TU Wien

Business Informatics Group
Institut für Softwaretechnik und Interaktive Systeme

Ubiquitäre Web-Anwendungen

Modellierung und Implementierung von Kontextinformation

Magisterarbeit zur Erlangung des akademischen Grades eines Magister der Sozial- und Wirtschaftswissenschaften (Mag. rer. soc. oec.)

eingereicht bei o. Univ.-Prof. Mag. Dipl.-Ing. Dr. Gerti Kappel
Mitbetreuung: Mag. Andrea Schauerhuber

Arnold Weissensteiner

Wien, 29. Januar 2007

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Wien, 29. Januar 2007

Danksagung

Im Besonderen möchte ich mich bei meinen Eltern Christa und Josef Weissensteiner für die finanzielle und moralische Unterstützung während meiner ganzen Ausbildung bedanken. Sie haben mir meinen gewählten Bildungsweg ermöglicht und sind immer hinter meinen Entscheidungen und mit Rat und Tat zur Seite gestanden.

Für die gute Betreuung und die nützlichen Anregungen bei der Erstellung dieser Magisterarbeit möchte ich mich bei Mag. Andrea Schauerhuber und o. Univ.-Prof. Mag. Dipl.-Ing. Dr. Gerti Kappel bedanken.

Ein ganz besonderes Dankeschön geht an Petra Brosch und Rudi Mayer für die tolle Zusammenarbeit während der gesamten Studienzeit und speziell während dieser Magisterarbeit.

Kurzfassung

Ubiquitäre Web-Anwendungen sollen, entsprechend dem *anytime/anywhere/ anymedia* Paradigma, dem Anwender, egal wann, wo und mit welchem Gerät er die Anwendung nutzt, einen individuell abgestimmten und auf die Rahmenbedingungen des Benutzers angepassten Inhalt liefern.

Im Rahmen einer Kooperation von drei Magisterarbeiten [Bros06, Maye06, Weis06] wurde ein ubiquitäres Tourismusinformationssystem entwickelt. Das Ziel dieses umfangreichen Projekts war die Konzeption, Modellierung und Implementierung einer Web-Anwendung mit Customizationunterstützung, dh. einer Web-Anwendungen, die aufgrund verschiedener Kontextfaktoren wie Benutzer, Zeit, Ort, Gerät, etc., mit der Adaptierung ihrer Dienste reagiert.

Hierbei ist eine entsprechende Kontextbehandlung insbesondere betreffend die Repräsentation, die Erfassung und die Auswertung von Kontextinformation von entscheidender Bedeutung. Darüber hinaus ergibt sich einerseits das Problem, dass sich die Kontexterfassung und die Auswertung des aktuellen Kontexts über die gesamte Web-Anwendung erstreckt. Andererseits sind diese Funktionalitäten meist auch fix im Sourcecode verankert, wodurch die Wartung und Flexibilität des Systems erheblich eingeschränkt wird. Daher sollen diese Komponenten möglichst gekapselt und separiert von der restlichen Anwendung implementiert werden, was in dieser Arbeit mit Hilfe von aspektorientierter Programmierung gelöst wird.

Für ubiquitäre Web-Anwendungen und die Customization ihrer Dienste ist der Kontext das Basiselement, ohne welchem keine Adaptierung möglich ist. Die Rolle des Kontext in ubiquitären Web-Anwendungen ist das Thema dieser Arbeit. Hierbei wird auf die Erfassung, Repräsentation und Auswertung von Kontextinformationen eingegangen, wobei Grundlagen verschiedener Kontextmodelle für die Repräsentation dargestellt werden und ein Einblick in Regelsysteme für die Auswertung von Kontextinformationen gegeben wird. Darüber hinaus werden bestehende Ansätze für Kontext verarbeitende Frameworks vorgestellt. Im Anschluss wird auf die praktische Umsetzung der Kontextbehandlung und das entwickelte Regelsystem für die erforderlichen Adaptierungen im entwickelten Forschungsprototyp eingegangen.

Abstract

Ubiquitous web applications adhering to the *anytime/anywhere/anymedia* paradigm are required to yield a customized content in respect of the user's context.

As a cooperation of three master theses [Bros06, Maye06, Weis06], we have developed a ubiquitous tourism information system. Our intention was to design and implement a ubiquitous web application with customization support, i.e., a web application which adapts its services according to several context properties such as user, time, location, device, etc.

To fulfil this task, it is crucial to handle context information in an appropriate way, particularly regarding its representation, acquisition, and interpretation. It has to be considered, that this functionality is scattered across the code of the whole system and that functionality concerning context handling is often hard-coded into the system, which consequently has a negative effect in terms of maintainability, extensibility, and changeability. Under these circumstances, separation of concerns is realized with aspect-oriented programming.

The key prerequisite for ubiquitous web applications and their customization is the actual context. Without context, no adaptation is possible. For this reason, the role of context in ubiquitous web applications is the main subject of this master theses. In this respect, the acquisition, representation and interpretation of context will be investigated. First, concerning context representation and interpretation, the basics of context models are presented and a short introduction to rule engines is given. Next, an overview of different existing context-aware frameworks is provided. The lessons learned from the theoretical exploration are feeded into the implementation of our ubiquitous web application prototype by means of which we then describe our solution for the mentioned challenges.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die immer weiter steigende Verbreitung des World Wide Web bringt ständig neue Herausforderungen mit sich. War es früher das Ziel mit einer umfassenden Web-Anwendung die Anforderungen möglichst vieler Benutzer auf einmal abzudecken, so versucht man heute vermehrt, jedem Benutzer seine eigene, speziell auf ihn abgestimmte, Web-Anwendung zu liefern. Diese Art von Web-Anwendungen nennt man *ubiquitäre Web-Anwendungen*.

Die Anpassung einer Web-Anwendung an ihren Nutzungskontext heißt *Customization*. Customization beinhaltet die Akquisition und Aufbereitung des aktuellen Kontext, sowie die Adaptierung, also die eigentliche Anpassung. Diese Adaptierung erfolgt mit Hilfe eines Regelsystems, mit welchem basierend auf dem gegebenen Kontext sinnvolle Anpassungen der Web-Anwendung ausgelöst werden.

1.1 Problem

Die Entwicklung von ubiquitären Web-Anwendungen ist komplex. Das liegt daran, dass Customization an jeder Stelle der Web-Anwendung denkbar ist. Bei der Entwicklung ubiquitärer Web-Anwendungen muss daher nach [FSK+02,SSK+06], im Vergleich zum „herkömmlichen“ Web-Engineering, zusätzlich zu den bekannten Dimensionen *Levels* (dh. Content-, Hypertext-, und Präsentations-Levels von Web-Anwendungen), *Features* (dh. Struktur und Verhalten von Web-Anwendungen) und *Phases* (dh. die Entwicklungsphasen von Web-Anwendungen) die *Customization* als eigene Dimension berücksichtigt werden.

Ausgangspunkt für jede Art von Customization ist die Kenntnis des aktuellen Kontext. Dieser bildet die Grundlage für die Adaptierung der Web-Anwendung und entscheidet somit auch über die Qualität der Customization. Hierbei ergibt sich einerseits das Problem, dass sich die Kontexterfassung genauso wie die Cu-

stomization verteilt über die gesamte Web-Anwendung erstreckt und somit die Wartung, Erweiterung und Wiederverwendung erschwert wird. Andererseits steht man bei der Kontextauswertung und dem Auslösen von passenden Adaptierungen wiederum vor dem Problem, dass diese meist fix im Sourcecode verankert sind was die Flexibilität und Wartung der Anwendung erheblich einschränkt. Dieses Problem hat man sowohl bei den meisten bestehenden Context-Aware Frameworks wie auch bei Eigenentwicklungen.

1.2 Ziel der Arbeit

Ziel dieser Arbeit ist es, ein ubiquitäres Tourismusinformationssystem zu konzipieren und implementieren, wobei die Customization durch aspektorientierte Programmierung realisiert werden soll. Wichtig hierbei ist, dass der Kontext möglichst flexibel erfasst wird und die dahinter stehenden Regelsysteme einfach und leicht änderbar gestaltet sind, damit die Anwendung entsprechend einfach auf geänderte Rahmenbedingungen angepasst werden kann. Ein passendes Beispiel hierfür ist das Hinzufügen und Entfernen von Regeln zur Laufzeit. Daher sollen diese Komponenten möglichst gekapselt von der restlichen Anwendung implementiert werden, wofür sich der Einsatz von aspektorientierter Programmierung (AOP) anbietet. Diese stellt ein neues Programmier-Paradigma dar, um so genannte Crosscutting Concerns zu modularisieren [KLM+97]. Crosscutting Concerns sind verschiedene Funktionalitäten im Programm, die mit den objektorientierten Konzepten wie zB Klassen nicht gekapselt werden können, sondern in vielen Klassen des Programms verteilt auftreten.

Das entwickelte Tourismusinformationssystem unterstützt die Anpassung an mehrere Kontextfaktoren wie Benutzer, Zeit, Ort und Endgerät. Im Speziellen wird in dieser Arbeit auf die Rolle des Kontext bei ubiquitären Web-Anwendungen, beginnenden bei der Akquirierung über die Repräsentation im System bis hin zur Auswertung, eingegangen und eine Lösung für die entwickelte Web-Anwendung dargestellt. Die Entwicklung des Tourismusinformationssystems geschah in Zusammenarbeit mit Petra Brosch [Bros06] und Rudolf Mayer [Maye06][1].

[1] Aus dieser Zusammenarbeit resultieren teilweise ähnliche bzw. idente Texte. Daher wird im Anhang in Tabelle A.4 im Detail aufgelistet, von welchem Autor welche Teile stammen.

1.3 Aufbau der Arbeit

In einer Einführung in die theoretischen Grundlagen von ubiquitären Web-Anwendungen werden unter anderem die Begriffe der Adaptierung und des Kontext vorgestellt (siehe Kapitel 2). In Kapitel 3 folgt eine detaillierte Betrachtung der Rolle des Kontextes in ubiquitären Web-Anwendungen. Danach wird in Kapitel 4 auf die Grundlagen der Aspektorientierung eingegangen und in Kapitel 5 schließlich das implementierte Tourismusinformationssystem und seine Funktionalität vorgestellt. Anschließend folgen in Kapitel 6 die Implementierungsdetails in Bezug auf Customization, die verwendeten Kontextfaktoren und das entwickelte Regelsystem des Tourismusinformationssystems. Mit einem Erfahrungsbericht und einem Ausblick schließt die Arbeit in Kapitel 7.

Kapitel 2

Web Engineering

Das Internet, im Speziellen das World Wide Web (WWW), ist aus unserem Alltag kaum mehr wegzudenken. Das WWW ist bereits in nahezu allen Bereichen unseres Lebens ein fester Bestandteil, ganz gleich ob in Beruf oder Freizeit. Es hat nicht nur unsere Gesellschaft nachhaltig beeinflusst, sondern hat auch eine neue Ära der Informatikdisziplin eingeleitet. Zu Beginn stellte das WWW ein dokumentenzentriertes *„read-only“* Informationsmedium mit statischen Inhalten dar. Heute wird das Web immer mehr für vollwertige, komplexe Anwendungen genutzt, die interaktive, datenintensive und personalisierbare Dienste zur Verfügung stellen und über verschiedene Endgeräte erreicht werden können. Web-Anwendungen unterscheiden sich von herkömmlichen Softwaresystemen in erster Linie durch ihren Einsatz im Web. In [KPRR04] wird der Begriff Web-Anwendung wie folgt definiert:

> Eine Web-Anwendung ist ein Softwaresystem, das auf Spezifikationen des World Wide Web Consortium (W3C) beruht und Web-spezifische Ressourcen, wie Inhalte und Dienste bereitstellt, die über eine Benutzerschnittstelle, den Web-Browser, verwendet werden.

Eine Web-Anwendung ist somit ein Softwaresystem, das von Beginn an für den Einsatz im WWW entwickelt wird und somit den Hypermedia Aspekt[1], also die Kombination von Hypertext, Multimedia und Anwendungslogik, mit einbezieht. Eine Menge von Web-Seiten sind erst dann eine Web-Anwendung, wenn sie über das klassische Request-Response Paradigma hinausgehen und zustandsbasiert sind, also den Einsatz einer *Session* erfordern [FSK+02].

[1] In diesem Kapitel wird der Aspekt-Begriff im herkömmlichen Sinn und nicht im Sinne der Aspektorientierung verwendet.

2.1 Historische Entwicklung von Web-Anwendungen

Web-Anwendungen werden aufgrund ihres Komplexitätsgrades und der Entwicklungshistorie in verschiedene Kategorien gegliedert, denen eindeutige Beispiele zugeordnet werden können. Wie Abbildung 2.1 aus [KPRR04] veranschaulicht, steigt der Komplexitätsgrad mit der zeitlichen Entwicklung, dh. höhere Entwicklungsstufen bauen auf den ihnen vorangegangenen auf.

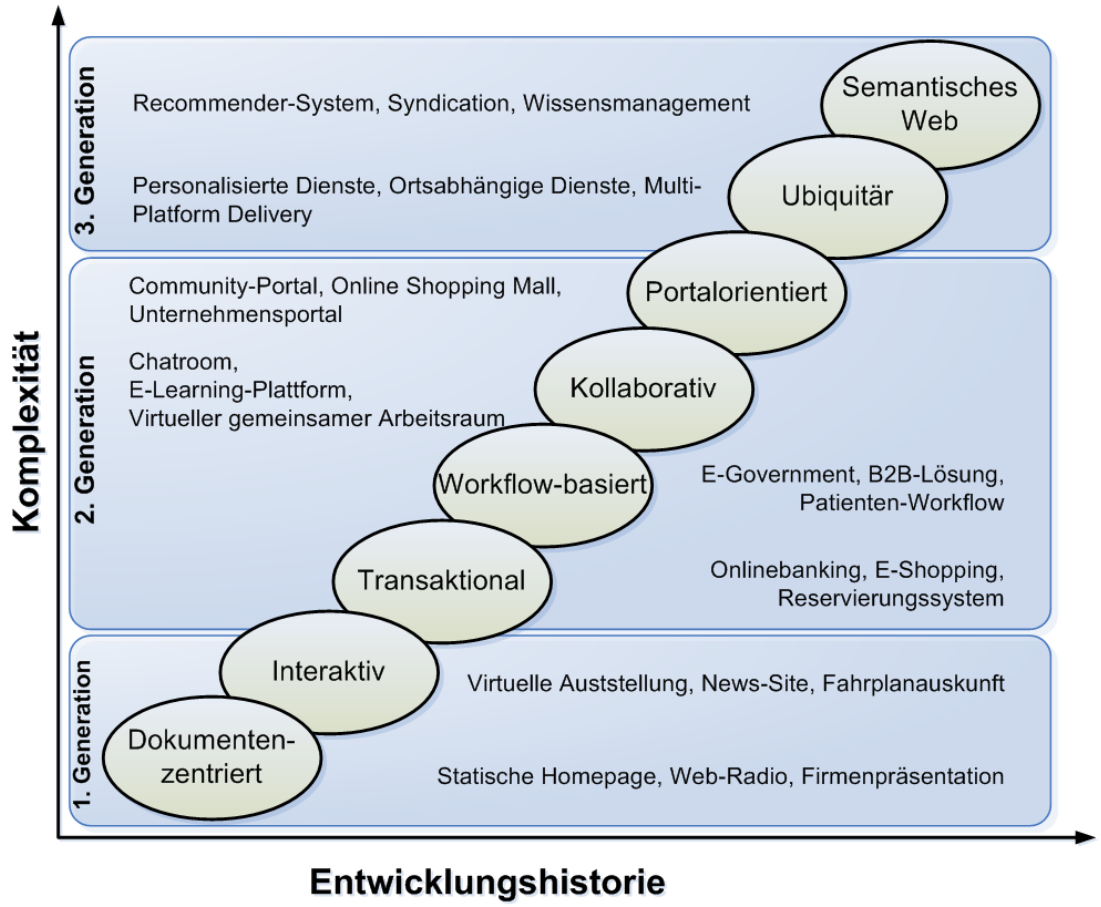

Abbildung 2.1: Kategorien von Web-Anwendungen

2.1.1 Generationen von Web-Anwendungen

Generell können diese Kategorien in drei Generationen von Web-Anwendungen eingeteilt werden, die sich vor allem durch die verwendete Technologie und den angebotenen Services voneinander abheben [HKRS02]. Die *erste Generation* umfasst „read-only“ Web-Seiten, die anonymen Benutzern der Informationsgewinnung dienen. Die *zweite Generation* sind bereits laut obiger Definition *„echte“*

Web-Anwendungen, also zustandsbasierte Softwaresysteme mit zugrunde liegender Datenbank, die Dienste mit Benutzertransaktionen zur Verfügung stellen. Hierzu gehören auch E-Commerce Anwendungen, die durch personalisierte Services, also auf den jeweiligen Benutzer angepasste Dienste, diese Generation besonders geprägt haben. Derzeit aktuell ist die *dritte Generation*, welche durch ubiquitäre Web-Anwendungen charakterisiert ist. Ubiquitäre Web-Anwendungen gehen einen Schritt weiter als Web-Anwendungen der zweiten Generation und bieten kontextabhängige Dienste über jedes beliebige Endgerät an, wodurch der Zugriff allgegenwärtig wird.

2.2 Ubiquitäre Web-Anwendungen

Ubiquitäre Web-Anwendungen sind spezielle Web-Anwendungen, die das *anytime/anywhere/anymedia Paradigma* umsetzen. Während die permanente Verfügbarkeit (*anytime*) in der Natur des Web liegt und nichts Neues ist, werden ubiquitäre Web-Anwendungen durch die Charakteristika *anywhere* und *anymedia* geprägt. Die Bezeichnung *ubiquitär* leitet sich vom lateinischen Wort *ubique* ab und bedeutet allgegenwärtig. Den Terminus *Ubiquitous Computing* hat Mark Weiser in seinem Artikel „The Computer for the 21st Century“ geprägt [Weis91].

Darin betont Mark Weiser die automatische Anpassung des Verhaltens von Computern an den aktuellen Standort (*anywhere*).

> [...] We have found two issues of crucial importance: location and scale. Little is more basic to human perception than physical juxtaposition, and so ubiquitous computers must know where they are. (Today's computers, in contrast, have no idea of their location and surroundings.) If a computer merely knows what room it is in, it can adapt its behavior in significant ways without requiring even a hint of artificial intelligence. [...] [Weis91]

Bei ubiquitären Web-Anwendungen spielt der Standort eine zentrale Rolle. Allerdings steht im Gegensatz zur Theorie von Mark Weiser nicht der Standort des Web-Servers, auf dem die Web-Anwendung installiert ist, im Vordergrund, sondern der Standort des Benutzers, der die Web-Anwendung nutzt.

Weiters beschreibt Mark Weiser seine Vision der totalen, unsichtbaren Vernetzung, in der jeder Gegenstand „intelligent“ ist (*anymedia*).

> [...] When almost every object either contains a computer or can have a tab attached to it, obtaining information will be trivial: „Who

> made that dress? Are there any more in the store? What was the name of the designer of that suit I liked last week?“ The computing environment knows the suit you looked at for a long time last week because it knows both of your locations, and, it can retroactively find the designer's name even if it did not interest you at the time. [...] [Weis91]

Ubiquitäre Web-Anwendungen können über jedes beliebige Endgerät (Handies, PDAs, PCs, etc.) benutzt werden. Sie passen ihre Darstellung und ihr Verhalten an die zur Verfügung stehenden Ressourcen der unterschiedlichsten Zugriffsmedien an. Dazu gehören unter anderem die Größe und Farbunterstützung des Displays, die Möglichkeiten zur Dateneingabe, Netzwerkverbindung und lokaler Speicherplatz.

Ubiquitäre Web-Anwendungen bieten neben der angepassten Darstellung auch an den aktuellen Nutzungskontext angepasste Inhalte. Der Nutzungskontext enthält Faktoren wie Ort, Zeit, Benutzer, etc. Die Aquisition dieser Kontextfaktoren und die daraus resultierende Adaptierung der Web-Anwendung heißt Customization.

2.2.1 Customization

Customization ist der Schlüssel für individuell auf den Benutzer abgestimmte Web-Anwendungen. Bei der Customization wird, basierend auf dem aktuellen Nutzungskontext (siehe 2.2.2), mittels einem Regelsystem (Mapping) (siehe 2.2.4) die Web-Anwendung spezifisch auf den jeweiligen Benutzer und seinen bekannten Kontext abgestimmt. Der Umfang der Customization kann hierbei sehr stark variieren. So ist von einer einfachen Anpassung der Darstellung der Web-Anwendung an das entsprechende Gerät des Benutzers bis hin zu einer kompletten Adaptierung von Content-, Hypertext- und Präsentations-Levels der Web-Anwendung alles möglich.

Ursprung der Customization

Historisch gesehen wurde der Begriff der Customization vor allem von der *Personalisierung* und dem *Mobile-Computing* beeinflusst [KPRS03]. In Folge wird auf einzelne Bereiche aus Abbildung 2.2 in Anlehnung an [KPRS03] eingegangen.

Personalisierung. Die Personalisierung war eine der ersten Customizationsaktionen die angedacht und umgesetzt worden ist und hat ihre Ursprünge

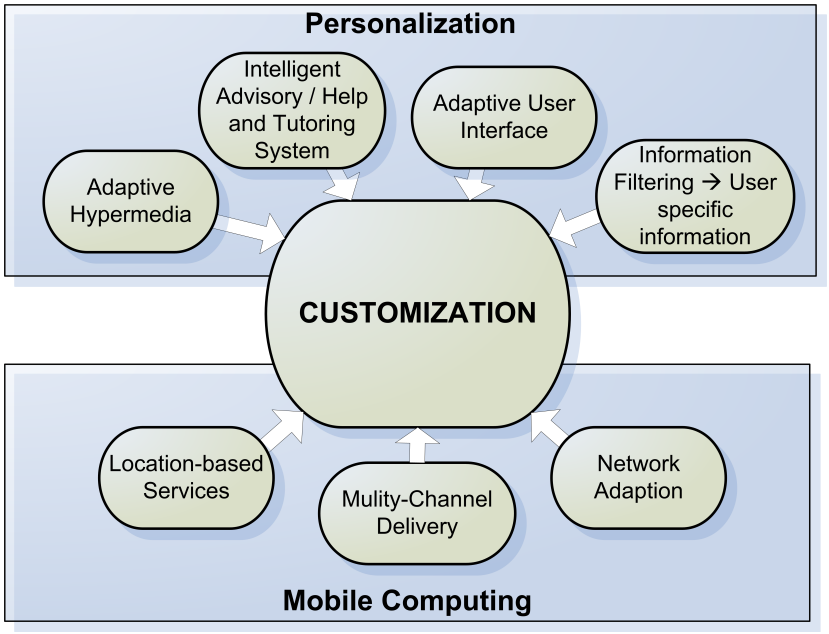

Abbildung 2.2: Ursprünge der Customization [KPRS03]

in den früheren Achtzigern. Hierbei stand zuerst die *Anpassung des User-Interfaces* im Mittelpunkt des Interesses. Das Ziel war, die Interfaces der Anwendungen an die Fähigkeiten, Aufgaben und Wünsche des Benutzers anzupassen [GoWJ84] beziehungsweise das Interface für den Benutzer anpassbar zu machen.

Danach folgten im Bereich der Personalisierung *intelligente Lehr- und Hilfssysteme*, welche ihre Funktion und Lehrstrategien auf die individuellen Bedürfnisse der Benutzer, je nach Wissensstand und Lernfortschritt, automatisch abstimmen.

Weiters kommen von der Personalisierungsseite die für die Customization immens wichtigen Bereiche der *Informationsfilterung* [AvZe97, LoTe92] und *Adaptive Hypermedia* [BrMa02]. Das finale Ziel dieser beiden Bereiche ist es, aus dem umfassenden Pool an Daten einer Web-Anwendung einerseits genau jene Informationen zu liefern, welche für den Benutzer interessant sind und diese andererseits auch benutzerspezifisch aufzubereiten und darzustellen.

Mobile-Computing. Das Mobile-Computing kann als zweites großes Standbein der Customization gesehen werden. Hierbei wurden die ersten Ansätze in den frühen Neunzigern gemacht, wobei zu Beginn alles im Zeichen von ortsabhängigen Diensten gestanden ist [WaSc01]. Dies ist naheliegend, da sich hierfür eine Fülle von Anwendungsgebieten geboten haben und noch immer bieten.

Ortsbasierte Dienste. Das Ergebnis dieser Entwicklung zeigt sich heute in der Vielzahl von ortsbasierten Diensten. Solche sind zum Beispiel im Outdoorbereich Flottenmanagementsysteme, Verkehrskontroll-, Leit- und Notrufsysteme sowie im Indoorbereich elektronische Museumsführer. Weitere Beispiele zu ortsbasierten Diensten findet man in [ChKo00, EPS+01, KPRS03, KDJ+04].

Für die Ermittlung der benötigten Ortsdaten stehen eine Reihe von Technologien zur Verfügung, welche sich je nach Anwendungsfall unterschiedlich gut eignen, wobei hier vor allem zwischen der Erfassung der Ortsdaten im Freien oder innerhalb von Gebäuden differenziert werden muss. Als Auswahl können hier GPS, Mobilfunkanlangen, Kameras, RFID, Magnetkarten und Barcodesysteme genannt werden [BaDR06].

Multi Channel Delivery. Ein weiterer wichtiger Bereich von Mobile Computing ist das so genannte *Multi Channel Delivery*. Hiermit wird es ermöglicht, die Web-Anwendung auf unterschiedlichsten (mobilen) Endgeräten nutzen zu können. Dabei müssen die verschiedenen Spezifikationen der Endgeräte auf Hardware- und auf Softwareseite erfasst und berücksichtigt werden, um eine angemessene Adaptierung der Web-Anwendung in Bezug auf die grafischen Benutzeroberfläche, Interaktionsmöglichkeiten für den Benutzer und den Inhalt an sich durchführen zu können. Solche Spezifikationen umfassen beispielsweise die Display-Größe und Auflösung, die Farbtiefe, die Speichergröße und Rechenleistung sowie das Betriebssystem und den verwendeten Browser des Endgerätes.

Um den Aufwand für die Erkennung der entsprechenden technischen Daten und Funktionen nicht ausufern zu lassen, haben schon sehr früh Standardisierungsbemühungen eingesetzt. Das *The World Wide Web Consortium (W3C)*[2] bietet mit seinen *Device Independence* Aktivitäten eine Reihe von Standardisierungsansätzen [W3C003], wobei hier speziell das *Composite Capability/Preference Profile (CC/PP)* [W3C004] hervorzuheben ist, welches standardisierte Profile für Endgerätespezifikationen und Benutzerpräferenzen darstellt. Für die Profilerstellung wird das *Resource Description Framework (RDF)*[3] benutzt. Hierfür existiert bereits ein umfangreiches Vokabular für die Spezifikation der einzelnen Eigenschaften der Geräte.

Network Adaptation. Nachdem Mobile-Computing mit drahtloser Datenübertragung einhergeht, bei welcher verschiedene Netzwerkbedingun-

[2] http://www.w3.org/
[3] http://www.w3.org/RDF/

gen zu berücksichtigen sind, liegt es nahe, dass auch diese bei der Customization mit berücksichtigt werden. Dieser immer bedeutender werdende Forschungsbereich nennt sich *Network Adaptation*.

Hierbei waren die ersten Ansätze darauf ausgelegt die Web-Anwendungen und Dienste so robust zu machen, dass diese mit unerwarteten Verbindungsabbrüchen umgehen und sich an diese anpassen können. Solche Abbrüche können bewusst durch den Benutzer herbeigeführt werden, in dem dieser zum Beispiel einfach die Anwendung beendet oder das Endgerät ausschaltet, oder aber auch durch einen leeren Akku oder einem Verbindungsabbruch im Netzwerk auftreten.

Darüber hinaus muss man davon ausgehen, dass die Bandbreiten der Datenübertragung zu unterschiedlichen Endgeräten verschieden hoch sind und auch fortwährenden Schwankungen unterliegen [BFK+00]. Darum soll man auch dieses Faktum bei der Customization mit einbeziehen und den Inhalt, welcher übertragen werden soll, entsprechend anpassen. Dies kann einerseits durch verschiedene Kompressionsmechanismen erreicht werden oder aber auch durch eine Filterung des Inhalts an sich. So kann man bei einer niedrigen Bandbreite zum Beispiel auf multimediale Inhalte komplett verzichten.

Wie man hier sieht, hat Customization in gewissem Maße in einzelnen Bereichen der Informatik schon lange fußgefasst. Der Punkt hierbei ist jedoch dass in Forschungsarbeiten genauso wie im Einsatz in der Produktivumgebung meist nur Teilgebiete der Customization umgesetzt werden. Hier bietet die Customization noch ein großes Potential für zukünftige Anwendungen.

Definition des verwendeten Customization Begriffes

In dieser Arbeit wird der Customization Begriff verwendet, wie er in [KaRS00] definiert worden ist:

> [...] In our terms, customization specializes a web application called core web application towards the particular circumstances of consumption called context. The result of the customization process represents a customized web application. [...]

Aus dieser Definition geht bereits hervor, wie umfassend sich der Einbezug von Customization auf die Entwicklung einer Web-Anwendung auswirken kann. In Abbildung 2.3 wird dies noch deutlicher gezeigt. Hierbei wird klar, dass sich die

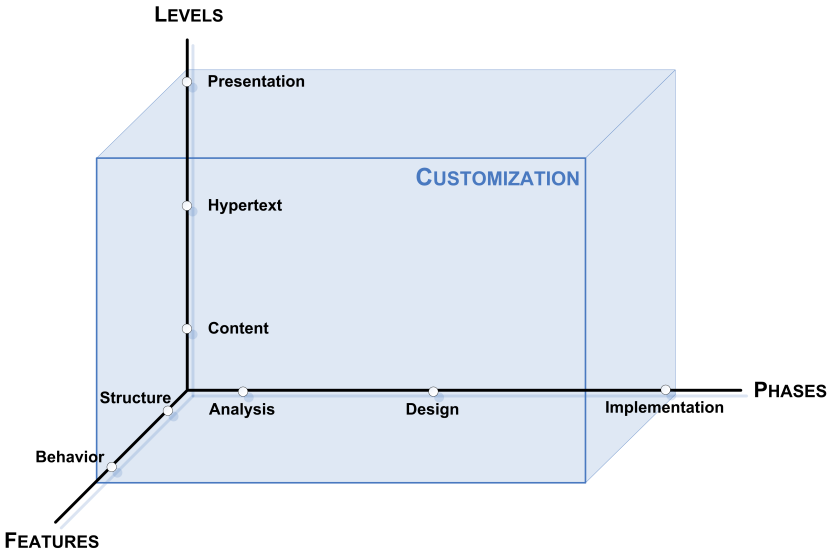

Abbildung 2.3: Umfang der Customization anhand der Modellierungsdimensionen von Web-Anwendungen [KaRS00]

Customization über alle bekannten Dimensionen bei der Entwicklung von Web-Anwendungen erstreckt. Die Customization muss also für alle *Levels*, *Features* und *Phases* einer Web-Anwendung mitberücksichtigt werden. Daher ist es nur sinnvoll, die Customization als eigene Dimension in die Entwicklung von Web-Anwendungen einzuführen [KaRS00].

Customization wird, wie in Abbildung 2.4 dargestellt, im Prinzip durch zwei Hauptcharakteristika bestimmt. Einerseits durch die Umstände der Verwendung der Web-Anwendung, was dem *Nutzungskontext*, auch kurz *Kontext* genannt, entspricht und andererseits durch die möglichen Änderungen an der Web-Anwendung an sich, der so genannten *Adaptierung* [KaRS00]. Beide Charakteristika können entweder statisch oder dynamisch sein, was sich direkt auf den Grad der Anpassbarkeit auswirkt. Web-Anwendungen, welche nur statischen Kontext und/oder statische Adaptierung unterstützen werden oft als *adaptable* bezeichnet wobei hingegen Anwendungen mit dynamischem Kontext/Adaptierung als *adaptive* bezeichnet werden [FiKS97]. Kontext und Adaptierung bestimmen gemeinsam das Ergebnis der Customization, wobei diese das Mapping zwischen den beiden Bereichen abbildet.

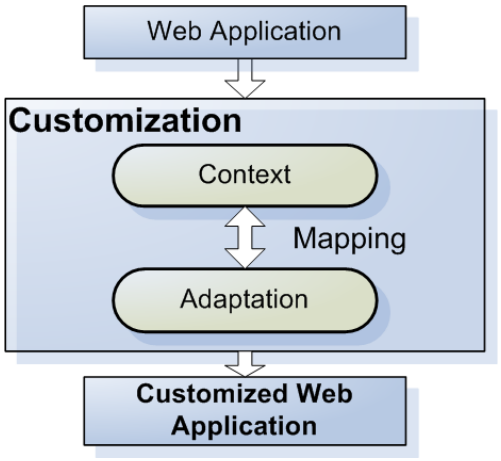

Abbildung 2.4: Customization

2.2.2 Kontext

Das *anytime/anywhere/anymedia* Paradigma von ubiquitären Web-Anwendungen und die damit verbundene Anpassung an den Kontext (Customization) stellt eine bei weitem größere Herausforderung dar, als die reine Personalisierung einer Web-Anwendung. Während bei der Personalisierung nur auf den Benutzer bzw. auf sein Verhalten Rücksicht genommen wird, bieten ubiquitäre Web-Anwendungen auch zeitabhängige, ortsabhängige, netzwerkabhängige und endgeräteunabhängige Services an [KPRS03], welche in Summe eine nach allen technisch machbaren Faktoren adaptierte und personalisierte Web-Anwendung ergeben.

Für die Adaptierung ist es somit wichtig zu wissen unter welchen Umständen die Anwendung verwendet wird. Diese Umstände (zB mit welchen Browser, auf welchem Gerät, zu welcher Zeit) werden im Folgenden als Kontext bezeichnet. Prinzipiell kann jede nur irgendwie verfügbare Information für die Adaptierung benutzt werden, aber natürlich sind nicht alle Informationen relevant. Es gilt somit zuerst herauszufinden, welche Kontextfaktoren relevant sind und wie sie für die Customization der Anwendung verwendet werden können [KaRS00]. Im Folgenden werden die allgemein für ubiquitäre Web-Anwendungen interessanten Kontextfaktoren in Anlehnung an [KRK+02] kurz dargestellt, wobei grundsätzlich zwischen dem physischen Kontext und dem daraus abgeleiteten logischen Kontext unterschieden wird.

Physischer Kontext

Der physische Kontext ist prinzipiell nicht modifizierbar und wird nicht von der Anwendung kontrolliert. Er bezieht sich auf die Umgebung und die Umstände un-

ter denen die Anwendung verwendet wird, welche sich natürlich in unregelmäßigen Abständen ändern können. Da die physischen Kontextfaktoren großteils anwendungsunabhängig bzw. generisch sind, können die für die Anwendung relevanten Faktoren je nach Art und Zweck der Anwendung unterschiedlich sein.

Natürlicher Kontext Der natürliche Kontext besteht aus den natürlichen Gegebenheiten, wie Ort oder Zeit, unter denen die Web-Anwendung verwendet wird.

Ort. Informationen über den Standort des Benutzers, von welchem aus er die Anwendung verwendet. Die Kenntnis über diesen Standort ist für 'Location Based Services' unabdingbar. Diese Informationen werden in der Regel nicht vom Endgerät selbst zur Verfügung gestellt, sondern werden von so genannten 'Location Server' abgefragt. Dazu kann einerseits die IP-Adresse oder bei mobilen Endgeräten, die über das GSM oder UMTS Netz eine Verbindung zum Internet herstellen, auch die 'CellId' ihrer Mobilfunkzelle verwendet werden.

Zeit. Der Kontextfaktor Zeit ermöglicht die Customization der Anwendung in Abhängigkeit bestimmter zeitlicher Einschränkungen, wie zum Beispiel die Öffnungszeiten von Geschäften oder die Fahrpläne von öffentlichen Verkehrsmitteln. Dabei ist zu beachten, dass sich der Benutzer der Anwendung durchaus in einer anderen Zeitzone befinden kann und somit die 'Serverzeit' nur bedingt verwendet werden kann.

Sprache. Informationen über die bevorzugte Sprache des Benutzers. Mit Hilfe dieser Informationen kann die Sprache der Anwendung an die des Benutzers angepasst werden. Die vom Benutzer bevorzugte Sprache hat er entweder selbst bekannt gegeben (siehe sozialer Kontext) oder wurde über den Browser automatisch festgestellt (siehe technischer Kontext).

Wetter. Das derzeitige Wetter, sowie die Vorhersage für eine bestimmte Anzahl an Tagen, für den Standort, an dem sich der Benutzer befindet. Diese Informationen, die in Abhängigkeit des Kontextfaktors Ort von einem 'Wetterserver' abgefragt werden, sind besonders für Tourismus und Freizeitportale interessant, denn damit kann das dem Benutzer präsentierte Angebot detaillierter angepasst werden.

Technischer Kontext Der technische Kontext besteht aus Informationen über den verwendeten Browser, das Endgerät, Netzwerk und die Anwendung selbst.

Endgerät. Wie bereits zuvor erwähnt, ist dieser Kontextfaktor besonders für mobile Endgeräte wichtig, denn diese verfügen oft nur über sehr beschränkte Ressourcen (zB Bildschirmgröße, Tastatur, etc.) für die Darstellung von Webseiten. Speziell Mobiltelefone, sind in der Regel nicht für die Dartstellung bzw. Eingabe längerer Texte geeignet. Somit ist es notwendig die Web-Anwendung jeweils speziell für diese, aber auch andere mögliche Endgeräte anzupassen.

Browser. Um dem *anymedia*-Anspruch von ubiquitären Web-Anwendungen gerecht zu werden, ist es notwendig, auch Informationen über das verwendete Endgerät (siehe Kontextfaktor Endgerät) und den Browser für die Customization auszuwerten. So können gerade Browser auf mobilen Endgeräten, wie Mobiltelefone oder Handheld-Computer, nicht immer Frames darstellen oder verfügen nur über einen geringen Speicher für die Darstellung von Webseiten.

Netzwerk. Dieser Kontextfaktor enthält Informationen über die Bandbreite der Netzwerkverbindung mit der die Anwendung verwendet wird. Gerade bei mobilen Endgeräten aber auch bei Computern, die eine Modemverbindung verwenden, spielt dies eine wichtige Rolle. So kann bei der Customization für Endgeräte mit geringer Bandbreite auf speicherintensive Elemente wie Videos oder Bilder nach Möglichkeit verzichtet werden.

Anwendung. Geben die bisherigen Kontextfaktoren nur Aufschluss über die Umgebung der Anwendung, so ist es für die Customization auch wichtig, Informationen über die Anwendung selbst zu haben. Beispielsweise über die derzeitige Server- oder Datenbankauslastung oder ob alle Komponenten der Anwendung fehlerfrei arbeiten.

Sozialer Kontext Der Soziale Kontext umfasst alle Informationen über den Benutzer, beziehungsweise welche den Benutzer direkt betreffen. Diese können sowohl von ihm selbst zur Verfügung gestellt als auch, bis zu einem gewissen Grad, automatisch von der Anwendung abgeleitet worden sein. Sie werden unter dem Kontextfaktor Benutzer zusammengefasst.

Benutzer. Für die Personalisierung der Anwendung ist es notwendig den Benutzer der Anwendung zu identifizieren. Dies kann je nach dem verwendeten Endgerät über eine Vielzahl von Techniken geschehen. Bei Mobiltelefonen über die Telefonnummer, bei normalen Computern über die IP-Adresse oder ein Cookie. Diese Methoden sind allerdings nicht hundertprozentig zuverlässig. Somit wird der Benutzer in der Regel durch einen

Benutzernamen und ein Passwort, die er beide selbst eingeben muss, identifiziert. Unabhängig davon wie der Benutzer identifiziert wird, müssen der Anwendung natürlich seine Daten bekannt sein. Diese werden in einem Benutzerprofil gespeichert, welches in der Regel selbst vom Benutzer angelegt wird und neben seinen Zugangsdaten auch Informationen über seine persönlichen Präferenzen bezüglich der Sprache der Anwendung, seine Interessensgebiete oder demographische Daten über ihn beinhalten. Mit Hilfe dieser Daten ist je nach Zweck der Anwendung und Qualität der Angaben im Benutzerprofil eine Adaptierung möglich.

Logischer Kontext

Wie bereits erwähnt, wird der logische Kontext aus dem physischen abgeleitet, um abstraktere Informationen für die Customization der Web-Anwendung zu erhalten. Beispielsweise ist die IP-Adresse des Benutzers als solche nicht besonders aussagekräftig. Kann man die IP-Adresse jedoch einem Ort bzw. einer Addresse zuordnen so kann dies für die Adaptierung durchaus sehr nützlich sein.

Der logische Kontext kann auf verschiedene Arten abstrahiert werden. Ist es möglich mehrere IP-Adressen einem Ort zuzuordnen, so wäre dies ein Beispiel für eine Aggregation. Auch die Fusion verschiedener Kontextfaktoren ist möglich. 'Wien bei Nacht' ist ein Beispiel für die Fusion der Kontextfaktoren Ort und Zeit. Weiters wird beim logischen Kontext zwischen dem anwendungsspezifischen Teil und dem anwendungsunabhängigen Teil unterschieden, welcher generisch ist und meist von externen Dienstleistern zur Verfügung gestellt wird (zB die Zuordnung von IP-Adressen zu Orten).

So wie die physischen Kontextfaktoren hauptsächlich automatisch aus den Umgebungsbedingungen der Anwendung gewonnen werden, sollte auch der logische Kontext automatisch abgleitet werden. In den zuvor angeführten Beispielen war dies immer möglich. Aber nicht immer sind alle Kontextinformationen vollständig vorhanden. Zum Beispiel wenn eine IP-Adresse nicht eindeutig einem Ort zugeordnet werden kann oder wenn ein Benutzer die Anwendung zum ersten Mal besucht und noch kein Benutzerprofil mit seinen Präferenzen angelegt hat. Für diesen Fall ist es notwendig allgemeine Default-Werte für die fehlenden Kontextinformationen bereit zu halten oder den Benutzer dazu aufzufordern die fehlenden Informationen zu ergänzen. Dennoch darf man nicht außer Acht lassen, dass gerade ubiquitäre Web-Anwendungen hochgradig dynamisch sind [KPRS03] und daher auch die Akquisition der Kontextinformationen dynamisch erfolgen sollte. Denn auch der Nutzungskontext eines der Anwendung bereits bekannten Nutzers kann sich 'unvorhergesehen' ändern. Zusätzlich hat sich auch gezeigt, dass es

nützlich ist nicht nur aktuelle Kontextinformationen zu speichern sondern auch 'historische' um Veränderungen im Nutzungsverhalten besser nachvollziehen zu können, aber auch um Vorhersagen bezüglich einzelner Kontextfaktoren (zB der Bandbreite) treffen zu können.

2.2.3 Adaptierung

Wie schon für die Customization im Allgemeinen, gilt auch für die Adaptierung, dass diese sich über alle Dimensionen der Web-Anwendungsentwicklung erstreckt. Als Beispiel kann hier die Anpassung des User Interfaces auf Präsentationsebene oder die Anpassung der Linkstruktur auf Hypertextebene genannt werden. Wie bereits angesprochen, kann die Adaptierung statisch oder dynamisch ausfallen.

Bei einer *statischen Adaptierung* wird lediglich aus einem Set an vorgefertigten Versionen der Web-Anwendung, beziehungsweise des zu adaptierenden Teils, eine passende gewählt. So können für unterschiedliche Endgeräte jeweils passende Templates für das User Interface vorliegen, wobei bei der Adaptierung lediglich das für das Endgerät zutreffende Template gewählt wird.

Im Gegensatz dazu wird bei einer *dynamischen Adaptierung* die angepasste Web-Anwendung zur Laufzeit generiert. So werden hier Bilder entsprechend verkleinert oder ausgeblendet, Inhalte ausgewählt, Linkstrukturen angepasst, etc., und das Alles genau abgestimmt auf den aktuellen Nutzungskontext des Benutzers.

Eine weitere Stufe bei der Adaptierung ist die "lernende Adaption", auch *Adaption of Adaption* genannt [KaRS00]. Hierbei soll die Adaptierung die Customization selbst automatisch an sich ändernde Rahmenbedingungen anpassen. Das kann von einer Anpassung der Adaptierungsregeln an sich, also zum Beispiel deaktivieren, aktivieren oder ändern der Parametern von Regeln, bis hin zur Änderung der Gewichtung der, für die Customization wichtigen, Kontextfaktoren gehen.

2.2.4 Mapping - Customizationregeln

Wie bereits angesprochen, bietet das Customization ein Mapping zwischen Kontextfaktoren und entsprechenden Customizationregeln. Ein möglicher Ansatz dazu ist basierend auf [KaRe98, FSK+02], so genannte Event/Condition/Action Regeln zu verwenden (siehe Abbildung 2.5). Hierbei drückt der Event mit der Condition den Kontext aus und die Action führt die Adaptierung durch.

Als Beispiel kann der Event einfach der Aufruf einer Website durch einen Benutzer sein, die Condition ist zum Beispiel die aktuelle Tageszeit und die Action führt die Anpassung an sich durch, wobei in diesem Fall ein, auf die Tageszeit

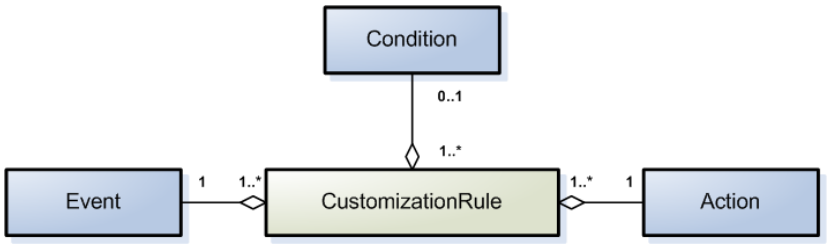

Abbildung 2.5: event/condition/action [FSK+02]

abgestimmter Content geliefert wird.

Customization wird in dieser Arbeit sehr umfassend gesehen, wie auch in [FSK+02]. Wie dort beschrieben muss Customization *Personalisation* und *Context-Aware Computing* gleichermaßen beinhalten.

Personalisation liefert jedem Benutzer einen individuell abgestimmten Inhalt und bietet ihm somit einen erheblichen Mehrwert. Darüber hinaus liefert aber die Customization dem Benutzer das Angebot auch auf diverse total verschiedene Endgeräte. Hier kann zwar die Darstellung aufgrund der technischen Gegebenheiten unterschiedlich ausfallen, aber der Punkt ist, dass der Inhalt und der durch die Customization gebotene Mehrwert bei jedem Endgerät gleich ist und nicht verloren geht.

Kapitel 3

Kontext - Basiselement der Customization

In diesem Kapitel wird detaillierter auf eines der Grundelemente der Customization, dem Kontext, eingegangen. Ohne Kontextinformationen ist eine ubiquitäre Web-Anwendung genauso adaptierbar wie eine normale statische HTML-Seite - nämlich gar nicht. Der physische und logische Kontext liefern jene Daten, welche durch ein entsprechendes Regelsystem ausgewertet werden und so zur Adaptierung der Web-Anwendung führen. Kontext muss also abgestimmt auf die jeweilige Web-Anwendung erfasst, gespeichert und ausgewertet werden. Die Akquirierung von Kontext kann hierbei über *physische* (zum Beispiel ein Temperatursensor) und *virtuelle Sensoren* (zum Beispiel Benutzertracking auf der Website) erfolgen, wobei hieraus wiederum logische Kontexdaten generiert werden können. Diese Daten müssen dann in der Web-Anwendung gespeichert werden, wobei hier ein passendes Kontextmodell für die Repräsentation der Daten im System für eine funktionierende ubiquitäre Web-Anwendung eine Grundvoraussetzung ist. Aufgrund dieser, im Kontextmodell erfassten, Daten ermittelt ein Regelsystem die nötigen Adaptierungen der Web-Anwendung und löst diese aus.

Im Folgenden wird zuerst ein Überblick über die grundlegenden Kontextmodellkategorien (Kapitel 3.1) und eine Einführung in Regelsysteme gegeben (Kapitel 3.2) um im Anschluss dazu einige repräsentative Ansätze für Context-Aware Frameworks vorzustellen (Kapitel 3.3).

3.1 Kontextmodelle im Überblick

Ein für die Anwendung passendes Kontextmodell ist Grundvoraussetzung für eine funktionierende ubiquitäre Web-Anwendung. Das Kontextmodell definiert und speichert sämtliche Kontextdaten in maschinenlesbarer Form. Dabei liegt

die Herausforderung bei der Entwicklung eines solchen Modells in dem praktisch unüberschaubaren Umfang von möglichen Kontextfaktoren, was dazu führt, dass das Modell möglichst flexibel konzipiert werden muss aber trotzdem einfach in der Bedienung bleiben soll. In Folge werden nun die wichtigsten Ansätze für Kontextmodelle nach [StLi04] aufgeführt.

3.1.1 Key-Value Modelle

Key-Value Modelle bilden den einfachsten Repräsentationsformalismus für die Kontextmodellierung. Hierbei werden über *Schlüssel-Wert Paare* Kontextdaten erfasst, welche über entsprechende Matching-Algorithmen ausgewertet werden. Es ist allerdings nur ein exaktes Matching möglich. Ein Beispiel für ein solches Schlüssel-Wert Paar ist in Listing 3.1 zu sehen. Diese Modelle sind einfach zu handhaben, eignen sich aber aufgrund der fehlenden weiter gehenden Strukturierungsmöglichkeit nicht für effiziente Kontextermittlungsalgorithmen. Als Beispiel kann hier das Kontextmodell des Context Frameworks „Context Toolkit“ (siehe Kapitel 3.3.1) genannt werden.

Listing 3.1: Einfaches Key-Value Beispiel

```
1 Name = John Doe
```

3.1.2 Markup Schema Modelle

Den *Markup Schema Modellen* liegen hierarchische Datenstrukturen zu Grunde. Diese sind aus Markup-Tags mit Attributen und Inhalten aufgebaut, wobei die Inhalte hierbei wieder Tags sein können, was zu einem rekursiven Aufbau der Datenstruktur führt.

Die bekanntesten Beispiel für Markup Schema Modelle bilden Profile, wie zum Beispiel das CC/PP[1] [W3C004] oder das UAProf[2] [OMA206]. Diese beiden Profile werden auch im entwickelten Tourismusinformationssystem verwendet. Ein Beispiel für ein UAProf Profil findet sich in Kapitel 6.3.1.

3.1.3 Grafische Modelle

Die Unified Modeling Language (UML[3] [UML206]) ist mit ihrer umfangreichen grafischen Komponenten und ihrer generischen Struktur gut für Kontextmodelle geeignet. Auch das ER-Modell [Chen76], welches die Struktur der Daten in

[1] Composite Capabilities/Preferences Profile
[2] User Agent Profile
[3] Unified Modeling Language

relationalen Datenbank darstellt, kann hierzu eingesetzt werden. Für die Modellierung von Kontextmodellen mit UML existieren bereits verschiedene Ansätze, zum Beispiel [ShBe05]. Ein anderer Ansatz für grafische Kontextmodellierung stammt von [HeIR03] welcher Kontexterweiterungen für die Object-Role Modeling (ORM) Sprache bietet. Hierbei wurde ORM insofern erweitert, dass das ORM Grundkonzept, die Fakten, kategorisiert werden können. Diese können *statisch* oder *dynamisch* sein, wobei diese wiederum nach der Quelle der Herkunft unterteilt werden können. Darüber hinaus existieren eigene Fakt-Typen für die Erfassung von historischem Kontext und für die Abbildung von speziellen Beziehungen, wo die Änderung eines Faktors zur automatischen Änderung eines anderen führt.

In Abbildung 3.1 wird ein Beispiel für die Notation von [HeIR03] gegeben. Hierbei sind im mittleren Teil die Fakt-Typen zu sehen, wobei die *Striche* - unter den Kästchen angeben, welche Art von Beziehung besteht. So ist eine 1-1 Beziehung gleichwertig mit einem durchgezogen Strich unter alle Kästchen, eine 1-n Beziehung ist ein Strich auf der Einerseite und m-n Beziehung ist gleichbedeutend mit keinem Strich. Das *o* gibt an, das die Information vom Benutzer selbst eingegeben worden ist, *M* gibt an, dass die Information von einem Sensor stammt und *[]* sind temporäre Informationen.

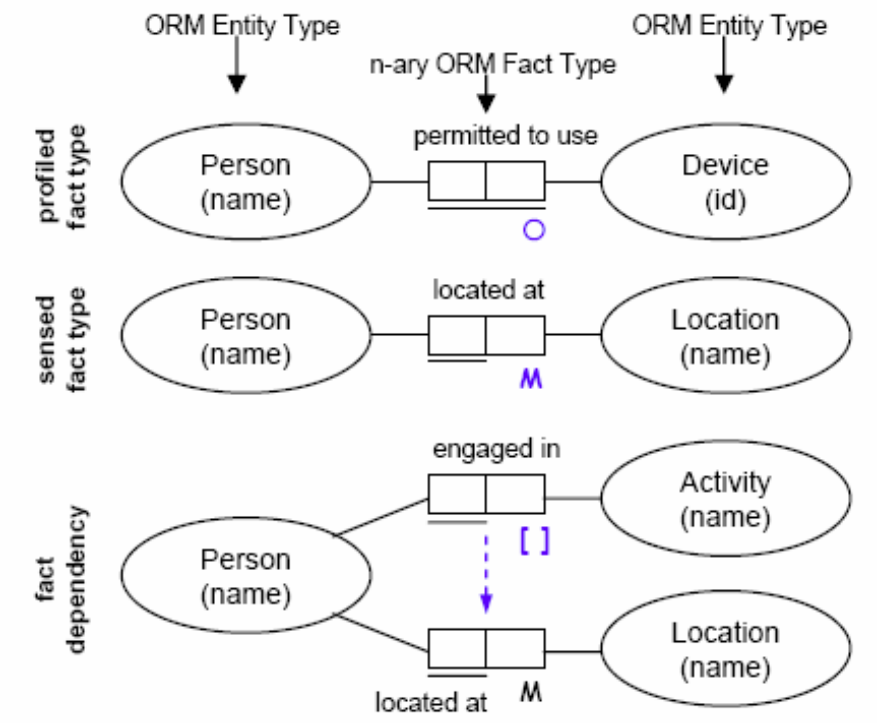

Abbildung 3.1: Object-Role Modeling erweitert mit Kontext [HeIR03]

3.1.4 Objektorientierte Modelle

Objektorientierte Modelle können, wie aus dem Namen schon hervor geht, alle Vorteile der Objektorientierung (wie zum Beispiel Datenkapselung, Vererbung, Wiederverwendung) voll nützen. Bei diesen Modellen werden verschiedenartige Objekte (wie zum Beispiel ein Wetterobjekt oder ein Ortobjekt) genutzt um die unterschiedlichen Kontexttypen abzubilden und die Details der Kontextverarbeitung und Repräsentierung zu kapseln, wobei ein Zugriff auf den Kontext und auf die Kontextlogik hierbei nur über eindeutig definierte Interfaces möglich ist.

Beispiele für Context-Aware Frameworks mit objektorientierten Kontextmodellen sind [HSP+03] (siehe Abbildung 3.2) und [ChMD99]. In [HSP+03] wird ein Framework basierend auf einer Drei-Schichtenarchitektur beschrieben. Das markanteste Merkmal dieses Ansatzes ist es, dass kein zentraler Server benötigt wird, sondern sämtliche Funktionalität am jeweiligen Endgerät selbst abläuft, wobei hier die Kontextdaten zum Beispiel per Wireless Lan oder Bluetooth ad-hoc ausgetauscht werden. Hieraus erkennt man auch den für dieses Framework gedachten Anwendungszweck, nämlich das Mobile Computing.

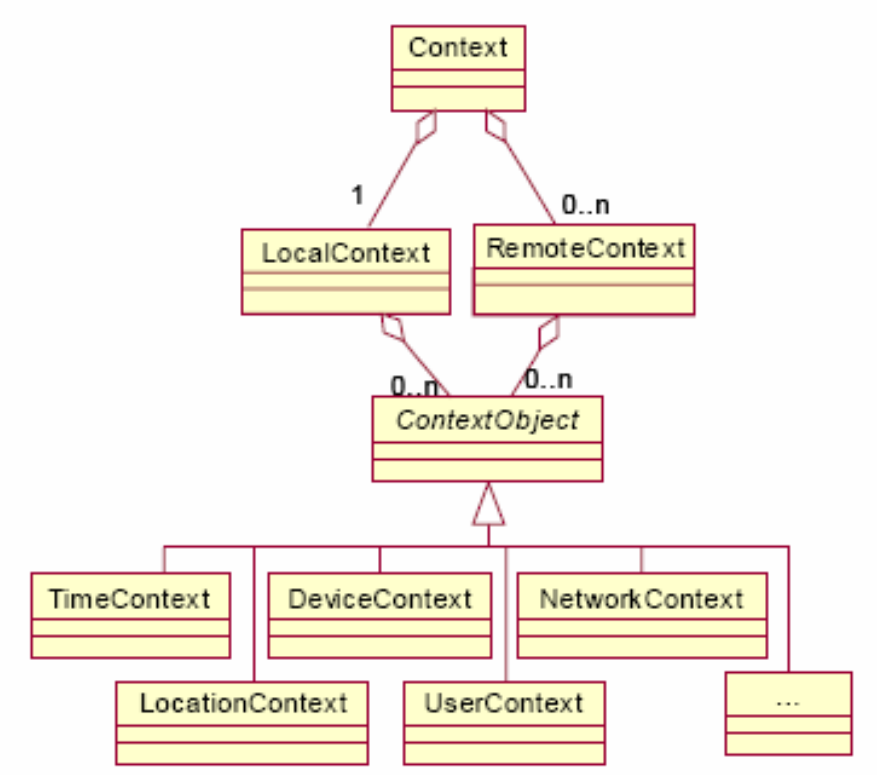

Abbildung 3.2: Beispiel für ein objektorientiertes Kontextmodell aus Hydrogen [HSP+03]

3.1.5 Logikbasierte Modelle

Eine Logik definiert die Bedingungen unter welchen eine Schlussfolgerung auf Grund einer Menge von anderen Ausdrücken oder Fakten erzielt werden kann. Diese Bedingungen werden mit Regeln in einem formalen System beschrieben. In einem logikbasierten Modell wird der Kontext immer durch Fakten, Ausdrücke und Regeln definiert welche mit einem logikbasierten System verwaltet werden über welches neue Fakten hinzugefügt, Fakten aktualisiert und gelöscht werden können. Der Folgerungsprozess kann für die Gewinnung von, auf existierende Regeln basierenden, neuen Fakten genutzt werden.

Einer der ersten Ansätze für logikbasierte Kontextmodelle war [McBu97]. Die grundlegende Aussage in diesem Ansatz ist *ist(c, p)* was besagt, dass die Behauptung p im Kontext c wahr ist. Bei diesem Modell wird auch immer davon ausgegangen, dass jede Aussage in einem „größeren“ Kontext eingebunden ist, wodurch man zu Aussagen wie *c0: ist(context-of(„Sherlock Holmes stories“), „Holmes is a detevtive“)* kommt. Hierbei ist c0 der so genannter *Outer Context*.

3.1.6 Ontologiebasierte Modelle

Ontologien sind ein Werkzeug für die Beschreibung von Konzepten und deren Beziehungen [UsGr96]. Die Inhalte von Ontologien können wiederum objektorientiert oder logikbasiert sein. Daraus ergibt sich eine gute Anwendbarkeit für die Modellierung von Kontextinformationen aufgrund der formalen Ausdrucksstärke und der Möglichkeit der Anwendung der Folgerungstechniken von Ontologien.

In Listing 3.2 ist ein Ontologiebeispiel aus der Context Broker Architecture (CoBrA) [ChFJ04] angeführt. In CoBra wird ein OWL-basierter Ansatz (Web Ontology Language [OWL004]) verwendet (CORBA-Ont [ChFJ03]), wobei die Struktur und das Vokabular in RDF[4] beschrieben sind.

Ein weiteres ontologiebasiertes Framework ist das Context Framework 3.3.3 von [Korp05].

Listing 3.2: COBRA-Ontologie Beispiel aus [BaDR06]

```
1  <loc:LocationContext>
2      <rdf:type rdf:resource="&tme;␣InstantThing"/>
3      <loc:locationContextOf>
4        <per:Person>
5          <per:name rdf:datatype="&xsd;␣string␣">
6            Harry Chen
7          </per:name >
8        </per:Person>
9      </loc:locationContextOf>
10     <loc:boundedWithin rdf:resource="&␣ebgeo␣;␣Japan␣"/>
```

[4] http://www.w3.org/RDF/

```
11      <tme:at rdf:datatype="&xsd;␣dateTime␣">
12        2004 -02 -23T11:23:00
13      </tme:at >
14  </loc:LocationContext>
```

3.1.7 Vergleich

Die hier vorgestellten Kontextmodelle geben einen Überblick über die verschiedenen Ansätze bei der Kontextmodellierung. Die direkte Vergleichbarkeit der Modellkategorien untereinander ist jedoch schwierig, da unterschiedliche Merkmale zur Einteilung getroffen worden sind. Somit können einzelne Modelle durchaus zwei Kategorien zugeordnet werden, was zum Beispiel bei den grafischen und objektorientierten Modellen deutlich wird.

Für die Eignung der Kontextmodelle für übiquitäre Web-Anwendungen werden in [StLi04] sechs Kriterien als essentiell genannt, anhand welcher man die Modellkategorien bewerten kann:

Verteilte Komposition Die Administration und Komposition eines Kontextmodells und dessen Daten in ubiquitäre Systemen erfolgt verteilt, dass heißt einerseits nicht nur zu verschiedenen Zeitpunkte, je nachdem wann die Daten verfügbar sind, sondern andererseits auch in verschiedenen, teilweise extern gelegenen Programmkomponenten. Genauso sind Kontextfaktoren oft nicht aus einer Quelle beziehbar, sondern müssen auf Grund von mehreren Datenquellen ermittelt werden.

Partielle Validierung In einem ubiquitären System soll es möglich sein Kontextdaten sowohl auf Struktur als auch auf Instanzebene gegen ein bestehendes Kontextmodell zu validieren. Dies ist aufgrund der unterschiedlichen, verteilt liegenden und über die Zeit wechselnden Kontextquellen nötig um sicher zustellen, dass die Kontextfaktoren für das Kontextmodell korrekt verarbeitbar sind.

Datenqualität Die Qualität einer, zum Beispiel von einem Sensor gelieferten, Information kann über die Zeit gesehen schwanken genauso wie unterschiedliche Sensoren für die Bestimmung ein und desselben Kontextfaktores unterschiedliche Daten liefern können. Daher sollen Kontextmodelle Techniken zur Qualitätsbestimmung und Qualitätssicherung unterstützen. Dies ist ein Punkt der bei allen ubiquitären Anwendungen viel stärker berücksichtigt werden muss, als es derzeit der Fall ist, da immer mehr Entscheidung in ubiquitären Anwendung nur auf Basis von automatisch erfassten und ausgewerteten Kontextinformationen getroffen werden.

Unvollständigkeit und Mehrdeutigkeit Da die Daten für die Bestimmung eines Kontextfaktors oft unvollständig oder nicht eindeutig sind, sollen im Kontextmodell Mechanismen vorgesehen sein, mit denen die Daten vervollständigt, eindeutig bestimmt oder wenn dies nicht möglich, entsprechend "sensibel" behandelt werden.

Formalisierungsgrad Kontextfaktoren und die Beziehungen dieser untereinander in einer exakten und nachvollziehbaren Art und Weise zu beschreiben ist eine große Herausforderung, auch unter dem Gesichtspunkt der Validierung von erfassten Daten. Man stelle sich hier vor, dass man von der Aufgabe „Finde ein Lokal in meiner Nähe" jeden Term genau definieren muss, also zum Beispiel was „Nähe" für „mich" bedeutet. Dies ist nötig, damit bei einer Interaktion von verschiedenen Beteiligten oder auch bei einer Interaktion mit externen Quellen das gleiche Verständnis besteht, also die gleiche Semantik verwendet wird, was für ein gutes Funktionieren von ubiquitären Anwendungen von entscheidender Bedeutung ist.

(Wieder)Verwendbarkeit in anderen Systemen Aus der Entwicklungssicht von ubiquitären Anwendungen ist es anzustreben, dass Kontextmodelle leicht in neue Anwendungen integriert und auch in bestehenden Systemen, zum Beispiel als Service Frameworks oder Web Services, verwendet werden können.

In Tabelle 3.1 werden die in diesem Kapitel genannten Kontextmodelle anhand der soeben aufgezählten Kriterien zusammengefasst dargestellt. Hierbei zeigt sich deutlich, dass objektorientiert und ontologiebasierte Modelle gegenüber den anderen Lösungen die hier angeführten Kriterien am besten erfüllen und sich somit gut für den Einsatz in ubiquitären Web-Anwendungen eignen. Dies bedeutet natürlich nicht, dass die anderen Ansätze ungeeignet sind, da je nach Anwendungsfall durchaus diese Ansätze den anderen vorzuziehen sein könnten. Ontologiebasierte Modelle zeichnen sich hierbei durch den hohen Formalisierungsgrad und die gute Unterstützung der partiellen Validierung gegenüber objektorientierten Modellen aus. Wie jedoch aus dieser Tabelle deutlich hervorgeht, liegen die objektorientierten Modelle sonst gleich auf mit den ontologiebasierten Modellen.

Daher, und unter der Berücksichtigung der Anforderung bei der Entwicklung der Web-Anwendung mit aspektorientierter Programmierung zu Arbeiten, wurde für die Implementierung ein objektorientiertes Kontextmodell gewählt, wobei die Daten an sich in einer relationalen Datenbank gespeichert werden.

Kontext Modelle	*Anforderungen an ubiquitäre Systeme*					
	Verteilte Komposition	Partielle Validierung	Daten-qualität	Unvollständig-keit und Mehrdeutigkeit	Formalisier-ungsgrad	Verwendbarkeit in existierenden Anwendungen
Key-Value Modelle	-	-	--	--	--	+
Markup Schema Modelle	+	++	-	-	+	++
Grafische Modelle	--	-	+	-	+	+
Objektorientierte Modelle	++	+	+	+	+	+
Logikbasierte Modelle	++	-	-	-	++	--
Ontologiebasierte Modelle	++	++	+	+	++	+

Tabelle 3.1: Gegenüberstellung der dargestellten Kontextmodelle aus [StLi04]

3.2 Kontextverarbeitung mit Regelsystemen

Die Erfassung von Kontext bildet den Grundstock für die Customization einer Anwendung. Für die Adaptierung der Anwendung ist es nötig diesen erfassten Kontext auszuwerten. Hierfür müssen entsprechende Regelsysteme entwickelt werden. Was man sich unter einem solchen System vorstellen kann, wird in Folge erklärt.

3.2.1 Was ist ein Regelsystem?

Regelsysteme stellen das Mapping zwischen dem aktuellen Nutzungskontext und der Adaptierung der Web-Anwendung her, sie analysieren also die Kontextdaten und lösen entsprechende Adaptierungen aus. Die grundlegende Funktionsweise von Regelsystemen lässt sich wohl am besten mit einem Vergleich mit dem altbekannten *If-Statement* in Programmiersprachen erklären. Ohne viel über die Folgen für die Wartbarkeit nachzudenken wird Geschäftslogik oft mit einer *if-then-else* Anweisung in den Sourcecode im wahrsten Sinne des Wortes eingebrannt. Oft geschieht dies bereits in einer Phase des Entwicklungsprozesses, in welchen noch nicht einmal das finale Pflichtenheft vorhanden ist. Auch wenn dies der Fall ist, so tritt nicht selten bei Vollendung des Projektes der Fall ein, dass sich bereits wieder Faktoren im schnelllebigen Geschäftsumfeld geändert haben und die Geschäftslogik angepasst werden muss. Nun muss man die meist weit verstreute komplexe und in verschachtelte Bedingungsbäume aufgeteilte Logik entsprechend anpassen, um dann nach einigen Monaten vielleicht schon wieder vor einem nötigen Refactoring zu stehen. Daraus ergibt sich die Forderung nach einer Kapselung

der Stellen im Sourcecode, welche sich ständig nach neuen Regeln richten sollen, womit wir beim Ansatz der *regelbasierten Software* angelangt sind.

> [...] For any IT application, the business rules change more frequently than the rest of the application code. Rules Engines or Inference Engines are the pluggable software components that separate the business rules from the application code. This allows the business users to modify the rules frequently without the need of IT intervention and hence allowing the applications to be more adaptable with the dynamic rules. [...] [Wiki06]

Ein *Regelsystem (Rule Engine)* ist nach [Wund06b] eine Softwarekomponente welche erlaubt

- Regeln in einer Regelsprache zu formulieren
- Regeln im Kontext von konkreten Daten und zur Verfügung gestellten Methoden zu prüfen und anzuwenden

Hieraus geht hervor, dass für die Benutzung eines Regelsystems die systemspezifische Logiksprache verstanden und angewandt werden muss und dass für die Ausführung der Regeln Teile des anwendungspezifischen Objektmodells für das Regelsystem zugreifbar sind.

Darüber hinaus können Regelsysteme je nach dem Einsatzgebiet um diverse Funtkionen erweitert werden, und anderem [Wund06b]:

- Organisation von Regeln, also die Anordnung von Regeln nach bestimmen Kriterien (z.B. nach Algorithmen und / oder fachlichen Gesichtspunkten)
- Berechtigungsverwaltung für das Erfassen, Ändern und Entfernen von Regel und Erfassung von Metadaten
- Austausch von Regeln zur Laufzeit des Systemes
- Erfassung von Regeln in der Business-Sprache (z.B. in natürlichsprachlicher Formulierung und Sätzen)
- Visualisierung von Regeln zum Beispiel in Form von UML-Diagrammen

Dem Regelsystem zu Grunde liegt ein anwendungsspezifisches Set an Regeln auf Basis welcher das Regelsystem unter Einbezug der, zum Zeitpunkt der Schlussfolgerung, bestehenden Fakten Schlüsse zieht und entsprechende Konsequenzen auslöst.

Eine Regel besteht im einfachsten Fall aus einer oder mehrerer Bedingungen und einer oder mehrerer Konsequenzen. Als Beispiel in der Literatur findet sich

Wenn es regnet und ich frei habe, gehe ich ins Kino [Wund06a].

Dieser Satz besteht aus zwei, über die Boolsche-Logik verbundene Teil-Bedingungen und einer daraus resultierenden Konsequenz. Im Optimalfall sollte das Regelsystem den Satz, so wie er hier geschrieben wurde, „verstehen“ und anwenden. Dem ist jedoch (noch) nicht so, ein passender Sourcecode, mit dem diese Regel im Regelsystem abgebildet werden kann, könnte also so aussehen 3.3:

Listing 3.3: Sourcecode einer einfachen Regel in Java

```
1 if (weather.isRaining() && me.hasADayOff()) {
2     me.goTo(Loaction.CINEMA);
3 }
```

Wie man hier sieht, ist noch ein großer Unterschied zwischen dem ursprünglichem Satz und der Interpretation im Regelsystem, wobei sich dieses Faktum derzeit noch über sämtliche Regelsystemansätze zieht. Dieses Mapping in die jeweilige Programmiersprache stellt die größte Schwierigkeit bei Regelsystemen dar. So müssen für das hier angegebene Beispiel sämtliche Satzteile wie „Kino“, „regnet“,... entsprechenden Sprachelemente (wie Variabeln und Klassen) vorhanden sein um diese in einer Regel verwenden zu können. Darüber hinaus muss auch die Mehrsprachigkeit berücksichtigt und ein entsprechendes Mapping erstellt werden (also im Beispiel z.B. von Kino auf Cinema).

Hat man diese Punkte alle berücksichtigt stößt man zwangsläufig auf ein zweites großes Problem: die semantische Lücken zwischen natürlichen Sprachen und Programmiersprachen. Regelsysteme sind nicht in der Lage abstrakte Begriffe kontextabhängig zuzuordnen, wie es der Mensch kann. Ändert man den vorher gebrachte Satz folgendermaßen ab, kann damit das Regelsystem nichts anfangen, es sei denn die relevanten neuen Begriffe sind definiert.

Wenn es am Abend zu nieseln anfängt und ich Zeit habe, sehe ich mir einen spannenden Film im Cineplexx an

Da jedoch in natürlichen Sprachen eine Unzahl an Variationen für ein und denselben Satz vorhanden sind ist es unmöglich alle Variationen im Regelsystem zu berücksichtigen.

Daher muss der Ersteller einer Regel genau wissen wie er die Regel im der Regelsprache definieren muss, damit diese einerseits vom System interpretiert werden kann und andererseits nach der Transformation zu eine syntaktisch korrekten und anwendbaren Programmsprachformulierung wird.

Dies stellt eine der größten Herausforderungen und auch eines der größten Risiken bei Regelsystemen dar. [Wund06a] [Wund06b]

3.2.2 Wann sind Regelsysteme sinnvoll?

Generell sind die möglichen Einsatzgebiete für Rule-Engines so umfassend wie für If-Statements. Natürlich macht es keinen Sinn, jedes If-Statement in eine Regel auszulagern, da einerseits Performanceverluste und natürlich vor Allem die Kosten-Nutzen Relation dagegen spricht. Sinn macht dies nur wenn abzusehen ist, dass diese Bedingung fortlaufenden Änderungen unterliegend und / oder sehr viele, von den Bedingungsteilen her sehr unterschiedliche, Regeln existieren welche von Experten häufig geändert werden sollen. Für solche Fälle eignen sich Regelsysteme mit dem RETE[5]-basierte Baumansatz von Charles Foley, mit welchem mittels Baummodelle optimierte Regelabfragen erzeugt werden.

Typische Anwendungsfälle sind hierbei:

- Modellierung und Umsetzung von Geschäftsprozessen (mittels Lösungen auf Basis von Service Oriented Architecture oder Business Process Management)
- Schlussfolgerungssysteme zur Analyse von Daten in der Medizin, Meteorologie oder am Aktienmarkt
- Marktabhängige Kalkulations, Angebots- und Warenwirtschaftsprogramme
- Steuerung beziehungsweise partielle Steuerung von autonomen Agenten und Teilbereiche einer künstlichen Intelligenz

3.2.3 Die JBoss Rules Engine

Nach der eben gegebenen kurzen Einführung in die Thematik der Regelsysteme soll nun, anhand einer konkreten Rule Engine und mit einem einfachen Beispiel die Anwendung eines solchen demonstriert werden. Als Rule Engine wird hierbei *JBoss Rules* [JBos06] verwendet, wobei in Folge ein Beispiel aus [Wund06a] dargestellt wird.

JBoss Rules ist eine Weiterentwicklung von Drools und stand zum Zeitpunkt der Erstellung dieser Arbeit in Version 3.0.4 zur Verfügung. Dieses Regelsystem ist in Java implementiert und steht unter der Apache Software License[6] als freie Software zur Verfügung. Die JBoss Rules Engine eignet sich sehr gut als Einstieg in Regelmaschinen, da dieses Produkt relativ einfach zu verstehen und auch gut dokumentiert ist. Die Drools Libraries und die JBoss Rules IDE-Erweiterung für Eclipse mit Beispielen sind frei verfügbar[7] und eine ausführliche Dokumentation

[5] http://en.wikipedia.org/wiki/Rete_algorithm
[6] http://www.apache.org/licenses/
[7] http://labs.jboss.com/portal/jbossrules/downloads

kann man unter [JBos06] abrufen. Wichtig hierbei ist auch, dass keine Abhängigkeiten zu anderen JBoss Libraries oder zum Application Server an sich bestehen und somit dieses Regelsystem leicht ausprobiert beziehungsweise in Projekte integriert werden kann.

Wie funktionieren nun die JBoss Rules? Um dies zu erklären, wird in Folge ein kleines Beispiel gebracht, anhand welcher die Grundelemente der Jboss Rules dargestellt werden. Ausgangspunkt ist eine einfache JavaBean Klasse „Produkt.java“ in Listing 3.4.

Listing 3.4: Product.java [Wund06a]

```
1  package ruleExample;
2
3  public class Product
4  {
5    private float price = 0.0f;
6    private String name = null;
7
8    public String getName()
9    {
10     return name;
11   }
12
13   public void setName(String name)
14   {
15     this.name=name;
16   }
17
18   public String getPrice()
19   {
20     return price;
21   }
22
23   public void setPrice(float price)
24   {
25     this.price=price;
26   }
27 }
```

Hierfür soll nun eine Regel definiert werden, durch die der Preis des Produktes „Katzenstreu“ um 10 Prozent gesenkt wird, wenn das Produkt teurer als 10 Euro ist. Diese Regeldatei ist in Listing 3.5 dargestellt. Nach den allfälligen Import-Statements wird die Regel mit dem Schlüsselwort *rule* gestartet. Danach folgt der Regelname und die Definition der Regel an sich. Hier im Beispiel umfasst die Regel eine zusammengesetzte Bedingung, eingeleitet durch *when*, und zwei Konsequenzteile, gekennzeichnet durch *then*. Beendet muss jede Regel immer mit dem Schlüsselwort *end* werden.

Listing 3.5: ProductRules.drl [Wund06a]

```
1  package ruleExample;
2  import ruleExample.Product;
3
4  rule "Produktregel"
5
6    when
7      product:Product(name=="Katzenstreu", price>10)
8    then
9      product.setPrice((float)(0.9*product.getPrice()));
10     System.out.println("Neuer␣Produktpreis:␣" + product.getPrice());
11
12 end
```

Wie die Regel beispielsweise angewendet werden kann ist in Listing 3.6 zu sehen.

Listing 3.6: TestRules.java [Wund06a]

```
1  package ruleExample;
2
3  import java.io.InputStreamReader;
4
5  import org.drools.RuleBase;
6  import org.drools.RuleBaseFactory;
7  import org.drools.WorkingMemory;
8  import org.drools.compiler.PackageBuilder;
9
10 public class TestRules {
11
12   public static void main(String[] args)
13   {
14     Product product=new Product();
15     product.setName("Katzenstreu");
16     product.setPrice(12.90f);
17
18     try {
19       PackageBuilder builder=new PackageBuilder();
20       builder.addPackageFromDrl(new InputStreamReader(
21         TestRules.class.getRessourceAsStream("/ProductRules.drl")));
22       RuleBase base = RuleBaseFactory.newRuleBase();
23       base.addPackage(builder.getPackage());
24       WorkingMemory wm = base.newWorkingMemory();
25       wm.assertObject(product);
26       wm.fireAllRules();
27     }
28     catch (Exception e) {
29       e.printStackTrace();
30     }
31   }
32 }
```

Damit sind auch schon alle grundlegende Anforderungen für die Regelerfassung und Umsetzung erklärt - natürlich kann man die Regeln noch um einiges ausbauen, aber das würde an dieser Stelle zu weit führen (hierfür sei auf [Wund06b] verwiesen).

Ein wichtiger Punkt ist jedoch noch die Erweiterung des Systems um eine *Domain Specific Language* (DSL[8]). Mit einer DSL hat man die Möglichkeit, die Regeln an sich mehr natürlichsprachig zu verfassen. Dazu muss eine DSL-Datei mit der Benennung des DSL-Moduls und verschiedenen Mappings erstellt werden. Für das hier gebrachte Beispiel ist eine solche DSL-Datei in Listing 3.7 zu sehen. Nutzt man diese DSL-Datei in der Regeldatei, so kann man in dieser jetzt wirklich Regeln in natürlicher Sprache definieren (siehe Listing 3.8).

Listing 3.7: ProductRules.dsl [Wund06a]

```
1
2 [when]das Produkt mit dem Namen {arg1} einen Preis von mehr als {arg2} Euro
      aufweist=product:Product(name=="{arg1}", price>{arg2})
3
4 [then]multipliziere den Originalpreis mit {arg1}=
5         product.setPrice((float)({arg1}*product.getPrice()));
6       System.out.println("Neuer␣Produktpreis:␣"+product.getPrice());
```

Listing 3.8: ProductRules.drl [Wund06a]

```
1  package ruleExample;
2
3  expander ProductRules.dsl
4
5  import ruleExample.Product;
6
7  rule "Produktregel"
8    when
9      das Produkt mit dem Namen Katzenstreu einen Preise von mehr als 10 Euro
           aufweist
10   then
11     multipliziere den Originalpreis mit 0.9
12 end
```

Zu beachten ist hierbei jedoch, dass man noch immer bei der Erstellung der Regeln sehr eingeschränkt ist, da man eigentlich nur wenige bestimmte Werte im der Regeln variieren kann, der Rest jedoch genau in der DSL-Datei vorgegeben ist und auch eingehalten werden muss, wenn die Regel funktionieren soll.

Auch ist die Interaktionsmöglichkeit und die Einbindung der Regeln in das restliche Programm hierbei nur bedingt flexibel, was wieder für den Einsatz von aspektorientierter Programmierung spricht.

3.3 Context-Aware Frameworks im Überblick

Bei der Entwicklung von ubiquitären Web-Anwendungen steht man immer vor der Entscheidung ob man das System komplett neu, und speziell auf die Pro-

[8] http://en.wikipedia.org/wiki/Domain-specific_programming_language

blemstellung abgestimmt entwicktelt, oder ob man ein bestehende Framework heranzieht und bei der Entwicklung das Konzept des Frameworks übernimmt.

Um einen Überblick über die bestehenden Systeme zu bekommen, werden in Folge vier repräsentative Umsetzungen von „Context-Aware Frameworks“ hinsichtlich ihrer verschiedenen Ansätze vorgestellt und Vor- und Nachteile aufgezeigt. Dies soll eine Einblick in das Thema geben und vor Allem ersichtlich machen, dass man nicht immer zwangsweise das Rad bei der Entwicklung einer ubiquitären Web-Anwendung neu erfinden muss, sondern das es durchaus Frameworks gibt, die einem die Arbeit hierbei um einiges vereinfachen können. Die Auswahl der vorgestellten Frameworks beruht auf [BaDR06] [SiCo06] [KPRS03].

3.3.1 Context Toolkit von Dey et al.

Das Context Toolkit Framework [SaAb99], [DeSA01] wurde mit der Zielsetzung entwickelt, ein umfangreiches und anwendungsunabhängiges, konzeptuelles Framework für die Repräsentierung und Verarbeitung von Kontext zu bieten.

Das Grundprinzip des Context Toolkit sind sogenannte *Context Widgets* welche über eine verteilte Infrastruktur verfügbar sind. In Abbildung 3.3 [DeSA01] ist eine Beispiel für eine mögliche Zusammensetzung der Context Toolkit Komponenten dargestellt. Context Widgets sind Softwarekomponenten, welche Anwendungen Kontextinformationen liefern, wobei der Kontextgewinnungsvorgang hierbei von den Widgets gekapselt und die Details vor der Anwendung versteckt werden. Ein Context Widget kann zum Beispiel die Ermittlung der aktuellen Raumtemperatur über entsprechende Sensoren, oder die Bestimmung des aktuellen Standortes auf Grund der Cell-ID oder der IP des Benutzers übernehmen. Diese Informationen können von Anwendungen einfach eingebunden werden, ohne dass man sich bei der Anwendungsentwicklung über die Umsetzung der Datenermittlung Gedanken machen muss. Context Widget funktionieren demnach nach dem gleichen Prinzip wie GUI[9] Widgets, durch welche sämtliche oder zumindest Teile der Benutzeroberfläche von der Anwendung gelöst und somit gesondert entwickelbar und wartbar sind.

Mit den *Aggregatoren* und *Interpretoren* werden beim Context Toolkit zwei unterschiedliche Abstraktionsmechanismen zur Verfügung gestellt, um generischen und anwendungsspezifischen Kontext zu erfassen. Die Aggregatoren sammeln und speichern logisch zusammengehörigen Kontext von verschiedenen Widgets wogegen die Interpretoren noch einen Schritt weiter gehen und aus bestehendem Kontext und Aggregatordaten übergeordnete Kontext abstrahieren. Durch diesen

[9] Graphical User Interface

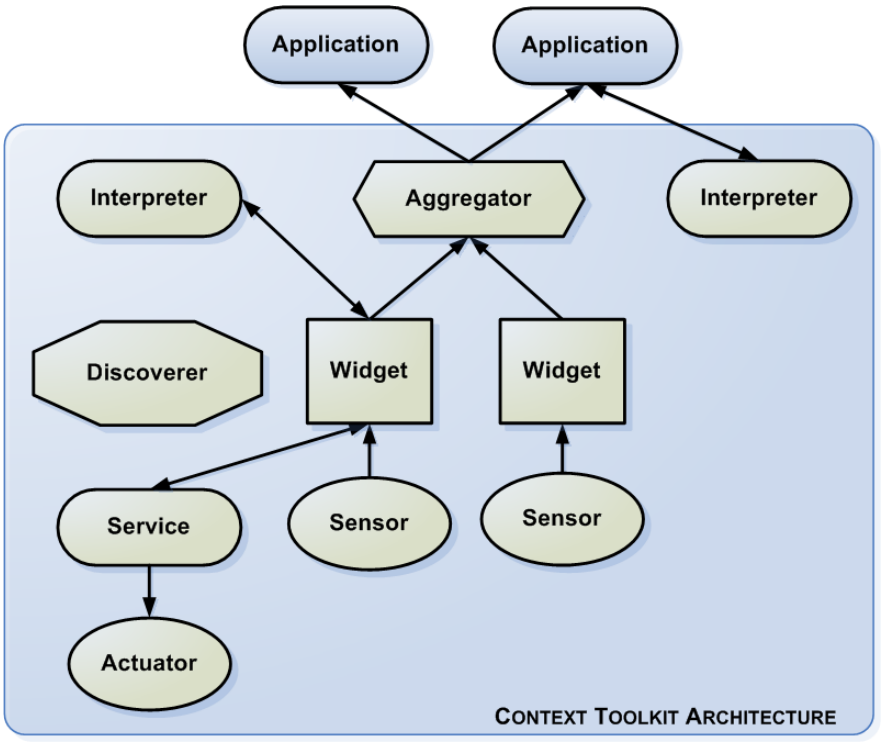

Abbildung 3.3: Beispiel einer Konfiguration der Komponenten des Context Toolkits

Aufbau bietet das Kontext Toolkit einen explizite Trennung zwischen physischen Kontext (durch die Context Widgets) und logischen Kontext (durch Aggregatoren und Interperetoren).

Für die Adaptierung beinhaltet das Context Toolkit *Context Services*, welche für die Ausführung der Adaptierungsvorgänge anstelle der Anwendung zuständig sind. Hierbei werden drei Kategorien von Context Services unterschieden: „Recommendations“ (zum Beispiel die Anzeige von möglichen Aktionen aufgrund des aktuellen Kontexts des Benutzers), „Trigger“ (zum Beispiel das automatische Auslösen von Ereignissen) und „Taggers“ (zum Beispiel die Ergänzung von Objektendaten mit aktuellen Kontextdaten für eine später Verwendung). Hierbei bilden so genannte *Aktuatoren* den Gegenpart zu den Sensoren bei den Context Widgets und setzen so kontextbezogene Adaptierungen, wie zum Beispiel die Regelung der Temperatur in einem Raum, um.

Darüber ist es noch wichtig zu erwähnen, dass alle Komponenten in einer Registry, dem *Discoverer*, verzeichnet sind, welche ein Look-Up Service bietet über welches auf das Context Toolkit zugegriffen werden kann [KPRS03] [SiCo06].

3.3.2 Context Broker Architecture

Die Context Broker Architecture (CoBrA) [ChFJ04] ist eine agentenbasierte Architektur (siehe Abbildung 3.4) für die Unterstützung von Context-Aware Systemen in sogenannten *intelligenten Räumen* wie zum Beispiel Wohnungen, Büros und Vorlesungsräume welche Dienste des *Pervasive Computing*[10], also zum Beispiel ansprechbare Sensoren für Temperatur und Licht bieten.

Den Kern dieser Architektur bildet der *Context Broker* welcher ein gemeinsames Kontextmodell für alle zum System gehörenden Agenten verwaltet und die Kontextdaten aufgrund von benutzerspezifischen Regeln im Sinne des Datenschutzes schützt. Hinter diesen Agenten können Anwendungen von mobilen Endgeräten (zum Beispiel die Regelung der Art des Handyklingeltones (Stumm, Vibrationsalarm, Normal,...) anhängig vom Aufenthaltsort), Dienste welche stationäre Geräte in einem Raum bieten (zum Beispiel eine Projektorkontrolle oder eine Temperaturregelung) oder Web-Services stehen. Für die Auswertung der erfassten Kontextdaten und die Ermittlung und Wartung von logischem Kontext setzt CoBrA auf eine regelbasiertes, logisches Schlusssytem [SiCo06] [BaDR06].

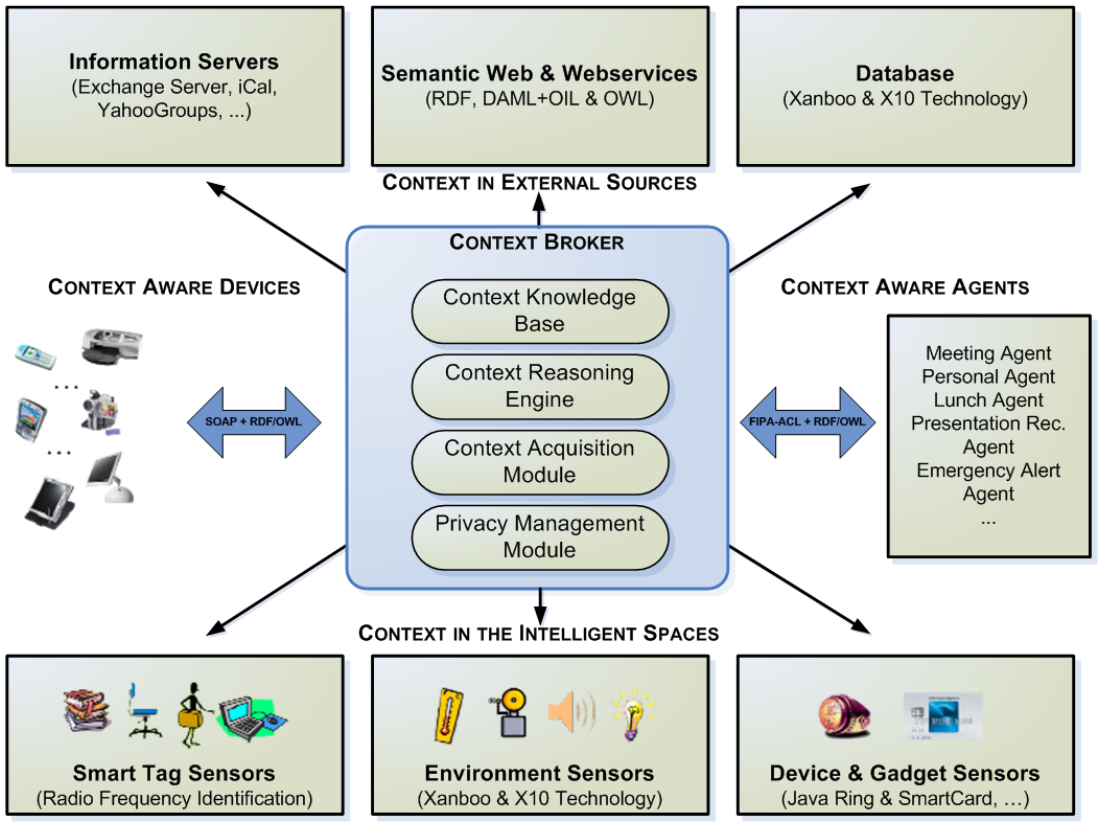

Abbildung 3.4: Schema der Context Broker Architecture [ChFJ04]

[10] http://en.wikipedia.org/wiki/Pervasive_Computing

Die Aufgaben des Context Brokers sind [ChFJ04]:

1. Die zur Verfügungstellung eines zentralen Kontextmodells, welches von allen Geräten, Diensten und Agenten im Raum benutzt werden kann
2. Eine globale Kontextinformationsakquisition von Daten, welche aufgrund von hohen Ressourcen-Anforderungen von einzelnen Endgeräten nicht ermittelt werden können
3. Ermittlung des logischen Kontext
4. Ermittlung und Bereinigung von inkonsistentem Kontext im zentralen Modell
5. Schutz der Privatsphäre der Benutzer aufgrund von individuell definierten Regeln für die Benutzung der Kontextinformationen

3.3.3 Context Framework nach Korpipää et al.

Der Architektur des Context Framework [Korp05] liegt ein *Blackboard*[11]-basierter *Context Manager* zu Grunde, um welchen die anderen Hauptkomponenten des Frameworks angeordnet sind (Abbildung 3.5, nämlich die *Application*, die *Context Source*, der *Context Abstractor*, der *Change Detector*, der *Application Controller* und der *Customizer*.

Der *Context Manager* speichert Daten, empfängt Anfragen, verwaltet Subskriptionen von jeglicher Komponenten das Context Frameworks und liefert entsprechende Antworten auf Kontextanfragen von Komponenten, Applikationen und Application Controllern. Anwendungen können entweder den Context Manager an sich nutzen und den daraus gewonnenen Kontext selbst verwerten oder können vom Applikation Controller gesteuert werden. Dieser wiederum kann über den Customizer zur Laufzeit entsprechend den gegebenen Anforderungen angepasst werden. Jede der anderen Komponenten kann entweder lokal oder getrennt davon auf einem zentralen Server implementiert werden.

Im zentralen Context Manager Blackboard Server laufen sämtliche Kontextdaten der einzelnen Komponenten zusammen und werden auch dort in Folge ausgewertet. Die Kontextdaten vom Blackboard stehen den Applikationen auf folgende drei Arten zur Verfügung:

[11] Im Zentrum des Blackboard-Architekturmodells [EnMo88] steht ein zentrales Messageboard auf welchem von diversen Komponenten, wie Services oder vollständige Anwendungen, Nachrichten eingetragen und abonniert werden können.

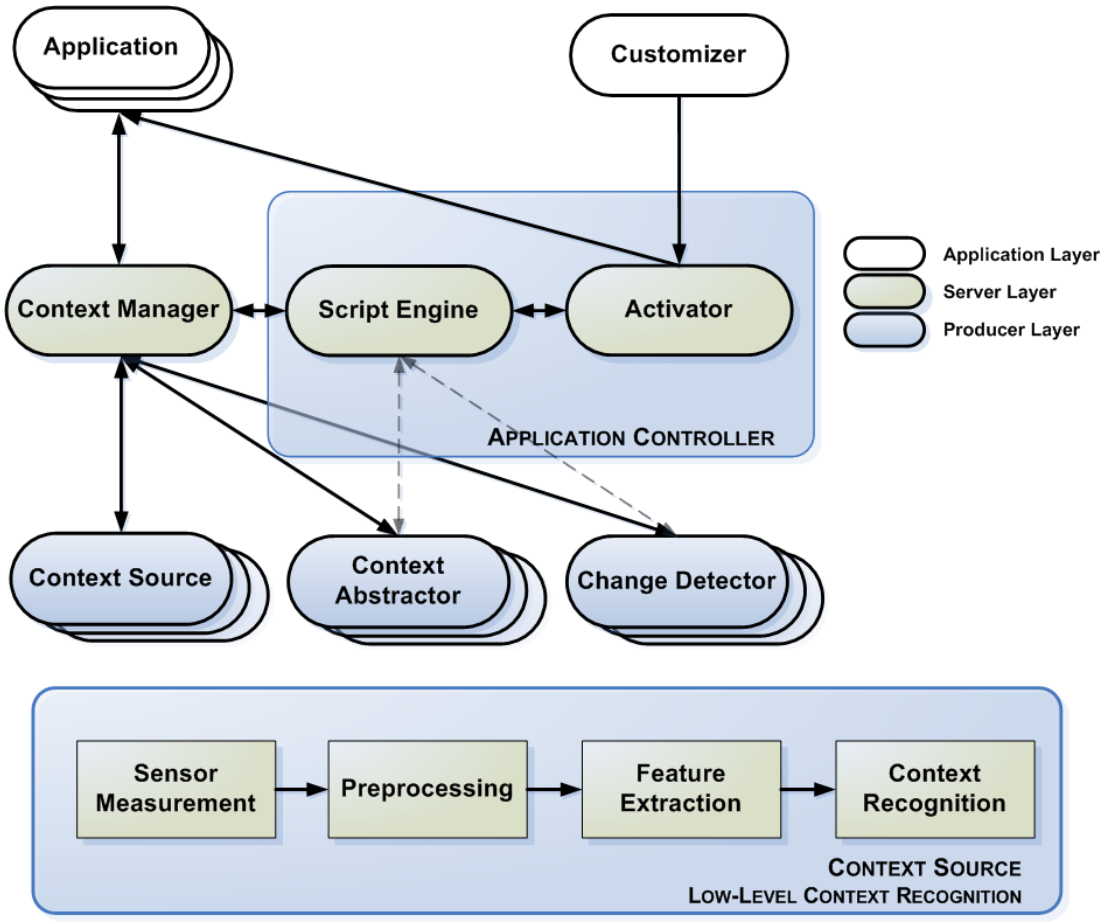

Abbildung 3.5: Schema des Context Framework nach [Korp05]

1. Direkte Abfragen der Daten aus der Datenbank des Context Managers
2. Anmeldung zu Benachrichtigungsdiensten (zum Beispiel für Benachrichtigungen bei definierten Kontextänderungen)
3. High-Level Kontext, wobei der Context Manager diesen für die Applikationen über entsprechende „Recognition Services“ ermittelt

Ein interessanter Gesichtspunkt bei diesem Ansatz ist, dass hier für unscharfe Sensordaten eine spezielle *Fuzzy-Logic* für die Ermittlung von entsprechendem logischen Kontext zum Einsatz kommt, wobei hingegen andere Systeme davon ausgehen, dass Kontextdaten so definiert werden können, dass diese eindeutig sind, genau zugeordnet und verwertet werden können [SiCo06] [BaDR06].

3.3.4 Das Hydrogen Framework

Der Hydrogen Ansatz [HSP+03](Abbildung 3.6) basiert auf einem objektorientiertem Kontextmodell wobei hier komplett auf eine zentrale Komponente zur

Kontextverwaltung und Auswertung verzichtet wird. Dies begründet sich auch durch das für dieses Framework gedachte Anwendungsgebiet, Mobile Computing. Daraus ergibt sich auch, dass das Framework komplett am jeweiligen Endgerät implementiert sein muss und dort sämtliche Aktionen ablaufen, wodurch gewisse Einschränkungen aufgrund der begrenzten Hardwareressourcen bei mobilen Endgeräten gemacht werden müssen.

Prinzipiell wird bei Hydrogen zwischen zwei Kontextarten unterschieden: dem *local* und dem *remote Kontext*. Der „local“ Kontext ist der Kontext des eigenen Gerätes und der „remote“ Kontext somit naheliegenderweise der Kontext, über den ein anderes Gerät verfügt. Die Erfassung der Kontextdaten erfolgt hierbei ad-hoc, wenn Endgeräte mit dem Hydrogen Framework in physische Nähe zueinander kommen zum Beispiel per Wireless Lan oder Bluetooth, was als *Context Sharing* bezeichnet wird.

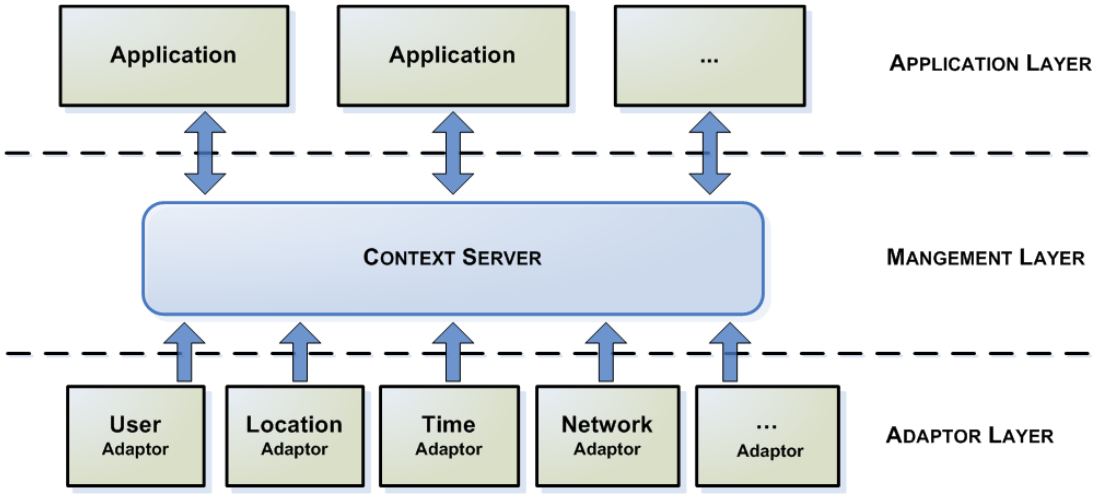

Abbildung 3.6: Die Hydrogen Architektur [HSP+03]

Die Architektur von Hydrogen 3.6 besteht aus drei Schichten. Der *Adaptor Layer* ist für die Ermittlung von den physischen Kontextdaten auf Basis von Sensoren zuständig wobei hier auch eine Anreicherung der Daten bis hin zu logischen Kontext möglich ist.

Im *Management Layer* wird sämtlicher, gerade aktueller Kontext des Endgerätes im *Context Server* gespeichert, welcher diesen den Anwendungen zur Verfügung stellt. Der Context Server kann diesen Kontext auch mit anderen Endgeräten, wenn diese in Empfangsreichweite sind, teilen und im Gegenzug auf den Kontext dieser Geräte zugreifen. Dadurch wird der zur Verfügung stehende Kontextumfang deutlich erhöht, da auf diese Art und Weise auch Kontextdaten, welche vom eigenen Gerät nicht ermittelt werden können mithilfe des remote Kontext anderen Geräte erfasst werden.

Der *Application Layer* umfasst alle Anwendungen, welche den Kontext des Management Layers und des Adaptor Layers nutzen.

3.3.5 Vergleich

Die Hauptcharakteristika der vier vorgestellten Context-Aware Frameworks sind in Tabelle 3.2 kurz zusammengefasst dargestellt. Prinzipiell zeigt sich klar, dass die Architektur der einzelnen Frameworks sich immer nach der Art der Kontexterfassung richtet, wobei alle präsentierten Ansätze das so genannte "Separation of Concerns" Paradigma umsetzen und eine saubere Trennung von Kontexterfassung/Auswertung und Benutzerkomponenten bieten.

Bei der Umsetzung der Datenerfassung setzt jedes Framework auf einen eigenen Ansatz, wobei hier generell noch kein Standard einer Description Language oder einer Ontologie für die Ermittlung und Übertragung von Kontextinformationen von verschiedenen Quellen existiert wodurch eine einfache Wiederverwendung dieser Daten in anderen Systemen möglich wäre.

Der wichtigste Teil eines intelligenten und anpassbaren Context-Aware Frameworks ist das verwendete Kontextmodell und die dahinter stehende Kontextverarbeitungslogik. Wie in Kapitel 3.1.7 bereits aufgezeigt, haben die verschiedenen Kontextmodelle unterschiedliche Ausprägungen und Stärken und Schwächen. Generell zeigt sich, das ontologiebasierte Modell im Allgemeinen die bessere Wahl sind, wobei ein entscheidender Nachteil von nichtontoligiebasierten Modellen die meist starke Verwebung und hohe Verteilung des Kontextmodell über die gesamten Quellcode der Anwendung ist. Dieser Nachteil kann mit aspektorientierter Programmierung jedoch deutlich reduziert oder im Optimalfall gänzlich beseitigt werden.

Die automatische Entdeckung und Aufnahme von neuen Ressourcen für die Kontexterfassung wird vor Allem in ubiquitären Web-Anwendungen immer wichtiger, da sich hier die verfügbaren Sensoren und Kontextquellen rasch ändern können. Fehlt ein solche Funktion in einem Framework ist das ein entscheidender Nachteil, da hierbei angenommen wird, dass immer dieselben Kontextquellen benutzt werden und diese auch immer verfügbar sind. Genauso können bei solchen Systemen zur Laufzeit keine Änderungen der Kontextprovider durchgeführt werden. Auch dieser Punkt kann mit aspektorientierter Programmierung einfach gelöst werden.

Das Wissen über vergangenen Kontext ist für eine gute und qualitativ hochwertige Customization von Anwendungen ein weiterer wesentlicher Faktor. Mit diesem Wissen wird die Möglichkeit der Implementierung von intelligenten Lernalgorithmen geschaffen, welche den Umfang der Adaptierung der Anwendung

	Architektur	Kontext-Erfassung	Kontext-Modell	Kontext-Auswertung	Kontext-ressourcen Ermittlung	Historischer Kontext	Sicherheit + Datenschutz
Context Toolkit	Wigetbasiert	Kontext Widgets	Key-Value	Kontext Interpertation und Aggregation	Discoverer Komponente	vorhanden	expliziter Kontextinhaber
CoBrA	Agentenbasiert	Kontext Akquisitions-modul	Ontologien (OWL)	Inferenzmaschine und Wissensdatenbank	nicht vorhanden	vorhanden	Rei Policy-Sprache
Contect Framework	Blackboard-basiert	Resourcen-server	Ontologien (RDF)	Kontext Erkennungsservice	Resourcen-server und Subskriptions mechanismus	nicht vorhanden	nicht vorhanden
SOCAM	Verteilt mit zentralen Server	Kontext Provider	Ontologien (OWL)	Vorwärst- und rückwärtsverkettetes regelbasiertes System	Service-findungsdienst	vorhanden	nicht vorhanden
Hydrogen	Drei Schichten-architektur	Adaptoren für diverse Kontexttypen	Objekt-orientiert	Nur Interpretation und Aggregation der Rohdaten	nicht vorhanden	nicht vorhanden	nicht vorhanden

Tabelle 3.2: Überblick über die vorgestellten Context-Aware Systeme [BaDR06]

erheblich steigern. Der Großteil der Frameworks unterstützt die Erfassung von historischem Kontext, wobei teilweise Einschränkungen bei der Vorhaltezeit gegeben sind. Frameworks, welche auf einen zentralen Server für die Kontextverwaltung verzichten, bieten oftmals auf Grund von beschränkten Hardwareressourcen auch keinen historischen Kontext.

Ein weiterer wichtiger Punkt ist der Umgang mit den erfassten Kontextdaten. Da hierbei oft auch sensible Benutzerdaten erfasst werden, sollen von den Frameworks Möglichkeit für den Schutz dieser gegeben sein. Dafür benutzt CoBrA die „Rei Policy“ Sprache um Sicherheitspolicies für Kontextinformationen verwirklichen zu können. Das Context Toolkit wiederum setzt auf das Konzept des „Kontexteigentümers“ welcher nur selbst auf seinen Kontext Zugriff hat und nur selbst Kontextdaten von sich für andere zugänglich machen kann. Alle anderen Systeme bieten keine Schutz der Kontextdaten, was ein entscheidender Nachteil dieser ist [BaDR06].

In Kapitel 6 wird das, für den Forschungsprototypen entwickelte Regelsystem erläutert, wobei hierbei keines der hier angeführten Frameworks eingesetzt wird, sondern ein eigenes System unter Nutzung von aspektorientierte Programmierung erstellt wird. Dies hat den Grund, dass eine maximale Kontextauswertung unter Beibehaltung einer großen Flexibilität angestrebt wird. Auch ist es mit dem gewählten Ansatz möglich, das Regelsystem im „worst-case“ aufgrund der Verwendung von AOP komplett auszutauschen oder zumindest grundlegend zu verändern. Dies ist bei den anderen Ansätzen nur unter erheblichem Mehraufwand, am einfachsten wohl auch wieder mit AOP, möglich.

Kapitel 4

Aspektorientierte Softwareentwicklung

Wie bereits in Kapitel 2.2.1 erläutert und in Abbildung 2.3 dargestellt ist, erstreckt sich die Customization über die gesamte Web-Anwendung. Dies hat wiederum zur Folge, dass auch der Code für die Customization und damit für die Kontextbehandlung zwangsweise über die gesamte Anwendung verstreut ist. Bei der Entwicklung des Forschungsprototypen wurde deshalb der Ansatz der *aspektorientierte Programmierung* verwendet, womit dieses Problem verhindert beziehungsweise großteils entschärft wurde.

In Folge wird eine kurze allgemeine Einführung in die aspektorientierte Programmierung gegeben. Im Kapitel 6 wird detailliert auf die Umsetzung und Anwendung der Prinzipien der aspektorientierten Software Entwicklung im entwickelten Tourismusinformationssystem eingegangen.

4.1 Grundlagen

Die aspektorientierte Software Entwicklung (Aspect-oriented Software Development, AOSD) ist eine Weiterentwicklung der objektorientierten Software Entwicklung, um so genannte *Crosscutting Concerns* zu modularisieren. Concerns sind Funktionalitäten, Konzepte oder Interessensgebiete im Programm, und werden in *Core Concerns* und *System-level Concerns* unterschieden. Core Concerns, die eigentlichen Kernfunktionalität, sind fachlicher Natur und können meist durch die Klassen der Geschäftslogik repräsentiert werden. System-level Concerns sind hingegen nicht fachlicher Natur und kommen an vielen Stellen im Programm verstreut vor. Sie ziehen sich durch (crosscut) die gesamte Anwendung, sind im Code der Core Concerns verteilt und können mit den Konzepten der objektorientierten Programmierung (OOP), dh. innerhalb von Klassen, nicht gekapselt wer-

den. Dies minimiert die Wiederverwendbarkeit und erschwert Änderungen. Diese Querschnittsbelange werden als Crosscutting Concerns bezeichnet. Typische Beispiele hierfür sind die System-level Concerns Logging, Authentifikation, Transaktionsmanagement und Error-Handling. Beispiel für crosscutting Core Concerns sind sich häufig ändernde Teile der Geschäftslogik, oder Code mit sehr vielen Fallunterscheidungen.

4.1.1 Vorteile von AOP

In Hinsicht auf die Grenzen der OO hat der Einsatz der AO einige Vorteile [Böhm06]. Durch AOP kann die *Wartbarkeit* des Code gesteigert werden, da Crosscutting Concerns modular in Aspekten, und nicht mehr verstreut im gesamten Programm, implementiert werden. *Änderungen* sind leichter und weniger fehleranfällig durchzuführen.

Der Code ist einfach *erweiterbar*, indem zusätzliche Funktionalitäten nachträglich über Aspekte eingebunden werden.

Durch das verbesserte SoC steigt der Grad der *Wiederverwendbarkeit*. Sowohl die Komponenten der Geschäftslogik, die nur noch die Kernfunktionalität beinhalten, als auch die Komponenten der Crosscutting Concerns können besser „recycled“ werden.

Ein weiterer großer Vorteil der AOP ist, dass Anpassung von Fremd-Software (auch ohne Änderungen am Source Code) möglich ist.

4.1.2 Nachteile von AOP

Die Aspektorientierung bringt durch die Einführung einer neuen Programmierpraxis (vor allem durch das Konzept des *Weavings*) auch neue Probleme und damit Nachteile mit sich.

Die *Fehlerlokalisierung* wird durch AOP erschwert. Da der später durch den Weaver eingefügten *Advice*-Code nicht direkt an der Stelle ersichtlich ist, an der er ausgeführt wird, ist das Programm schwerer nachvollziehbar. Auch zu weit definierte *Pointcuts*, dh. die Stellen an denen der Code des Aspekts einfgefügt werden soll, oder falsche Reihenfolgen bei der Hintereinanderausführung mehrerer Pointcuts können das Debuggen erschweren. Hier ist gute Tool-Unterstützung notwendig, die aktuelle Softwareentwicklungstools noch nicht, oder nur unzureichend bieten.

Durch den Weaving-Prozess werden im generierten Programm Methodenaufrufe eingefügt, was zu einem *Overhead* und in weiterer Folge zu *Performanzeinbußen* führen kann. Allerdings wären diese Methodenaufrufe wahrscheinlich auch in rei-

nen OOP Projekten zur Erfüllung der geforderten Funktionalität erforderlich - um die Flexibilität zu wahren wäre vielleicht sogar noch ein zusätzlicher Layer nötig.

Die *Modellierung* von AO Software ist *nur durch proprietäre Notationen* möglich. Der derzeit aktuelle UML 2.0 Standard[1] enthält keine Notation, um aspektorientierte Konzepte darzustellen. Es existieren aber schon sehr konkrete Modellierungsansätze [SSK+06, Stei02].

4.2 Motivation

Im Folgenden wird anhand eines einfachen Beispiels aus [Böhm06] der Nutzen von AOP erläutert. Listing 4.1 zeigt eine einfache Klasse `Konto`, die lediglich ein Attribut, den Kontostand, hat und Methoden für die Kontobewegungen einzahlen, abheben und überweisen zur Verfügung stellt.

Listing 4.1: Einfaches Konto Bsp.

```
1   public class Konto {
2       private double kontostand = 0.0;
3
4       public double abfragen() {
5           return kontostand;
6       }
7       public void einzahlen(double betrag) {
8           kontostand = kontostand + betrag;
9       }
10      public void abheben(double betrag) {
11          kontostand = kontostand - betrag;
12      }
13      public void ueberweisen(double betrag, Konto anderesKonto) {
14          abheben(betrag);
15          anderesKonto.einzahlen(betrag);
16      }
17  }
```

Diese Klasse ist noch sehr klar strukturiert und gut lesbar, da nur der Code der eigentlichen Geschäftslogik vorhanden ist. Erweitert man das Beispiel um Fehlerbehandlung und Logging, wird der Code unübersichtlich.

Listing 4.2: Erweitertes Konto Bsp.

```
1   public class Konto {
2       private double kontostand = 0.0;
3
4       public double abfragen() {
5           return kontostand;
6       }
7       public void einzahlen(double betrag) {
8           System.out.println(betrag + "␣soll␣eingezahlt␣werden");
```

[1] UML, http://www.uml.org/

```
9             if (betrag < 0) {
10                System.err.println("Einzahlung␣mit␣neg.␣Betrag␣verweigert");
11                throw new IllegalArgumentException("Betrag␣negativ");
12            }
13            kontostand = kontostand + betrag;
14            System.out.println(betrag + "␣auf␣Konto␣eingezahlt.");
15        }
16        public void abheben(double betrag) {
17            System.out.println(betrag + "␣soll␣ausgezahlt␣werden");
18            if (betrag < 0) {
19                    System.err.println("Abheben␣mit␣neg.␣Betrag␣verweigert");
20                    throw new IllegalArgumentException("Betrag␣negativ");
21                }
22                    if (kontostand < betrag) {
23                         System.err.println("Konto␣nicht␣gedeckt,␣Auszahlung␣
                               verweigert");
24                         throw new RuntimeException("keine␣Deckung");
25                     }
26                     kontostand = kontostand - betrag;
27                     System.out.println(betrag + "␣von␣Konto␣ausgezahlt.");
28            }
29            public void ueberweisen(double betrag, Konto anderesKonto) {
30                System.out.println(betrag + "␣soll␣ueberwiesen␣werden");
31                  this.abheben(betrag);
32                  anderesKonto.einzahlen(betrag);
33                  System.out.println(betrag + "␣auf␣" + anderesKonto + "␣
                        ueberwiesen");
34            }
35    }
```

In der Konto Klasse aus Listing 4.2 ist die eigentliche Geschäftslogik deutlich schwerer herauszulesen - dabei sind die für ein reales Konto notwendigen Funktionalitäten für Spesen, Zinsen und andere Rahmenbedingungen noch nicht berücksichtigt. Selbst durch Herausziehen der Logik (zB. die Überprüfung, ob der Betrag positiv ist) in eigene Methoden, beinhaltet die Konto Klasse zumindest alle Aufrufe zu solchen Methoden. Mit AOP können Fehlerbehandlung und Logging in einen so genannten *Aspekt* ausgelagert werden, dh. der entsprechende Code wird aus der Konto-Klasse entfernt und mit der Information, wo er ausgeführt werden muss, im Aspekt gespeichert. Zu einem späteren Zeitpunkt können dann mit Hilfe des *Weavers*, das ist ein Compiler, Fehlerbehandlung und Logging automatisch an den richtigen Stellen im Code wieder eingefügt werden. Wie die Fehlerbehandlung aus Listing 4.2 mit AOP gelöst werden kann, ist in Abbildung 4.1 dargestellt.

4.3 Einführung in AspectJ 5

Für die Entwicklung des Tourismusinformationssystems als Forschungsprototyp dieser Arbeit (siehe Kapitel 5, 6) haben wir uns für AspectJ (Version 5) entschie-

den.

AspectJ wurde ursprünglich bei Xerox PARC von Cristina Lopes und Gregor Kiczales, dem „Vater der aspektorientierten Programmierung“, entwickelt. Seit Ende 2002 wird AspectJ als Eclipse Projekt[2] weiterentwickelt.

AspectJ ist eine aspektorientierte Erweiterung für Java, ersetzt Java also nicht, sondern baut darauf auf. AspectJ Programme sind weitestgehend normale Java Programme mit Klassen, um Core Concerns zu strukturieren und Aspekten, um Crosscutting Concerns zu modularisieren. AspectJ erweitert Java um einige neue Konzepte. Die wesentlichsten Begriffe werden im Folgenden kurz erläutert.

4.3.1 Aspektorientierte Konzepte

Aspekt. Aspekte sind eine Erweiterung des Klassen-Konzepts und dienen als „Container“ für AOP Code, können aber auch OOP Code enthalten.

Joinpoint. Joinpoints (Verbindungspunkte) sind bestimmte Ereignisse im Programmablauf, an denen der ursprüngliche Code - mit Hilfe von AOP - manipuliert werden kann. Solche Ereignisse können beispielsweise der Aufruf einer Methode oder der Zugriff auf eine Variable sein.

Pointcut. Pointcuts (Schnittpunkte) selektieren beliebige Joinpoints, an denen der Programmablauf manipuliert werden soll.

Advice. Advices (Empfehlungen) beschreiben die Manipulation am ursprünglichen Code, die bei einem, vom Pointcut selektierten, Joinpoint durchgeführt werden soll. Ein Advice kann entweder vor (*before*), nach (*after*) oder statt bzw. rund um (*around*) einen Joinpoint ausgeführt werden.

Intertype Deklaration. Intertype Deklarationen ermöglichen die Erweiterung bestehender Klassen um Attribute, abstrakte oder konkrete Methoden und Konstruktoren. Weiters können dadurch Default-Implementierungen für Interfaces in Aspekten hinterlegt werden.

Weaving. Ein eigener Compiler, bzw. Pre-Processor, der *Aspect Weaver*, vereinigt den AOP Code mit dem OOP Code. Dieser Prozess wird in Abbildung 4.1 veranschaulicht und heißt *Weaving*. Das Weaving kann zu verschiedenen Zeitpunkten passieren: *Compile-Time Weaving* verbindet den Advice-Code bereits beim Kompilieren mit dem Byte-Code der (Java-)Klassen, *Post-Compile-Time Weaving* wird eingesetzt um bestehende Class- oder Jar-Dateien mit Aspekten anzureichern. Beim *Load-Time Weaving* geschieht die

[2] AspectJ Project, http://www.eclipse.org/aspectj/

Integration erst durch den Classloader, damit können zB fremde Bibliotheken angepasst werden, ohne sie direkt zu verändern. Diese Ansätze werden als „Static AOP“ bezeichnet. *Run-Time Weaving* bindet den AOP Code erst zur Laufzeit in der Virtual Machine ein und heißt „Dynamic AOP“. Derzeit gibt es nur wenige Produkte, die umfangreiche Unterstützung für Run-Time Weaving bieten.

4.3.2 Kontobeispiel

In Abbildung 4.1 wird gezeigt, wie das Konto-Beispiel aus Listing 4.1 mit AOP um Fehlerbehandlung erweitert werden kann, ohne den ursprünglichen Code zu verändern.

Hierzu wird ein Aspekt, der „ErrorHandlingAspekt“, erstellt in welchem ein *Pointcut* und ein *Before-Advice* definiert sind.

Die Überprüfung des Betrags in den Methoden `einzahlen()` und `abheben()` wird in den *Before-Advice* ausgelagert. Die Prüfung selbst ist der selbe Java-Code wie in Listing 4.2, wobei das Schlüsselwort `before` angibt, dass der Code *vor* dem Pointcut *kontobewegung* ausgeführt werden soll.

Der Pointcut *kontobewegung* definiert die Stellen, an denen der Before-Advice ausgeführt werden soll, also den Aufruf (`call`) der öffentlichen Methoden `einzahlen()` und `abheben()` in der Klasse `bank.Konto` mit dem Rückgabewert `void` und einem Parameter vom Typ `double`. Durch `args(betrag)` kann im Advice auf den Parameter `double betrag` der beiden Methoden zugegriffen werden, was für die Überprüfung notwendig ist.

Durch das Weaving wird die Funktionalität des Aspekts schließlich im Byte-Code in die Konto Klasse eingefügt, wie in Abbildung 4.1 ersichtlich ist.

Weitaus umfassender wird dieses Thema und die Anwendung von AOP bei ubiquitären Web-Anwendungen in Kombination mit den Erfahrungen bei der Entwicklung des Forschungsprototypen in [Bros06] behandelt.

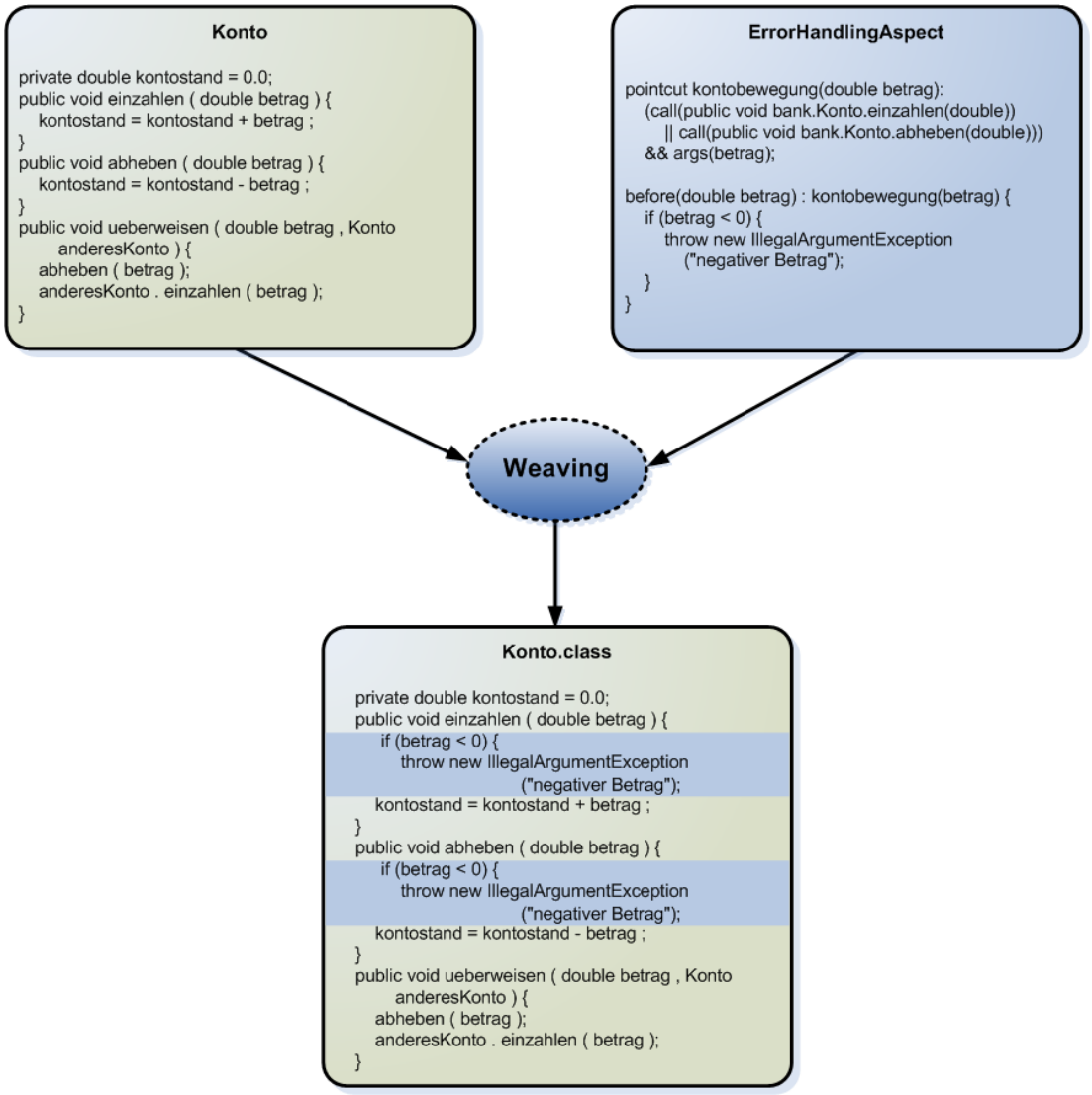

Abbildung 4.1: Weaving

Kapitel 5

Technische Konzeption und Funktionsumfang des entwickelten Tourismusinformationssystems

Im Zuge dieser Arbeit wurde eine praxisnahe, ubiquitäre Web-Anwendung entwickelt. Um möglichst viele Beispiele zur Customization zu zeigen, entschlossen wir uns, als Forschungsprototyp ein Tourismusinformationssystem zu implementieren.

Gerade die Bedürfnisse eines Urlaubsgasts sind ein Paradebeispiel für die Notwendigkeit des *anytime/anywhere/anymedia* Paradigma von ubiquitären Web-Anwendungen. Dieser hat den Wunsch, sich im Urlaub überall mit seinem Handy/PDA oder über einen beliebigen Internetzugang über Freizeitangebote und aktuelle Angebote in seiner Urlaubsregion zu informieren. Die ubiquitäre Web-Anwendung passt sich dabei immer an die momentane Situation des Benutzers an und liefert ihm zusätzlich noch die für ihn interessantesten Informationen direkt auf der Startseite.

5.1 Allgemeine Beschreibung

Die Grundidee besteht darin, ein Tourismusinformationssystem zu entwickeln, das dem Benutzer objektive und tagesaktuelle Informationen zu den verschiedenen Urlaubsorten liefert. Es soll eine Plattform geboten werden, die dem Benutzer einerseits *vor dem Urlaub* eine Entscheidungshilfe, wo er seinen Urlaub verbringen möchte, und andererseits *während des Urlaubs* ein ständiger Begleiter und Ratgeber ist. Darin liegt auch der große Unterschied zu bereits vorhandenen Tourismusinformationssystemen, da der Gast mit dem System umfangreiche und tagesaktuelle Informationen zu allen möglichen Freizeitaktivitäten, dem Wetter,

der Schneelage, Events, Lokale, usw. einer Region seiner Wahl geliefert bekommt und somit auch während seinem Aufenthalt eine ständige und zuverlässige zentrale Informationsquelle zur Verfügung hat.

Gerade im Vergleich mit anderen Tourismusinformationssystemen[1][2]wird sofort klar, dass diese großteils nur 'Presales'-Informationen zur Verfügung stellen. Bereits auf der Startseite finden sich eine Vielzahl an Preisen und Buchungsmöglichkeiten. Das erklärte Ziel der Anwendungen ist, den Benutzer auf Urlaub in Österreich zu schicken, aber man hat immer den Eindruck es soll primär ein bestimmtes Angebot verkauft werden. Für detaillierte Informationen über einzelne Regionen und deren Angebote wird meist auf die Seite des lokalen Tourismusverbands verwiesen. Das von uns entwickelte System sollte sich genau dieser Problematik annehmen und die Angebote abseits der Hotelzimmer zentral und objektiv präsentieren. Mit Zusatzfunktionalitäten für registrierte Benutzer soll eine 'Community' geschaffen werden, welche auch die Möglichkeit hat, einzelne Angebote zu bewerten und zu kommentieren. Gerade der Kontrast von den aufdringlichen Angeboten auf den bereits vorhandenen Anwendungen zu den eher unauffälligen Möglichkeiten der gezielten Information bei einer ubiquitären Web-Anwendung macht den Reiz an der Entwicklung eines solchen Systems aus.

5.2 Basisfunktionalität

Die Web-Anwendung teilt sich in zwei getrennte Anwendungsgebiete, die Content-Erstellung (*Backend*) und die Content-Nutzung (*Frontend*). Das Frontend dient primär dazu, dem Endbenutzer Informationen anzuzeigen, die im Backend von Redakteuren und anderen Content-Erstellern eingegeben und gepflegt werden. Der Funktionsumfang der Web-Anwendung wird im Folgenden anhand von UML Use Case Diagrammen näher beschrieben.

5.2.1 Funktionen für den Endbenutzer (Frontend)

Abbildung 5.1 zeigt die möglichen Anwendungsfälle für Benutzer des Frontend. Dieser Funktionsumfang steht im System angemeldeten Usern zur Verfügung. Anonyme Benutzer können die Hauptfunktionalität - Content browsen, nach Informationen suchen, Sprache einstellen und Newsletter abonnieren - ausführen, sowie sich als Benutzer anmelden um auch Ratings abgeben und den Reisekoffer nutzen zu können.

[1] http://www.tiscover.at/
[2] http://www.austria.info/

Abbildung 5.1: Frontend Use Case

Content browsen. Content browsen ist die Grundfunktion im Frontend. Hier kann der Benutzer nach Regionen oder Urlaubs- und Aktivitätsthemen gegliedert die verschiedenen Informationen durchstöbern. Die Informationen werden in Form von Artikeln dargestellt und bestehen aus Text und optionalen Bildern. Die Artikel werden in einer an den Benutzer angepassten Reihenfolge geliefert, dh. Artikel, die den Benutzer mehr interessieren könnten, werden in der Liste zuerst angezeigt.

Reisekoffer anzeigen. Angemeldete Benutzer können sich interessante Artikel in ihrem persönlichen „Reisekoffer“, ähnlich einem Warenkorb in bekannten E-Commerce Anwendungen, speichern um später einfacher auf die Information zugreifen zu können.

Artikel drucken. Die Artikel können gedruckt werden.

Artikel löschen. Die Artikel im Reisekoffer können wieder gelöscht werden.

Erinnerung aktivieren. Darüber hinaus können zu bestimmten Artikeln im Reisekoffer Erinnerungen aktiviert werden, wodurch der Benutzer wahlweise per E-Mail oder SMS, rechtzeitig vor Beginn des Events, der Aktivität, etc. informiert wird.

Kommentar abgeben. Angemeldete Benutzer können zu einem Artikel ein Kommentar und eine Empfehlung für verschiedene „Urlaubstypen“ abgeben. Urlaubstypen sind beispielsweise „Familie mit Kindern“, „Ältere Urlauber“ oder „Extremsportler“.

Artikel suchen. Der Benutzer kann eine Volltextsuche in allen Artikeln durchführen.

Benutzer Anmelden. Um die Web-Anwendung noch besser auf seine persönlichen Anforderungen anzupassen und auch Ratings abgeben und den Reisekoffer nutzen zu können, kann sich der Benutzer im System anmelden. Vor der ersten Anmeldung ist eine Registrierung notwendig.

Benutzer registrieren. Bei der Registrierung kann der Benutzer eine bevorzugte Sprache und einen „Urlaubstyp“ wählen, der auf ihn zutrifft.

Profil aktualisieren. Die gespeicherten Informationen können jederzeit wieder bearbeitet werden.

Newsletter abonnieren. Jeder Benutzer hat die Möglichkeit, sich zu einem Newsletter an- und abzumelden. Basierend auf den erfassten Daten des Benutzer und anderen Faktoren, wie zum Beispiel die aktuelle Jahreszeit, spezielle Events, ... werden die Newsletter für den einzelnen Benutzer individuell vom System angepasst und in regelmäßigen Abständen versandt.

Sprache einstellen. Der Benutzer kann zwischen verschiedenen Sprachen wählen, in denen die Web-Anwendung angezeigt wird. Dabei kann er entweder direkt in der Weboberfläche umschalten. Wenn der Benutzer angemeldet ist, kann er in seinem Profil die gewünschte Sprache speichern. Beim Start der Web-Anwendung wird bei angemeldeten Benutzern die im Profil eingestellte Sprache gewählt und bei nicht angemeldeten Benutzern wird die Sprache aus dem Request Header erkannt.

Geplante Funktionen

Folgende Frontend-Funktionen sind für spätere Versionen der Web-Anwendung geplant:

User-Page. Die User-Page ist eine Möglichkeit für den angemeldeten Benutzer, seine Urlaubserfahrungen mit anderen zu teilen. Hier kann eine Liste der Lieblings-Artikel veröffentlicht werden, Urlaubsberichte und Bildergalerien erstellt werden, etc.

Wo kann ich ... Die Funktion „Wo kann ich ...“ erweitert die Volltextsuche und erlaubt eine spezielle Suche nach Orten und Regionen, in denen vom Benutzer bevorzugte Aktivitäten möglich sind.

5.2.2 Funktionen für die Redaktion (Backend)

Das Backend stellt die Funktionalität zur Content-Erstellung bereit. Hier werden zwei Rollen unterschieden, „Redakteur“ und „Tourismusverband“ (TVB). Die Rolle Redakteur ist ein systeminterner Mitarbeiter und für die Administration der Web-Anwendung zuständig. Der Redakteur verwaltet Kategorien, Benutzer der Rolle TVB und kann das Layout und die Struktur der Web-Anwendung für verschiedene Endgeräte anpassen. Benutzer mit der Rolle TVB können für ihre Region Inhalte erstellen und verwalten. In Abbildung 5.2 sind alle Anwendungsfälle dargestellt.

Artikel bearbeiten. Dies ist die Hauptfunktion des Backend und die einzige, die ein Benutzer mit der Rolle TVB durchführen kann. Es können Artikel zu bestimmten Orten und Kategorien (zB Wintersport, Kultur, etc.) in verschiedenen Sprachen angelegt, bearbeitet und gelöscht werden. Die Informationen können durch verschiedene Contentparts strukturiert und mit Hilfe eines in der Anwendung integrierten HTML WYSIWYG[3]-Editors formatiert werden. Contentparts sind Teile eines Artikels und können derzeit Text und Bild enthalten. Contentparts mit Multimediainhalten sind ebenso denkbar. Die Bilder, die zu den Artikel gespeichert werden können, werden automatisch in der Größe und im Corporate Design der Weboberfläche (Rahmen, Schlagschatten, leichte Drehung) angepasst. Auf Wunsch wird ein kurzer Titel auf das Bild geschrieben. Zu jedem Artikel kann die Kommentar-Funktion ein- oder ausgeschaltet werden und der Artikel als unsichtbar markiert werden. Weiters kann eine Tageszeit angegeben werden, zu der der Artikel besonders interessant ist.

[3] What You See Is What You Get

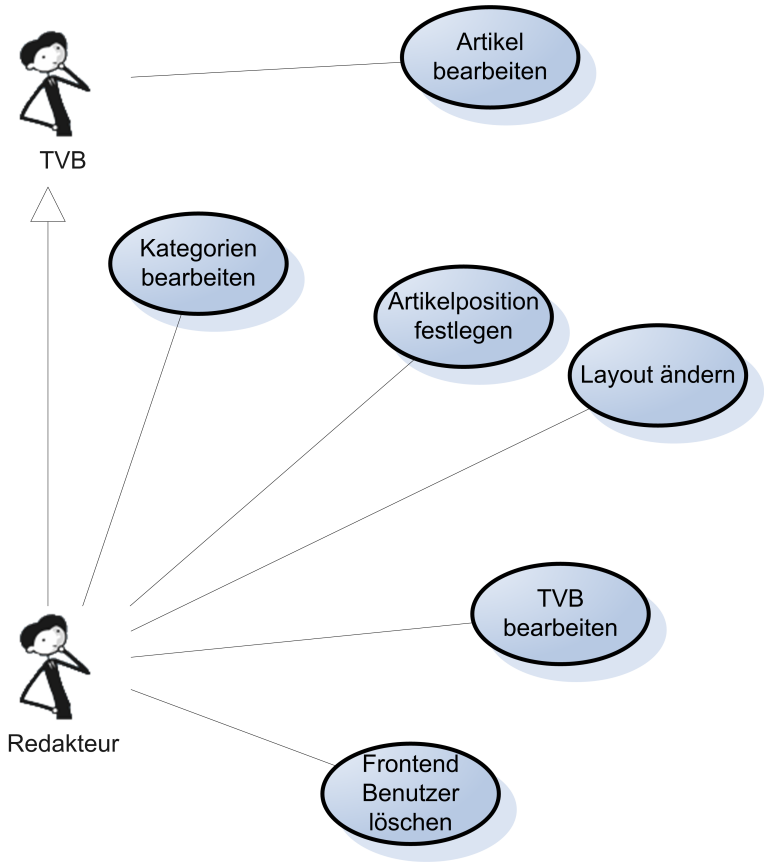

Abbildung 5.2: Backend Use Case

Artikelpositionen festlegen. Der Redakteur kann für bestimmte Artikel die an den Benutzer angepasste Reihenfolge ausschalten und fix an die oberste Stelle in der Artikelliste platzieren. Dies ist beispielsweise für bezahlte Inhalte (Werbung) nützlich.

Kategorien bearbeiten. Der Redakteur kann in verschiedenen Sprachen Kategorien anlegen, bearbeiten und löschen. Kategorien werden zu Saisonen (Sommer/Winter) zugeordnet und gliedern Urlaubs- und Aktivitätsthemen, zu denen die Artikel angelegt werden können. Kategorien sind zB Schifahren, Kultur oder Wellness.

Layout ändern. Der Redakteur kann das Layout, also die Struktur der Web-Anwendung für verschiedene Endgeräten verändern. Die Struktur gibt die Reihenfolge und Position der einzelnen Elemente der Web-Anwendung an.

TVB bearbeiten. Der Redakteur kann Benutzer der Rolle TVB anlegen, bearbeiten bzw. löschen und ihre Berechtigungen verwalten.

Frontend Benutzer löschen. Der Redakteur kann Benutzer des Frontends löschen.

Anwendungsfälle, die keinen Einfluss auf die Customization haben, wurden zwar in der Datenbank berücksichtigt, aber nicht implementiert. Nicht implementierte Use Cases im Frontend sind Artikel suchen, Newsletter abonnieren und Erinnerung aktivieren, im Backend wurde auf die Anwendungsfälle TVB bearbeiten, Frontend Benutzer löschen und Layout ändern verzichtet.

5.3 Evaluierung der Customizationfunktionalität

In diesem Kapitel wird der Umfang der Customizationfunktionalität des entwickelten Forschungsprototypen anhand des *Evaluation-Frameworks für ubiquitäre Web-Anwendungen* aus [KPRS03] analysiert. Prinzipiell umfasst dieses Framework die beiden Dimensionen der Customization, namentlich den Kontext und die Adaptierung, wobei diese anhand eines umfangreichen Kritierenkatalogs für die jeweilige Web-Anwendung erfasst werden. Aus diesen Kriterien kann in Folge der Umfang der Customization und allfällige Stärken und Schwächen abgeleitet werden. Diese sind hierbei so gewählt, dass einerseits Anforderungen, welche bei der Entwicklung von ubiquitären Web-Anwendungen umgesetzt werden sollten, erfasst werden. Anderseits sind aber auch Kriterien aufgeführt, welche notwendig sind, damit bestehende Ansätze in diesem Framework mit berücksichtigt werden können und somit eine gemeinsame Basis für die Charakterisierung von Web-Anwendungen unter dem Gesichtspunkt der Ubiquität möglich ist.

In Folge wird nun auf den Kontext, die Adaptierung und ihre Kriterien nach [KPRS03] im Detail eingegangen und der Forschungsprototyp anhand dieser charakterisiert. In den Tabellen 5.1 und 5.2 werden die Ergebnisse übersichtlich zusammengefasst gezeigt.

5.3.1 Charakteristik des Kontext

Hierbei werden die Umstände der Benutzung der Web-Anwendung erfasst und anhand von, im Framewerk vorgegebenen, Kriterien analysiert. Eine Beschreibung des Kontextes von ubiquitären Web-Anwendungen ist bereits in Kapitel 2.2.2 erfolgt. In dem Framework erfolgt eine Verdichtung des Gesamtkontextes auf bestimmte Eigenschaften, welche die Umgebung der Anwendung und diese selbst beschreiben und den Umfang der Customization bestimmen. Des Weiteren werden die Kontext-bezogenen Kriterien anhand ihres *Umfanges*, ihrer *Repräsentation*, ihrer *Akquisition* und ihrer *Zugriffsmechanismen* unterteilt. In Tabelle 5.1 sind die Ergebnisse der Evaluierung des Forschungsprototypen hinsichtlich der Kontextkriterien zusammengefasst.

Kontextumfang

Dieser umfasst nicht nur die unterschiedlichen *Kontexteigenschaften*, welche vom System bei der Customization unterstützt werden und die damit verbundene *Erweiterbarkeit*, sondern auch die *Zeitdimension* des Kontext in Bezug auf die *Chronologie* und *Gültigkeit*.

Kontextfaktor Die relevanten Kontexteigenschaften sind natürlich sehr stark von der zu entwickelnden Web-Anwendung abhängig, jedoch haben sich in der einschlägigen Literatur Standardfaktoren für ubiquitäre Web-Anwendung herauskristallisiert, nämlich die *Ort*, *Zeit*, *Gerät*, *Netzwerk*, *Benutzer* und die *Anwendung*. (Für eine Erläuterung dieser Faktoren siehe 2.2.2.)

Andere Diese Kategorie existiert im Framework nicht. Um aber den Kontextfaktor *Wetter*, welchem im Forschungsprototypen eine entscheidende Bedeutung zu kommt, abbilden zu können, wurde diese Kategorie eingeführt. Mit Ausnahme der „Anwendung“ werden alle hier aufgezählten Kontextfaktoren von der entwickelten Web-Anwendung berücksichtigt.

Erweiterbarkeit Dieses Kriterium wird dem Anspruch gerecht, dass zusätzliche Kontextfaktoren, welche nicht zu den genannten Standardfaktoren zählen, bei bestimmten Anwendungen mit berücksichtigt werden müssen. Nachdem nicht vorhergesehen werden kann, welche Faktoren das eventuell sein könnten, muss es möglich sein das bestehende System mit zusätzlichen Kontextfaktoren zu erweitern. So können zum Beispiel die Schneelage oder die Wassertemperatur für gewisse Anwendungen durchaus als wichtige Kontextfaktoren in Betracht kommen. Eine Erweiterung der Kontextfaktoren bei dem Forschungsprototypen ist durch den Einsatz von aspektorientierter Programmierung welche die Kontextdetektoren in das System einbindet und in Kombination mit dem erweiterbar gestalteten Datenbankschema einfach möglich. In diesem Zusammenhang ist natürlich auch die Einbindung von externen Datenquellen möglich.

Chronologie Die Erfahrung hat gezeigt, dass nicht nur der *aktuelle Kontext* sondern auch der *historische Kontext* für die Customization eine entscheidende Rolle spielt. So macht es Sinn, die Bandbreite der Verbindung des Benutzers über die Zeit zu verfolgen, um eine Durchschnittsbandbreite ermitteln und somit die Anwendung darauf abstimmen zu können. So kann man gegebenfalls Bilder verkleinern oder multimedia Inhalte ausblenden, um den Benutzer bei niedriger Bandbreite trotzdem eine schnelle Benutzung der Web-Anwendung zu ermöglichen. Aber nicht nur der historische sondern

auch der *zukünftige Kontext* kann für bestimmte Anwendungen sehr wertvoll sein. So ist es beim Videostreaming neben der durchschnittlichen vergangenen Bandbreite genauso wichtig abzuschätzen wie sich diese in der Zukunft entwickelt, um das Video so anzupassen, dass es zu keiner Unterbrechung und zu einer konstanten Übertragung kommt. Auch das historische Benutzerverhalten liefert wichtige Daten für die Customization. So werden im Rahmen des Forschungsprototpyen die Artikel, welche der Benutzer über die Zeit aufgerufen oder in seinem Reisekoffer gegeben hat, gespeichert um daraus ein Interessensprofil für die Adaptierung des Inhaltes ableiten zu können. Die Chronologie der Endgerätnutzung wird jedoch nicht erfasst, genausowenig wie die Vorhersage von zukünftigem Kontext implementiert ist.

Gültigkeit Kontextfaktoren sind nicht immer unendlich gültig und können auf eine ganz bestimmte Zeitspanne beschränkt sein. Im mobilen Bereich wäre hierfür ein Beispiel, dass, wenn sich der Standort des Gerätes innerhalb einer bestimmten Zeitspanne nicht ändert, dieses vielleicht nicht mehr online und somit die Ortsinformation nicht mehr gültig ist. Auch geänderte Öffnungszeiten zu verschiedenen Jahreszeiten stellen eine Einschränkung der Gültigkeit dieser Information dar und führen dazu, dass für einzelne Kontextfaktoren Zeitperioden, innerhalb welcher diese valide sind, spezifiziert und berücksichtigt werden müssen. Dieses Kriterium wird beim Forschungsprototypen bei dem logischen Kontextfaktor der Saison und tageszeitlich eingeschränkten Artikel berücksichtigt.

Kontextrepräsentation

Die Repräsentation des Kontext verbindet zwei wichtige Themen, nämlich Mechanismen für die Verbesserung der Wiederverwendung der Kontextrepräsentation und das Abstraktionsniveau des Kontexts.

Wiederverwendbarkeit Hierbei soll der Kontext explizit im System aufgeführt sein und nicht nur vermischt mit der Adaptierung an sich vorliegen. Dies hat den Vorteil, dass der Kontext in anderen Systemen einfach wiederverwendet werden kann. Auch soll der Kontext, der durch den Customization Ansatz geliefert wird, generisch sein, also anwendungsunabhängig, wobei ein solcher Kontext auch von externen Anbietern geliefert werden kann. So kann hier zum Beispiel auf externe GIS[4] Daten zurückgegriffen werden. Bei dem entwickelten Forschungsprototypen wurde hierfür auf Aspektorientierung

[4]Geografisches-Informationssystem

gesetzt, womit untersucht wurde, ob die angesprochene Trennung des Kontextes von der Programmlogik an sich mit AOP sinnvoll durchzuführen ist. Mit diesem Ansatz ist auch die Wiederverwendung in anderen ubiquitären Web-Anwendungen relativ einfach möglich. Siehe dazu auch 6.3.

Abstraktion Hierunter fallen die Unterscheidung und Behandlung von logischen und physischen Kontext. Siehe hierzu 2.2.2. Als Beispiel im Forschungsprototypen kann hier einerseits für den physischen Kontext die IP, das Datum und die angesehenen Artikel aufgezählt werden. Der logische Kontext umfasst unter anderem den aktuellen Standort in Form einer Adresse, die Saison und das Wetter.

Kontextakquisition

Hier wird der Grad der Automation und der Grad der Dynamizität der Kontextakquisition behandelt.

Automation Hierbei unterscheidet man wer für die Sammlung des Kontext verantwortlich ist. Ist es ein Mensch, so spricht man von *manueller Akquisition*, ist es ein System, dann ist es die *automatische Akquisition*. Im Falle einer Kombination von beiden nennt man es *halbautomatische Akquisition*. Prinzipiell ist es bei ubiquitären Web-Anwendungen erstrebenswert soviel wie möglich an Information automatisch zu generieren, um die Benutzerinteraktion in diesem Bereich gering zu halten. Physischer Kontext kann großteils automatisch von der Umgebung erfasst und der daraus resultierende logische Kontext vom System abgeleitet werden. Aber natürlich funktioniert das nicht überall und so ist es oft nötig, die Kontextfaktoren halbautomatisch zu erfassen. Dabei wird zum Beispiel vom Benutzer sein Profil ausgefüllt, woraus das System den logischen Benutzer-Kontext ermittelt (zum Beispiel eine Benutzerkategorie) oder fehlende Daten zu ergänzen versucht. In der realisierten Web-Anwendung werden zum Beispiel Benutzerdaten teilweise manuell erfasst, das heißt der Benutzer gibt Daten zu seiner Person selbst ein wie zum Beispiel seinen Namen, sein Geburtsdatum oder seinen Heimatort. Darüber hinaus wird zum Beispiel die Userkategorie halbautomatisch, die Endgeräteeigenschaften und die maximale Auflösung, automatisch erfasst.

Natürlich können wichtige Daten auch fehlen, wenn zum Beispiel der Benutzer zum ersten Mal die Web-Anwendung aufruft. Aus diesem Grund muss das System so ausgelegt sein, dass es auch mit diesem Fall umgehen kann.

Ein Standardansatz hierbei ist es, einen im System definierten Default-Kontext zu benutzen, was auch beim Forschungsprototypen so umgesetzt wurde.

Dynamizität Ein weiterer wichtiger Punkt ist, wann der Kontext erfasst wird. Der *statische Kontext* wird nur einmal erfasst - also zum Beispiel beim Start der Anwendung - und in Folge nicht mehr verändert, was zum Beispiel bei der Ermittlung des Gerätes mit dem die Anwendung benutzt wird, ausreichend ist. Ist der *Kontext dynamisch*, so wird bei jeder Änderung während der Laufzeit auch der Content angepasst, wobei hier wieder die Bandbreite ein gutes Beispiel ist. Im Prinzip ist bei ubiquitären Web-Anwendungen eine dynamische Kontextermittlung nötig, jedoch ist aufgrund von Performanzüberlegungen auch statischer Kontext in gewissem Maße sinnvoll. Hierbei wird in der Anwendung wiederum beides erfüllt. So wird zum Beispiel das Gerät nur beim Start der Benutzersession ermittelt, wobei hingegen das Wetter bei jedem Request dynamisch erfasst wird.

Kontextzugriff

Zugriffsmechanismus Hierbei wird die Art der Veranlassung der Adaptierung auf Basis eines geänderten Kontextes betrachtet. Dies kann durch den *push-* oder *pull*-Ansatz realisiert werden. Bei dem push-Ansatz werden die betreffenden "Clients" im Falle einer Änderung des spezifizierten Kontextes sofort automatisch verständigt. Beim pull-Ansatz hingegen muss der aktuelle Kontext von dem Adaptierungsmechanisums selbst angefordert werden, wenn dieser benötigt wird. Aus Gründen der Flexibilität sind hier Systeme die beide Ansätze unterstützen, die bessere Wahl. Beim Forschungsprototpyen wurde hier nur der pull-Ansatz verwirklicht. Dies hat auch zur Folge, dass eine Customization nur auf einen Request vom Benutzer hin erfolgt und nicht automatisch wenn sich ein Kontextfaktor ändert. So müssen zum Beispiel bei einer einfachen Customization nach einem Ort, einem Benutzer und einer Saison sämtliche benötigten Daten von der Adaptierung „geholt" werden.

5.3.2 Charakteristik der Adaptierung

In der zweiten Dimension des Evaluation-Frameworks von [KPRS03] wird die Art der Adaptierung, also was für Operationen durchgeführt werden müssen, der Gegenstand der Adaptierung und wie das zu geschehen hat, im Prozess der Adaptierung behandelt.

	Kontext Umfang											Kontext Repräsentation		Kontext Akquisition					Kontext Zugriff	
	Eigenschaft							Erweiterbarkeit	Chronologie		Gültigkeit	Wiederverwendbarkeit	Abstraktion	Automation			Dynamizität		Zugriffs-mechanismus	
	Ort	Zeit	Gerät	Netzwerk	Benutzer	Anwendung	Andere		Historie	Zukunft				manuell	halbautomatisch	automatisch	statisch	dynamisch	push	pull
Forschungs-prototyp "TIP"	✔	✔	✔	✔	✔	✘	✔	✔	✔	✘	✔	✔	✔	✔	✔	✔	✔	✔	✘	✔

✔ ... unterstützt | ✘ ... nicht unterstützt

Tabelle 5.1: Evaluierung der Kontext-Charakteristiken nach [KPRS03]

Art der Adaptierung

In Tabelle 5.2 sind die Ergebnisse der Evaluierung des Forschungsprototypen hinsichtlich der Kontextkriterien zusammengefasst.

Operation Ubiquitäre Web-Anwendungen sollen zumindest eine Bibliothek an Adaptierungsoperationen für die Behandlung der Standardkontextfaktoren bieten. Beispiele aus dem entwickelten Forschungsprototypen hierfür sind Contenfilterung, die Anpassung der Auflösung von Bildern oder die Anpassung der Navigation.

Erweiterbarkeit Gleich wie bei den Kontextfaktoren sollen auch die eingebauten Adaptierungsvorgänge erweiterbar sein, um geänderten Anforderungen gerecht werden zu können. Dies wird in der Anwendung durch den Einsatz von aspektorientierter Programmierung und des entwickelten Regelsystems erreicht. Siehe 2.2.2 und 6.3.2.

Effekt Mit Hilfe der Adaptierungsoperation sollen zumindest drei unterschiedliche Ergebnisse erreicht werden können. Einerseits soll es möglich sein, bestimmte Teile zur Anwendung *hinzu zu fügen*, die vorher nicht existent waren, wie zum Beispiel eine personalisierte Werbung. Des Weiteren sollen Teile *entfernt* werden können, wie zum Beispiel sämtlicher multimedialer Inhalt oder *angepasst* werden können, wie zum Beispiel die Bildgröße und Qualität an die Bandbreite der Verbindung des Benutzers. Dies Alles erfüllt

der Forschungsprototyp - es können zum Beispiel Contentparts hinzugefügt oder gelöscht und Bilder an das jeweilige Endgerät angepasst werden.

Komplexität In der Customization sollen nicht nur *einfache* Adaptierungsoperationen, welche bei einem bestimmten Kontext greifen, möglich sein, sondern auch *komplexe* Operationen, welche mehrere Adaptierungsvorgänge auf das Selbe oder auch auf verschiedene Subjekte angewendet werden. Als Beispiel kann hierfür die Transformation von der Web-Anwendung auf ein mobiles Endgerät mit einer kleinen Bandbreite, wo die Navigation, die grafische Oberfläche und der Inhalt angepasst werden muss, genannt werden. Auch diese beiden Arten der Adaptierung sind mit dem Forschungsprototypen möglich. So kann eine Adaptierung definiert werden, welche sich nur den Ort als Kontext heranzieht und auf diese Basis die Anwendung anpasst - oder aber genauso komplexe Adaptierungen, welche zum Beispiel die Artikel auf Basis vielen Kontextfaktoren, wie der Saison, dem Ort, dem Wetter, der Benutzerkategorie,... auswählen und auf Basis der Kontextfaktoren vom Gerät anpassen.

Subjekt der Adaptierung

Das betroffene Subjekt der Adaptierung wird in dem, dieser Analyse zugrunde liegenden, Framework charakterisiert durch den *Level* der Web-Anwendung, genauso wie durch konkrete *Elemente* und durch die Unterscheidung nach der Anzahl der betroffenen Elemente, der *Granularität*.

Level Prinzipiell sollen alle Ebenen einer Web-Anwendung adaptierbar sein [BrMa02] [KaRS00]. Diese Forderung erfüllt auch der Forschungsprototyp. So wird auf der Inhaltsebene die Artikelauswahl, auf Hypertextebene die Linkstruktur und auf der Präsentationsebene die Darstellung je nach Kontext angepasst.

Element Jede Ebene einer Web-Anwendung beinhaltet eine Menge an Anwendungselementen, wie zum Beispiel *Seiten*, *Links*, *Eingabefelder*, *Listen* und *multimediale Inhalte*. Jedes dieser Elemente sollte auch anpassbar sein. In den Bereich *Andere* fällt bei dem Forschungsprototypen das Rating-Element.

Granularity Gibt die Nummer der, durch eine Adaptierung, betroffenen Elemente einer Web-Anwendung an. Hier wird zwischen *Mikroadaptierung*, welche nur eines oder wenige Elemente betrifft (wie zum Beispiel das Ausblenden eines Links) und *Makroadaptierung*, wobei viele, verschiedene Teile der

Anwendung verändert werden, unterschieden. Als Beispiel für eine Makroadaptierung kann hier das Ändern sämtlicher Texte, auch in Bildern und der Navigation, auf der Website in eine andere Sprache herangezogen werden. Im Extremfall wäre eine Makroadaptierung auch die komplette Auswechslung einer Anwendung, wenn diese besser zu dem aktuellen Kontext passt. Hierbei würde es zum Beispiel ein Anwendung für Desktopbenutzer und ein eigene Anwendung für Benutzer von mobilen Geräten geben. Hier treffen auch wieder beide möglichen Ausprägungen auf die entwickelte Web-Anwendung zu. Einerseits werden zum Beispiel einzelne Contentparts oder Links ausgeblendet, was einer Mikroadaptierung entspricht, oder es wird die ganze Seite verändert, wenn der Benutzer eben mit einem mobilen Endgerät die Anwendung benutzt, was wiederum einer Makroadaptierung entspricht.

Prozess der Adaptierung

Aufgaben Der Adaptierungsprozess umfasst eine Reihe von Aufgaben, welche zum Zweck der gezielten Steuerung ihrer Automation und Dynamizität aufgeteilt werden sollen. Mit diesem Kriterium wird angegeben, ob die Adaptierung eine Unterteilung in Aufgaben vorsieht und unterstützt. Folgende Aufgaben werden in dem Framework aus [KPRS03] genannt: Die *Initiierung*, welche angibt wer oder was den Adaptierungsprozess auslöst. Der *Adaptierunsvorschlag*, also was für Möglichkeiten der Adaptierung bestehen, wie zum Beispiel drei unterschiedliche Bildervarianten (schwarzweiß, klein, groß). Dann die *Vorschlagsselektion*, welche aufgrund des aktuellen Kontextes einen Vorschlag auswählt, welcher in der *Produktion* entweder dynamisch zur Runtime erstellt oder statisch durch das Laden einer vorgefertigten Version durchgeführt wird. Dies wird in der *Präsentation* dem Benutzer angezeigt, welcher unter Umständen über die *Revision* die eben durchgeführte Änderung wieder rückgängig machen kann.

Automation Vergleichbar mit der Automation bei dem Kontext wird auch hier zwischen manueller, halbautomatischer und automatischer Adaptierung unterschieden (siehe auch 2.2.1). Hierbei macht es zum Beispiel Sinn, dem Benutzer zwar eine angepasste Version seiner Startseite einer Web-Anwendung vorzuschlagen, diesem aber auch die Möglichkeit zu geben, diese nach seinem Willen zu verändern, was einer halbautomatischen Adaptierung entspricht. In dem Forschungsprototypen kann der Benutzer nur zum Teil eingreifen, rein manuell ist nicht möglich. So kann der Benutzer aber zum

Beispiel die Sprache oder eine Benutzerkategorie einstellen, was zu einer halbautomatischen Adaptierung führt. Große Teile der Adaptierung laufen jedoch vollkommen automatisch ab, als Beispiel sei hier die Anpassung an das jeweilige Endgerät genannt.

Dynamizität Wie schon beim Kontext zwischen statisch und dynamisch unterschieden worden ist, so wird auch bei der Adaptierung zwischen *statischer* und *dynamischer Adaptierung* unterschieden. Statische Adaptierung bedeutet, dass diese abhängig von gewissen Kontextfaktoren fix vordefiniert ist und somit nicht geändert werden kann. Dynamische Adaptierung hingegen bedeutet, dass diese zur Runtime, abhängig von den gerade gegebenen Kontextfaktoren, durchgeführt wird. Ein Beispiel ist hier die Adaptierung von Bildern. Werden diese nur zum Beispiel in drei verschiedenen Größen zur Verfügung gestellt, so spricht man von statischer Adaptierung - wird die Bildgröße dynamisch, je nach aktueller Anforderung, erzeugt, ist es hingegen die erwähnte dynamische Adaptierung. In der entwickelten Web-Anwendung kommen beide Ansätze zum Einsatz. Einerseits werden hierbei zwei Grundlayouts für die Darstellung der Web-Anwendung geboten, welche je nach Endgerät gewählt werden und statisch vorgegeben sind. Andererseits wird der Inhalt, also die Auswahl der Artikel, aber auch die Darstellung dieser, im entsprechenden Layout dynamisch generiert.

Adaptierungsschritte Die Adaptierung muss nicht jedes mal auf das Neue erfolgen. So kann das Ergebnis einer Adaptierung persistent gehalten werden und muss bei einer erneuten Anforderung nicht nochmal durchgeführt werden, was eine erhebliche Performanzsteigerung bewirken kann. Als Beispiel kann wieder die Kompression eines Videos genannt werden. Ansätze hierfür sind im Forschungsprototypen umgesetzt. So wird zum Beispiel die Linkstruktur der Kategorien und Orte entsprechend der Sprache und vorhandenen Artikel zwischengespeichert und nur im Bedarfsfall, zum Beispiel bei Änderungen im Backend, neu geladen und adaptiert.

	Art der Adaptierung							Subjekt der Adaptierung										Prozess der Adaptierung						
	Operation	Erweiterbarkeit	Effekt			Komplexität		Level			Element					Granularität		Aufgaben	Automation			Dynamizität		Adaptierungsschritte
			hinzufügen	entfernen	anpassen	einfach	komplex	Inhalt	Hypertext	Präsentation	Text	Bild	Audio	Video	Link	Andere	micro	macro		automatisch	halb-automatisch	manuell	statisch	dynamisch	
Forschungs-prototyp "TIP"	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✘	✘	✔	✔	✔	✔	✘	✔	✔	✘	✔	✔	~

✔ ... unterstützt | ~ ... teilweise unterstützt | ✘ ... nicht unterstützt

Tabelle 5.2: Evaluierung der Adaptierungs-Charakteristiken nach [KPRS03]

5.4 Architektur

5.4.1 Schichten-Architektur der realisierten Web-Anwendung

Der Forschungsprototyp wurde nach dem MVC[5]-Model 5.3 konzipiert und implementiert eine Schichten-Architektur, welche die wohl gebräuchlichste Methode ist, um komplexe Systeme in weniger komplexe Teile aufzuteilen. Der große Vorteil besteht darin, dass die eine Schicht immer nur mit der nächsten unter ihr kommuniziert, aber nichts von den Funktionen der darunterliegenden Schicht wissen muss. Das bedeutet, dass jede Schicht die Schichten unter ihr vor den darüberliegenden Schichten verbirgt. Dadurch werden die Abhängigkeiten unter den Schichten minimiert, jede Schicht ist ein in sich schlüssiges System und die Möglichkeiten zur unabhängigen Modellierung, Entwicklung und eventuell späteren Standardisierung werden vereinfacht [Demo06].

Im Folgenden werden die Schichten des verwendeten Fünf-Schichten-Modells kurz erläutert, siehe auch Abbildung 5.4.

Presentation Layer Ist jene Schicht, mit der der Benutzer interagiert. Im konkreten Fall sind dies JSP Seiten die HTML-Code erzeugen, der im Browser des Benutzers dargestellt wird. Alle Eingaben und Aktionen werden vom Benutzer auf dieser Schicht getätigt und dann von den darunter liegenden Schichten ausgeführt bzw. bearbeitet.

Controller Layer Alle Benutzereingaben werden hier validiert und in die jewei-

[5]Model View Controller

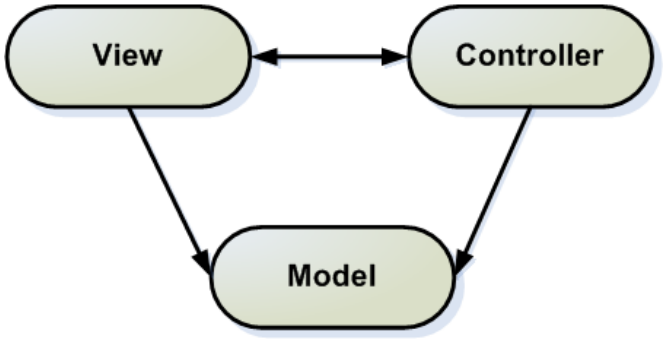

Abbildung 5.3: MVC-Modell

ligen Klassen der darunter liegenden Schicht, dem Business Layer weitergeleitet, danach entscheidet der Controller mit Hilfe der Informationen aus der darunter liegenden Schicht, welche Seiten und Inhalte angezeigt werden und in welcher Reihenfolge dies geschieht.

Business Layer Dieser ist der komplexeste Teil, hier befindet sich der Großteil der Prozesslogik einer Anwendung. Alle Auswertungen finden auf dieser Schicht statt, zum Beispiel das Erfassen des physischen Kontextes und die darauf folgende Ableitung des logischen Kontextes.

Data Mapper Layer Diese Schicht kapselt die CRUD[6]-Funktionen einer Datenbank wie *insert*, *update*, *select* oder *delete* von den Objekten des Business Layers ab und stellt eigene Klassen zur Verfügung, die den Tabellen in der Datenbank entsprechen. Somit sind alle datenbankspezifischen Funktionen auf einer Schicht angesiedelt, ein nicht zu übertreffender Vorteil, wenn die Anwendung beispielsweise mit Datenbanken verschiedener Hersteller funktionieren soll.

Data Layer Im Data Layer geschieht die eigentliche persistente Datenhaltung. Dies kann auf verschiedenste Arten geschehen zum Beispiel durch Datenbanken, XML Dateien oder andere Dateisysteme.

5.4.2 Datenbank-Schema

Bei der Konzeption der Datenbank wurde speziell auf die Anforderungen eines Tourismusinformationssystems eingegangen, wobei dies von Beginn an unter dem Gesichtspunkt einer maximalen Customization erfolgt ist. Damit wurde sichergestellt, dass einerseits der von Anfang an geplante Kontext entsprechend gespeichert wird und so gestaltet ist, dass dieser auch im nach hinein einfach erweitert

[6] **C**reate **R**ead **U**pdate **D**elete

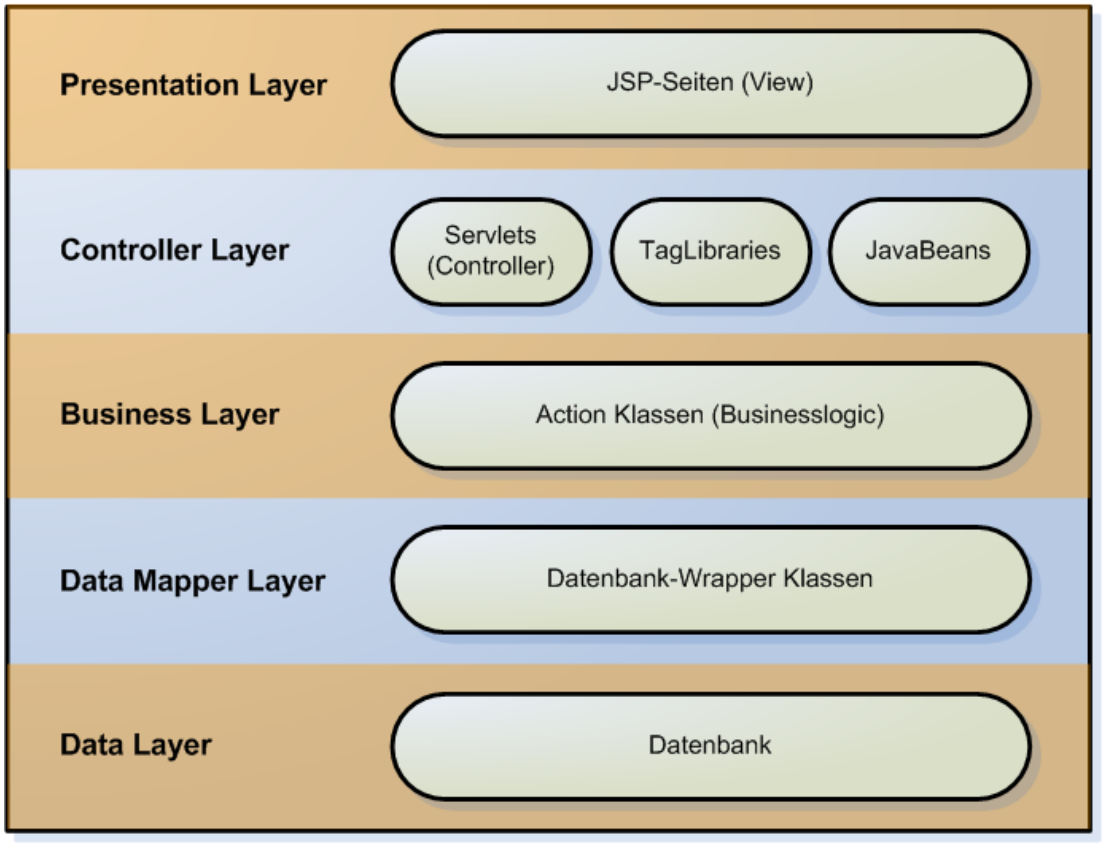

Abbildung 5.4: Architektur

werden kann. Mit eine Folge von dieser Herangehensweise war es, dass bei dem Entwurf des Datenbankschemas im Speziellen darauf geachtet wurde die Daten gekapselt und möglichst redundanzfrei zu speichern, um auch auf der Ebene der Datenbasis den „Seperation of Concerns“ Gedanken umzusetzen.

In der Abbildung 5.5 ist die Datenbankstruktur dargestellt, wobei aus Gründen der Übersichtlichkeit hierbei die Übersetzungstabellen für die einzelnen Entitäten nicht abgebildet sind. Mit Hilfe dieser Übersetzungstabellen ist es möglich, das System mit beliebig vielen Sprachen basierend auf beliebigen Zeichensätzen zu erweitern. Welche Tabellen eine eigene Übersetzungstabelle besitzen wird durch den Stereotyp *dict* angezeigt.

Die Kerndaten des Systems sind die Artikeldaten. Diese werden hierbei verteilt auf mehrere Tabellen gespeichert, was den Vorteil hat, dass ein Artikel einfach aus unterschiedlichen Teilen bestehen und je nach Anforderung zusammengestellt werden kann. Das Grundgerüst eines Artikels ist der *ArticleFrame*, welcher auch jene Metadaten zu einem Artikel beinhaltet, die sprachübergreifend gelten und welche wiederum Einfluss auf die Customization haben. Ein Beispiel ist hier das Attribut *IsOutdoor*: Bezieht sich ein Artikel auf eine Aktivität im Freien, so ist dieses Attribut wahr - was bei schlechtem Wetter zur Folge hat, dass dieser Artikel

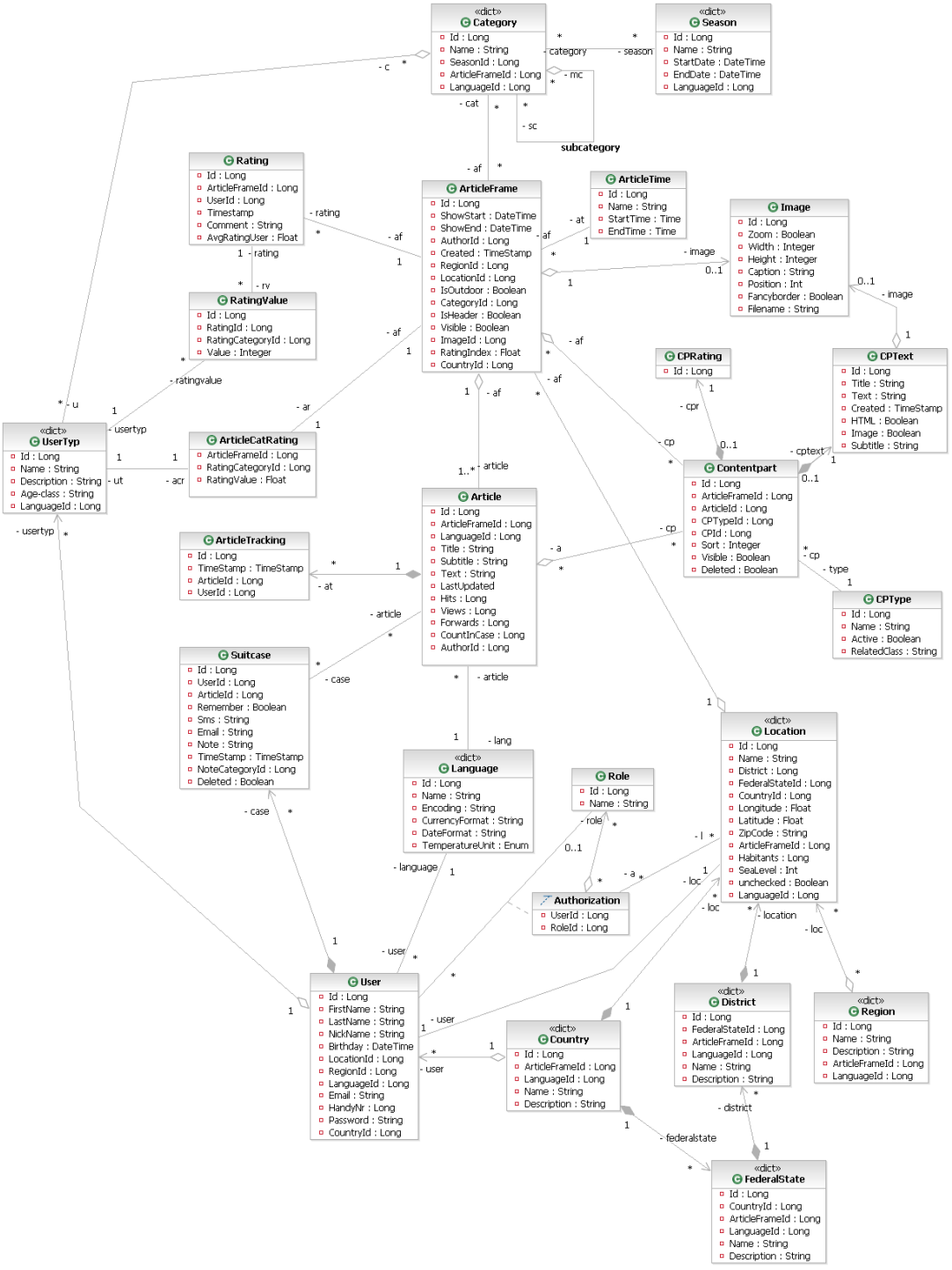

Abbildung 5.5: Datenbank-Schema

nicht oder zumindest nur nach Artikel, welche nicht im Freien sind, angezeigt wird.

Der Inhalt eines Artikels an sich wird sprachspezifisch in der *Article* Tabelle gespeichert.

Mit der *Contentpart* Tabelle und den damit verbundenen Tabellen ist es möglich, einerseits einen Artikel um einzelne, vorgegebene Teile zu erweitern, wie zum Beispiel einem Rating-Modul, oder aber auch einfach zusätzlich neue Module für die Artikelgestaltung im Nachhinein zur Web-Anwendung hinzuzufügen, wie zum Beispiel ein Galerie-Modul.

Benutzerdaten werden genauso erfasst, wie auch ein Benutzer einer Kategorie zugewiesen wird, was wiederum, neben seinem aufgezeichnetem Interaktionsverhalten in der Web-Anwendung (angesehene Artikel, Artikel in seinem Reisekoffer, abgegebene Ratings), natürlich in die Customization einfließt. Hierbei spielen natürlich auch die Kategorien, die Jahreszeit und der Ort eine wichtige Rolle.

5.5 Adaptierungsszenario

Im folgenden Abschnitt soll der Unterschied zwischen einer herkömmlichen und einer ubiquitären Web-Anwendung verdeutlicht werden. Hierzu wird dasselbe Szenario einmal mit und einmal ohne, den für die Customization nötigen Modulen und Daten, dargestellt. Abbildung 5.6 zeigt den Ablauf eines Requests ohne Aspekte und somit auch ohne Customization. Dabei werden die ausgewählten Artikel nur nach den Navigationsparametern (wie zum Beispiel die Kategorie oder den Ort auf den der Benutzer gerade geklickt hat) und gespeicherten Userdaten (wie zum Beispiel Usertyp) ausgewählt. Um die Unterschiede besser darstellen zu können wird folgendes Szenario definiert.

- Ein dem System bereits bekannter Benutzer loggt sich am Donnerstag, dem 9.3.2006 um 09:00 Uhr von einem PC aus in die Web-Anwendung ein. Die aktuelle Saison ist die Wintersaison, der Standort des Users ist Haus im Ennstal. Das aktuelle Wetter: -6°C und leichter Schneefall. Der Terminal verfügt über eine Internetverbindung mit hoher Bandbreite. Der User hat in seinem Profil folgende Daten bekannt gegeben:

Alter:	24
Heimatregion:	Mödling, Niederösterreich
Bevorzugte Sprache:	Deutsch
Ich würde mich selbst Charakterisieren als:	Jung und Aktiv

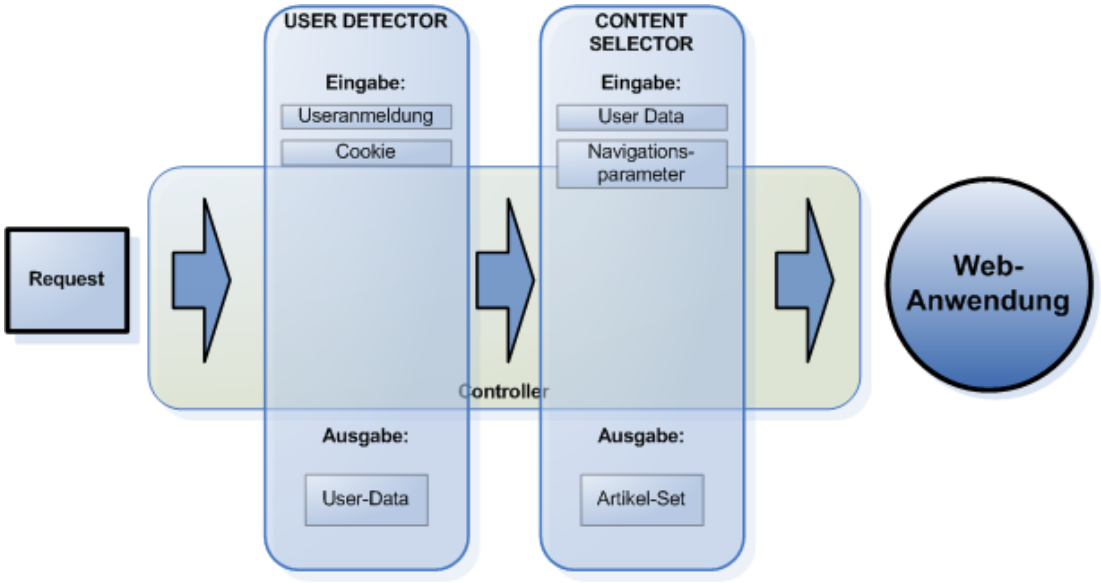

Abbildung 5.6: Requestablauf ohne Adaptierung

Wird das definierte Szenario auf den Requestablauf in Abbildung 5.6 angewandt, so würden für eine Adaptierung nur die Benutzerdaten und jeweilige Navigationsparameter für die Artikelauswahl zur Verfügung stehen. Nach dem Login würde der Benutzer wahrscheinlich die gleiche Startseite wie alle anderen Benutzer seiner Region erhalten. Das beschriebene Szenario lässt aber darauf schließen, dass der Benutzer nicht auf der Suche nach, beispielsweise Wintersportangeboten in Niederösterreich ist, sondern sich bereits auf Urlaub in Haus im Ennstal befindet und eventuell gerne Informationen über die Pistenverhältnisse und Events in seinem Wintersportort hätte. Um das zu erreichen muss die Web-Anwendung bei der Artikelauswahl für die Startseite mehr Informationen berücksichtigen.

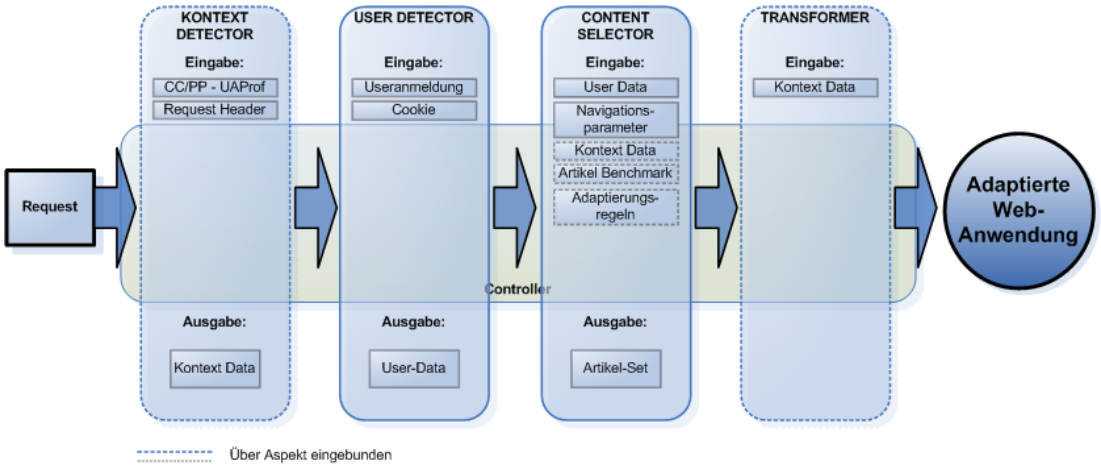

Abbildung 5.7: Requestablauf mit Adaptierung

Abbildung 5.7 zeigt nun wie der Controller der Anwendung erweitert werden muss um dies zu erreichen. Die zusätzlichen Module werden über Aspekte in die Anwendung eingebunden. Bei Instanzierung der Session liest der Kontext Detector eventuelle CC/PP[7] Informationen aus, diese betreffen größten teils das verwendete Endgerät also Beispielsweise: Hersteller und Typ des Geräts, Bildschirmauflösung und Art der Tastatur usw. Weiters liest der Kontext Detector alle wichtigen Informationen aus dem Request Header, also IP-Adresse, bevorzugte Sprache, Browsersprache usw. All diese Kontextdaten stehen die ganze Session über zur Verfügung. Zusätzlich zu den Kontextinformationen werden nun auch noch Daten aus dem Artikel-Benchmark und die Adaptierungsregeln für die Auswahl der Artikel berücksichtigt. Sie werden mit Aspekten direkt in den Content Selector eingebunden. Der Artikel-Benchmark errechnet sich Nutzungsdaten zB wie oft ein Artikel angesehen wurde, oder wie oft er zu einem Reisekoffer hinzugefügt wurde, und auch daraus wie die Benutzer den Artikel bewerten. Mit Hilfe der Adaptierungsregeln wird nun festgelegt wie all diese Informationen für die Artikelauswahl untereinander gewichtet sind. Sie sind sozusagen die Stellgrößen mit denen man die Auswahl zusätzlich manuell beeinflussen kann, um zB bestimmte bezahlte Artikel weiter nach oben zu reihen. Zuletzt kommt der Transformer zum Einsatz, er passt den ausgewählten Content an das Endgerät des Benutzers an, so werden zB Bilder verkleinert oder komplett entfernt oder große Tabellen umstrukturiert damit sie auf einen kleinen Bildschirm passen.

Für das zuerst definiert Szenario bedeutet das, dass nun alle zur Verfügung stehenden Informationen in die Artikelauswahl mit einfließen. Somit kann mit Hilfe der IP-Adresse der Ort (Haus im Ennstal) festgestellt werden, damit kann bei einem externen Anbieter das Wetter abgefragt werden (-6°C und leichter Schneefall) und auf Grund der hohen Bandbreite und den Informationen aus dem Request Header (unterstützte Plugins) weiß die Anwendung, dass sie dem User auch Multimedia-Content anzeigen kann. Die somit am wahrscheinlichsten erstgereihten Artikel werden die aktuellen Pistenverhältnisse, die aktuelle Wettervorhersage und eine der Webcams vom Hauserkaibling beinhalten. Danach würden aufgrund des Alters (24) und der Benutzerkategorie (Jung und Aktiv) aktuelle Abendprogramme, wie zB spezielle Events in den lokalen Diskotheken, gereiht nach dem Artikel-Benchmark angezeigt.

Dieses Beispiel zeigt, wie groß die Unterschiede zwischen einer adaptierten und einer nicht bzw. nur personalisierten Web-Anwendung sind. Ohne dass der Benutzer zusätzliche Informationen preisgegeben hat (es wurden nur sowieso vorhandene Umgebungsfaktoren in die Artikelauswahl miteinbezogen), bekommt er den

[7] Composite Capabilities/Preferences Profile

für interessantesten Inhalt. Dies war nur eines von einer Vielzahl an möglichen Adaptierungsszenarien, sofern die Anwendung über ausreichende Informationen über den Nutzungskontext verfügt, ist jedes nur erdenkliche Szenario möglich.

Kapitel 6

Implementierung

In diesem Kapitel werden Details zur Implementierung des Forschungsprototypen aufgezeigt. Zuerst werden die verwendeten Technologien und Bibliotheken kurz vorgestellt (siehe Abschnitt 6.1). Es folgt eine Beschreibung der grundlegenden Elemente der Web-Anwendung in Abschnitt 6.2. Der konkrete Einsatz der aspektorientierten Programmierung für die Customization der Web-Anwendung wird anhand von Codebeispielen in Abschnitt 6.3 gezeigt. Hierbei wird im speziellen auf die Aufteilung in Kontextaspekt 6.3.1, Adaptierungsaspekt mit dem Regelsystem 6.3.2 und Transformationsaspekt 6.3.3 eingegangen. Zum Abschluss werden in Abschnitt 6.4 Vor- und Nachteile der Implementierung untersucht.

6.1 Verwendete Technologien

Die verwendeten Technologien und Bibliotheken werden im Folgenden kurz beschrieben. Abbildung 6.1 zeigt, in welchen Schichten unserer Architektur diese Technologien eingesetzt werden.

Java. [1] Der Forschungsprototyp wurde mit Java (J2SE 5.0 JDK) entwickelt. Die Realisierung der Web-Schnittstelle basiert auf den Java Enterprise Technologien Servlet, JavaServer Pages (JSP) und Tag Libraries.

AspectJ 5. [2] AspectJ wird als Erweiterung für Java eingesetzt, um besseres Separation of Concerns des Crosscutting Concerns Customization zu erlangen (siehe Abschnitt 6.3).

MySQL. [3] Die persistente Speicherung der Artikel-, User-, Kategorie-, Regions- und Trackingdaten erfolgt mit Hilfe der relationalen Datenbank MySQL 5.0.

[1] http://java.sun.com
[2] http://www.eclipse.org/aspectj
[3] http://www.mysql.org

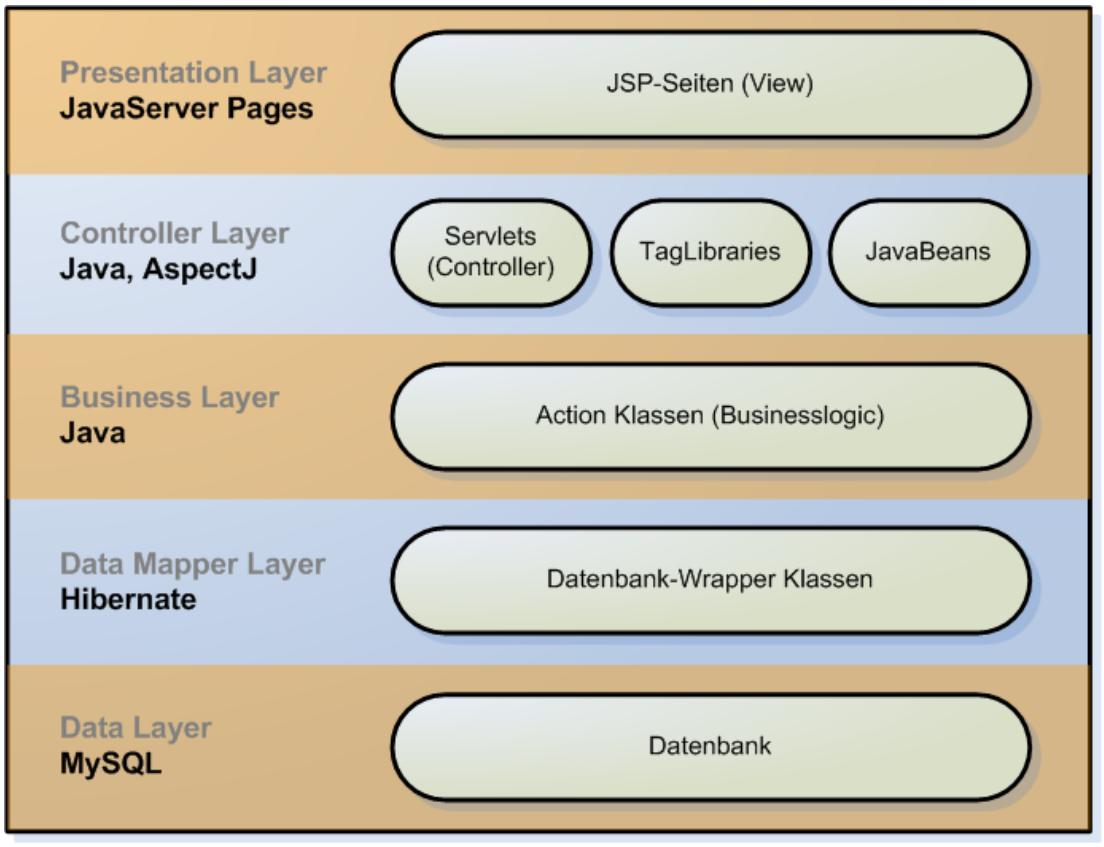

Abbildung 6.1: Eingesetzte Technologien

Hibernate. [4] Für den Datenbankzugriff wird das objekt-relationale Persistenz-Framework Hibernate verwendet. Dies ermöglicht eine leichtgewichtige und transparente Persistierung der Entitäten, die als Plain Old Java Objects (POJOs) implementiert sind. Somit kann mit einem objektorientierten Domänenmodell gearbeitet werden. Abfragen erfolgen entweder durch ein Query Object, das Hibernate Criteria Object, mit dem schrittweise durch Hinzufügen von Kriterien eine Abfrage zusammengebaut werden kann, oder mit der Hibernate Query Language (HQL), die eine objektorientierte Erweiterung von SQL darstellt.

Bibliotheken. Folgende Third-Party Libraries kamen bei der Entwicklung des Forschungsprototyps zum Einsatz:

FCKeditor.[5] Die FCKeditor wird zum Anlegen/Bearbeiten von Artikel im Backend verwendet. Es stellt einen WYSIWYG[6] oder einfachen Text Editor in Form eines HTML Input Forms zur Verfügung.

[4] http://www.hibernate.org
[5] http://www.fckeditor.net
[6] What You See Is What You Get

Apache Commons Fileupload.[7] Diese Bibliothek kann Requests mit MultipartContent verarbeiten. Dies wird im Backend benötigt, um Bilder zu den Artikeln auf den Server zu laden.

JH Labs Java Image Filters.[8] Die Bilder zu den Artikeln werden automatisch mit einem Rahmen, einem Schlagschatten und einer leichten Drehung versehen. Der Schlagschatten wird mit Hilfe der Klasse `ShadowFilter` aus dem JH Labs Library realisiert.

MaxMind GeoIP.[9] Die MaxMind GeoIP Bibliothek wird dazu eingesetzt, um den Standort der Benutzer im Frontend auf Basis der IP Adresse zu erkennen.

DELI: Delivery context library.[10] Diese Bibliothek dient dazu, Composite Capabilities/Preferences Profiles (CC/PP) aus dem Request Header auszulesen. Dadurch kann das verwendete Endgerät des Benutzers erkannt, und in weiterer Folge ein dafür geeignetes Template geladen werden.

6.2 Allgemeine Beschreibung

Abbildung 6.2 zeigt die Startseite des entwickelten Forschungsprototypen im Winterdesign. Die grundlegenden Elemente werden im Anschluss kurz beschrieben. Die Zahlen der folgenden Aufzählung entsprechen jenen aus Abbildung 6.2.

1. Hier wird die Übersicht eines besonders interessanten oder ev. bezahlten Artikel angezeigt. Der Text wird automatisch vom Transformer 6.3.3 auf eine passende Länge gekürzt.

2. Unter Regionen und Aktivitäten befinden sich Javascript Menüs (in der Abbildung nicht sichtbar) in denen die Regionen, Orte usw. bzw. Aktivitäten, Sportarten usw., zu denen Angebote vorhanden sind, angezeigt und ermöglichen eine schnelle Auswahl des gewünschten Angebots.

3. Das linke Menü dient zur Navigation durch die vorhandenen Artikel und zeigt immer die jeweilige Kategorie oder den jeweiligen Ort zu dem in der Mitte (Pkt. 6) angezeigten Inhalt an.

[7] http://jakarta.apache.org/commons/fileupload
[8] http://www.jhlabs.com/ip/filters/index.html
[9] http://www.maxmind.com/app/ip-location
[10] http://www.hpl.hp.com/personal/marbut/deli

Abbildung 6.2: Screenshot des Forschungsprototypen im Winterdesign

4. Registrierte Benutzer können sich hier mit ihrem Benutzernamen und Passwort anmelden oder registrieren, sie bleiben automatisch auch über die Session durch ein Cookie angemeldet, bis sie sich wieder abmelden.

5. Zeigt ein Beispiel für eine mögliche Werbeeinschaltung. Diese sind in den beiden äußeren Spalten an den meisten Stellen problemlos möglich.

6. In der mittleren Spalte wird immer eine Liste von Artikel angezeigt, außer der Benutzer wählt einen speziellen Artikel aus, dann wird nur dieser angezeigt. Ein Artikel der Artikelliste besteht aus dem Schlagtext (bzw. der Einführung), einem Bild das automatisch an die erforderliche Größe angepasst, mit einem Rahmen, Schatten und einer leichten Drehung versehen wird, und der besten Durchschnittsbewertung anderer Benutzer. Weiters werden auch noch der Inhalt des Reisekoffers[11] und die Benutzerregistrierung in dieser Spalte angezeigt.

7. Das Wetter für den aktuellen Tag und die Vorhersage für den nächsten Tag werden immer passend zu dem ausgewählten Artikel angezeigt oder bei einer Artikelliste passend zum ersten.

8. Hier besteht die Möglichkeit die Sprache zu wechseln.

9. Die Volltextsuche sucht in allen Artikel und zeigt die Treffer in der mittleren Spalte an.

10. Sowohl in der linken als auch in der rechten Spalte (in der Abbildung nur in der rechten Spalte) besteht die Möglichkeit unter den bereits beschriebenen Elementen eine Liste aus sehr kurzen Artikelvorschauen (mit/ohne Bild) anzuzeigen.

6.3 Customization Aspekte

In dieser Arbeit wird die Customization der Web-Anwendung als Crosscutting Concern im Sinne der aspektorientierten Programmierung behandelt. Um Separation of Concerns zu erreichen, wird der Code, der zur Customization beiträgt und in nahezu allen Komponenten der Web-Anwendung verteilt ist, in Aspekte ausgelagert. Die objektorientierten Komponenten der Web-Anwendung beinhalten nur den Code, der zur grundlegenden Funktionalität nötig ist und in der

[11] Der Benutzer hat die Möglichkeit Artikel die ihn interessieren in einen Reisekoffer, ähnlich dem Warenkorb bei einem Webshop, zu geben um sie später leichter wiederzufinden, auszudrucken oder an Freunde zu versenden.

„Bestimmung“ der Komponente liegt, dh. im Controller Servlet werden Parameter aus dem Request validiert und auf die entsprechende View weitergeleitet, in der JSP Seite werden ein CSS Stylesheet und Tag Libraries eingebunden und in den Tag Libraries werden die einzelnen Teile der HTML Seite erzeugt. Wenn die Customization in diesen Komponenten vermischt wäre, müsste das Controller Servlet Informationen zum Kontext extrahieren, die Tag Libraries müssten Fallunterscheidungen zu den verschiedenen Darstellungen auf den unterschiedlichen Endgeräten beinhalten, die Adaptierung des Inhalts wäre starr in den Klassen der Geschäftslogik verankert, usw. Änderungen und Erweiterungen wären dann nur sehr schwer durchzuführen.

Unser Ansatz basiert auf den drei Aspekten *Kontextaspekt*, *Transformationsaspekt* und *Adaptierungsaspekt* um diese Bereiche zu kapseln. Abbildung 6.3 zeigt das Zusammenspiel der Aspekte mit den objektorientierten Komponenten anhand eines, an der UML-Notation angelehnten, Sequenzdiagramms. Die Pfeile welche von den Aspekten ausgehen, zeigen, dass ein Advice an dieser Stelle ausgeführt wird.

6.3.1 Kontextaspekt

Der Kontextaspekt hat die Aufgabe, relevante technische Kontextdaten zu erfassen. Diese Daten werden aus dem Request Header und der IP-Adresse des Benutzers extrahiert und umfassen derzeit das verwendete Endgerät und Browser, die verfügbare Bandbreite, die bevorzugte Sprache und den aktuellen Standort des Benutzers. Der Kontext wird beim Start der Session erfasst und in Anlehnung an die objektorientierten Kontextmodelle 3.1.4 in einem Objekt, dem *ContextBean*, gespeichert.

Die Adaptierung der Weboberfläche an die möglichen Endgeräte findet automatisch in drei Schritten statt. Zuerst werden alle das Gerät betreffenden Informationen gesammelt und ausgewertet. Danach wird das Grundlayout an das Gerät angepasst bzw. ein passendes Layout aus den vorhandenen ausgewählt. Zuletzt werden die anzuzeigenden Artikel in ihrem Format an das gewählte Layout angepasst. Im Folgenden werden die drei Schritte der endgerätespezifischen Adaptierung kurz beschrieben. Eine ausführliche Behandlung dieses Themas findet sich in [Maye06].

Erkennung der Geräteeigenschaften

Der wahrscheinlich wichtigste Teil bei der Adaptierung an verschiedene Endgeräte ist die Erkennung der Eigenschaften und Fähigkeiten des verwendeten Geräts. Bei

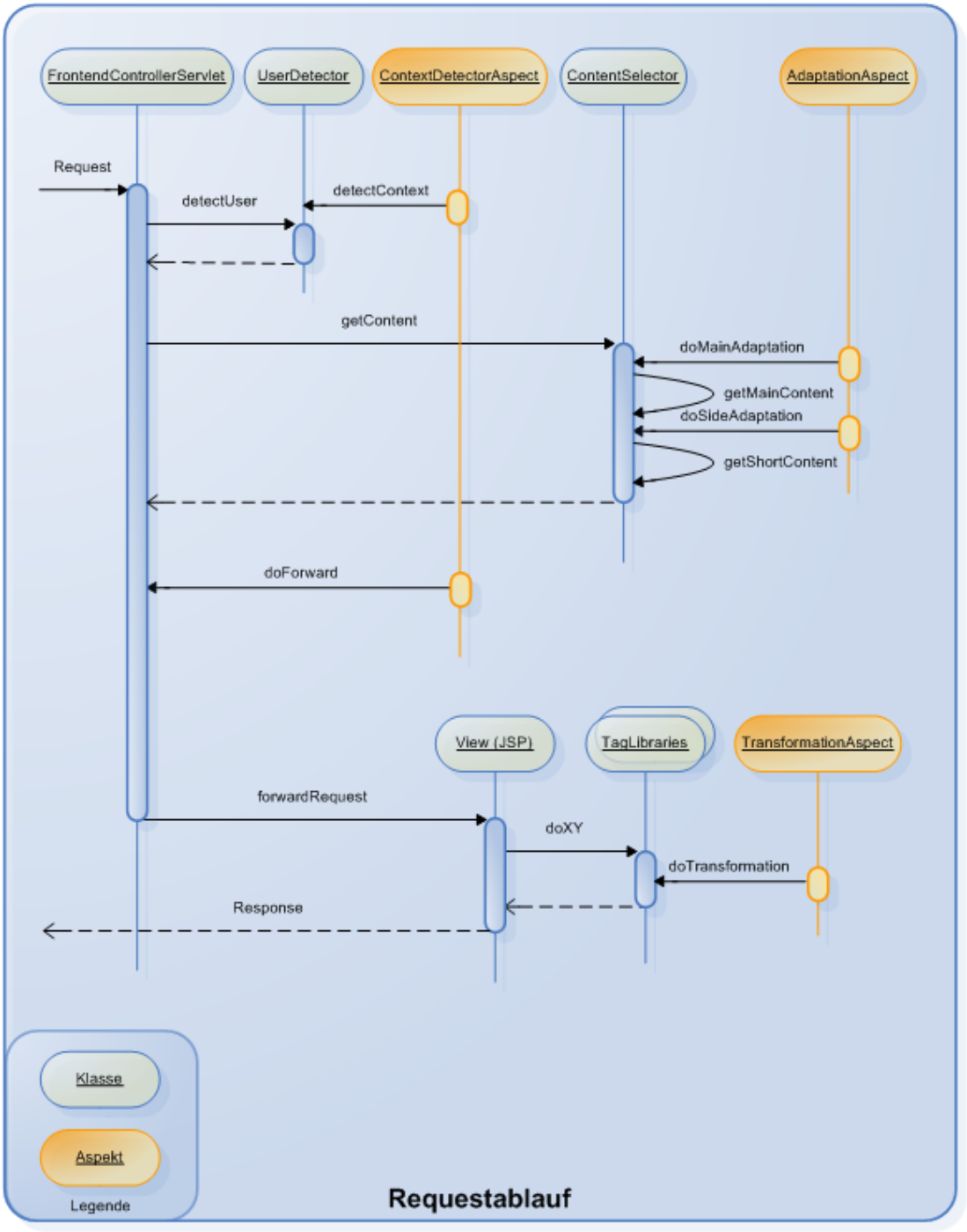

Abbildung 6.3: Requestablauf

dem entwickeltem Forschungsprototyp wurden dazu zwei verschiedene Möglichkeiten implementiert.

Request Header Erweiterungen In Listing 6.1 ist der Request Header eines Pocket Internet Explorers unter Windows CE dargestellt. Bei den Feldern UA-OS, UA-Color usw. handelt es sich um proprietäre Header Erweiterung des Herstellers, aus welchen man die wichtigsten Eigenschaften des Geräts ablesen kann. Sie haben den Vorteil dass sie relativ leicht auszulesen sind und daher einen geringen Implementierungsaufwand benötigen. Der Nachteil besteht darin, dass es bei gerätespezifischen Header-Erweiterungen keine Standards gibt und die Felder daher je nach Hersteller unterschiedlich benannt sein können oder unter dem gleichen Namen andere Werte enthalten können.

Composite Capabilities/Preferences Profiles (CC/PP) Da es bei den Header-Erweiterungen keine einheitlichen Standards gab bzw. bis heute nicht gibt, hat das World Wide Web Consortium (W3C) 2004 den auf RDF[12] basierenden Standard CC/PP [W3C004] verabschiedet. Damit sollte ein einheitlicher Austausch von Benutzerpräferenzen und Geräteeigenschaften zwischen Client und Server ermöglicht werden. Mittlerweile stellen so gut wie alle Hersteller von Mobiltelefonen Profile für ihre Geräte zur Verfügung. Dabei handelt es sich meist um User Agent Profiles (UAProf) [OMA206] welche von der Open Mobile Alliance auf dem CC/PP Standard aufbauend entwickelt wurden, UAProf kann somit als Implementierung von CC/PP verstanden werden. Listing 6.2 zeigt einen Teil eines solchen Profils.

Listing 6.1: Request Header Pocket Internet Explorer

```
1  GET http://localhost:8080/tip/main.do HTTP/1.1
2  Accept: */*
3  UA-OS: Windows CE (POCKET PC) - Version 3.0
4  UA-color: color16
5  UA-pixels: 240x320
6  UA-CPU: ARM SA1110
7  UA-Language: JavaScript
8  Accept-Encoding: gzip, deflate
9  User-Agent: Mozilla/2.0 (compatible; MSIE 3.02; Windows CE;
10 240x320)
```

[12] http://www.w3.org/RDF/

Listing 6.2: Teil eines UAProf-Profils

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:prf="http://www.openmobilealliance.org/tech/profiles/UAPROF/ccppschema
    -20021212#"
xmlns:mms="http://www.wapforum.org/profiles/MMS/ccppschema-20010111#">
   <rdf:Description rdf:ID="Profile">
      <prf:component>
         <rdf:Description rdf:ID="HardwarePlatform">
            <rdf:type rdf:resource="http://www.openmobilealliance.org/tech/
                profiles/UAPROF/ccppschema-20021212#HardwarePlatform"/>
            <prf:BluetoothProfile>
               <rdf:Bag>
                  <rdf:li>Headset Profile</rdf:li>
                  <rdf:li>Handsfree Profile</rdf:li>
                  <rdf:li>Dial-up Networking Profile</rdf:li>
                  <rdf:li>Basic Imaging Profile</rdf:li>
                  <rdf:li>Basic Printing Profile</rdf:li>
                  <rdf:li>Object Push Profile</rdf:li>
                  <rdf:li>File Transfer Profile</rdf:li>
                  <rdf:li>Human Interface Device Profile</rdf:li>
                  <rdf:li>General Access Profile</rdf:li>
                  <rdf:li>Service Discovery Profile</rdf:li>
                  <rdf:li>Serial Port Profile</rdf:li>
                  <rdf:li>General Object Exchange Profile</rdf:li>
               </rdf:Bag>
            </prf:BluetoothProfile>
            <prf:BitsPerPixel>18</prf:BitsPerPixel>
            <prf:ColorCapable>Yes</prf:ColorCapable>
            <prf:CPU>ARM</prf:CPU>
            <prf:ImageCapable>Yes</prf:ImageCapable>
            <prf:InputCharSet>
               <rdf:Bag>
                  <rdf:li>ISO-8859-1</rdf:li>
                  <rdf:li>ISO-10646-UCS-2</rdf:li>
                  <rdf:li>US-ASCII</rdf:li>
                  <rdf:li>UTF-8</rdf:li>
               </rdf:Bag>
            </prf:InputCharSet>
            <prf:Keyboard>PhoneKeyPad</prf:Keyboard>
            <prf:Model>N70-1</prf:Model>
            <prf:NumberOfSoftKeys>2</prf:NumberOfSoftKeys>
            <prf:OutputCharSet>
               <rdf:Bag>
                  <rdf:li>ISO-8859-1</rdf:li>
                  <rdf:li>ISO-10646-UCS-2</rdf:li>
                  <rdf:li>US-ASCII</rdf:li>
                  <rdf:li>UTF-8</rdf:li>
               </rdf:Bag>
            </prf:OutputCharSet>
            <prf:PixelAspectRatio>1x1</prf:PixelAspectRatio>
            <prf:PointingResolution>Pixel</prf:PointingResolution>
            <prf:ScreenSize>176x208</prf:ScreenSize>
            <prf:ScreenSizeChar>15x6</prf:ScreenSizeChar>
            <prf:StandardFontProportional>Yes</prf:StandardFontProportional>
            <prf:SoundOutputCapable>Yes</prf:SoundOutputCapable>
            <prf:TextInputCapable>Yes</prf:TextInputCapable>
            <prf:Vendor>Nokia</prf:Vendor>
            <prf:VoiceInputCapable>Yes</prf:VoiceInputCapable>
```

```
56            </rdf:Description>
57          </prf:component>
58      ....
```

Aus den Header-Erweiterungen und den CC/PP Profilen stehen nun genügend Informationen zur für die Adaptierung zur Verfügung. Nachdem sie bei der Instanzierung der Session eingelesen wurden, stehen sie allen Anwendungsmodulen zur Verfügung.

Der Joinpoint befindet sich im Controller Servlet beim Aufruf des *UserDetectors*, der auch für die herkömmliche Web-Anwendung aufgerufen wird. Der UserDetector prüft, ob ein Cookie für die automatische Anmeldung vorhanden ist. Der Advice benötigt Zugriff auf den Request und auf das ContextBean. Dieses Ereignis wird durch den nachstehenden Pointcut `detectContext` selektiert. Durch `args(ctx, request)` werden die der Methode `detectUser(ContextBean, HttpServletRequest)` übergebenen Parameter für den Advice sichtbar gemacht.

Listing 6.3: Pointcut detectContext

```
1  public pointcut detectContext(ContextBean ctx, HttpServletRequest request) :
2      call(ContextBean UserDetector.detectUser(ContextBean, HttpServletRequest))
3      && (args(ctx, request));
```

Bei diesem Pointcut wird ein Around-Advice durchgeführt. Dadurch kann der Rückgabewert der Methode `detectUser()` nach der Ausführung der Methode, aber noch vor der tatsächlichen Rückgabe des Objekts und des Kontrollflusses, verändert werden. Es werden Daten zum Endgerät, der Bandbreite und dem Standort extrahiert und zusätzlich im `ContextBean` gespeichert. Wenn der Benutzer automatisch angemeldet werden kann, wird die im Profil gespeicherte Sprache für die Darstellung der Web-Anwendung verwendet. Wenn kein Cookie existiert, wird zusätzlich die Sprache aus dem Request Header gelesen. Passt keine Sprache aus dem Request Header mit den im System verfügbaren überein, wird die Web-Anwendung in Deutsch angezeigt.

Listing 6.4: Advice detectContext

```
ContextBean around(ContextBean ctx, HttpServletRequest request) :
  detectContext(ctx, request) {

    ctx = proceed(ctx, request);

    if (ctx.getUser() == null) {
        long langId = doDetectLanguageFromHeader(request);
        ctx.setLanguage(langId);
    }
    ctx.setDeviceBean(doParseProfile(request)); //CC/PP Infos Parsen
    ctx.setDevice(doDetectDevice(ctx.getDb()));
    String ip = request.getRemoteAddr();
    ctx.setIp(ip);
    ctx.setBandwith(doDetectBandwith(ip));
    ctx.setLocation(doDetectLocation(ip));

    return ctx;
}
```

Abhängig vom erkannten Endgerät wird die Web-Anwendung in einem passenden Layout dargestellt. Für jedes Layout existiert eine eigene View, das ist eine JSP Seite, die HTML Code für das Design und Tag Libraries für dynamische Inhalte enthält. Das Controller Servlet muss den Request auf die richtige View weiterleiten. Um aber im Controller nicht die Concerns zu vermischen, wird die Entscheidung, welche View die richtige ist, im Kontextaspekt getroffen.

Der Pointcut `doForward` aktiviert dazu beim Aufruf der Methode `getRequestDispatcher()`, sofern sie innerhalb der Methode `doGet()` in der Klasse `FrontendControllerServlet` aufgerufen wird, einen Around-Advice.

Listing 6.5: Pointcut doForward

```
public pointcut doForward(HttpServletRequest request) :
    call(RequestDispatcher HttpServletRequest.getRequestDispatcher(*))
    && withincode(protected void FrontendControllerServlet.doGet(..))
    && (target(request));
```

Der Advice prüft, welcher Kategorie (groß/klein) das Gerät zugeordnet ist, und liefert ein passendes `RequestDispatcher` Objekt zurück.

Listing 6.6: Advice doForward

```
RequestDispatcher around(HttpServletRequest request) : doForward(request) {
    RequestDispatcher dispatcher;
    ContextBean ctx = (ContextBean)request.getSession().getAttribute("ctxBean");
    if (ctx.getDevice() == GlobalValues.Device.MIDI) {
        dispatcher = request.getRequestDispatcher("index_mobile.jsp");
    }
    else {
        dispatcher = request.getRequestDispatcher("index_front.jsp");
    }
    return dispatcher;
}
```

6.3.2 Adaptierungsaspekt

Der Adaptierungsaspekt ist das Bindeglied zwischen den Regeln der Customization bezüglich Inhalt und der Klasse `ContentSelector`, die für Artikelabfrage und -aufbereitung, Navigation, sowie Breadcrumb Menü zuständig ist.

Für die Auswahl und Reihung der Artikel im Forschungsprototypen wurde, basierend auf den theoretischen Grundlagen aus den Kapiteln 2, 3 und 5, im Rahmen dieser Arbeit ein eigenes Customizationsystem entwickelt, welches in Abbildung 6.4 dargestellt ist und in Folge kurz erläutert wird.

Regelsystembeschreibung

Bei der Entwicklung der ubiquitären Web-Anwendung war es das Ziel, möglichst viele Kontextfaktoren mit einzubeziehen und darüber hinaus das System so zu gestalten, dass es leicht erweiterbar und dass eine Konfiguration des Einflusses der Kontextfaktoren auf die Adaptierung „von Außen“ möglich ist, also ohne Eingriff in den Programmcode.

Mit diesem System wird die Artikelauswahl für die gesamte Web-Anwendung getroffen und bietet damit jedem Benutzer jederzeit ein speziell abgestimmtes Angebot, wobei natürlich ein Navigieren durch alle Artikel über die Kategorie- oder Ortsnavigation trotzdem weiterhin möglich ist.

Dabei wird prinzipiell zwischen zwei unterschiedlichen Kontextfaktorgruppen hinsichtlich ihres Einflusses auf die Adaptierung unterschieden, nämlich den *Knock-Out Faktoren* und den *Reihungsfaktoren*.

Knock-Out Faktoren Diese Kontextfaktoren entscheiden, welche Artikel überhaupt für eine Anzeige in Frage kommen. Es wird über diese Faktoren also eine Selektion von, auf den Kontext des Benutzers, passende Artikeln durchgeführt. Zu dieser Faktorgruppe zählen im Forschungsprototypen der *Ort*, die *Saison*, die *Sprache* und der *Benutzertyp*. Am Beispiel der Saison wird schon deutlich, warum dies ein Knock-Out Faktor ist: Im Winter wird ein Benutzer mit großer Wahrscheinlichkeit nicht an Sommerangeboten interessiert sein, darum soll er diese überflüssige Information auch gar nicht angezeigt bekommen. Eine ausführlichere Erklärung in dieser Gruppe der Kontextfaktor bedarf der *Benutzertyp*. Dieser gibt eine grobe Interessensausrichtung und Einstufung des Benutzers an. Mögliche Benutzertypen sind hierbei „Jung&Aktiv“, „Familien mit Kindern“, „Romantik&Wellness“ oder Ähnliche. Diesen Typen wiederum sind passende Kategorien von Artikeln zugeordnet, wie zum Beispiel Jung&Aktiv die Kategorien Sport, Nightlife und Szene, Action und Fun umfasst. Die Erfassung und Einstufung eines

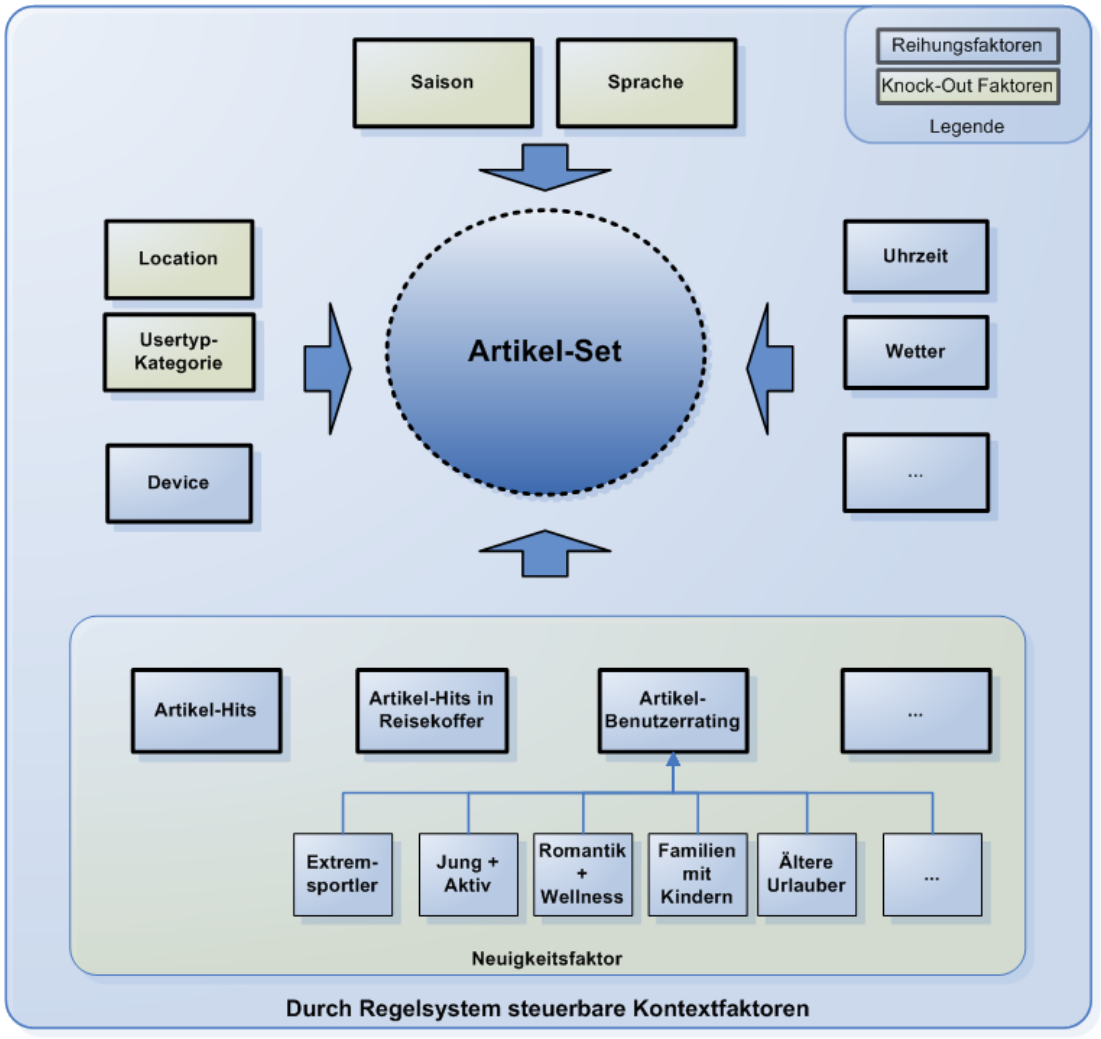

Abbildung 6.4: Schema Adaptierungsvorgang der Artikel basierend auf sozialen, natürlichen und logischen Kontext

Benutzers zu einem Benutzertypen erfolgt halbautomatisch, kann also vom Benutzer selbst eingegeben oder über das System automatisch erfasst werden.

Abhängig von der Einstellung der Gewichtung dieser Faktoren werden diese jedoch wiederum automatisch schrittweise zurückgenommen, wenn eine Berücksichtigung aller Faktoren keine oder nur eine unzureichende Treffermenge an Artikeln ergibt.

Reihungsfaktoren Das durch die Knock-Out Faktoren vorgewählte Artikel-Set wird über diese Faktoren, wie der Name schon sagt, gereiht. So werden zum Beispiel bei schönem Wetter Outdoor Events nach vor, oder Artikel, welche zur Zeit nicht aktuell sind, weil diese an Events mit Öffnungszeiten gebunden sind, nach hinten gereiht. Ein besonderer Reihungsfaktor ist der *Artikelbenchmark*.

Artikelbenchmark Hierbei werden Trackingdaten zu den einzelnen Artikeln über ein Benchmarksystem, welches die erfassten Daten normalisiert und gewichtet, bewertet. Mit in diese Bewertung fließt auch ein „Neuigkeitsbonus“ für neu erstellte Artikel ein, damit diese prominent platziert und nicht unter Umständen vom Regelsystem übergangen werden. Der daraus erhaltene Benchmarkwert pro Artikel dient unter Berücksichtigung der anderen Reihungsfaktoren, je nach Konfiguration des Regelsystems, für die Reihung des vorgewählten Artikel-Sets.

Technische Umsetzung

Das oben beschriebene System zur Adaptierung der Artikel besteht im Prinzip aus drei Komponenten. Einerseits aus einer erweiterbaren Gruppe an *Regelklassen*, einer *Konfigurationsdatei* für die Definition der Reihenfolge und Gewichtung der anzuwendenden Regeln und einem *Adaptierungsaspekt*.

Regelklasse Je eine Regelklasse definiert für einen Kontextfaktor die, für die Adaptierung benötigten, HQL[13] *FROM, WHERE und ORDER BY*-Fragmente. Diese Regelklassen werden vom Adaptierungsaspekt über *Reflection*[14] aufgerufen, was zur Folge hat, dass neue Regeln ganz einfach, auch zur Runtime, hinzugefügt werden können. Einzige Bedingung hierfür ist, dass diese in der Konfigurationsdatei enthalten sein müssen, da ansonst der Aspekt nichts von der neuen Regel weiß.

[13] Hibernate Query Language
[14] http://java.sun.com/docs/books/tutorial/reflect/index.html

Konfigurationsdatei In dieser XML[15]-Datei sind die Regelklassen der Kontextfaktoren mit optionalen Gewichtungen enthalten. Über diese Datei erfährt der Adaptierungsaspekt, welche Regelklassen in welcher Reihenfolge und zu welcher Gewichtung anzuwenden sind.

Hier ein Beispiel für eine einfache Konfigurationsdatei des Regelsystems, wobei diejenige Regelklasse mit dem niedrigsten Wert am höchsten gewichtet ist:

Listing 6.7: Beispiel für eine Konfigurationsdatei des Regelsystems

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
3   <properties>
4     <comment>this xml file contains the rule-classes</comment>
5     <entry key="LanguageRule">0</entry>
6     <entry key="LocationRule">1</entry>
7     <entry key="CategoryRule">2</entry>
8   </properties>
```

Adaptierungsaspekt Die in der Konfigurationsdatei aufgeführten Regelklassen werden von dem Adaptierungsaspekt ausgeführt und die erhaltenen HQL-Fragmente zu einem HQL-Statement zusammengeführt. Dieses wird in die entsprechende Stelle im Contentselector eingefügt und von diesem ausgeführt. Daraufhin wird vom Aspekt geprüft, ob das erhaltene Artikel-Set ausreichend groß ist um den Hauptcontent und die Side-Artikel zu befüllen. Ist dies nicht der Fall wird ein neues HQL-Statement erstellt, welches die am schwächsten gewichtete Regel verwirft und ein neues Artikel-Set liefert.

Aufgrund dieses modularen Aufbaues kann das Regelsystem einfach erweitert werden, in dem man eine neue Regelklasse entsprechend den Vorgaben einfügt und diese auch in der Konfigurationsdatei berücksichtigt. Am Quellcode an sich sind somit keine Änderungen nötig.

Im Adaptierungsaspekt existieren zwei einfache Pointcuts, `doMainAdaptation()` und `doSideAdaptation()` mit jeweiligem Before-Advice, die die entsprechenden Datenbank-Abfragen für die große Artikelliste in der mittleren Spalte und die kleinen seitlichen Artikellisten zusammen bauen.

Der Pointcut selektiert die Ausführung der Methode `getMainContent()` in der Klasse `ContentSelector`. Zusätzlich zu den Parametern der Methode wird eine Referenz auf das ContentSelector Objekt dem Advice übergeben.

[15] Extensible Markup Language

Listing 6.8: Pointcut doMainAdaptation

```
1  public pointcut doMainAdaptation(ContentSelector cs,
2    ContentBean cb, ContextBean ctx) :
3        execution(private * tip.main.action.ContentSelector
4        .getMainContent(ContentBean, ContextBean))
5      && (this(cs))
6      && (args(cb, ctx));
```

Der Before-Advice lädt mittels Dynamic Class Loading die Regelklassen, die in der Konfigurationsdatei angegeben sind. Die einzelnen Regelklassen sind von der abstrakten Klasse `AbstractRule` abgeleitet und implementieren somit alle die Methoden `getHqlFrom()`, `getHqlWhere()` und `getHqlOrder()`. Eine konkrete Instanz von AbstractRule ist zB `LanguageRule` (siehe Listing 6.9), die als WHERE Klausel eine Einschränkung der Artikelsprache auf die vom Benutzer gewählte Sprache erzeugt.

Listing 6.9: LanguageRule

```
1  public class LanguageRule extends AbstractRule {
2
3    public void generateHql(ContentBean cb, ContextBean ub) {
4      hqlWhere = "articles.language='" + GlobalValues.Language.valueOf(ub.
           getLanguage()).ordinal() + "'";
5    }
6    public String getHqlFrom() {
7      return hqlFrom;
8    }
9    public String getHqlWhere() {
10     return hqlWhere;
11   }
12   public String getHqlOrder() {
13     return order;
14   }
15 }
```

Der so zusammen gestellte Query String wird durch `cs.hqlMain = hqlFrom + hqlWhere + hqlOrder;` an die Variable `hqlMain` im ContentSelector Objekt weitergegeben.

Listing 6.10: Advice doMainAdaptation

```
1  before(ContentSelector cs, ContentBean cb, ContextBean ctx) :
2    doMainAdaptation(cs, cb, ctx) {
3
4      hqlFrom = "SELECT␣DISTINCT␣articles␣FROM␣Article␣articles␣";
5      hqlWhere = "␣WHERE␣";
6      hqlOrder = "";
7
8      // load rule-config
9      Properties prop = loadProperties();
10     Enumeration enumProp = prop.propertyNames();
11
12     while (enumProp.hasMoreElements()) {
13       try {
```

```
14          Object ruleClass = Class.forName("tip.customization.adaptation.rule." +
                (String)enumProp.nextElement()).newInstance();
15
16          // invoke methods for generating query string...
17
18        }
19        catch (Exception ex) { }
20      }
21
22      // set query string in ContentSelector
23      cs.hqlMain = hqlFrom + hqlWhere + hqlOrder;
24    }
```

Die Customization der seitlichen Artikellisten erfolgt analog dazu.

6.3.3 Transformationsaspekt

Im Transformationsaspekt werden eventuell notwendige Änderungen zur Anpassung des Layouts für mobile Endgeräte vorgenommen. Der Transformationsaspekt „befreit“ die Klassen der Tag Libraries von Entscheidungen zur Endgeräteanpassung.

Layoutauswahl

Nachdem die Eigenschaften des Geräts bekannt sind wird ein dazu passendes Layout ausgewählt. Das Layout gibt die grobe Seitenstruktur vor und besteht aus JSP Seiten die wiederum Tags enthalten durch die dann die eigentlichen Inhalte erstellt werden. Für den entwickelten Forschungsprototypen wurden derzeit zwei Templates bereitgestellt. Eines für alle herkömmlichen Browser und Bildschirme mit einer Auflösung von mindestens 800px in der Breite, und eines für mobile Endgeräte oder Bildschirmauflösungen unter 800px Breite. Theoretisch wäre es möglich beliebig viel Templates zur Verfügung zu stellen, der praktische Nutzen sei jedoch dahingestellt. Die Kombination aus Templates und Tags macht es möglich das komplette Layout zu ändern ohne in die Programmlogik der Anwendung eingreifen zu müssen. Denn das Template gibt die Struktur und das Design vor, und an den Stellen, an denen dann das Menü erscheint, wird der dementsprechende Tag eingebunden, welcher wiederum ausschließlich designunabhängiges HTML erzeugt. Die Entscheidung für das entsprechende Template wird im Controller aufgrund der Informationen über das Endgerät und darauf angewendeten Regeln (zB Bildschirmbreite $<$ 800px: Template für PDAs) getroffen.

Transformation

Die Transformation ist der letzte Schritt in der Adaptierung für verschiedene Endgeräte. Dabei werden die aus der Datenbank geladenen Inhalte für das jeweilige Endgerät aufbereitet. Dies ist notwendig, da das Backend, aus Gründen der besseren Benutzbarkeit, nur die Möglichkeit bietet Inhalte in einer Formatierung (für herkömmliche Browser) zu speichern. Dabei steht im Backend ein WYSIWYG-HTML-Editor zur Verfügung, mit dem der Benutzer die angelegten Artikel komfortabel und ohne HTML-Kenntnisse editieren kann. Die Formatierung die der Autor des Artikels vornimmt, wie zum Beispiel große Tabellen oder Bilder, können aber nicht immer auf allen Endgeräten gleich angezeigt werden. Daher ist es notwendig diese in Abhängigkeit der Geräteeigenschaften zu transformieren. Nachdem die Inhalte aus der Datenbank geladen wurden wird nun überprüft, ob sie Elemente enthalten, die für die Darstellung auf dem verwendeten Endgerät problematisch sind. Wird ein solches Element gefunden, zB ein Bild, wird es vom Transformator durch ein kleineres in der passenden Auflösung ersetzt. Der Transformator besitzt auch die Fähigkeit Texte logisch zu kürzen. Dies kommt vor allem bei der Artikelübersicht zum Tragen wo nur die Schlagtexte zu den ersten 3-5 Artikeln angezeigt werden. Der Schlagtext, der die Einleitung des Artikels darstellt, besteht zwar in der Regel nur aus 100 bis 200 Wörtern, was auf einem herkömmlichen Bildschirm kein Problem darstellt. Für ein Gerät mit einer Bildschirmhöhe von zum Beispiel nur 320 px ist dies, vor allem bei einer Übersicht, jedoch viel zu viel. Der Transformator kürzt den Schlagtext daher je nach Möglichkeit auf 20-50 Wörter ab. Abbildung 6.5 zeigt das Ergebenis der endgerätspezifischen Adaption auf einem PDA.

In der Artikelliste sollen auf kleinen Endgeräten keine Contentparts angezeigt werden, die nur allgemeine Informationen beinhalten, wie zB die zusammengefasste Empfehlung aus den abgegebenen Ratings, für welchen Urlaubstyp ein Artikel geeignet ist. Der Pointcut `doListSystemCPTransformation` adressiert den Aufruf der Methode `CPTag.performContentparts()`, die das Rendering aller Contentparts für die Artikelliste veranlasst.

Listing 6.11: Pointcut doListSystemCPTransformation

```
1 public pointcut doListSystemCPTransformation(ArticleListTag alt) :
2     call (String CPTag.performContentparts(ArticleFrame, ..))
3   && (this(alt));
```

Da auf mobilen Devices keiner der Contentparts in der Liste angezeigt werden soll, bricht der Around-Advice entweder mit einem Leerstring das Rendern der

Abbildung 6.5: Screenshot der für einen PDA adaptierten Anwendung im Sommerdesign

Contentparts ab, oder wenn es sich nicht um ein kleines Endgerät handelt, wird durch den Aufruf von `proceed()` der Vorgang ordnungsgemäß durchgeführt.

Listing 6.12: Advice doListSystemCPTransformation

```
1   String around(ArticleListTag alt) :
2       doListSystemCPTransformation(alt) {
3
4     Device d = alt.ctx.getDevice();
5
6     if (d == GlobalValues.Device.MIDI)
7       return "";
8     else
9       return proceed(alt);
10  }
```

In der Detailansicht eines Artikels werden alle abgegebenen Ratings inkl. Kommentar aufgelistet. Für kleine Endgeräte soll stattdessen ein Durchschnittsrating angezeigt werden. Der Pointcut `doRating` adressiert die Ausführung der Rating Rendering Methode innerhalb des Kontrollflusses des `ArticleTag`.

Listing 6.13: Pointcut doRatingCPTransformation

```
1   public pointcut doRatingCPTransformation(ArticleTag at, ContextBean ctx) :
2     execution (String CPRating.performFullOutput(ContextBean))
3     && cflowbelow (execution (public int ArticleTag.doStartTag()) && (this(at)))
4     && (args(ctx));
```

Der Around-Advice bewirkt bei mobilen Endgeräten die Anzeige des durchschnittlichen Ratings statt der gesamten Liste.

Listing 6.14: Advice doRatingCPTransformation

```
1   String around(ArticleTag at, ContextBean ctx) :
2       doRatingCPTransformation(at, ctx) {
3     Device d = ctx.getDevice();
4
5     if (d == GlobalValues.Device.MIDI) {
6
7       Object[][] articles = at.cb.getMainArticles();
8       Article a = (Article) articles[0][0];
9
10      RatingBean rb = new RatingBean();
11      rb.loadAvgRating(a.getArticleFrame().getId());
12
13      RatingAction ra = new RatingAction();
14      return ra.getAvgRatingCategories(rb, ctx);
15    }
16    else
17      return proceed(at, ctx);
18  }
```

6.4 Kritische Würdigung

Durch die drei Aspekte *Kontextaspekt*, *Adaptierungsaspekt* und *Transformationsaspekt* können die Bereiche der Customization sauber gekapselt werden. Die großen Vorteile sind, dass der Code nicht vermischt ist, jede Komponente nur die Funktionalität beinhaltet, die auch in ihr vermutet wird und Änderungen an zentraler Stelle durchzuführen sind.

Die Transformation ist insofern flexibel, da auch das Hinzufügen eines neuen Contentparttyps, der noch nicht unterstützt wird, wie zum Beispiel Audio- oder Video-Dateien, einfach möglich ist. Für neue Contentparts muss nur eine Entität in der Datenbank und eine Hibernate-Mapping Klasse angelegt werden. Diese Mapping Klasse implementiert das Interface `IContentpart` mit den Methoden `performListOutput()` und `performFullOutput()` für das Rendering in der Artikelliste bzw. in der Vollansicht. Diese beiden Methoden dienen als Joinpoint. Eine Adaptierung für verschiedene Endgeräte erfolgt analog zu der Transformation des Ratings (siehe Listing 6.14).

Die Adaptierung bezüglich Inhalt erlaubt ebenfalls größtmögliche Flexibilität. Durch Hinzufügen neuer Regeln, beziehungsweise durch Ändern der Reihenfolge der Regeln in der Konfigurationsdatei, wird die Anpassung beeinflusst. Werden die Regeln in Zukunft so komplex, dass eine sequentielle Abarbeitung nicht mehr effizient ist und ein RETE-basierter Baumansatz nötig ist, kann lediglich durch Änderungen in den Advices `doMainAdaptation` und `doSideAdaptation` eine externe *Rule Engine*, wie zB JBoss Rules[16], statt unserer Eigenentwicklung eingesetzt werden. Aktuell sind einige grundlegende Regeln des vorgestellten Regelsystems implementiert, wodurch die Funktion und Anwendbarkeit des Systems gezeigt werden konnte. Hier gibt es noch einigen Raum für Erweiterungen der Regelklassenbasis.

Schwieriger gestaltet sich das Hinzufügen neuer Kontextfaktoren. Hier genügt es nicht, nur den Advice zu ändern, da die neuen Kontextfaktoren im `ContextBean` gespeichert werden und in weiterer Folge für die Adaptierung genutzt werden. Somit ist eine Erweiterung des ContextBeans und die Erstellung neuer Regeln notwendig. Die Erweiterung des ContextBeans um Attribute für die neuen Kontextfaktoren kann entweder durch den Aspekt instrumentiert werden oder direkt in der Klasse erfolgen.

Änderungen bei der Extraktion der bestehenden Kontextfaktoren sind aber relativ einfach möglich. Hier sind nur Änderungen im Kontextaspekt nötig. So kann beispielsweise die Lokalisierung des Benutzers, die derzeit aufgrund der IP-

[16] http://labs.jboss.com/portal/jbossrules

Adresse erfolgt, auch für die Verwendung von GSM oder GPS Informationen ausgebaut werden.

Ein Punkt der nicht berücksichtigt wurde, ist der Umgang mit Kontextdaten in Bezug auf den Datenschutz. Natürlich werden die Daten der Benutzer nur im Rahmen der Anwendung verwendet und auch nicht weitergegeben, jedoch hat der Benutzer auch keine Möglichkeit, die Verarbeitung seiner Daten, oder Teile davon, innerhalb der Anwendung zu kontrollieren oder Daten teilweise für die Verarbeitung zu sperren.

Kapitel 7

Conclusio

Ubiquitäre Web-Anwendungen treten in der einen oder anderen Form immer mehr in den Vordergrund. Waren es zu Beginn erste Pilotprojekte, so beinhalten mittlerweile schon zahlreiche Web-Anwendungen Customizationfunkionalität. Hierbei steht jedoch meist der Benutzer an sich im Mittelpunkt wobei im Speziellen sein sozialer Kontext erfasst und ausgewertet wird. Teilweise werden auch noch Faktoren aus dem technischen Kontext berücksichtigt, wie zum Beispiel die Anpassung an das jeweilige Endgerät. Darüber hinaus werden aktuell noch sehr viele Kontextinformationen einfach nicht erfasst und ausgewertet und das, obwohl die Technologien dazu schon verfügbar und durchaus im Produktivbetrieb einsetzbar sind.

Das Ziel bei der Entwicklung des Tourismusinformationssystems war es zu zeigen, was an Customization bei Web-Anwendungen heute bereits möglich ist, dass mit Hilfe der aspektorientierten Programmierung der Kontext flexibel und von den anderen Programmkomponenten getrennt erfasst werden kann und genauso die Adaptierung mit dem Regelsystem gekapselt implementierbar ist. Dadurch wird einerseits die Wartung erleichtert und andererseits können leichter Kontextfaktoren hinzugefügt und das Regelsystem erweitert bzw. angepasst werden.

7.1 Erfahrungsbericht

Für eine gute ubiquitäre Web-Anwendung ist ein passendes Kontextmodell von entscheidender Bedeutung. Hierbei wurde bei dem entwickelten System auf ein objektorientiertes Kontextmodell gesetzt. Grund dafür war einerseits die bessere Kombinierbarkeit mit AOP und andererseits das einfach zu verstehende Grundmodell.

Die Hauptschwierigkeit bei der Umsetzung des Regelsystems auf Basis des Kontextmodells liegt darin, dieses so zu gestalten, dass die Regelerstellung auch von

Nicht-Experten verstanden und durchgeführt werden kann.

Für die Erfassung des Kontext und auch in der Einbindung des Regelsystems hat AOP seine Stärken gezeigt. Das Regelsystem und auch die einzelnen Adaptierungen sind hierbei vom restlichen Code getrennt bzw. können, ohne Eingriff in den Ursprungscode, relativ einfach erweitert und geändert werden. Auch ist es Dank der Verwendung von AOP möglich, im „Worst-Case“ das Regelsystem komplett auszutauschen, ohne die komplette Anwendung überarbeiten zu müssen.

Aber wo Licht ist, ist bekanntlich auch Schatten. AOP ist noch ein relativ junges Forschungsgebiet, weshalb auch die Toolunterstützung und Dokumentation teilweise nur eingeschränkt vorhanden sind. Dadurch kam es vor Allem bei dem AJDT[1] Plugin für Eclipse zu Problemen, da dieses nicht immer, wie erwartet und laut Dokumentation versprochen, funktioniert hat. Auch die fehlenden Debugging Möglichkeiten von Pointcut-Definition erleichtern die Arbeit nicht.

7.2 Ausblick

Ubiquitäre (Web-)Anwendungen sind bereits ein fixer Bestandteil der IT-Welt und werden auch immer mehr an Bedeutung gewinnen, nur gehen sie im aktuellen Hype um „Web 2.0[2]“ etwas unter - zu unrecht, wie ich behaupten möchte, denn diese Anwendungen bieten viel mehr als Web 2.0. Dieses Schlagwort steht eigentlich nur für die geänderte Interaktionsweise der Benutzer mit den Web-Anwendungen. Grob gesagt stellen bei Web 2.0 Anwendungen die Benutzer sämtliche Inhalte, der Betreiber der Seite hingegen stellt nur die Infrastruktur zur Verfügung (typische Beispiele sind Wikis[3] und Blogs[4]). Dabei kommt teilweise massiv Javascript (zB bei Ajax[5]) zum Einsatz, was dazu führt, dass die Anwendungen nur auf aktuellen Browsern mit aktiviertem Javascript benutzen werden können - ältere Versionen von Browsern bzw. mobile Endgeräte mit älter Versionen von Javascript bleiben bei solchen Anwendungen sehr oft ausgesperrt. Dadurch und durch die geänderte Interaktionsweise, bei welcher der Benutzer im Mittelpunkt steht, liegt es auf der Hand, die Prinzipien von ubiquitären Web-Anwendungen auch hier anzuwenden. Durch das anytime/anywhere/anymedia Paradigma können für diese Anwendungen entscheidende Verbesserungen erzielt werden und dann kann man, wenn die jetzigen so genannten Web 2.0 Anwendungen zu ubiquitären Anwendungen werden, vielleicht wirklich von Web 2.0

[1] Eclipse AspectJ Development Tools
[2] http://en.wikipedia.org/wiki/Web_2
[3] http://de.wikipedia.org/wiki/Wiki
[4] http://en.wikipedia.org/wiki/Blog
[5] Asynchronous JavaScript and XML

sprechen! An sonst bleibt es wohl nur ein Schlagwort unter vielen anderen.

Ein großes Problem besteht generell für diese neuen Generationen von Web-Anwendungen: der Datenschutz. Bei Web 2.0 Anwendungen, wie MySpace[6], openBC/XING[7] oder friendster[8], geben die Benutzer *freiwillig*, oft auch einfach aus Unkenntnis und ohne Berücksichtigung der möglichen Folgen, sämtliche Daten zu ihrer Person & Umgebung preis. Dies wird bei ubiquitären Anwendungen noch durch automatische Kontexterfassung und Kontextabstraktion ergänzt. Dabei werden sehr viele Daten der Benutzer erfasst und sind in Folge meist für andere Benutzer zugänglich oder werden zumindest von der Anwendung für unterschiedliche Zwecke ausgewertet.

Hier müssen von allen Seiten in Zukunft entsprechende Bemühungen unternommen werden, damit der Benutzer auch im WWW über seine Daten und die Verarbeitung dieser bestimmen kann, und somit der Datenschutz weiterhin gewährleistet wird und nicht zu einer Farce verkommt.

Die Entwicklung des „Ubiquitous Computing“ wird weitergehen und vielleicht sind wir irgendwann einmal dort, wo Mark Weiser diesen Begriff eigentlich gesehen hat, jedoch unter Berücksichtigung der Privatsphäre von uns Allen, damit wir nicht zum viel zitierten „Gläsernen Menschen“ werden ...

> [...]Ubiquitous computing names the third wave in computing, just now beginning. First were mainframes, each shared by lots of people. Now we are in the personal computing era, person and machine staring uneasily at each other across the desktop. Next comes ubiquitous computing, or the age of calm technology, when technology recedes into the background of our lives.[...] *Mark Weiser*[9]

[6] http://www.myspace.com
[7] http://www.xing.com
[8] http://www.friendster.com
[9] http://www-sul.stanford.edu/weiser/Ubiq.html

Anhang A

Sourcecode Statistik

Dateien	132
Klassen	128
Methoden	1448
Durchschnittliche Anzahl an Methoden pro Klasse	11
Maximale Anzahl an Methoden in einer Klasse	48
Minimale Anzahl an Methoden in einer Klasse	1
Codezeilen Total	17624
Durchschnittliche Anzahl an Codezeilen pro Datei	135
Maximale Anzahl an Codezeilen pro Datei	883
Minimale Anzahl an Codezeilen pro Datei	7

Tabelle A.1: Statistik Java Dateien

Dateien	16
Zeilen Total	1313
Durchschnittliche Anzahl an Zeilen pro Datein	82
Maximale Anzahl an Zeilen pro Datei	200
Minimale Anzahl an Zeilen pro Datei	7

Tabelle A.2: Statistik JSP Dateien

Dateien	54
Zeilen Total	2613
Durchschnittliche Anzahl an Zeilen pro Datein	48
Maximale Anzahl an Zeilen pro Datei	186
Minimale Anzahl an Zeilen pro Datei	14

Tabelle A.3: Statistik XML Dateien

Kapitel und Autoren

Kapitel	Petra Brosch	Rudolf Mayer	Arnold Weissensteiner
1			X
2.1	X		
2.2.1			X
2.2.2		X	
2.2.3			X
2.2.4			X
3			X
4	X		
5.1		X	
5.2	X		
5.3			X
5.4		X	
5.4.2			X
5.5		X	
6.1	X		
6.2		X	
6.3.1	X	X	
6.3.2	X		X
6.3.3	X	X	
7			X

Tabelle A.4: Kapitel und Autoren

Anhang B

Screenshots Frontend

Abbildung B.1: Startseite

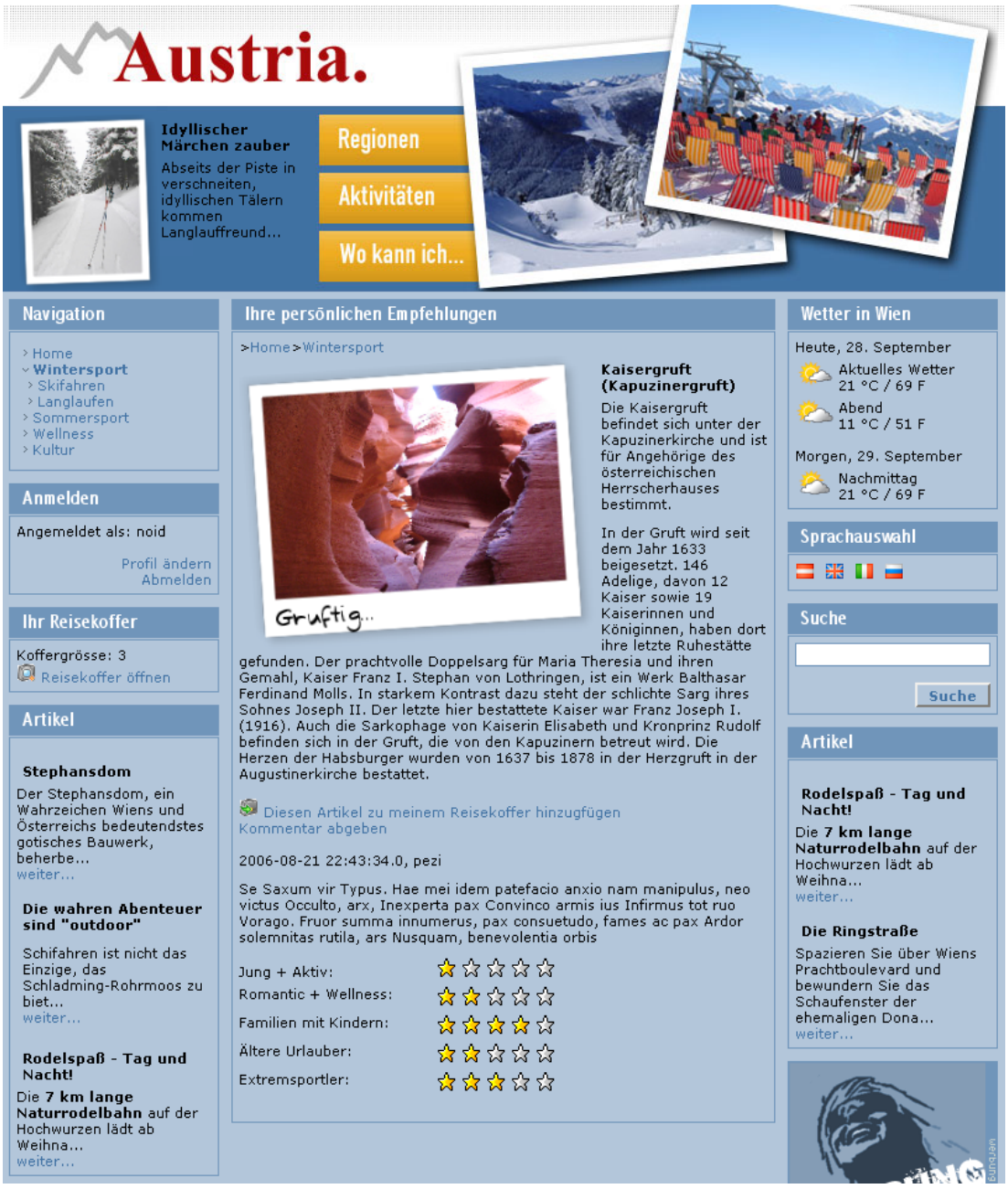

Abbildung B.2: Artikelansicht

Abbildung B.3: Artikelbewertung

Abbildung B.4: Reisekoffer ansehen

Anhang C

Screenshots Backend

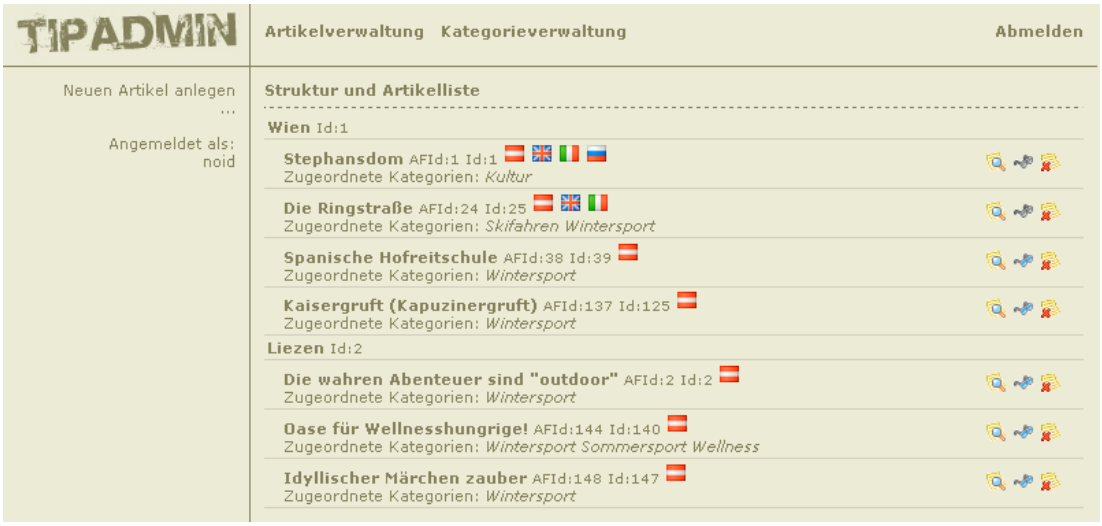

Abbildung C.1: Artikelzentrale

Abbildung C.2: Artikelübersicht

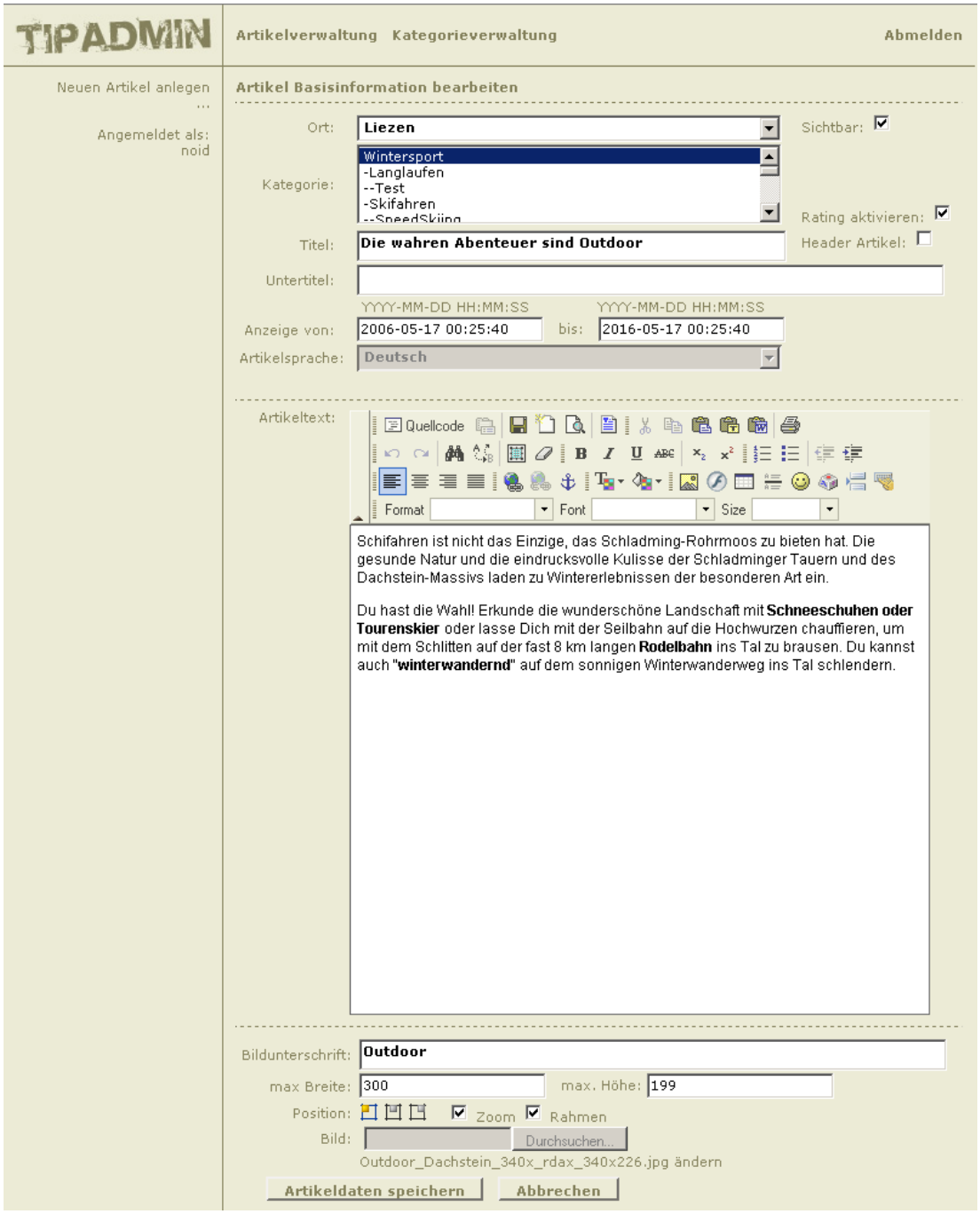

Abbildung C.3: Artikel bearbeiten/anlegen

Abbildung C.4: Contentpart Text bearbeiten/anlegen

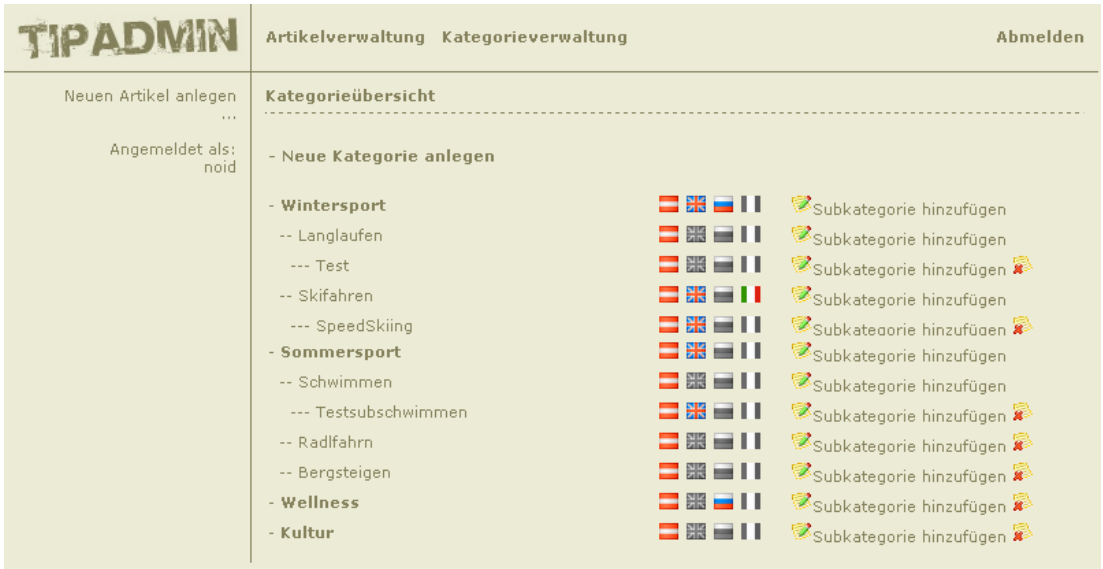

Abbildung C.5: Kategorien bearbeiten/anlegen

Tabellenverzeichnis

Listings

Abbildungsverzeichnis

Literaturverzeichnis

[AvZe97] Avery, C. und Zeckhauser, R.: *Recommender Systems for Evaluating Computer Messages.* Communications of the ACM (CACM), 40(3), März 1997.

[BaDR06] Baldauf, M., Dustdar, S. und Rosenberg, F.: *A Survey on Context-Aware Systems.* International Journal of Ad Hoc and Ubiquitous Computing, forthcoming, 2006.

[BFK+00] Badrinath, B., Fox, A., Kleinrock, L., Popek, G., Reiher, P. und Satyanarayanan, M.: *A Conceptual Framework for Network and Client Adaptation.* IEEE Mobile Networks and Applications, 5(4):221–231, 2000.

[Böhm06] Böhm, O.: *Aspektorientierte Programmierung mit AspectJ 5*, Band 1. dpunkt.verlag, Heidelberg, 2006.

[BrMa02] Brusilovsky, P. und Maybury, T.: *From adaptive hypermedia to adaptive Web.* Communications of the ACM (CACM), 45(5):31–33, Dezember 2002. Special Issue on the Adaptive Web.

[Bros06] Brosch, P.: *Ubiquitäre Web-Anwendungen - Realisierung von Adaptierung mit Hilfe aspektorientierter Programmierung.* Diplomarbeit, Technische Universität Wien, 2006.

[Chen76] Chen, P.-S.: *The entity-relationship model - toward a unified view of data.* ACM Trans. Database Syst., 1(1):9–36, 1976.

[ChFJ03] Chen, H., Finin, T. und Joshi, A.: *An ontology for context-aware pervasive computing environments.* Knowl. Eng. Rev., 18(3):197–207, 2003.

[ChFJ04] Chen, H., Finin, T. und Joshi, A.: *Semantic Web in the Context Broker Architecture.* In: *Proceedings of the Second IEEE international Conference on Pervasive Computing and Communications (Percom'04)*, 2004.

[ChKo00] Chen, G. und Kotz, D: *A Survey of Context-Aware Mobile Computing Research*. Technischer Bericht TR2000-381, Dept. of Computer Science, Dartmouth College, November 2000.

[ChMD99] Cheverst, K., Mitchell, K. und Davies, N.: *Design of an object model for a context sensitive tourist GUIDE*. Computers and Graphics, 23(6):883–891, 1999.

[Demo06] Demolsky, M.: *State of the Art in Java Enterprise Web Application Development*. Diplomarbeit, Technische Universität Wien, 2006.

[DeSA01] Dey, A.K., Salber, D. und Abowd, G.D.: *A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications*. anchor article of a special issue on Context-Aware Computing Human-Computer Interaction (HCI) Journal, 16(2-4):97–166, 2001.

[EnMo88] Engelmore, R. und Morgan, T.: *Blackboard systems.* Addison-Wesley, 1988.

[EPS+01] Espinoza, F., Persson, P., Sandin, A., Nyström, H., Cacciatore, E. und Bylund, M.: *GeoNotes: Social and Navigational Aspects of Location-Based Information Systems*. In: *Proceedings of the 3rd International Conference on Ubiquitous Computing, Atlanta, Georgia, USA*, Seiten 2–17. Springer Berlin / Heidelberg, 2001.

[FiKS97] Fink, J., Kobsa, A. und Schreck, J.: *Personalized Hypermedia Information Provision Through Adaptive and Adaptable System Features: User Modelling, Privacy and Security Issues*. In: *Proceedings of the Fourth International Conference on Intelligence in Services and Networks*, Seiten 459–467. Springer Berlin / Heidelberg, 1997.

[FSK+02] Finkelstein, A., Savigni, A., Kappel, G., Retschitzegger, W., Schwinger, W. und Feichtner, C.: *Ubiquitous Web Application Development - A Framework for Understanding*. In: *Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics (SCI), Orlando, Florida*, Seiten 431–438, 2002.

[GoWJ84] Good, M. D., Wixon, D. R. und Jones, S. J.: *Building a User-Derived Interface*. Communications of the ACM (CACM), 27(10), Oktober 1984.

[HeIR03] Henricksen, K., Indulska, J. und Rakotonirainy, A.: *Generating Context Management Infrastructure from Context Models.* In: *4th International Conference on Mobile Data Management (MDM), Industrial Track Proceedings*, Seiten 1–6, Melbourne, January 2003.

[HKRS02] Hitz, M., Kappel, G., Retschitzegger, W. und Schwinger, W.: *Ein UML-basiertes Framework zur Modellierung ubiquitärer Web-Anwendungen.* 2002.

[HSP+03] Hofer, T., Schwinger, W., Pichler, M., Leonhartsberger, G., Altmann, J. und Retschitzegger, W.: *Context-Awareness on Mobile Devices - the Hydrogen Approach.* In: *HICSS '03: Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03) - Track 9*, Seite 292.1, Washington, DC, USA, 2003. IEEE Computer Society.

[JBos06] *JBoss Rules Documentation Library.* http://labs.jboss.com/portal/jbossrules/docs, 2006.

[KaRe98] Kappel, G. und Retschitzegger, W.: *The TriGS Active Object-Oriented Database System - An Overview.* SIGMOD Rec., 27(3):36–41, 1998.

[KaRS00] Kappel, G., Retschitzegger, W. und Schwinger, W.: *Modeling Customizable Web Applications - A Requirement's Perspective.* 2000.

[KDJ+04] Kerer, C., Dustdar, S., Jazayeri, M., Gomes, D., Szego, A. und Caja, J. A. B.: *Presence-aware infrastructure using web services and RFID technologies.* In: *Proceedings of the 2nd European Workshop on ObjectOrientation and Web Services, Oslo, Norway.* Springer LNCS, 2004.

[KLM+97] Kiczales, G., Lamping, J., Menhdhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M. und Irwin, J.: *Aspect-Oriented Programming.* In: Akşit, Mehmet und Matsuoka, Satoshi (Herausgeber): *Proceedings European Conference on Object-Oriented Programming*, Band 1241, Seiten 220–242. Springer-Verlag, Berlin, Heidelberg, and New York, 1997.

[Korp05] Korpipää, P.: *Blackboard-based software framework and tool for mobile device context awareness. Doctoral thesis.* Nummer 579. VTT Electronics, Espoo, 2005.

[KPRR04] Kappel, G., Pröll, B., Reich, S. und Retschitzegger, W.: *Web Engineering*, Band 1. dpunkt.verlag, Heidelberg, 2004.

[KPRS03] Kappel, G., Pröll, B., Retschitzegger, W. und Schwinger, W.: *Customisation for Ubiquitous Web Applications - A Comparison of Approaches*. Int. Journal of Web Engineering and Technology (IJWET), 1:79–111, 2003.

[KRK+02] Kappel, G., Retschitzegger, W., Kimmerstorfer, E., Pröll, B., Schwinger, W. und Hofer, T.: *Towards a Generic Customisation Model for Ubiquitous Web Applications*. 2002.

[LoTe92] Loeb, S. und Terry, D.: *Information Filtering*. Communications of the ACM (CACM), 35(12), Dezember 1992.

[Maye06] Mayer, R.: *Ubiquitäre Web-Anwendungen*. Diplomarbeit, Technische Universität Wien, 2006.

[McBu97] McCarthy, J. und Buvač, S.: *Formalizing Context (Expanded Notes)*. In: Buvač, Sasa und Iwańska, Łucia (Herausgeber): *Working Papers of the AAAI Fall Symposium on Context in Knowledge Representation and Natural Language*, Seiten 99–135, Menlo Park, California, 1997. American Association for Artificial Intelligence.

[OMA206] *User Agent Profile, Approved Version 2.0*. http://member.openmobilealliance.org/ftp/Public_documents/BAC/UAPROF/Permanent_documents/OMA-TS-UAProf-V2_0-20060206-A.zip, 2006.

[OWL004] *OWL Web Ontology Language*. http://www.w3.org/TR/owl-features/, 2004.

[SaAb99] Salber, D. Dey, A.K. und Abowd, G.D.: *The Context Toolkit: Aiding the Development of Context-Enabled Applications*. In: *Proceedings of the ACM CHI, Pittsburgh, PA.*, Seiten 434–441, 1999.

[ShBe05] Sheng, Q.Z. und Benatallah, B.: *ContextUML: A UML-Based Modeling Language for Model-Driven Development of Context-Aware Web Services Development*. In: *ICMB '05: Proceedings of the International Conference on Mobile Business (ICMB'05)*, Seiten 206–212, Washington, DC, USA, 2005. IEEE Computer Society.

[SiCo06] Singh, A. und Conway, M.: *Survey of Context aware Frameworks - Analysis and Criticism.* http://its.unc.edu/teap/tap/core/caf_review.pdf, 2006.

[SSK+06] Schauerhuber, A., Schwinger, W., Kapsammer, E., Retschitzegger, W., Wimmer, M. und Kappel, G.: *A Survey on Aspect-Oriented Modeling.* 2006.

[Stei02] Stein, D.: *An Aspect-Oriented Design Model Based on AspectJ and UML.* Diplomarbeit, Universität Essen, 2002.

[StLi04] Strang, T. und Linnhoff-Popien, C.: *A Context Modeling Survey.* In: *First International Workshop on Advanced Context Modelling, Reasoning And Management at UbiComp 2004, Nottingham, England, September 7*, 2004.

[UML206] *Unified Modeling Language.* http://www.uml.org/, 2006.

[UsGr96] Uschold, M. und Grüninger, M.: *Ontologies: principles, methods, and applications.* Knowledge Engineering Review, 11(2):93–155, 1996.

[W3C003] *Device Independence Principles.* http://www.w3.org/TR/di-princ/, September 2003. W3C Note.

[W3C004] *Composite Capabilities/Preference Profiles (CC/PP): Structure and Vocabularies 1.0.* http://www.w3.org/TR/CCPP-struct-vocab/, Januar 2004. W3C Recommendation.

[WaSc01] Want, B.N. und Schilit, B.N.: *Expanding the Horizons of Location-Aware Computing (Guest Editor's Introduction).* Computer, 34(8):31–33, August 2001.

[Weis91] Weiser, M.: *The Computer for the 21st Century.* Scientific American, Seite 265, 1991.

[Weis06] Weissensteiner, A.: *Ubiquitäre Web-Anwendungen - Modellierung und Implementierung von Context Informationen.* Diplomarbeit, Technische Universität Wien, 2006.

[Wiki06] *Business rules engine.* http://en.wikipedia.org/wiki/Rule_engine#IT_use, 2006.

[Wund06a] Wunderlich, Lars: *Java Rules Engines - Entwicklung regelbasierter Systeme - Des Entwicklers Traum.* Javamagazin, 10, September 2006.

[Wund06b] Wunderlich, Lars: *Java Rules Engines - Entwicklung von regelbasierten Systemen.* entwickler.press, 2006.