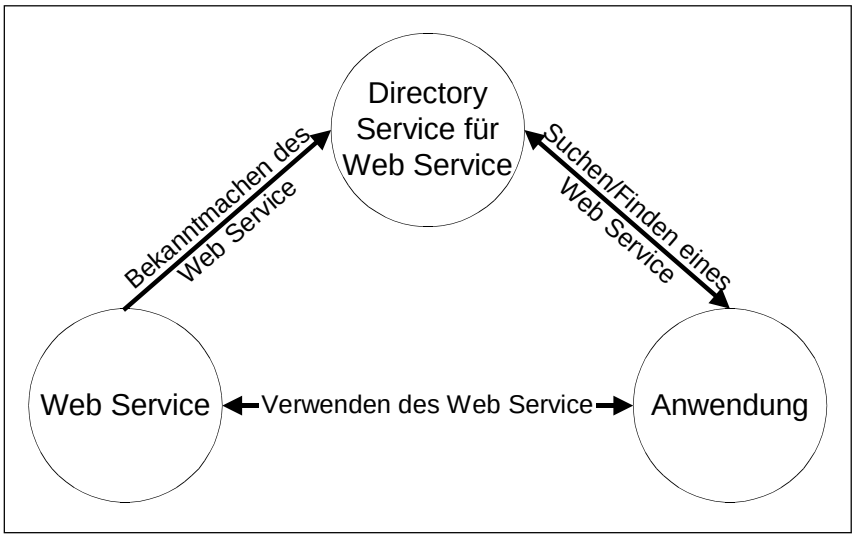

Abbildung 6 - Zusammenspiel von Web Service, Directory Service und Anwendung (vgl. [Snell et al. 2002, Abb.1-4, S. 5] und [LublinskyFarell 2001, Abb.1, S. 61])

Diese Architektur wird in drei Schritten verwendet:

1. Der Anbieter eines Web Services macht diesen in einem Directory Service bekannt.
2. Eine Anwendung bzw. deren Entwickler sucht im Directory Service durch Übermittlung einer genauen Spezifikation nach einem Web Service, der diese erfüllt und bekommt dessen Adresse und Aufrufspezifikation übermittelt.
3. Die Anwendung ruft unter Zuhilfenahme dieser Informationen den Web Service auf und verwendet ihn.

4.2 Aufbau eines Web Service

Nach der allgemeinen Definition von Web Services folgt hier zum besseren Verständnis ein Überblick über die gesamte Web Services Architektur, die dabei auf folgendem 5 Schichten Modell aufbaut (siehe Abbildung 7 und vgl. [Snell et al. 2002, Abb. 1-5, S. 5 und S. 7-9]):

1. *Discovery*
   Diese Schicht soll es ermöglichen, einen Anbieter für einen gesuchten Service zu finden. Die derzeit wichtigste Implementation ist das Universal Description, Discovery, and Integration (UDDI) System.

2. *Description*
   Diese Schicht soll Informationen zum verwendeten Netzwerk, Transport- und Verpackungsrotokoll eines Web Services liefern. Derzeitiger Standard ist die Web Service Description Language (WSDL).

3. *Packaging*
   Diese Schicht übernimmt das Verpacken der Nachrichten, die zwischen einem Web Service und dessen Kunden versendet werden, so dass beide Seiten die Nachrichten verstehen. Derzeit kommen hauptsächlich SOAP und XML-RPC zum Einsatz, wobei das w3c SOAP derzeit als Standard verabschiedet.

4. *Transport*
   Diese Schicht besteht aus den verwendeten Transport Protokollen (HTTP, SMTP, TCP, usw.)

5. *Network*
   Das Netzwerk, über das die Kommunikation erfolgt.

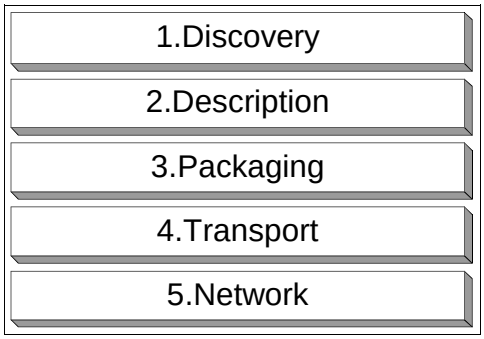

Abbildung 7 - 5 Schichten Modell der Web Service Architektur

Die Web Service Architektur hat zum Ziel, ein Zusammenspiel der einzelnen Komponenten zu erreichen, wie es in Abbildung 6 und 8 dargestellt ist.

Dies bedeutet, dass ein Serviceanbieter seinen Web Service in einem Serviceverzeichnis öffentlich bekanntmacht (z.B. UDDI). Jetzt kann ein Servicekonsument diesen finden und anhand von z.B. WSDL Informationen z.B. über SOAP einbinden und verwenden.

In den folgenden Punkten beschreibe ich den grundsätzlichen Aufbau eines Web Service:

- Es gibt ein Stück ausführbaren Programmcode in einer beliebigen Programmiersprache, der den eigentlichen Service in Form eines Algorithmuses realisiert
- Es gibt einen über ein Netzwerk erreichbaren Proxy (Web Service) für diesen Programmcode
- Dieser Web Service ruft in der Sprache, in der der eigentliche Programmcode realisert ist, diesen auf und liefert die Rückgaben des Programmcodes an die aufrufende Anwendung zurück
- Die Aufrufe finden via SOAP bzw. XML-RPC, Programmiersprachen unabhängig, in XML statt.

- SOAP bzw. XML-RPC kommunizieren dabei über ein Netzwerk in einem weit verbreiteten Protokoll, wie z.B. HTTP, SMTP, Jabber, usw. mit ihrem jeweiligen Gegenpart in der entfernten Anwendung
- Ein SOAP-Service hat Informationen darüber, welche Web Services durch ihn verfügbar sind und über welche Schnittstellen sie verfügen
- Die Anwendung, die den Web Service verwendet, kann in jeder beliebigen Programmiersprache programmiert sein
- Sie benötigt eine Implementation der benutzten Kommunikationsservices (SOAP oder XML-RPC)
- Um einen Web Service verwenden zu können, benötigt eine Anwendung nur Informationen über den „Ort“ des SOAP-Services, sowie die Parameter und den Namen der Schnittstelle des zu verwendenden Web Services

Daraus ergibt sich eine Kommunikation, wie sie in Abbildung 8 dargestellt ist.

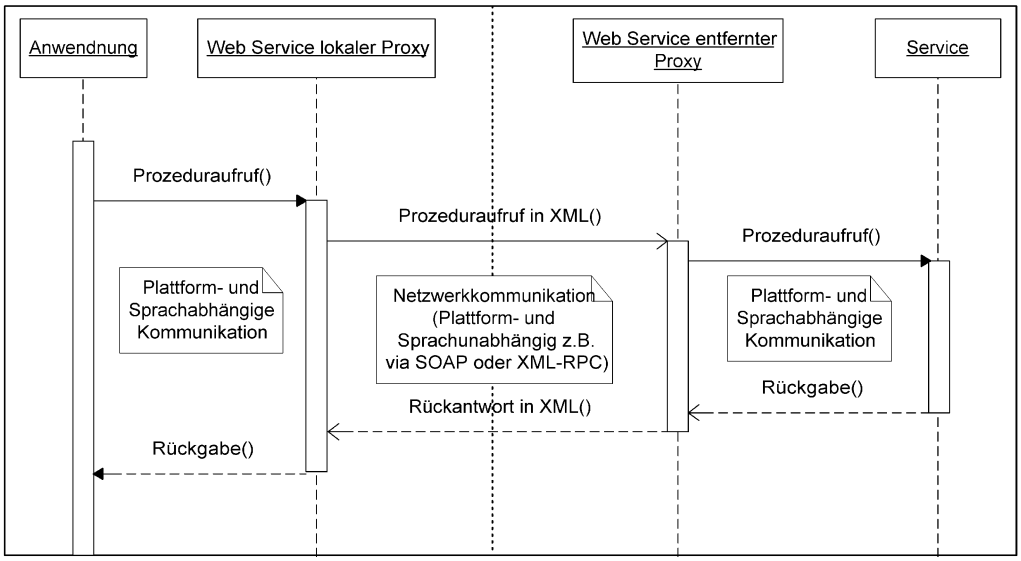

Abbildung 8 - Kommunikation von Web Service und Anwendung

4.3 Einordnung des Architekturmodells dieser Arbeit im Hinblick auf Web Services

Wenn man die ersten drei Punkte der von mir oben festgehaltenen Web Service Definition betrachtet, so kann man meine Webapplikationen als Web Service ansehen:

Zu Punkt 1: In der Regel wird es sich bei Webapplikationen um Applikationen mit CGI Scripten oder Java Code handeln. Im Extremfall kann eine Webapplikation allerdings mal zu einer XML-Dokument im Filesystem eines Webservers degenerieren, welches der Aussage widerspräche, dass es sich um ausführbaren Programmcode handelt.

Zu Punkt 2: Zur Kommunikation verwende ich einfache HTTP-Aufrufe oder alternativ SOAP. Die Daten sind dabei in einem von mir definierten XML-Format kodiert und werden

entweder als einfaches XML-Dokument übertragen oder bei der Verwendung von SOAP in diesem verpackt.

Zu Punkt 3: Durch die Verwendung von XML-Dokumenten zur Übertragung der Inhalte und Informationen zu dem jeweiligen Zustand der Webapplikationen und der Verwendung von HTTP und SOAP ist auch mein Architekturmodell grundsätzlich plattformunabhängig. Mein Framework für Webinformationssysteme ist in Java implementiert, es ist aber auch eine Implementation in anderen Sprachen denkbar.

Zu Punkt 4: Das Ziel dieser Arbeit ist es, ein Architekturmodell für die flexible Einbindung von Webapplikationen in Webinformationssysteme zu schaffen, das es ermöglicht, Webapplikationen problemlos in verschiedenen Webinformationssystemen einsetzten zu können, also eine lose Kopplung einer Webapplikation zum Portal mit einer möglichst großen Wiederverwendbarkeit zu erreichen.

Zu Punkt 5: Ich benutze keinen Directory Service, es wäre aber möglich. Dies ergibt sich im besonderen dadurch, dass in meinem Ansatz ein Anwender meines Frameworks zum Verwenden einer fremden Webapplikation im Normalfall nur deren Start URL benötigt. Ein Directory Service müsste eine Beschreibung, das verwendete Protokoll der Webapplikation, deren Start URL und falls benötigt, Parameter für den Aufruf des Startzustandes beinhalten.

Da Web Services SOAP bzw. XML-RPC verwenden, habe ich unter anderem SOAP als möglichen Transportservice implementiert, auch wenn die Verwendung bei meinem Ansatz keinen großen zusätzlichen Gewinn bringt, da ich nicht nur einfache Funktionsaufrufe mit Werten als Rückgabe durchführe, sondern komplette XML-Dokumente von einer Webapplikation zurück erhalte.

Der Einsatz von SOAP führt nur dazu, dass das Webapplikations XML-Dokument in ein SOAP-XML-Dokument eingepackt wird, was zu einem zusätzlichen Overhead an Daten führt.

Der einzige Vorteil den SOAP mitbringt, ist sein Typkonzept für die Parameter der Funktionsaufrufe und deren Rückgabewerte und das sprachübergreifend.

Zusätzlich benötigt man für die eingesetzten Sprachen bei Verwendung von SOAP jeweils einen Proxy auf den Anwendungssystemen.

Bei der Verwendung meiner einfachen HTTP-Lösung benötigt man hingegen nur meine Software auf dem integrierenden Webinformationssystem und keinen zusätzlichen SOAP Proxie auf den Systemen mit den Webapplikationen.

Der Unterschied in der Zielsetzung des Web Service-Ansatz zu meinem Modell ist ‚dass es bei den Web Services hauptsächlich darum geht, Remote Procedure Calls durchzuführen, um einen Service zu nutzen. Daher die Verwendung der Protokolle XML-RPC und SOAP. Bei meinem Architekturmodell geht es aber um mehr. Es geht darum, eine Möglichkeit zu schaffen, um Eingaben und Aufrufe von Folgezuständen einer Webapplikation in einem Webinformationssystem entgegenzunehmen und an die entsprechende Webapplikation in einem anderen System weiterzuleiten. Dabei sollen andere gleichzeitig in diesem Webinformationssystem eingebundene Webapplikationen jedoch nicht beeinträchtigt werden und ihren Zustand behalten. Dazu kommt ein dem Web Services vergleichbares System zur Übertragung dieser Daten an die Webapplikationen sowie ein Rücktransport der Antworten mit den Informationen über deren neuen Zustand an das Webinformationssystem. Außerdem bietet das Modell einen Ansatz, die Inhalte einer Webapplikation, die deren Zustand wiedergeben, ohne Layoutinformationen zu übertragen und im Webinformationssystem nach HTML zu transformieren und dort mit Layoutinformationen zu versehen.

Abschließend ist festzuhalten, dass die Webapplikationen in meinem Architekturmodell den Web Services vergleichbar sind, wie oben zu den Punkten 1 bis 5 festgestellt. Ansonsten geht mein Modell über das des Web Services hinaus und ist als eine Anwendung desselben zu verstehen.

In diesem Kapitel habe ich den Begriff des Web Services vorgestellt und eine aus 5 Punkten bestehende Liste von Anforderungen, die einen Web Service charakterisieren, erarbeitet. Anschließend habe ich die dem Begriff zugrunde liegende Architektur vorgestellt und meinem Architekturmodell mit Webappliaktionen gegenübergestellt. Dabei bin ich zu dem Schluss gekommen, dass mein Ansatz als eine konkrete Implementation eines Web Services zu betrachten ist, obgleich ich nicht alle Schichten der Web Services Architektur ausnutze beziehungsweise in der vorliegenden Implementation unterstütze. Im nächsten Kapitel werde ich den Web Services verwandten Begriff des Fachlichen (Domain) Services näher betrachten.

5 Fachliche (Domain) Services

Nach dem nun eine grundsätzliche Klärung des Begriffs Web Service und eine Einordung meines Webap Frameworks in diesem Zusammenhang im vorigen Kapitel erfolgt ist, gibt es noch einen weiteren zu betrachtenden Service Begriff: Den des Fachlichen (bzw. Domain) Service (siehe [OttoSchuler 2000], [Lipert et al. 2001]). Außerdem ist die Positionierung von Web Service und Webapplikation im Zusammenhang mit diesem zu untersuchen. Doch zunächst, was ist ein „Fachlicher Service“ und in welchem Kontext ist dieser Begriff entstanden?

5.1 Das Problem der Trennung von Funktion und Interaktion

Der Begriff des Fachlichen Service ist im Kontext des Werkzeug & Material-Ansatz (WAM) [Züllighoven 1998] erarbeitet worden. Bei der Entwicklung von Anwendungen, die Benutzungsschnittstellen auf verschiedenen Plattformen wie Web, Desktop, Mobile, usw. besitzen, stellte es sich heraus, dass eine der zentralen Forderungen des WAM Ansatzes, die der Trennung der fachlichen Funktionskomponente FK von der Benutzeroberfläche (Interaktionskomponente) IAK, in diesem Zusammenhang mit den bisherigen Methoden nicht ausreichend unterstützt wurde. Wie sich zeigte, war die Trennung von FK und IAK nicht deutlich genug, um Oberflächen mit verschiedenen Benutzungsmodellen die gemeinsame Nutzung einer FK zu ermöglichen. Ein besonderes Problem in diesem Zusammenhang stellt das Verhalten von Oberflächen basierend auf dem HTTP-Protokoll dar, da dies standardmäßig nur ein „request – response“ Verhalten von Seiten der IAK her zulässt. Das bedeutet, dass keine automatische Benachrichtigung von Seiten der FK her erfolgt. Dies führt vor allem bei der nebenläufigen Verwendung von Materialien zu Inkonsistenzen, da die Änderungen nicht sofort in allen IAKs sichtbar werden, sondern erst nach einem erneuten „request“ dieser IAKs von der FK her eine „response“ mit den geänderten Materialien erfolgen kann.

Zur Begriffsklärung hier einige wichtige Definitionen des WAM Ansatzes nach [Züllighoven 1998, S. 6 und S. 236]:

- *Werkzeug*
  „...unterstützt wiederkehrende Arbeitsabläufe und -handlungen. ... Ein Werkzeug wird von seinem Benutzer je nach den Erfordernissen einer Situation gehandhabt... Als Softwarewerkzeug ermöglicht es den interaktiven Umgang mit den Arbeitsgegenständen.“
- *Material*
  „... Arbeitsgegenstände die schließlich zum Arbeitsergebnis werden. ...Softwarematerialien verkörpern 'reine' anwendungsfachliche Funktionalität. ...“
- *Automat*
  „... erledigt eine vorab vollständig festgelegte Aufgabe und produziert ein definiertes Ergebnis. ...“
- *Funktionskomponente (FK)*
  „...ist der bewirkende und sondierende Teil des Werkzeugs. In ihr wird die fachliche Funktionalität des Werkzeugs festgelegt. ... Um den Arbeitszusammenhang unterstützen zu können, verwaltet die FK einen Arbeitszustand...“

- *Interaktionskomponente (IAK)*
  „... legt die Benutzungsschnittstelle des Werkzeugs fest. Dazu nimmt sie Ereignisse entgegen, ruft die FK und steuert die Präsentation an der Oberfläche. ...“

5.2 Was genau ist ein Fachlicher Service?

Ein Fachlicher Service ist nach den Definitionen von [OttoSchuler 2000] bzw. [Lipert et al. 2001] Software, die eine Kapselung fachlichen Wissens darstellt, die über eine genau definierte Schnittstelle ansprechbar ist und so einem Konsumenten einen speziellen Service anbietet. Dabei soll dieser Serviceanbieter keinerlei Annahmen über den ihn verwendenden Konsumenten (GUI, anderer Service) machen. Gleichzeitig soll der Service eine parallele Benutzung durch mehrere Konsumenten gleichzeitig unterstützen. Des Weiteren sollte ein Session Mechanismus vorhanden sein, um über einzelne Aufrufe hinweg arbeiten zu können. Dies ergibt sich aus der Tatsache, dass ein Fachlicher Service aufgerufen wird, seinen Service erbringt und sich anschließend beendet. Schließlich soll ein Fachlicher Service durch seine Schnittstelle seine Wiederverwendbarkeit erleichtern und die Implementation austauschbar machen.

Dies wird mit den folgenden fünf Anforderungen auf den Punkt gebracht:

1.) „Fachliche Services sollen die fachliche Funktionalität und das fachliche Wissen eines Anwendungsbereichs zusammengefasst und gekapselt anbieten.“

2.) „Fachliche Services sollen die fachliche Funktionalität eines Anwendungsbereichs handhabungs und präsentationsunabhängig anbieten.“

3.) „Fachliche Services sollen die fachliche Funktionalität eines Anwendungsbereichs mehrbenutzerfähig und verteilt anbieten.“

4.) „Fachliche Services sollen die fachliche Funktionalität eines Anwendungsbereichs mit einem der Aufgabe und technischen Umgebung angemessenen Zustands- und Sitzungsmodell anbieten.“

5.) „Fachliche Services sollen die fachliche Funktionalität eines Anwendungsbereichs wieder verwendbar und austauschbar anbieten.“
(siehe [OttoSchuler 2000, S. 160/1])

Die Tabelle 3 zeigt wie eine Systemarchitektur aus 3 Schichten mit Fachlichen Services aussehen kann.

Schicht	Aufgaben der Schicht
Benutzungsschnittstelle	Stellt dar und interagiert
Interaktionsschicht	• Ansteuerung Benutzungsschnittstelle • Übersetzt Wissen über Umgang mit Fachlichen Services • Koordination zwischen mehreren Fachlichen Services
Fachliche Services	Kapseln und bündeln fachliche Funktionalität und fachliches Wissen

Tabelle 3 - Systemarchitektur mit Fachlichen Sevices nach [OttSch 2000, Abb. 17, S. 69]

5.3 Relation von Fachlichen Services und Web Services

Nach der Klärung der Begriffe des Fachlichen und Web Services, bietet es sich an, darüber nachzudenken, in welcher Relation die beiden zueinander stehen. Bei den Web Services handelt es sich um das allgemeinere Konzept der beiden. Fachliche Service verstehen sich speziell als ein Modell im Rahmen des WAM-Ansatzes. Fachliche Services sollen speziell das Fachwissen und die Handlungslogik eines Anwendungsbereiches kapseln. Dagegen wird beim Web Services Modell keinerlei Annahme darüber gemacht, welche Art von Service erbracht werden wird. Das Web Service Modell ist spezieller, da es eine komplette Infrastruktur definiert, auf deren Grundlage ein Service implementiert werden kann. Der Schwerpunkt liegt dabei auf Inter- und Intranet Architekturen, die das HTTP-Protokoll verwenden, obwohl grundsätzlich andere Protokolle möglich sind. All dies spricht nicht dagegen, einen Fachlichen Service in der Web Services Architektur zu implementieren. Die Web Services Architektur bietet eine gute Möglichkeit, Fachliche Services zu implementieren. Web Services mit RPC aufrufen via SOAP oder XML-RPC z.B. bieten eine sauber getypte Schnittstellen Definition und fördern die lose Koppelung der einzelnen Komponenten. Die Wiederverwendbarkeit und Austauschbarkeit eines einzelnen Service wird durch Beschreibungssprachen wie WSDL gefördert.

5.4 Relation von Fachlichen Services und meinen Webapplikationen

In welcher Relation mein Ansatz zu den Fachlichen Services steht, zeigt Tabelle 4. In ihr sieht man, wie eine Architektur unter Verwendung meines WebAp Frameworks und dem Einsatz von Fachlichen Services aussehen würde. Die Webapplikation kann aber jeden beliebigen anderen internen Aufbau besitzen.

Wenn man sich Tabelle 3 und Tabelle 4 zum Vergleich der Beispielarchitekturen ansieht, so stellt man fest, dass die Schicht der Fachlichen Services für die Kapselung der fachlichen Dienstleistungen in der Architektur sorgt.

Mein Webapplikations Framework soll dagegen mehrere Softwaresysteme (Webapplikationen) mit jeweils einer eigenen Weboberfläche in eine zusammenfassende gemeinsame Weboberfläche integrieren helfen. Diese integrierende Weboberfläche wird im Normalfall ein Webinformationssystem sein. Dabei ist es das Ziel meines Ansatzes, den Inhalt der Webseiten einer Webapplikation möglichst ohne Layoutinformationen, aber mit den für eine Interaktion notwendigen Informationen in XML-Dokumenten zu übertragen. Dies ist die Voraussetzung dafür, dass eine Webapplikation ohne großen Aufwand des Webapplikationsanbieters in verschiedene Webinformationssysteme integriert werden kann und auch Änderungen an der Webapplikation nicht immer den Betreiber des Webinformationssystem dazu zwingen, Änderungen am Webinformationssystem vorzunehmen, um die Applikation weiterhin nutzen zu können. Durch diesen Ansatz benötigt der Webinformationssystembetreiber kein fachliches Wissen über die Oberfläche der Anwendung. Er kann sich daher voll und ganz auf das eigene Webinformationssystem und dessen Aussehen und Aufbau konzentrieren.

Fachliche Services und mein WebAp Framework haben beide die Zielsetzung, fachliches Wissen zu kapseln, setzen aber an verschiedenen Ebenen an. Während Fachliche Services die eigentlichen Handlungen mit Werkzeugen und Materialien betreffen, beinhaltet mein Ansatz die Trennung von Webapplikationen und Webinformationssystemen. Dies bedeutet eine Trennung der Systeme, die Oberflächeninhalte erzeugen und Eingaben verarbeiten, von denen, die das eigentliche Layout einer Oberfläche generieren und die Benutzereingaben entgegennehmen.

	Schicht	Aufgaben der Schicht
	Benutzungsschnittstelle	Stellt dar und interagiert
	Integrationsschicht (Webinformationssystem mit WebAp Framework)	Benutzt mein WebAp Framework um: • Die Webapplikation zu integrieren • Für den korrekten Kontrollfluss zwischen den Webapplikationen zu sorgen • HTML für die Weboberfläche aus den Inhalten der Webapplikationen und den Layoutvorgaben des Webinformationssystems zu generieren
Webapplikation	*Interaktionsschicht*	• Erstellt den XML-Code für die Integrationsschicht • Übersetzt Wissen über den Umgang mit Fachlichen Services • Koordination zwischen mehreren Fachlichen Services
	Fachliche Services	Kapseln und bündeln fachliche Funktionalität und fachliches Wissen

Tabelle 4 - Webapplikations Architektur

In diesem Kapitel habe ich das Konzept des Fachlichen (domain) Services vorgestellt und mit Web Services und meinem WebAp Framework verglichen. Dabei bin ich zu dem Ergebniss gekommen, dass Fachliche Services durch Web Services implementiert werden können und in einer Software Architektur mit meinem WebAp Framework auf verschiedenen Ebenen zusammen eingesetzt werden können. Im nächsten Kapitel werde ich basierend auf den Erkenntnissen der vorherigen Kapitel meinen Lösungsansatz für das Problem der flexiblen Einbindung von Webapplikationen in Webinformationssysteme vorstellen.

6 Das Architekturmodell

In diesem Kapitel werde ich meine Lösung des Problems der „flexiblen Einbindung von Webapplikationen in Webinformationssysteme“ vorstellen und genauer untersuchen. Mein Lösungsansatz besteht aus zwei Teilen. Zum einen, dem Java WebAp Framework, welches für die Integration der einzelnen Webapplikationen in einem Webinformationssystem zuständig ist und zum anderen, aus der WebAp-DTD, die den Aufbau der XML-Dokumente beschreibt, mit denen die Webapplikationen mit meinem WebAp Framework kommunizieren.

Doch zunächst die Anforderungen, die sich aus den vorhergehenden Kapiteln ergeben haben.

6.1 Die Anforderungen

Wie ich bereits im Kapitel 3, S. 28 festgestellt habe, entstehen mehrere zu beachtende Probleme bei der flexiblen Einbindung von Webapplikationen in Webinformationssysteme.

Hieraus ergeben sich direkt die folgenden Anforderungen:

1. Der Anbieter einer Webapplikation sollte nicht jedes Mal für einen neuen Einsatzkontext Änderungen an dieser vornehmen müssen. Das heißt, dass das Layout einer Applikation ohne Eingriff in den Applikationscode selbst änderbar sein muss. Daraus folgt Layout und Inhalte zu trennen, um so die Inhalte ohne Programmlogik an das Webinformationssystem übertragen zu können.
2. Um beliebige Webapplikationen in einem Webinformationssystem einsetzen zu können, muss der Kontrollfluß und der Namensraum der Variablen der Applikation in verschiedenen Einsatzkontexten gewährleistet sein. Dies bedeutet unter anderem, dass Eingaben im Portal an einer bestimmten Webapplikation nicht zu unkontrollierten Änderungen an anderen Webapplikation führen dürfen.
3. Eine Webapplikation sollte weiterhin dezentral auf den Servern ihres Herstellers laufen können, damit dieser die volle Kontrolle über seine Applikation und damit sein Know-how behalten kann.

Bei weiteren Überlegungen ergeben sich aus wirtschaftlicher und praktischer Sicht noch die folgenden Punkte:

4. Die verwendeten Kommunikationstechniken sollten nach Möglichkeit keine zusätzlichen Hard- und Softwaresysteme erfordern, sich also in bestehende Systeme integrieren und keine weiteren Kosten erzeugen.
5. Für den Betreiber eines Webinformationssystems ist es außerdem von Interesse, dass er den Zustand der integrierten Webapplikationen zwischen den Aufrufen innerhalb einer Session cachen kann. So wird unnötige Kommunikation über das Netzwerk vermieden und Kosten und Zeit gespart.
6. Auf Grund der Vielzahl der im Web verwendeten Technologien (vgl Kapitel 3.4, S. 28 und [Turau 1999]) zum Realisieren und Einsetzen von Webapplikationen muss eine Form der Integration und Kommunikation gewählt werden, die es ermöglicht, diese Technologien zu verbinden.

Wie im Kapitel 5 dargelegt, geht meine Lösung für Webapplikationen dabei über den bisher verwendeten Ansatz für die Trennung von Funktion und Interaktion, wie er im WAM-Anstatz (basierend auf dem Model View Controler Paradigma siehe [Gamma et al. 1996, S. 4-7] und [Züllighoven 1998, S. 234ff]) und Fachlichen Services verwendet wird, noch hinaus und trennt die Interaktionskomponente noch einmal auf. In meinem Ansatz wird die Interaktionskomponente dabei so aufgeteilt, dass der eine Teil alles fachliche Wissen enthält und der andere Teil lediglich eine grafische Aufbereitung desselben durchführt. Dieser Teil bietet zusätzlich eine Unterstützung zur Integration mehrerer Webapplikationen in ein Webinformationssystem. Zur Verdeutlichung der dabei entstehenden Architektur siehe die folgende Tabelle:

Schicht	Aufgaben der Schicht
Webbrowser	Stellt dar und interagiert
Webinformationssystem	Benutzt mein WebAp Framework um: • Webapplikationen zu integrieren • Für den korrekten Kontrollfluss zwischen den verschiedenen Webapplikationen zu sorgen • HTML für die Weboberfläche aus den Layout-vorgaben des Webinformationssystems zu generieren
Webapplikation	• Erzeugt die anzuzeigenden Inhaltsseiten als XML ohne genaue Layoutvorgaben • Stellt eine Schnittstelle für das WebAp Framework zur Verfügung (z.B. über HTTP oder SOAP) • Kapselt das fachliche Wissen der Applikation, das zur Darstellung notwendig ist und abstrahiert von tiefer liegenden Schichten

Tabelle 5 - Webapplikations Architektur

Aus diesen Anforderungen ergibt sich zum einen, dass die Kommunikation zwischen Webapplikation und Webinformationssystem ein zentraler Aspekt ist und zum anderen, dass die Hauptarbeit der Integration der Webapplikationen auf Seiten des Webinformationssystems durch die Lösung gestützt stattfindet.

6.2 Die Auswahl der Kommunikationstechnologien

Es erschien mir sinnvoll, dass sich die Webapplikationen, sowohl was ihren Aufruf als auch die Rückgabe ihrer Daten betrifft, möglichst nah am Design gewöhnlicher Webapplikationen orientieren, auf Basis von HTML unter Verwendung des HTTP-Protokolls, insbesondere an der seitenartigen Struktur der übertragenen Daten. Zum einen erleichtert dies das Anpassen bestehender Webapplikationen und zum anderen besitzen die Webinformationssysteme, in die die Web-applikation integriert werden, ebenfalls einen auf Seiten basierenden Aufbau.

Auf Grund der Forderung nach einer Trennung von Inhalten und Layout und der Forderung nach der einfachen Anpassung des Layouts an ein Webinformationssystem, ist die direkte Verwendung von HTML ungeeignet. Eine Übergabe von Daten auf Basis einer Programmiersprache (durch RPC oder RMI) erschien mir ungeeignet, da hierdurch zum einen in den meisten Fällen eine Festlegung

auf eine spezielle Programmiersprache hätte erfolgen müssen und zum anderen auch noch die Kommunikation auf ein dazu gehöriges anderes Protokoll als HTTP festgelegt gewesen wäre[8].

Die Wahl eines anderen Protokolls als HTTP hätte zusätzlich zu einer Erschwerung der Kommunikation führen können, da Firewalls in den Firmen häufig nur Standard Protokolle wie HTTP ungehindert zulassen. Außerdem findet die Kommunikation bestehender eigenständiger Webapplikationen mit den Webbrowsern der Benutzer ohnehin über HTTP statt, so dass eine entsprechende Infrastruktur meistens bereits existiert.

Als Datenformat bei der Übertragung wird XML anstelle von HTML verwendet. Dies habe ich auf Grund der engen Verwandtschaft von HTML und XML und der inzwischen weiten Verbreitung von XML-Tools getan, außerdem wegen der guten Eigenschaften zur Beschreibung der enthaltenen Daten, sowie der Möglichkeit zur Konvertierung zwischen verschiedenen Formaten und der Verfügbarkeit in existierenden Programmiersprachen.

Die erste Variante für die Kommunikation zwischen Webinformationssystem und Webapplikation sieht folgendermaßen aus:

> Zuerst wird auf Seiten des Webinformationssystems durch das WebAp Framework per HTTP GET/POST eine Anfrage an eine Webapplikation gesendet. Diese ist entweder ein Aufruf einer Startseite oder eine durch eine Benutzereingabe initiierte Seitenanfrage.
>
> Im nächsten Schritt erzeugt die Webapplikation als Antwort auf die Anfrage in HTTP ein XML-Dokument. Dies beinhaltet einen „header“, in dem Metadaten über die Webapplikation angegeben werden können (Verfasser, Inhaltsangabe, Erstellungszeitpunkt, usw.). Dazu kommen die von diesem Zustand der Webapplikation aus erreichbaren Zustände und deren Parameter, sowie eine Liste aller in diesem Zustand existierenden Übergabeparameter, deren Vorgabewerte und deren Typ. Außerdem beinhaltet dieses Dokument als Hauptteil alle anzuzeigenden Daten und Eingabeaufforderungen, jedoch keine speziellen Layoutangaben, abgesehen von Angaben zur Strukturierung der Daten. Diese Antwort wird danach über HTTP zurückgesendet.
>
> Auf Seiten des Webinformationssystems werten Objekte der dafür implementierten Klassen des WebAp Frameworks das empfangene XML-Dokument aus und erzeugen durch XSL-Transformationen den auszugebenden HTML-Code. Ferner werden die Parameter und Adressen der möglichen Ziele verwaltet, die in diesem Zustand existieren. Nachfolgend ein kurzes Beispiel, wie ein übertragenes XML-Dokument ungefähr aussieht (siehe Listing 2):

Im Laufe der Arbeit stellte es sich als sinnvoll heraus, den Transport so zu implementieren, dass alternative Techniken einfach einzufügen sind. Als Ergebnis kann jetzt eine Webapplikation verschiedene Transportservices einsetzen. Standardmäßig stehen durch meine Implementation eine Kommunikation in SOAP und eine direkt in HTTP zur Verfügung.

8 Eine Ausnahme bildet z.B. SOAP, siehe Kapitel Web Services

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE WebAp:webapplication SYSTEM ...>
<WebAp:webapplication version="1.0" ...>
  <WebAp:header>
    <rdf:Description rdf:about="http://...">
      <dc:title>
        MeinWetter.de
      </dc:title>
      ...
    </rdf:Description>
    <WebAp:name creationtime="11.01.2003 17:07:19" updateinterval=""
     haslogin="true" isloggedin="false" >
      MeinWetter.de
    </WebAp:name>
    <WebAp:linkurl id="link1" url="http://..." transport="SOAP"
      parameter ="uid"/>
    <WebAp:linkurl id="link3" url="http://..." transport="DirektHTTP"/>
    <WebAp:targeturl id="target1" url="http://..." transport="SOAP"
      parameter="uid"/>
    <WebAp:parameter id="uid" value="" type="String"/>
  </WebAp:header>
  <WebAp:application>

Hier steht der eigentliche Inhalt, der angezeigt werden soll

  </WebAp:application>
```

Listing 2 – XML Codefragment – hervorgehoben sind dc Dublin Core Metadata, link – und targeturl und parameter Elemente bzw. Attribute von diesen

In Listing 2 können folgende Entwurfsentscheidungen betrachtet werden:

1. Im XML-Dokument findet eine Trennung zwischen dem darzustellenden Inhalt im Element „application“ und den zusätzlichen Programminformationen, um Funktionsaufrufe durchzuführen, im Element „header“ statt.

2. Im „header“ befinden sich die Informationen, um mögliche Funktionsaufrufe für Folgezustände durchzuführen („linkurls“ und „targeturls“) und eine Auflistung der in dem aktuellen Zustand der Webapplikation enthaltenen Variablen und Konstanten („parameter“).

3. Es findet eine Unterscheidung von Links („linkurls“) und Targets („targeturls“) statt. Bei Links handelt es sich um Informationen, um einen Funktionsaufruf ohne Parameter bzw. mit Konstanten als Parametern aus dem WebAp Framework durchzuführen. Bei den Targets hingegen handelt es sich um Informationen, um Funktionsaufrufe mit Variablen als Parametern zu erzeugen.

4. Parameter werden mit ihren Werten und Typen angegeben. Auf Grund der eingeschränkten Möglichkeiten zur Typisierung von Parametern in HTML und Webbrowsern, diese kennen nur Strings, unterstützt mein Framework zur Zeit nur die Basistypen Boolean, String, Integer und Real sowie Arrays von diesen.

6.3 Die Auswahl der Implementationssprache

Durch meine Entscheidungen für eine Kommunikation auf Basis von XML und HTTP zu Beginn der Planung gab es praktisch keine Einschränkung, was die Auswahl der Implementationssprachen betraf. So kamen sowohl Skriptsprachen, wie z.B. PHP, Pearl oder TCL, als auch Programmiersprachen wie z.B. Java oder C# in Frage. Da ich mich für einen objektorientierten Entwurf entschieden hatte, erschienen mir die Skriptsprachen mit ihren meist noch in den Anfängen stehenden Objektmodellen nicht geeignet. Um jedoch eine größt mögliche Plattformunabhängigkeit zu erreichen und auf Grund meines Vorwissens, erschien mir Java am geeignetsten. Dazu kommt, dass Java mit seiner J2EE Plattform[9] verschiedene Mechanismen für die Programmierung des Web anbietet. Diese reichen von Applets und Servlets (bzw. Java Server Pages) bis hin zu Enterprise JavaBeans. Mein WebAp Framework kommt ohne diese Erweiterungen aus, kann aber in Verbindung mit diesen genutzt werden. Ich verwende nur zur Demonstration der Verwendungsweise meines Frameworks Servlets. Für die Implementation des Frameworks selbst kommt Java 2 zum Einsatz. Unterstützt wird dies durch verschiedene Java Pakete des Apache Projekts zur Verarbeitung von XML und XSLT sowie dem SOAP-Protokoll. Außerdem setze ich noch die HTTPMessage Klasse aus dem „com.oreilly.servlet“ Paket für die direkte HTTP-Kommunikation ein. Dies sollte den Einsatz meines Frameworks in nahezu jedem Java 2 fähigen Web Content Management Systems und Applikationsserver ermöglichen.

6.4 Die Aufgaben des WebAp Frameworks

Nach der Klärung des Datenformates, der zu verwendenden Kommunikationstechnologien und der Programmiersprache bleibt noch darzulegen, wie die anderen Anforderungen durch mein WebAp Framework gelöst werden. Es muss ein Transportservice angeboten werden. Dies war, wie oben beschrieben, zunächst ein auf HTTP beschränkter Service, der im Verlauf zu einem erweiterbaren Serviceanbieter ausgebaut worden ist, der zusätzlich über das SOAP-Protokoll verfügt.

Um die empfangenen XML-Dokumente zu verarbeiten, ist ein entsprechender Service von Nöten. Dieser transformiert die inhaltlichen Daten in HTML und passt die Verweise innerhalb des Zustands der Webapplikation an das umgebende Webinformationssystem an.

Die Transformation erfolgt unter Verwendung von XSL-Templates. So kann das Layout der jeweiligen Webapplikation geändert werden, ohne das der Programmcode geändert werden muss. Außerdem werden die link- und targeturls sowie die jeweils benötigten Parameter inklusive eventueller Vorgabewerte ausgelesen.

Zur Koordinierung des Transports und der Verarbeitung des XML gibt es das WebAp Objekt. Zusätzlich sorgt es dafür, dass die Daten durch einen Sessionmanager gespeichert und wieder beschafft werden. Es stellt die öffentliche Schnittstelle dar, über die auf die einzelnen Webapplikationen durch das Framework zugegriffen werden.

Der implementierte Session Manager für die WebAp Objekte ist dazu gedacht, entweder in eine javax.servlet.http.HttpSession oder eine Datenbank zu speichern. Als Beispielimplementation für das Speichern in Datenbanken ist eine MySQL Anbindung realisiert.

Den Abschluss des WebAp Frameworks nach außen zu einem Webinformationssystem hin stellt schließlich der für die Integration der einzelnen WebAps zuständige WebApConstructor dar. Dieser erhält vom Webinformationssystem eine Liste der zu verwendenden Webapplikationen, das zu verwendende Session Objekt und, falls vorhanden, Parameter aus Benutzeraufrufen. Diese ordnet er über die verwendeten Namen dem jeweiligen WebAp Objekt zu und fordert dieses dazu auf, den

9 Java 2 Platform, Enterprise Edition

entsprechenden nächsten Zustand der Webapplikation aufzurufen. Anschließend kann das Webinformationssystem den aktuellen HTML-Code des Zustands der Webapplikation abrufen.

Nach dem ich in diesem Kapitel die Anforderungen an meinen Lösungsansatz und die daraus resultierende Lösung dargelegt habe, werde ich im nächsten Kapitel die einzelnen Klassen näher vorstellen. Dazu werde ich die entsprechenden Klassendiagramme zeigen und verwendete Entwurfsmuster benennen. Abschließend gehe ich auf meine Beispiel Webapplikationen ein.

7 Das WebAp Framework

In diesem Kapitel werde ich die Implementation meines WebAp Framework Prototyps darlegen, auf einige der Klassen näher eingehen und die Struktur der Lösung anhand von Klassendiagrammen erläutern.

Bei dem Prototypen handelt es sich um ein Pilotsystem (vgl. [Kieback et al. 1992]), welches in einem evolutionären Prozeß (vgl. Evolutionäres Prototyping [Floyd et al. 1984]) entstanden ist.

Zunächst die Definition des Begriffs des Frameworks nach [Gamma et al. 1996, S. 31-32 und S. 396]:

> „Ein Framework ist eine Menge kooperierender Klassen, welche die Elemente eines wiederverwendbaren Entwurfs für eine bestimmte Art von Software darstellen. Ein Framework bietet eine Architekturhilfe beim Aufteilen des Entwurfs in abstrakte Klassen und beim Definieren ihrer Zuständigkeiten und Interaktionen. Ein Entwickler passt das Framework für eine bestimmte Anwendung an, indem er Unterklassen der Frameworkklassen bildet und ihre Objekte zusammensetzt.“

7.1 Die verwendeten Entwurfsmuster

Bei dem vorliegenden Entwurf habe ich mich auf Entwurfsmuster, wie sie in [Gamma et al. 1996] beschrieben werden, gestützt. In der folgenden Tabelle 6 findet sich eine Übersicht über diese:

Aufgabe	Entwurfsmuster
Erzeugungsmuster	• Abstrakte Fabrik [Gamma et al. 1996, S. 93-102] • Erbauer [Gamma et al. 1996, S. 103-113] • Singleton [Gamma et al. 1996, S. 139-146]
Strukturmuster	• Adapter [Gamma et al. 1996, S. 151-163] • Kompositum [Gamma et al. 1996, S. 213-225]
Verhaltensmuster	• Strategie [Gamma et al. 1996, S. 333-343]

Tabelle 6 - Verwendete Entwurfsmuster

7.2 Der Aufbau des Frameworks

Das WebAp Framework besteht hauptsächlich aus den folgenden Klassen und Interface Definitionen (Namen von Interface Definitionen enden auf `I`):

- `WebApConstructor`
- `WebApFactoryI` und einer Bsp. Implementation `WebApFactory`
- `WebApI` und einer Bsp. Implementation `WebAp`
- `XMLServicesFactoryI` und einer Bsp. Implementation `XMLServicesFactory`
- `XMLServicesI` und einer Bsp. Implementation `XMLServices`

- `MessageServiceProviderI` und den Bsp. Implementationen `Messaging-Service`, `DirectHTTPService` und `SOAPService`
- `SessionI` und den Bsp. Implementationen `HTTPServletSessionMgr` und `DBSessionMgr`
- `SessionDatabaseI` mit der Bsp. Implementation `MySQLSessionDatabase`

Dazu kommen noch einige kleinere Klassen zur Unterstützung,die die Objekte zum Datenaustausch zwischen den obigen Klassen und nach außerhalb des Frameworks implementieren:

- `DCDataContainer` – dient der Aufbewahrung der Dublin Core Meta Daten einer Webapplikation, nach dem Auslesen aus einem XML-Dokument.
- `MyMultiProperties` – ermöglicht es, angelehnt an die `java.util.Properties` Klasse, mehrere benannte Werte zu einem Schlüssel zu speichern, anstelle nur eines einfachen Schlüssel - Wert Paares.
- `Parameter` – beinhaltet alle Informationen über einen Übergabeparameter einer Webapplikation. Dessen für die Webapplikation eindeutige Parameter-ID, sowie dessen Typ und Wert repräsentiert als `String`.
- `URLContainer` – enthält alle Informationen zu einer vom aktuellen Zustand der Webapplikation aufrufbaren URL. Dies sind deren eindeutige ID, der Name des zu verwendenden Kommunikationsservice und die IDs der benötigten Parameter.

Die Abbildung 9 zeigt nach UML-Notation als Klassendiagramm eine Übersicht über die wichtigste Klasse und Schnittstellen des Frameworks, die in den folgenden Unterkapiteln näher erläutert werden. Im Diagramm ist die Abhängigkeit der einzelnen Klassen und deren zu erzeugenden Objekte zu sehen. `WebApConstructor` und `WebApI` nehmen dabei zentrale Rollen ein. Objekte einer konkreten `WebApI` Implementationen stellen dabei die lokale Schnittstelle zu einer entfernten Webapplikation und koordinieren alle Operationen an dieser. Der WebApConstructor wiederum erzeugt und koordiniert alle im Webinformationssystem vorhandenen Repräsentationen von entfernten Webapplikationen.

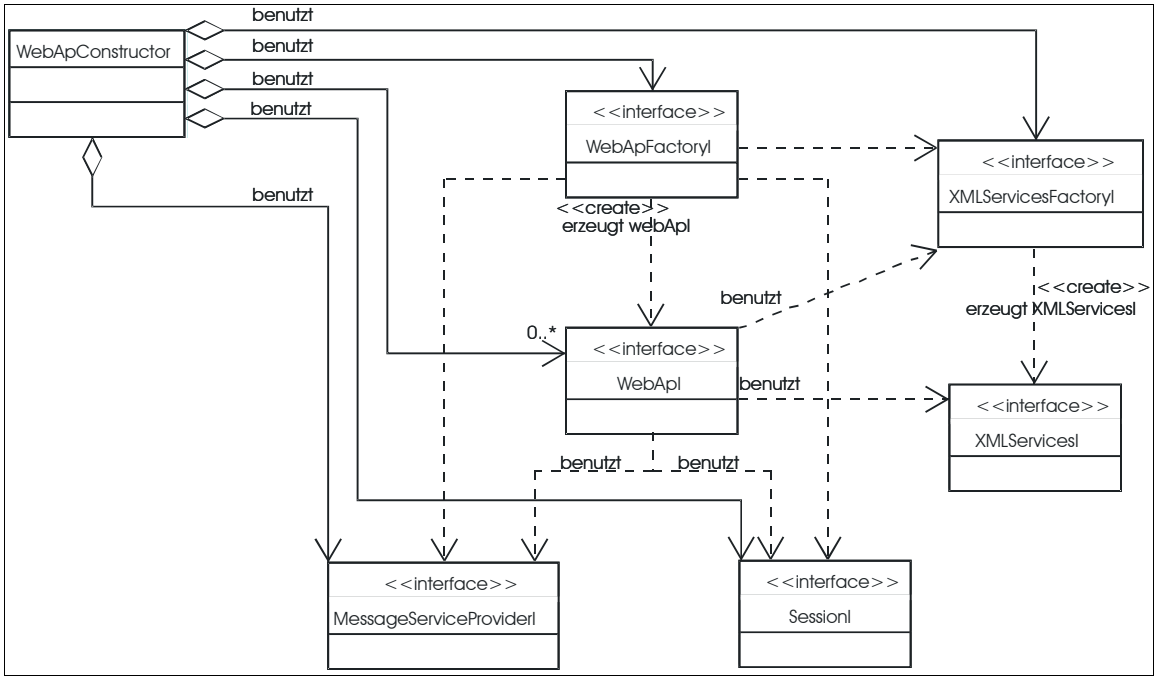

Abbildung 9 - Zentrale Klasse und Schnittstellen

7.2.1 Die WebApConstructor Klasse

Beim `WebApConstructor` handelt es sich um die zentrale Klasse des WebAp Frameworks. Ein Objekt dieser Klasse ermöglicht es, mehrere Objekte des Typs `WebApI` zu erzeugen und zu verwalten. Über seine Schnittstelle können ihm dazu verschiedene *Fabriken* – vom Typ `WebApFactoryI` und `XMLServicesFactoryI` – übergeben werden. Des Weiteren können ihm verschiedene *Strategien* für die Kommunikation über das Netz – vom Typ `MessageServiceProviderI` – und die Speicherung – vom Typ `SessionI` – übergeben werden. Standardmäßig verwendet ein als *Erbauer* fungierendes `WebApConstructor` Objekt dabei meine Beispielimplementationen. Dies sind die *Fabriken* `WebApFactory` und `XMLServicesFactory`, sowie der als *Kompositum* fungierende `MessagingService`, der die Kommunikations-*Strategien* `DirectHTTPService` und `SOAPService` anbietet. Bei der Erzeugung eines `WebApConstructor` Objekts wird diesem ein konkretes Objekt der zu verwendenden Speicherungs-*Strategie* vom Typ `SessionI` übergeben. Als Beispielimplementationen stehen der `HTTPServletSessionMgr` und der `DBSessionMgr` zur Verfügung. Letzterer benutzt als *Adapter* für die tatsächlich verwendete Datenbank ein Objekt vom Typ `SessionDatabaseI`. Als einen solchen Adapter biete ich die Klasse `MySQLSessionDatabase` für die Verwendung einer MySQL-Datenbank an. Die Namen der Webapplikationen und andere Parameter für die Erzeugung der Objekte vom Typ `WebApI` werden dem `WebApConstructor` Objekt durch ein `MyMultiProperties` Objekt übergeben. Durch den Aufruf der Methode *`buildWebAps`* wird ein `Hashtable` zurückgegeben, der als *Schlüssel* die Namen der Webapplikationen und als *Werte* `String`s mit dem erzeugten HTML-Code der jeweiligen Webapplikation enthält.

Mit den durch das `MyMultiProperties` Objekt übergebenen Parametern der zu verwendenden Webapplikationen kann eine Login- und eine Navigationswebapplikation für das Webinformationssystem definiert werden.

Mit der Loginwebapplikation besteht die Möglichkeit, einen Benutzer an allen anderen Webapplikationen an- und abzumelden. Diese Lösung ist als kleine Demonstration zu verstehen. Dies liegt daran, dass die endgültigen Formalien eines oder mehrerer Loginparameter mit jedem Anbieter einer Webapplikation einzeln ausgehandelt werden müssen sowie, ob ein expliziter Logout zum Ende einer Session aus dem System erfolgen muss. Bei der jetzigen Beispiellösung wird zum Login einfach eine von der Loginwebapplikation erhaltene Login-ID an alle anderen Webapplikationen übertragen, verbunden mit dem Aufruf ihres jeweiligen Startzustands. Zum Logout werden einfach alle Objekte vom Typ `WebApI` durch Aufruf der entsprechenden Methode am Implementationsobjekt vom Typ SessionI gelöscht und der Startzustand der Webapplikationen ohne Login-ID aufgerufen. Der Name der Loginwebapplikation kann durch das Webinformationssystem vom `WebApConstructor` durch eine entsprechende Methode erfragt werden.

Der Name der Navigationswebapplikation und der der Webapplikation, die vom Benutzer über die Navigation ausgewählt wurde, kann über entsprechende Methodenaufrufe vom `WebAp-Constructor` erfragt werden.

7.2.2 Das WebApFactoryI Interface

Das `WebApFactoryI` Interface beschreibt die *Abstrakte Fabrik* für den im Interface `WebApI` beschriebenen Typ von Webapplikationen. Durch die Wahl einer Abstrakten Fabrik ist ein einfaches Austauschen der konkreten `WebApI` Implementation zur Laufzeit gegeben. Die Schnittstelle der Fabrik erlaubt es, die Namen der verfügbaren Produkte zu erfragen und ein Produkt als Standard zu definieren. Entsprechend gibt es eine Fabrikmethode, die das Standardprodukt erzeugt und eine, bei der der Name des zu erzeugenden Produktes angegeben werden muss. Jedes Objekt vom Typ `WebApI` wird bei der Erzeugung jeweils mit einem Exemplar der Implementation einer `XMLServicesFactoryI`, eines `MessageServiceProviderI` und eines `SessionI` initialisiert. Als konkrete Implementation existiert derzeit die `WebApFactory,` die `WebAp` und `WebApWithWindowControls` Objekte erzeugen kann. Diese kann durch das Hinzufügen eines Produktnamens mit einem dazugehörigen Java Klassennamen um dieses Produkt erweitert werden, wenn es das `WebApI` Interface implementiert.

7.2.3 Das WebApI Interface

Das `WebApI` Interface dient als Vorlage für die Implementation der Repräsentation der Webapplikationen im lokalen System. Über diese Schnittstelle können die verschiedenen Zustände einer Webapplikation aufgerufen werden. Für diesen Zweck stehen die *`followLink`* und *`submitToTarget`* Methoden zur Verfügung. Ihre Namen sind angelehnt an den Begriff Link und das Attribut target des Form-Tags für Formulare in HTML-Dokumenten.

- *`followLink`* – es wird ein nachfolgender Zustand durch den Aufruf einer Methode bzw. Funktion mit nicht variablen Parametern angefordert, d.h. sie sind nicht von einer aktuellen Eingabe des Benutzers abhängig. Dies soll eine ähnliche Symantik zum Ausdruck bringen, wie sie bei einem Link in HTML-Dokumenten vorliegt, jedoch auf den Kontext eines Methoden- bzw. Funktionsaufrufs übertragen.
- *`submitToTarget`* – bei dieser Methode wird der nachfolgende Zustand durch einen Aufruf mit aktuellen Eingaben eines Benutzers angefordert. Dabei wird die Eingabe von HTML-Formulardaten auf einen Methoden- bzw. Funktionsaufruf mit Parametern übertragen.

Als weitere zentrale Methoden sind *getHTML* – zum Abfragen der HTML-Repräsentation des aktuellen Zustands, *setParameterValue* – zum Übergeben einer Benutzereingabe und *getName* – zum Abfragen des Namens der Webapplikation definiert.

Außerdem werden Methoden definiert, um die im XML enthaltenen Dublin Core Meta Daten zu erhalten. Zusätzlich werden Methoden definiert, um abzufragen, ob die Webapplikation ein Login unterstützt, ob der Benutzer eingeloggt ist, um die Namen der in dem aktuellen Zustand definierten Parameter zu erhalten und um sich in die Session zu speichern bzw. daraus zu laden.

Des Weiteren kann, falls dies von der entfernten Webapplikation angegeben worden ist, der Erzeugungszeitpunkt des Zustandes und ein Intervall, nach dem ein erneutes Abfragen des Zustandes sinnvoll ist, durch die Methoden *getCreationTime* und *getUpdateInterval* abgefragt werden.

Als konkrete Beispielimplementationen existieren die `WebAp` und `WebApWithWindow-Controls` Klassen.

Die folgende Abbildung 10 zeigt ein vereinfachtes Sequenzdiagramm eines *buildWebAps* Aufrufs an einem `WebApConstructor` Objekt mit Benutzereingaben. Um die Benutzereingaben von der betreffenden Webapplikation verarbeiten zu lassen wird *submitToTarget* bzw. *followLink* am entsprechenden `WepApI` implementierenden Objekt aufgerufen. Dies fordert, über den bei dieser Webapplikation zu verwendenden `MessagingServiceProvider`, das den neuen Zustand der entfernten Webapplikation repräsentierende XML-Dokument an.

Dieses XML-Dokument gibt es an das `XMLServicesI` Interface implementierende Objekt weiter und holt sich von diesem alle im Dokument enthaltenen Informationen und die nach HTML transformierte Ansicht des Zustandes. Das WebApConstructor Objekt gibt anschließend den HTML-Code der Webapplikationen an seinen Aufrufer zurück.

Nachstehend ein Kontrollfluss wie bei einem Web Service (vgl. Abbildung 8).

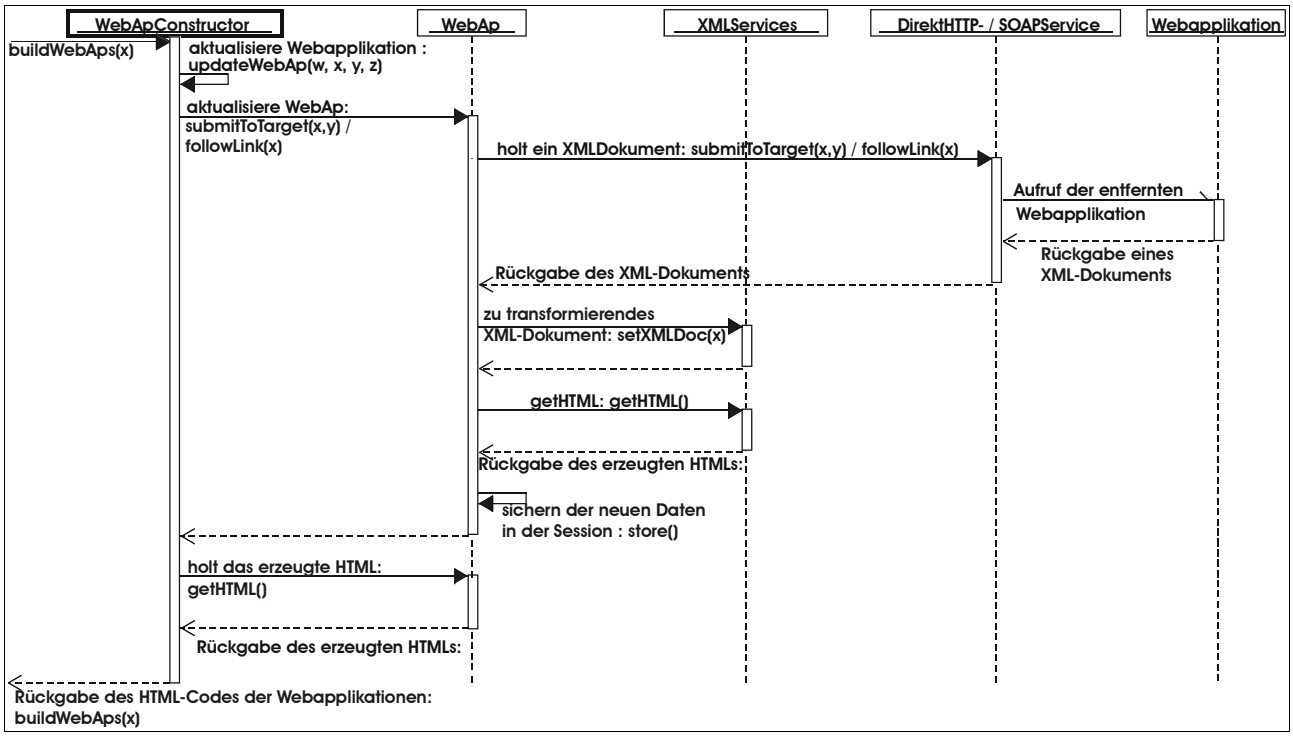

Abbildung 10 - Sequenzdiagramm eines buildWebAps(Hashtable: reqParameter) Aufrufs am WebApConstructor

7.2.4 Das XMLServicesFactoryI Interface

Das `XMLServicesFactoryI` Interface definiert die *Abstrakte Fabrik* für den im Interface `XMLServicesI` beschriebenen Typ von XMLServices, wie sie von den `WebApI` Objekten zur Verarbeitung der XML-Dokumente verwendet werden. Als Grund für die Verwendung einer a*bstrakten Fabrik* stand die leichte Austauschbarkeit der Produkte im Vordergrund.

Als konkrete Implementation existiert die `XMLServicesFactory` Klasse.

7.2.5 Das XMLServicesI Interface

Das `XMLServicesI` Interface beschreibt die Operationen, die auf dem zur Kommunikation verwendeten XML-Dokument möglich sind. Dies sind unter anderem:

- *`getHTML`* – liefert den nach HTML transformierten aktuellen Zustand der Webapplikation.
- *`getLinks`* – liefert die Links des aktuellen Zustands.
- *`getTargets`* – liefert die Targets des aktuellen Zustands.
- *`getParameterAsMMP`* – liefert die Parameter des aktuellen Zustands. Es werden die Namen, Vorgabewerte und Typen der Parameter zurückgegeben.
- *`getDCMData`* – liefert, falls angegeben, die Dublin Core Metadaten des aktuellen Zustands.
- *`getCreationTime`* – liefert, falls vorhanden, den Erzeugungszeitpunkt des aktuellen Zustands
- *`getUpdateInterval`* – liefert, falls vorhanden das Intervall, nach dem der Zustand mit neuen Inhalten abgerufen werden kann.

7.2.6 Das MessageServiceProviderI Interface

Das `MessageServiceProviderI` Interface definiert den Typ für die möglichen Kommunikations-*Strategien* des Frameworks mit den Webapplikationen. Es werden dazu zwei Methoden vorgegeben:

- *`followLink`* – bildet die entsprechende Methode eines Objekts vom Typ `WebApI` auf die jeweilige *Strategie* ab.
- *`submitToTarget`* – bildet ebenfalls die entsprechende Methode eins Objekts vom Typ `WebApI` auf die jeweilige *Strategie* ab.

Als konkrete Implementationen existieren die Strategien `DirektHTTPService` und `SOAP-Service`. Diese implementieren eine einfache Kommunikation via HTTP POST und GET Aufrufe und eine auf SOAP basierend.

Als Ergebnis des Aufrufs der beiden Methoden erhält man einen `String,` der das von der Webapplikation zurückgesendete XML-Dokument enthält.

Zusätzlich zu diesen zwei Implementationen habe ich noch den `MessagingService` als *Kompositum* implementiert. Dieser erzeugt, in der derzeitigen Implementation, je nach Bedarf ein Exemplar der tatsächlich benötigten *`MessageServiceProviderI` Strategie.* Für die Zukunft sind auch Implementationen als *Singleton* mit einem Pool von Exemplaren der jeweiligen *Strategien* denkbar, um die maximale Anzahl von Verbindungen kontrollieren zu können.

Für ein Klassendiagramm des `MessageServiceProviderI` Interfaces und seiner Implementationen siehe Abbildung 11. Es zeigt, dass alle drei Klassen das `MessageService-ProviderI` Interface implementieren und die `MessageService` Klasse zusätzlich die anderen zwei benutzt.

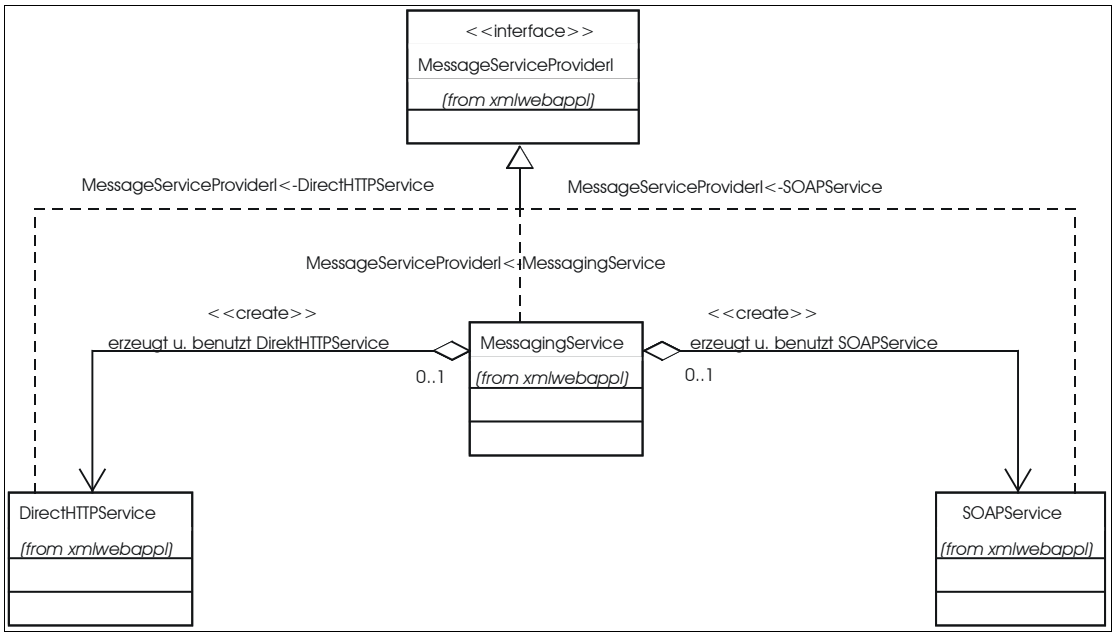

Abbildung 11 - die MessageServiceProviderI Klassenhierarchie

7.2.7 Das SessionI Interface

Das `SessionI` Interface definiert den Typ für mögliche *Strategien,* um `WebApI` Objektdaten in einer Session persistent zu machen. Dazu bietet es die Methoden *`store`* – zum Speichern, *`restore`* – zum Wiederholen und *`clear`* – zum Löschen von Daten in der Session.

Als konkrete Implementationen liegen `MyHTTPServletSessionMgr` und `MyDBSessionMgr` vor. Die `MyHTTPServletSessionMgr` *Strategie* ist für die Speicherung in einer zu einem Servlet gehörigen Session zuständig, und die `MyDBSessionMgr` *Strategie* ermöglicht es, die Session in einer Datenbank zu führen.

Die *Strategie* `MyDBSessionMgr` implementiert dazu die Methoden *`saveToDB`* und *`loadFromDB.`* Diese dienen dazu, die Anzahl der Zugriffe auf die darunter liegende Datenbank so gering wie möglich zu halten und ermöglichen es, den gesamten Session Inhalt gezielt zum Beginn der Bearbeitung aus der Datenbank zu lesen und an seinem Ende wieder in die Datenbank zu schreiben. Zur Abstraktion von einer bestimmten Datenbank wird dazu das `SessionDatabaseI` verwendet.

Das dazugehörige Klassendiagramm kann in Abbildung 12 betrachtet werden.

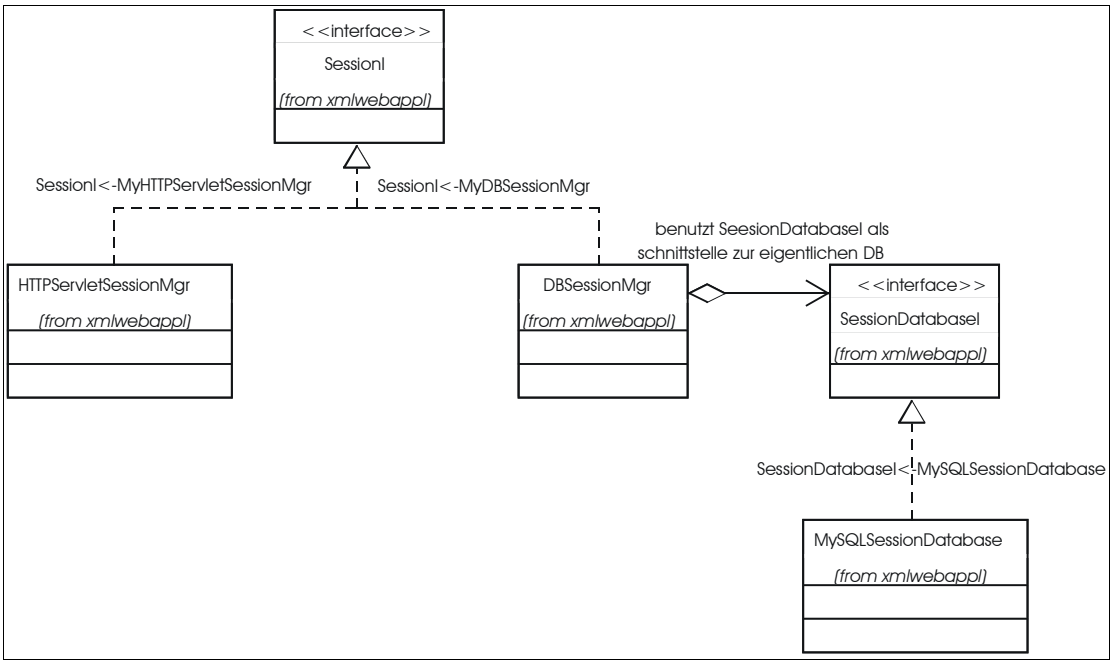

Abbildung 12 - die SessionI Hierarchie

7.2.8 Das SessionDatabaseI Interface

Für den Zugriff auf eine spezielle Datenbank dient das als *Adapter* fungierende Interface `SessionDatabaseI`. Dies ermöglicht es zu überprüfen, ob in der zugrunde liegenden Datenbank Daten zur Session existieren – *`sessionExists`*, diese in der Datenbank neu zu erzeugen – *`createSession`*, die Daten auszulesen – *`getSessionData`* – oder in der Datenbank auf den neusten Stand zu bringen – *`updateSessionData`*.

Als Beispielimplementation liegt ein *Adapter* für eine MySQL Datenbank vor.

7.3 Das Beispiel-Webinformationssystem

Zur Demonstration des WebAp Frameworks existiert ein von mir implementiertes Java Servlet, das ein Webinformationssystem mit dem Namen w3.MeinPortal.de erzeugt.

Zusätzlich sind folgende Webapplikationen zur Demonstration der Funktionalität des Frameworks implementiert:

- *portalNavi* – als Navigation über dem Webinformationssystem
- *portalLogin* – als Loginwebapplikation für das Webinformationssystem
- *MeineVersicherung.de* – als Versicherungswebapplikation
- *MeinWetter.de* – als Wetterberichtwebapplikation
- *HeiseNewsService* – als Nachrichtenservice, der die aktuellen Nachrichten von www.heise.de anzeigt.

Mit der Webapplikation *portalNavi* kann über dem Beispiel Webinformationssystem navigiert werden. Dazu wird diese dem `WebApConstructor` Objekt im System als Navigation bekannt

gemacht. Dieses wertet nun die Variable selectedlink der Navigation aus, um zu erfahren, welche Webapplikation ausgewählt worden ist. Über die Methode `getNaviSelectedWebApName` kann das Webinformationssystem dies vom `WebApConstructor` abfragen und entsprechend anzeigen. Im vorliegenden System wird diese Webapplikation in der Mitte des Webinformationssystems angezeigt (siehe Abbildung 13).

Die Webapplikation *portalLogin* ermöglicht es einem Benutzer, sich am Webinformationssystem anzumelden, dazu wird diese dem `WebApConstructor` Objekt als Loginwebapplikation bekannt gemacht. Anschließend verwendet dieses die Variablen justLoggedIn, justLoggedOut und portalUID des Login und versucht, den Benutzer an alle anderen Webapplikationen, die dies unterstützen, mit der portalUID anzumelden. Entsprechend werden bei einer Abmeldung alle Webapplikationen ohne portalUID neu initialisiert.

Während die beiden bisher genannten Webapplikationen, auf Grund ihrer besonderen Funktionen, Einfluss auf das gesamte Webinformationssystem haben, zeigen die folgenden, dass Eingaben an einer Webapplikation normalerweise nur diese selbst betreffen.

MeineVersicherung.de ist eine Webapplikation die zeigt, wie eine Webapplikation einer Versicherungsagentur aussehen könnte. Man kann Angaben zu seiner Person machen und bekommt dann verschiedene Versicherungstypen vorgeschlagen.

MeinWetter.de ermöglicht es einem angemeldeten Benutzer, Orte aus einer Datenbank auszuwählen, zu denen er dann die aktuellen Temperaturen angezeigt bekommt – derzeit zufällig generierte Werte. Bei jedem neuen Login werden diese Orte erneut mit den aktuellen Temperaturen angezeigt.

Der *HeiseNewsService* ist als ein über SOAP eingebundener Web Service realisiert und zeigt die Nachrichten des Tages an, die auf der Webseite www.heise.de präsentiert werden.

Die folgende Abbildung 13 stellt das Beispiel Webinformationssystem w3.MeinPortal.de mit den oben beschriebenen Webapplikationen dar. Es zeigt außerdem die gleichzeitige Verwendung verschiedener Implementationen des `WebApI` Interfaces. Die Implementation `WebAp` bietet nur die einfache Darstellung des Inhalts der Webapplikation, wogegen die `WebApWithWindow-Controls` Implementation zusätzlich einen Rahmen um die Webapplikation legt und eine Titlelzeile erzeugt. Diese Titelzeile enthält neben dem Namen der Webapplikation links ein Feld, um weitere Informationen über diese abzurufen und rechts ein Feld, um sie zu minimieren, so dass nur noch die Titelzeile angezeigt wird.

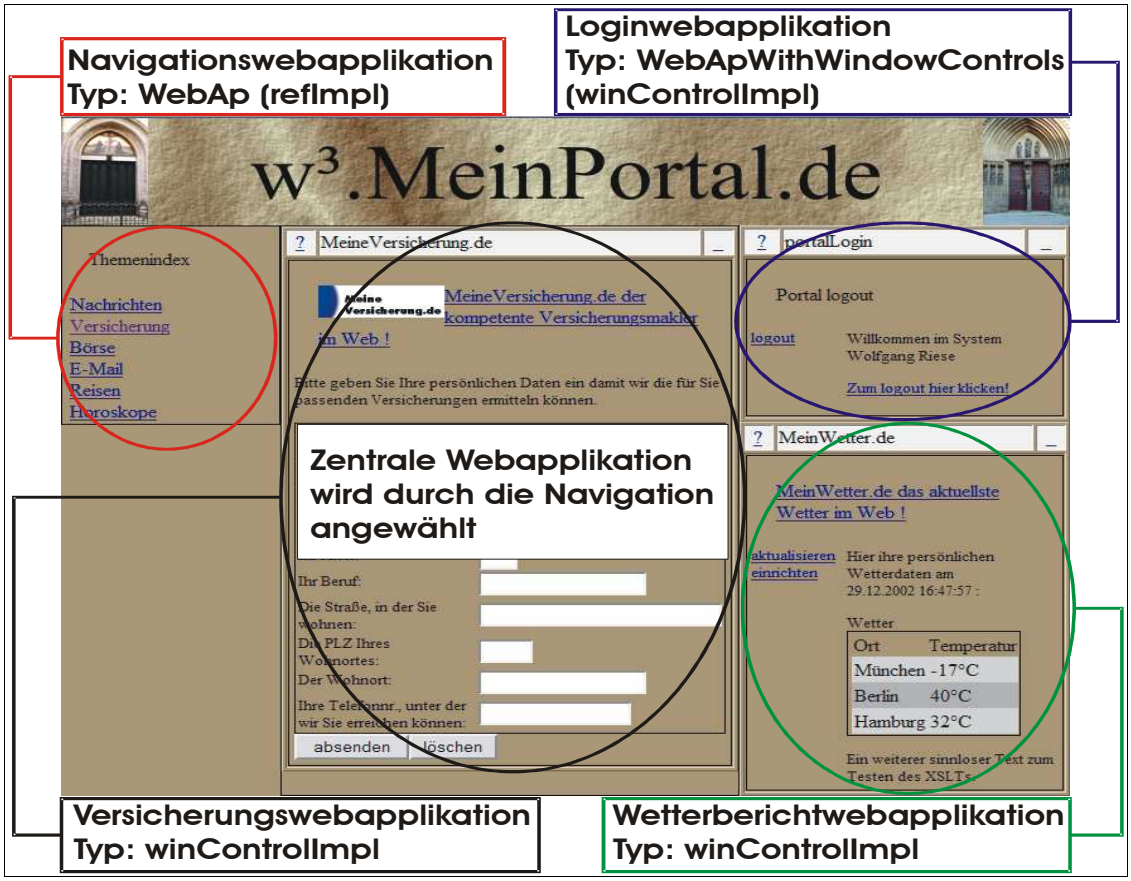

Abbildung 13 - Ansicht des Beispiel Webinformationssystems w3.MeinPortal.de

Im abschließenden Kapitel „Resümee“ werde ich noch einmal die gesamte Arbeit betrachten und alle Ergebnisse zusammenfassen. Des Weiteren werde ich Überlegungen dazu anstellen, welche Themen sich daraus für die Zukunft ergeben könnten.

8 Resümee

Zunächst werde ich den Inhalt und die Ergebnisse der Arbeit, beginnend mit Kapitel 2, kapitelweise zusammenfassen. Daran anschließend werde ich einen Ausblick auf Themen für zukünftige Untersuchungen im Zusammenhang mit dieser Arbeit geben.

8.1 Zusammenfassung der Arbeit

In Kapitel 2 Webinformationssysteme – Portale habe ich zunächst die für meine Arbeit grundlegenden Begriffe Webapplikation und Webinformationssystem bzw. Portal sowie Service vorgestellt und im Kontext der Arbeit definiert.

Dabei hat sich herausgestellt, dass vor allem für die Begriffe Webapplikation und Service in der Literatur keine einheitlichen Definitionen zu finden sind. Im Besonderen der Begriff des Service ist sehr stark vom Kontext abhängig, in dem er verwendet wird.

Im Folgenden noch einmal die Definitionen, die ich als Ergebnisse festgehalten habe:

- *Webapplikation*:
  Eine Softwareanwendung mit einem Frontend im Web, aber keine Sammlung von statischen Webseiten. Schließt auch Anwendungen mit ein, die erst nachträglich mit einem Frontend für das Web versehen wurden.

- *Service*:
  Im Zusammenhang mit Webinformationssystemen eine Leistung, die dem Benutzer des Systems erbracht wird, z.B. ein Wetterbericht, ein Börsenbericht, Webmail oder Ähnliches.

  Im softwaretechnischen Zusammenhang ein Programm X, welches Aufträge eines Konsumenten (Programm Y) erledigt auf Grundlage seiner Spezifikation (vgl. „Design by Contract“ [MeyerB 1997, S. 331ff] bzw. „Vertragsmodell“ [Züllighoven 1998, S. 44ff]).

- *Webinformationssysteme – Portale*:
  Eine Web Site, auf der Besucher verschiedene integrierte Serviceleistungen, Informationen und Nachrichten unter einer Oberfläche erhalten können. Dies erfolgt durch die Integration verschiedener *Webapplikationen* unter eine einheitliche Oberfläche.

  Der zur Zeit im Web vorherrschende Typ von Webinformationssystem ist das Portal. Dieser Typ von Webinformationssystem hat die Zielsetzung, dem Benutzer einen Mehrwert zu bieten, sowie einen Einstieg in das Internet bzw. einen bestimmten Themenkomplex zu bieten und zu erleichtern, zumeist vor einem kommerziellen Hintergrund. Es wird dabei meistens noch zwischen horizontalen- und vertikalen Portalen mit offenen bzw. geschlossenen Benutzerkreis und nach der Zielgruppe des Systems unterschieden.

Abgeschlossen habe ich das zweite Kapitel mit zwei Beispielen für Webinformationssysteme. Dazu habe ich „My Netscape“ und „hamburg.de“ untersucht. Ich habe dabei einen ersten Überblick darüber gewonnen, welche Art von Webapplikationen in Webinformationssystemen zum Einsatz kommen. Außerdem habe ich einen unterschiedlich hohen Grad an Personalisierung bei beiden Portalen feststellen können.

Im dritten Kapitel habe ich das zentrale Thema der Arbeit – Das Problem der flexiblen Einbindung von Webapplikationen in Webinformationssystemen – herausgearbeitet.
Zunächst habe ich nach einer Klassifikation von Webapplikationen in der Literatur gesucht. Diese orientieren sich zumeist am Inhalt der Webapplikationen. Für die Arbeit bleiben daraus folgende Typen festzuhalten:

1. Webapplikationen mit einem Minimum an Interaktion – sie präsentieren hauptsächlich Informationen.
2. Webapplikationen mit größerer Funktionalität – sie präsentieren nicht nur Informationen sondern bieten größeres Maß an Interaktivität.
3. Softwareapplikationen die nachträglich mit einem Frontend im Web versehen worden sind. Auch sie bieten meistens ein hohes Maß an Interaktivität.
4. Webapplikationen des Typs 2 und 3 mit einem gesteigerten Bedarf an Sicherheit, wie er im Bereich E-Business für den Zahlungsverkehr notwendig ist.

Anschließend habe ich mich mit den grundlegenden Problemen beim Design von Webapplikationen beschäftigt. Von mir und in der Literatur gesehenen Hauptprobleme sind der vielfach vorliegende Mangel an Trennung von Anwendungs- bzw. Geschäftslogik (Business Logic), Inhalten (Content) und Layout (vgl. [KererKirda 2001, S. 135-137]) bei der Entwicklung von Webapplikationen.

Des Weiteren bin ich auf die Interessen und Probleme der Betreiber von Webinformationssystemen und Webapplikationen eingegangen.
Aus der Sicht des WIS-Betreibers sind dies:

> Er möchte möglichst viele Serviceleistungen anbieten, aber warum sollte er alle selbst entwickeln? Dabei spielt vor allem der Zeitaufwand für die Entwicklung und Aneignung von Know-how eine wichtige Rolle. Also, warum nicht die bestehenden Auftritte von externen Serviceanbietern integrieren.

Aus der Sicht des Webapplikationsanbieters sind es folgende:

> Er möchte seinen Service einem möglichst großen Nutzerkreis zugänglich machen, aber wie erreicht man dies? Die Lösung ist es, den Service über ein WIS einem großen Nutzerkreis anbieten. Hier stellt sich die Frage, warum er das Know-how aus der Hand geben sollte und wie ein Auftritt in mehreren WIS erreicht werden kann? Dazu kommt die Frage, wie für jedes WIS und den eigenen Web-Auftritt die Applikation immer neu angepasst werden kann ohne unnötigen Aufwand?

Als nächstes habe ich vier mögliche Lösungen für die Problematik vorgestellt. Die vierte davon dient als Ausgangspunkt für Entwicklung des WebAp Frameworks, wie es in den Kapiteln 6 und 7 vorgestellt ist.
Diese hat zum Ziel, die Webapplikation weiterhin komplett im System ihres Anbieters zu belassen, sie aber trotzdem in das Webinformationssystem zu integrieren, so dass sie nicht nur über einen Link erreichbar ist. Dafür müssen beide Seiten eine gemeinsame Schnittstelle finden.
Der Anbieter der Webapplikation und der Betreiber des Webinformationssystems tauschen Informationen darüber aus, welche Daten zur Personalisierung der jeweiligen Webapplikation zu

übertragen sind und einigen sich darüber, wie die Webapplikation vom Layout her in das System eingepasst werden soll.

Die wichtigste Aufgabe der zu definierenden Schnittstelle zwischen Webinformationssystem und Webapplikation ist es, den Austausch der Inhalte der verschiedenen Zustände der Webapplikation mit dem Webinformationssystem zu ermöglichen. Bei dieser Übertragung müssen zusätzlich Variablendefinitionen und Adressen für Aufrufe von Folgezuständen übermittelt werden. Außerdem muss die Schnittstelle es möglich machen, technisch unterschiedlich realisierte Systeme miteinander zu verbinden. Dabei müssen die Sessions der beiden Systeme für das jeweils andere transparent sein. Des Weiteren soll es die Schnittstelle dem Betreiber des Webinformationssystems die Möglichkeit bieten, gleichzeitig verschiedene Webapplikationen von unterschiedlichen Anbietern zu integrieren und es genauso dem Anbieter der Webapplikation ermöglichen, seine Webapplikation in verschiedenen Kontexten ohne weitere Änderungen einsetzen zu können, immer vorausgesetzt, dass alle diese einheitliche Schnittstelle verwenden.

Es besteht weiterhin der Vorteil der Integration der Webapplikation in das Webinformationssystem, so dass der Besucher es nicht verlassen muss, um die Webapplikation zu nutzen. Dadurch hat der Nutzer den Eindruck, es mit demselben Ansprechpartner zu tun zu haben, was sich positiv auf das Vertrauensverhältnis auswirken dürfte.

Die technische Verfügbarkeit der Webapplikation ist davon abhängig, dass nicht nur das System des Webinformationssystems, sondern auch das des Anbieters der Webapplikation verfügbar ist.

Die folgenden Probleme zu minimieren, die sonst bei der Einbindung von Webapplikationen in Webinformationssystemen zum tragen kommen, hilft diese Lösung:

- Es gibt vielfach Vermischungen von Layoutelementen und Anwendungslogik. Daraus resultiert zudem das Fehlen einer eindeutigen Trennung von Inhalten und Oberflächenstrukturelementen. Dies hat zur Folge, dass das Anpassen von Anwendungen an andere Layouts erschwert wird, da die Gefahr besteht, versehentlich Anwendungslogik und Inhalte zu verändern.
- Bei der Integration von Webapplikationen in Webinformationssysteme besteht zusätzlich das Problem der Anpassung des Kontrollflusses der Applikation an das Webinformationssystem bzw. des Webinformationssystems an die integrierten Webapplikationen.
- Es besteht das Problem, zu viel eigenes Know-how an den Geschäftspartner preisgeben zu müssen und sich selbst so auf lange Sicht überflüssig bzw. Konkurrenz zu machen.
- Andersherum gesehen muss der Geschäftspartner evtl. Zeit opfern, um sich mit Dingen zu beschäftigen, die für sein eigenes Aufgabenfeld nicht relevant sind.

Ein weiteres Problem liegt in der Vielzahl der verwendeten Server-Technologien, wie Web Content Management Systemen, Applikationsservern und Webservern, bei denen es zu Überschneidungen der Einsatzgebiete kommt und den vielen verschiedenen eingesetzten Programmier- und Scriptsprachen in diesen Systemen.

Im vierten Kapitel habe ich den Begriff des Web Services vorgestellt und eine Definition desselben bestehend aus den folgenden 5 Punkten erarbeitet:

1. Es handelt sich um ausführbaren Programmcode
2. Dieser ist über ein Netzwerk (Web) mit einem Standardprotokoll ansprechbar (HTTP, SMTP, Jabber usw.), und die Daten werden in XML übertragen
3. Auf Grund der Übertragung der Daten in XML und der Verwendung von Standard Protokollen ist eine plattformübergreifende Verwendung von Web Services möglich
4. Durch seine Schnittstelle soll eine lose Koppelung des Programms möglich sein und dessen Wiederverwendbarkeit sichergestellt werden
5. Web Services können über Directory Services auffindbar sein, und es sollte eine manuelle oder automatische Einbindung des gefundenen Web Services in andere Systeme möglich sein

Anschließend habe ich die dem Begriff zugrunde liegende 5 Schichten Architektur vorgestellt und mein WebAp Framework in diesem Zusammenhang betrachtet. Dabei bin ich zu dem Schluss gekommen, dass man mein WebAp Framework mit Webapplikationen als eine konkrete Implementation eines Web Services betrachten kann, auch wenn ich nicht alle Schichten der Web Services Architektur ausnutze beziehungsweise in der vorliegenden Implementation unterstütze.

In Kapitel 5 habe ich das Konzept des Fachlichen (Domain) Services vorgestellt und mit Web Services und meinem WebAp Framework verglichen. Dabei bin ich zu dem Ergebnis gekommen, dass Fachliche Services durch Web Services implementiert werden können und in einer Software Architektur mit meinem WebAp Framework auf verschiedenen Ebenen zusammen eingesetzt werden können. Zum Überblick über die entstehenden Schichten und ihre Aufgaben in dieser Architektur hier die Tabelle 7.

	Schicht	Aufgaben der Schicht
	Benutzungsschnittstelle	Stellt dar und interagiert
	Integrationsschicht (Webinformationssystem mit WebAp Framework)	Benutzt mein WebAp Framework um: • Die Webapplikation zu integrieren • Für den korrekten Kontrollfluss zwischen den Webapplikationen zu sorgen • HTML für die Weboberfläche aus den Inhalten der Webapplikationen und den Layoutvorgaben des Webinformationssystems zu generieren
Webapplikation	*Interaktionsschicht*	• Erstellt den XML-Code für die Integrationsschicht • Übersetzt Wissen über den Umgang mit Fachlichen Services • Koordination zwischen mehreren Fachlichen Services
	Fachliche Services	Kapseln und bündeln fachliche Funktionalität und fachliches Wissen

Tabelle 7 - Webapplikations Architektur im Hinblick auf Fachliche Services

Im sechsten Kapitel habe ich zunächst auf Grundlage der im dritten Kapitel herausgearbeiteten Probleme die folgenden Anforderungen an meine Lösung festgehalten:

1. Der Anbieter einer Webapplikation sollte nicht jedes Mal für einen neuen Einsatzkontext Änderungen an der Webapplikation vornehmen müssen. Das heißt, dass das Layout einer Applikation ohne Eingriff in den Applikationscode selbst änderbar sein muss. Daraus folgt Layout und Inhalte zu trennen, um so die Inhalte ohne Programmlogik an das Webinformationssystem übertragen zu können.

2. Um beliebige Webapplikationen in einem Webinformationssystem einsetzen zu können, muss der Kontrollfluß und der Namensraum der Variablen der Applikation in verschiedenen Einsatzkontexten gewährleistet sein. Dies bedeutet unter anderem, dass Eingaben im Portal an einer bestimmten Webapplikation nicht zu unkontrollierten Änderungen an anderen Webapplikation führen dürfen.

3. Eine Webapplikation sollte weiterhin dezentral auf den Servern ihres Herstellers laufen können, damit dieser die volle Kontrolle über seine Applikation und damit sein Know-how behalten kann.

4. Die verwendeten Kommunikationstechniken sollten nach Möglichkeit keine zusätzlichen Hard- und Softwaresysteme erfordern, sich also in bestehende Systeme integrieren und keine weiteren Kosten erzeugen.

5. Für den Webinformationssystembetreiber ist es außerdem von Interesse, dass er den Zustand der integrierten Webapplikationen zwischen den Aufrufen innerhalb einer Session cachen kann. So wird unnötige Kommunikation über das Netzwerk vermieden und Kosten und Zeit gespart.

6. Auf Grund der Vielzahl der im Web verwendeten Technologien (vgl Kapitel 3.4, S. 28 und [Turau 1999]) zum Realisieren und Einsetzen von Webapplikationen muss eine Form der Integration und Kommunikation gewählt werden, die es ermöglicht, diese Technologien zu verbinden.

Aus diesen Anforderungen habe ich den Schluss gezogen, die Lösung des Problems in zwei Teilen zu betrachten. Der eine Teil betrifft die Kommunikation und beinhaltet zum einen die von mir entwickelte WebAp-DTD und zum anderen die Überlegung, die Kommunikation mit Hilfe des Protokolls HTTP über Inter- und Intranet zu erledigen. Die DTD beschreibt dabei das Format der von mir zur Datenübertragung genutzten XML-Dokumente. Zusammen mit den Überlegungen aus Kapitel 4 und 5 habe ich mich dazu entschlossen, bei der Kommunikation via HTTP nicht nur direkt HTTP mit POST und GET zu verwenden, sondern auch das von den Web Services her stammende SOAP-Protokoll zu nutzen, welches wiederum HTTP benutzt. Der andere Teil betrifft sowohl die Schnittstelle zur Integration von Webapplikationen in ein Webinformationssystem, als auch die Frage nach der Unabhängigkeit der einzelnen Webapplikationen bei dieser Integration, sowie die Frage nach der Aufbereitung der übertragenen Daten. Für die Erfüllung dieser Anforderungen habe ich das WebAp Framework in Java geschrieben. Dieses bietet eine Sammlung von Schnittstellen und Beispiel-Implementationen an, mit denen ein Webinformationssystem mit verschiedenen Webapplikationen aufgebaut werden kann.

Im siebten Kapitel habe ich schließlich die von mir entworfenen Java Interfaces und Klassen und die dabei von mir verwendeten Entwurfsmuster vorgestellt. Zur Verdeutlichung dieser habe ich dazu UML Klassen Diagramme verwendet. Zum Abschluss habe ich das von mir zur Demonstration erstellte w^3.MeinPortal.de Webinformationssystem und die dafür entwickelten Webapplikationen vorgestellt.

8.2 Ausblick

Als mögliche Themen für zukünftige Untersuchungen ergeben sich aus meiner Arbeit meines Erachtens folgende Punkte:

- Wie verhält sich das Framework im realen Einsatz eines Webinformationssystems mit hohen Zugriffzahlen von Benutzern?

- Würde das Verhalten eines Webinformationssystems bei Nebenläufigkeit der einzelnen WebAp Objekte durch Threading positiv beeinflusst werden, und würde es in diesem Zusammenhang Sinn machen, die MessageingService Klasse als Singleton mit einem Resourcenpool an Verbindungen zu realisieren?

- Wie wird die noch nicht veröffentlichte Portlet Spezifikation des Java Community Process Nr. 168 aussehen, und wie unterscheidet sie sich von meinem Ansatz?

- Wie kann man das Framework erweitern, damit es den größten Nutzen aus den in der Web Services Architektur vorgesehenen Directory Services ziehen kann?

Gibt es überhaupt eine Möglichkeit, bestehende Directory Services wie das UDDI ohne weiteres zu nutzen? In der Web Services Architektur ist es vorgesehen, dass Konsumenten automatisch aus einem Directory Service einen für ihre Anforderungen passenden Service anhand von „Ontologien“

auswählen können. Funktioniert dies für Webapplikationen nach meiner Definition, oder welche Anpassungen müssten auf welcher Seite vorgenommen werden?

Literaturverzeichnis

[Blank 2001] Timmy Blank: *Prozessunterstützung für eGovernment-Services mit Java und XML am Beispiel von www.hamburg.de.* Diplomarbeit am Fachbereich Informatik, Arbeitsbereich Softwaretechnik, Uni-Hamburg, 2001

[Bleek 2001b] W.-G. Bleek: *Situations in Life to Support the Use and Modelling of Municipal Information Systems*, Dan Remenyi and Frank Bannister, Proceedings of the European Conference on Electronic Government, S. 49-60, Trinity College Dublin, Ireland, 2001

[BleekFloyd 2000] W.-G. Bleek, C. Floyd: *Lebenslagen - Evaluation und Konzeption*, 2000

[BleekFloyd 2001] W.-G. Bleek, C. Floyd: *Lebenslagen - Redaktionskonzep*t, 2001

[Bodendorf 1999] F. Bodendorf: *Wirtschaftsinformatik im Dienstleistungsbereich*. Springer-Verlag, 1999.

[Caillliau 1995] R. Caillliau: *A Little History of the World Wide Web*, W[3]Consortium, http://www.w3c.org/History.html – Stand: 2002-09-20

[Cagle 2001] K. Cagle: Rethinking *Web Services – Can Web services live up to the royal hype?* Teil 1: Okt. 2001 S. 54-57 und Teil 2: Nov. 2001 S. 52-56 in [WSJ 2001], 2001

[Christodoulou et al. 2001] S. Christodoulou, P. Zafiris, T. Papatheodorou: *Web Engineering: The Developers' View and a Practinioner's Approach*. S 170-187 in [MurugesanDeshpande 2001], 2001

[DublinCore 1999] *Dublin Core Metadata Element Set, Version 1.1: Reference Description*. Dublin Core Metadata Initiative 1999 http://dublincore.org/documents/dces/ – Stand: 2001-11-02

[EcksteinCassabianca 2001] R. Eckstein, M. Casabianca: *XML Pocket Reference,* Second Edition. Sebastopol: O'Reilly & Associates, 2001

[Floyd et al. 1984] C. Floyd, R. Budde, K. Kuhlenkamp, L. Mathiassen, H. Züllighoven (Hrsg.): *A Systematic Look at Prototyping.* Approaches to Prototyping. S. 1-18, Berlin, Heidelberg: Springer-Verlag, 1984

[Floyd 1994] C. Floyd: *Software-Engineering - und dann?* Informatik Spektrum 17(1), S. 29-37, Berlin, Heidelberg: Springer-Verlag, 1994

[Floyd et al. 1989] C. Floyd, F.-M. Reisin, G. Schmidt: *STEPS to Software Development with Users*. ESEC 1989: S. 48-64, 1989

[Gamma et al. 1996] E. Gamma, R. Helm, R. Johnson, J. Vlissides: *Entwurfsmuster -Elemente wiederverwenbarer objektorientierter Software*. Übersetzung von D. Riehle, Bonn: Addison Wesley, 1996

[GellersenGaedke 1999] H. Gellersen, M. Gaedke: *Object-oriented Web Application Development*, IEEE Internet Computing, 01/1999, S. 60-80, 1999
http://computer.org/internet/ – Stand: 2002-06-18

[Gnaho 2001] C. Gnaho: *Web-Based Information Systems Dvelopment – A User Centered Engineering Approach*. S. 105-118 in [MurugesanDeshpande 2001], 2001

[Goldman 1999] *Internet Portals in Europe*. London: Goldman Sachs Investment Research, 1999

[Gryczan 1996] G. Gryczan: Prozeßmuster zur Unterstützung kooperativer Tätigkeit, Diss. Uni-Hamburg, Wiesbaden: Deutscher Universitäts-Verlag GmbH, 1996

[Howard 2001] D. Howard: *Leveraging Web Services – Where are the Web Services that are going to change the face of computing?* Dez. 2001 S. 7-10 in [WSJ 2001], 2001

[HunterCrawford 1998] J. Hunter, W. Crawford: *JAVA Servlet Programming*. Sebastopol: O'Reilly & Associates, 1998

[Isakowitz et al. 1998] T. Isakowitz, M. Bieber, and F. Vitali: *Web Information Systems.* Communications of the ACM, 41(7), S. 78-80, 1998

[ISOC 2000] B. M. Leiner, V. G. Cerf, D. D. Clark, R. E. Kahn, L. Kleinrock, D. C. Lynch, J. Postel, L. G. Roberts, S. Wolff: *A Brief History of the Internet, version 3.31*. Reston: Internet Society (ISOC), 2000

[JCP 168] *Portlets.* Java Community Process (JCP) Nr. 168
http://www.jcp.org – Stand: 2002-09-19

[KererKirda 2001] C. Kerer, E. Kirda: *Layout, Content und Logic Seperation in Web Engineering.* S. 135-147 in [MurugesanDeshpande 2001], 2001

[Kieback et al. 1992] A. Kieback, H. Lichter, M. Schneider-Hufschmidt, H. Züllighoven: Prototypen in Industriellen Software-Projekten – Erfahrungen und Analysen. Informatik-Spektrum, Bd. 15, Nr. 2, 1992

[Klischewski 2000] R. Klischewski: *Abstrakte Bedürfnisse und konkrete Beziehungen oder: Wie man Services (nicht) modelliert.* In: J. Ebert, U. Frank (Hrsg.): Modelle und Modellierungssprachen in Informatik und Wirtschaftsinformatik. Proceedings Modellierung 2000, S. 19-22, Koblenz: Fölbach, 2000

[Kuri 1999] J. Kuri: *Gemischtwarenladen - Kostenlose Dienstleistungen über Portal Sites*: c't 4/1999, S.122ff. Hannover: Verlag Heinz Heise, 1999

[LibertyAlliance 2002] *Liberty Alliance Releases Version 1.1 Specification Draft for Public Review*
http://www.projectliberty.org – Stand 2002-11-28

[Lipert et al. 2001] M. Lippert, H. Wolf, H. Züllighoven: *Domain Services for Multichannel Application Software*. In: Proceedings of the Hawaii Interantional Conference On System Sciences 2001, HICSS 34, IEEE Computer Society, 2001

[Longshaw 2001] A. Longshaw: *Exploding Web Service Myths – Unrealistic expectations may be the real vilain*. Dez. 2001 S.20-21 in [WSJ 2001], 2001

[LublinskyFarell 2001] B. Lublinsky, M. Farell: *Web Services and Distributed Component Platforms Part II*. Dez. 2001 S.56-64 in [WSJ 2001], 2001

[Manes 2001] A. T. Manes: *Enabling Open, Interoperable, and Smart Web Services - The Need for Shared Context*. Palo Alto: Sun Microsystems, Inc., 2001

[McLaughlin 2000] B. McLaughlin: *JAVA and XML*. Sebastopol: O'Reilly & Associates, 2000

[Meyer 1987] Meyers Lexikonredaktion: *Meyers grosses Taschenlexikon in 24 Bd.*: 2., neu bearbeitete Auflage: Mannheim: B.I.-Taschenbuchverlag, 1987

[MeyerB 1997] B. Meyer: *Object-oriented software construction*, second edition. New Jersey: Prentice Hall, Inc., 1997

[Microsoft 2002a] Microsoft *.net Homepage*
http://www.microsoft.com/net/ – Stand: 2002-09-17

[Microsoft 2002b] Microsoft *.net Passport Homepage*
http://www.microsoft.com/netservices/passport/ – Stand: 2002-11-28

[Monson-Haefel 1999] R. Monson-Haefel: *Enterprise JavaBeans*, Second Edition. Sebastopol: O'Reilly & Associates, 2000

[MurugesanDeshpande 2001] S. Murugesan, Y. Deshpande (Hrsg.): *WebEngineering 2000*, LNCS 2016, Berlin-Heidelberg: Springer-Verlag, 2001

[Murugesan et al. 2001] Y. Deshpande, A. Ginge, S. Hansen, S. Murugesan: *Web Engineering: A New Discipline for Development of Web-based Systems*. S. 3-13 in [MurugesanDeshpande 2001], 2001

[Noah 065] Noah Lite Ver. 0.65: ArsLexis: http://www.arslexis.com – Stand: 2002-09-17

[Oesterreich 1997] Objektorientierte Softwareentwicklung mit der Unified Modeling Language, 3., aktualisierte Auflage: München; Wien: Oldenbourg Verlag, 1997

[OttoSchuler 2000] M. Otto, N. Schuler: *Fachliche Services: Geschäftslogik als Dienstleistung für verschiedene Benutzungsschnittstellen-Typen*. Diplomarbeit am Fachbereich Informatik, Arbeitsbereich Softwaretechnik, Uni-Hamburg, 2000

[Gómez et al. 2000] J. Gómez , C. Cachero, O. Pastor: *Extending a Conceptual Modelling Approach to Web Application Design*. B.Wangler, L. Bergman (Hrsg.):CAiSE 2000, LNCS 1789, S. 79-93. Berlin-Heidelberg: Springer-Verlag, 2000

[Cachero et al. 2001] C. Cachero, J. Gómez, O. Pastor: *Object-Oriented Conceptual Modeling of Web Application Interfaces: the OO-HMethod Presentation Abstract Model*. K. Bauknecht, S. Kumar Madria, G. Pernul (Hrsg.): EC-Web 2000, LNCS 1875, S. 206-215, Berlin-Heidelberg: Springer-Verlag, 2001

[Penguin 1993] L. A. Hill (Hrsg.): *The Penguin English Student's Dictionary*. München: Deutscher Taschenbuch Verlag GmbH & Co. KG, 1993

[Pierer 1862] H. A. Pierer (Hrsg.): *Pierer's Universal-Lexikon der Vergangenheit und Gegenwart oder Neuestes encyclopädisches Wörterbuch der Wissenschaft, Künste und Gewerbe.*: Vierte, umgearbeitete und stark vermehrte Auflage: 15. Band: Altenburg: Verlagsbuchhandlung von H. A. Pierer, 1862

[Raines 1998] P. Raines: *Tcl/Tk Pocket Reference*. Sebastopol: O'Reilly & Associates, 1998

[RDF 1999] *Resource Description Framework (RDF) Model and Syntax Specification,* W3C Recommendation 22 February 1999
http://www.w3.org/TR/REC-rdf-syntax – Stand: 2002-10-21

[RSS0.91 1999] D. Libby: *RSS 0.91 Spec,* revision 3, Netscape Communications 1999
http://my.netscape.com/publish/formats/rss-spec-0.91.html – Stand: 2001-11-02
http://my.netscape.com/publish/formats/rss-0.91.dtd – Stand: 2002-04-12

[RSS1.0 2001] RDF Site Summary RSS 1.0
http://purl.org/rss/1.0/spec – Stand: 2002-10-21

[SchumacherSchwickert 1999] M. Schumacher, A. C. Schwickert: *Web-Portale – Stand und Entwicklungstendenze*, Arbeitspapiere WI, Nr. 4/1999, Mainz: Lehrstuhl für Allg. BWL und Wirtschaftsinformatik, Johannes Gutenberg-Universität, 1999

[SleeperRobins 2001] B. Sleeper, B. Robins: *Defining Web Services - An Analysis Memo from The Stencil Group*, 2001:
http://www.stencilgroup.com – Stand: 2002-08-10

[Snell et al. 2002] J.Snell, D. Tidwell, P. Kulchenko: *Programming Web Services with SOAP*. Sebastopol: O'Reilly & Associates, 2002

[SOAP 2000] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Frystyk Nielsen, S. Thatte, D. Winer: *Simple Object Access Protocol (SOAP) 1.1 - W3C Note 08 May 2000*, W3C, 2000
http://www.w3.org/TR/SOAP – Stand: 2002-12-22

[Sun 2002] *Sun ONE Architecture Guide – Delivering Services on Demand*. Palo Alto, Sun Microsystems, Inc., 2002
http://www.sun.com/sunone/ – Stand: 2002-09-17

[Turau 1999] V. Turau : *Techniken zur Realisierung Web-basierter Anwendungen.* In: Informatik Spektrum 22, 3-12, Springer-Verlag, 1999

[TDNE 2001] *Trend Duden – Wörterbuch der New Economy*, Herausgegeben und bearbeitet vom Trendbüro Hamburg, ISBN 3-411-71171-X, 2001
http://www.neweconomy-duden.de – Stand: 2002-08-10

[Tomcat 3.2.2] *Tomcat 3.2.2 FAQ, Webapplikation*
http://jakarta.apache.org

[UML1.4 2001] *OMG Unified Modeling Language Specification*, OMG Headquarters 250 First Avenue, Suite 201, Needham, MA 02494, 2001
http://www.omg.org

[Whatis 2002] Suchbegriffe: *push technology / webcasting / netcasting*
http://www.whatis.com – Stand: 2002-12-19
http://searchwebservices.techtarget.com – Stand: 2002-12-19

[W3C 2002] *Web Site des World Wide Web Consortium (W3C)*
http://www.w3c.org – Stand: 2002-09-19

[W3CWS 2002] *Web Services Activity* – Web Site des World Wide Web Consortium (W3C)
http://www.w3c.org /2002/ws/ – Stand: 2002-09-18

[Wegner 2002] H. Wegner: *Analyse und objektorientierter Entwurf eines integrierten Portalsystems für das Wissensmanagement*. Dissertation, TU Hamburg-Harburg, 2002

[WSJ 2001] *Web Service Journal*, SYS-CON Publication, Inc., 135 Chestnut Ridge Road, Montvale, NJ 07645
http://www.sys-con.com – Stand: 2002-09-17

[Szyperski 1999] C. Szyperski: *Component Software Beyond Object-Oriented Programming*. Harlow: Pearson Education Limited, 1999

[Züllighoven 1998] H. Züllighoven: *Das objektorientierte Konstruktionshandbuch nach dem Werkzeug- & Material-Ansatz*. Heidelberg: dpunkt, 1998