Ein UML-basiertes Framework zur Modellierung ubiquitärer Web-Anwendungen

Die Autoren

Martin Hitz
Gerti Kappel
Werner Retschitzegger
Wieland Schwinger

Dipl. Ing. Dr. Martin Hitz,
Institut für Informatik-Systeme,
Universität Klagenfurt,
Universitätsstraße 65–67,
A-9020 Klagenfurt,
E-Mail: martin.hitz@uni-klu.ac.at;
Dipl. Ing. Mag. Dr. Gerti Kappel,
Mag. Dr. Werner Retschitzegger,
Institut für Softwaretechnik und
Interaktive Systeme,
Technische Universität Wien,
Favoritenstraße 9–11/188,
A-1040 Wien, E-Mail:
{gerti | werner}@bi.tuwien.ac.at;
Mag. Dr. Wieland Schwinger,
Software Competence Center
Hagenberg (SCCH), Softwarepark
Hagenberg, Hauptstraße 99,
A-4232 Hagenberg,
E-Mail: wieland.schwinger@scch.at

1 Motivation

Softwaresysteme, deren Zustand durch Web-basierte Benutzerinteraktionen beeinflusst werden („Web-Anwendungen" [Cona99]), machen einen Großteil der heute entwickelten Individualsoftware aus. Es wird jedoch immer häufiger kritisiert, dass in diesem bedeutenden Segment softwareingenieurmäßige Methoden der Anwendungsentwicklung zum Teil nicht vorhanden sind und zum Teil nicht konsequent eingesetzt werden [BaLa01]. Diese Kritik wiegt umso schwerer, als auf Grund der Aussicht, zu jeder Zeit an jedem Ort mit jedem Medium eine Web-Anwendung benutzen zu können, sowohl die Entwicklung als auch die Verwendung so genannter *ubiquitärer Web-Anwendungen* immer weiter vorangetrieben wird, was bei dem unterstellten Mangel an Methodik eine neue Auflage der Softwarekrise zur Folge haben könnte.

Ziel dieser Arbeit ist es, dieser Kritik partiell zu begegnen und ein UML-basiertes Framework vorzustellen, das als Ausgangspunkt zur Modellierung ubiquitärer Web-Anwendungen herangezogen werden kann. Zu diesem Zweck wird im restlichen Teil dieses Abschnitts zunächst auf die Charakteristika und Anforderungen ubiquitärer Web-Anwendungen eingegangen und erörtert, dass die dynamische Anpassung der jeweiligen Web-Anwendung an den aktuellen Anwendungskontext eine notwendige Voraussetzung darstellt, um Ubiquität voll ausschöpfen zu können. In Abschnitt 2 wird die Architektur eines UML-basierten Frameworks zur *Anpassungsmodellierung* vorgestellt. Die wesentlichen Komponenten, nämlich das *Kontextmodell* und das *Anpassungsregelmodell*, werden in den Abschnitten 3 bzw. 4 behandelt. Alle dabei angeführten Beispiele beziehen sich auf die Domäne eines Konferenzorganisationssystems. Abschnitt 5 schließt mit einem Ausblick auf weitere Entwicklungen in diesem Gebiet.

Der in diesem Beitrag beschriebene Ansatz wird im Rahmen des EU-Projekts *UWA* (*Ubiquitous Web Applications,* IST-2000-25131) und des nationalen Forschungsprojektes *CustWeb* (*Customizable Web Applications,* Software Competence Center – SCCH Hagenberg) entwickelt. Im Rahmen dieser Projekte wird zurzeit evaluiert, inwieweit das vorgeschlagene Framework den tatsächlich auftretenden Anforderungen aus der Praxis gerecht wird. Erste Erfahrungsberichte können erst nach Abschluss dieser Projekte (Ende 2002) vorgelegt werden.

1.1 Ubiquitäre Web-Anwendungen

Ein Rückblick auf die kurze Geschichte des World Wide Web zeigt, dass drei Generationen von Web-Anwendungen unterschieden werden können, die sich vor allem durch die verwendete Technologie und die zur Verfügung gestellten Dienste auszeichnen [Cona99]. Die *erste Generation* von Web-Anwendungen ist charakterisiert durch reine Informationsgewinnung (read-only applications), wobei diese Web-An-

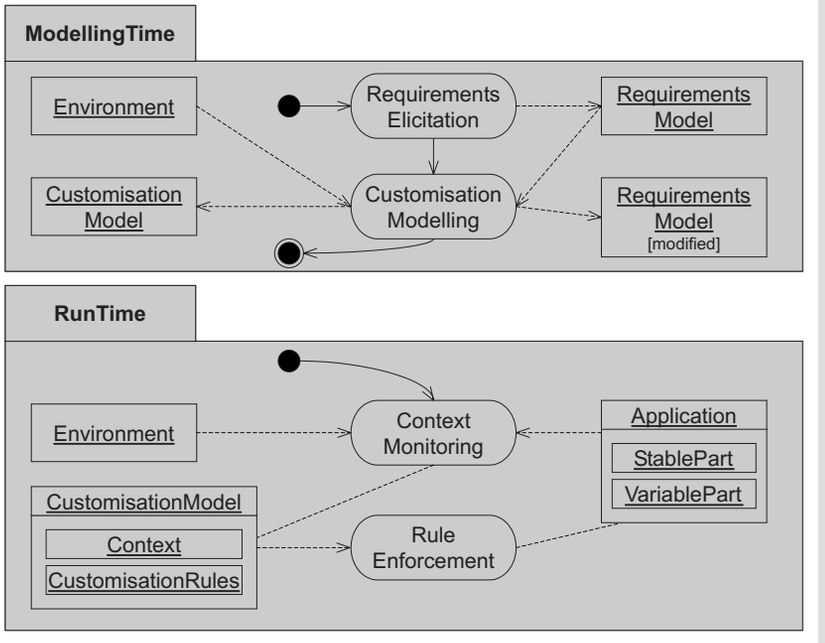

Bild 1 Prinzipien der Anpassungsmodellierung [siehe Anhang UML: Punkte i und x]

wendungen von anonymen Benutzern verwendet wurden. Die *zweite Generation* umfasst bereits „vollwertige" Softwareanwendungen, die verschiedene Dienste zur Verfügung stellen, zur Realisierung von Benutzertransaktionen zustandsbasiert arbeiten und die verwendeten Daten in der Regel in einer zugrundeliegenden Datenbank ablegen [EhKa97]. Nicht zuletzt auf Grund des Einsatzes dieser Anwendungen im Bereich des E-Commerce ist die Personalisierung von Diensten zu einem wesentlichen Charakteristikum dieser Anwendungen geworden [Kobs01]. Die zurzeit aktuelle *dritte Generation* von Web-Anwendungen sind so genannte ubiquitäre Web-Anwendungen.

Eine *ubiquitäre Web-Anwendung* stellt personalisierte Dienste zu jeder Zeit an jedem Ort für jedes Medium zur Verfügung, womit ein allgegenwärtiger Zugriff auf diese Web-Anwendung möglich wird. *Ubiquitous Computing* wurde erstmals von Mark Weiser in den frühen 90er Jahren definiert [Weis91]. Er ging dabei von der Durchdringung unserer Umwelt (Alltagsgegenstände, Fahrzeuge, Gebäude, etc.) mit Computertechnologie aus, die somit für den Benutzer transparent erscheinen sollte. Diese noch immer visionäre Form von Ubiquität wird in diesem Beitrag außer Acht gelassen. Der Fokus liegt vielmehr auf vorhandener Technologie, womit der Zugriff auf Web-Anwendungen nicht nur von einem stationären Gerät, sondern auch von unterschiedlichen mobilen Endgeräten aus erfolgen kann.

Die große Herausforderung bei der Entwicklung einer ubiquitären Web-Anwendung ist es, Dienste personalisiert für jeden Benutzer zeit-, orts- und mediengerecht zur Verfügung zu stellen [Fisc01]. Eine wesentliche Voraussetzung dafür ist, dass eine ubiquitäre Web-Anwendung den *Kontext* kennt, in dem sie benutzt wird, um auf Änderungen dieses Kontextes adäquat reagieren zu können. In diesem Sinn wird in der vorliegenden Arbeit unter dem Begriff ubiquitäre Web-Anwendung eine Web-Anwendung verstanden, für welche die dynamische, d. h. zur Laufzeit stattfindende Anpassung an sich ändernde Kontexte ein zentrales Merkmal darstellt.

1.2 Warum UML?

Trotz der aktuellen Bedeutung ubiquitärer Web-Anwendungen für Bereiche wie E-Commerce und M-Commerce gibt es nur wenige Modellierungsansätze zur Entwicklung von Web-Anwendungen; Ubiquität und die dazu notwendige *Anpassungsmodellierung* werden fast gänzlich außer Acht gelassen (für einen Überblick siehe [KaRe00]). In dieser Arbeit wird daher ein UML-basiertes Framework zur Modellierung ubiquitärer Web-Anwendungen vorgestellt, wobei speziell der Aspekt der Anpassungsmodellierung behandelt wird (für eine weiterführende detaillierte Diskussion siehe [Schw01]). Es wird davon ausgegangen, dass für jene Teile einer ubiquitären Web-Anwendung, die vom Kontext unabhängig sind, bereits entsprechende Analyse- und Entwurfsmodelle in UML vorliegen und als Basis für die vorgestellten Konzepte zur Anpassungsmodellierung dienen können.

UML (unified modeling language [HiKa99] [RuJa98]) ist der objektorientierte Modellierungsstandard der Softwareentwicklung. Es liegt daher nahe, UML auch zur Modellierung von Web-Anwendungen einzusetzen. Dabei auftretende spezifische Anforderungen an die Modellierungssprache (z. B. der hypermediale Aspekt) können durch die integrierten Erweiterungsmechanismen von UML erfüllt werden, wobei bereits verschiedene Vorschläge für entsprechende Erweiterungen vorliegen [BaKo99] [Cona99] [KaRe01]. Die benützten UML-Modellelemente werden im Anhang (i–x) kurz erläutert, wobei in den Legenden der Abbildungen jeweils jene Abschnitte des Anhangs angegeben werden, in denen die in der Abbildung (erstmals) verwendeten Modellelemente beschrieben sind.

2 Anpassungsmodellierung

Das im Folgenden vorgestellte Framework umfasst eine Reihe von UML-Modellen, die sowohl den Aspekt der Personalisierung als auch Fragen der Anpassung an Orts-, Zeit- und Medienspezifika der Web-Anwendung abdecken. *Bild 1* zeigt eine Übersicht über die grundlegenden Zusammenhänge im Prozess der Anpassungsmodellierung.

Zur *Modellierungszeit* wird nach der üblichen Anforderungsanalyse das Anpassungsmodell (CustomisationModel) erzeugt, und zwar auf Basis des Anforderungsmodells der Anwendung (RequirementsModel) einerseits und der erwarteten Umgebung der Anwendung (Environment)

andererseits. Dabei ist allerdings damit zu rechnen, dass die Anpassungsmodellierung auch in Änderungen des Anforderungsmodells resultiert (RequirementsModel [modified]).

Zur *Laufzeit* stellt das Kontextmodell (ContextModel) als Teil des Anpassungsmodells detaillierte Informationen über die Umgebung der Anwendung (Environment) und über die Anwendung selbst (Application) zur Verfügung. Änderungen des Kontextes lösen über einen regelbasierten Mechanismus (CustomisationRuleModel) die eigentliche Anpassung der Anwendung aus. Jene Teile der Anwendung, die dem Anpassungsmechanismus unterworfen sind, werden als variabler Teil (VariablePart) der Anwendung bezeichnet, während die kontextunabhängige Funktionalität der Anwendung dem so genannten stabilen Teil (StablePart) zugeordnet wird.

Durch die möglichst frühzeitige, explizite Trennung der anpassungsrelevanten Aspekte der Anwendung (ContextModel, CustomizationRuleModel, VariablePart) vom stabilen Teil wird der dynamischen Natur einer ubiquitären Web-Anwendung in geeigneter Weise Rechnung getragen und damit Wiederverwendbarkeit und Änderungslokalität unterstützt [Abow99].

Diesen Überlegungen folgend, wird ein *Modellierungsframework* vorgeschlagen, das für den Entwurf einer ubiquitären Web-Anwendung geeignete vordefinierte Modellelemente bereit stellt, die im Bedarfsfall durch Unterklassenbildung verfeinert und ergänzt werden können.

Bild 2 zeigt die Grobstruktur dieses Modellierungsframeworks mit dessen Abhängigkeiten von anderen Modellen einer ubiquitären Web-Anwendung.

Das Kontextmodell (ContextModel) besteht aus einem physischen (PhysicalContextModel) und einem logischen Teilmodell (LogicalContextModel), die sich im Wesentlichen durch den Abstraktionsgrad der Kontextinformationen unterscheiden, die durch die vordefinierten Klassen der einzelnen Teilmodelle beschrieben werden. Als dritte Komponente ist das Regelmodell zur Definition spezifischer Anpassungen vorgesehen (CustomisationRuleModel). Die «access»-Abhängigkeiten zu anderen Modellen einer ubiquitären Web-Anwendung sollen das Zusammenspiel mit der Anforderungsdefinition und dem eigentlichen Anwendungsmodell widerspiegeln.

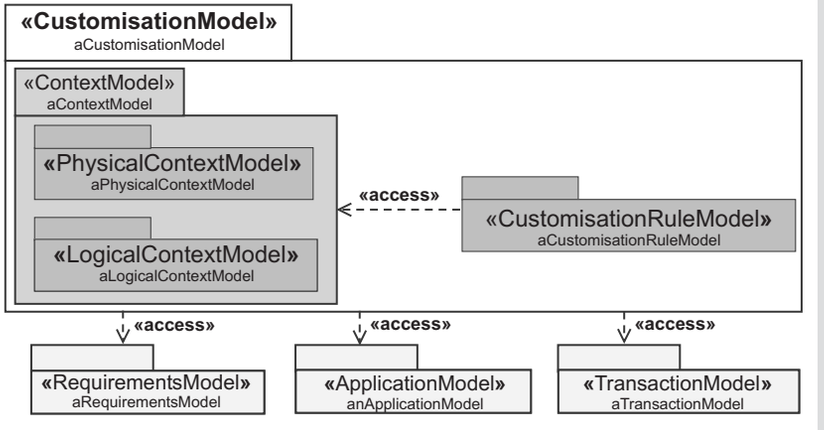

Bild 2 Generisches Modellierungsframework [UML: i, ii]

In den folgenden Abschnitten werden die einzelnen Komponenten des vorgeschlagenen Frameworks ausführlich erörtert.

3 Kontextmodell

In der Literatur über Anpassungsmodellierung werden die folgenden Aspekte des Kontexts einer Anwendung als berücksichtigungswürdig betrachtet:

Benutzer: Charakteristika und Vorlieben einzelner Benutzer und Benutzergruppen;

Endgeräte und Netzwerk: Annahmen über die technische Ausstattung des Endgeräts bzw. über dessen Anbindung an den Server und die Netzwerkeigenschaften (z. B. Bildschirmgröße, Speicherausstattung, installierte Software, Bandbreite etc. [OpSp99]);

Lokation: Angaben über den Aufenthaltsort des Benutzers, sowohl in physischer Hinsicht (z. B. Adresse) als auch in logischer Hinsicht (z. B. Unterscheidung zwischen Arbeitsplatz und Wohnung) [KuBl99];

Zeit: Auch der Zeitpunkt der Nutzung eines Services kann einen wesentlichen Einfluss auf die Services einer ubiquitären Web-Anwendung haben, wobei wiederum

Kernpunkte für das Management

Für die Modellierung von Web-Anwendungen, die sich dynamisch an den Kontext ihrer Anwender anpassen, wird ein objektorientiertes Framework vorgeschlagen. Damit wird die derzeit bestehende methodische Lücke bei der Entwicklung derartiger Anwendungen wenigstens teilweise geschlossen. Der Vorschlag basiert auf folgenden Schlüsselkonzepten:

- Trennung von kontextunabhängigen und kontextabhängigen Teilen der Anwendung
- Separate Modellierung von zeit-, orts-, benutzer-, geräte- und netzwerkspezischer Profilinformation
- Einsatz eines Regelmodells auf Basis von Ereignissen, Bedingungen und Aktionen zur Überwachung von Kontextänderungen und Auslösung entsprechender Reaktionen

Stichworte: ubiquitäre Web-Anwendung, objektorientiertes Modellierungsframework, Unified Modeling Language (UML), Event/Condition/Action-Regeln (ECA-Regeln)

physische (z. B. Tageszeit, Zeitzone) als auch logische Aspekte (z. B. Unterscheidung zwischen Arbeitszeit und Freizeit) eine Rolle spielen.

Für das vorgeschlagene Modellierungsframework sei der Begriff *Kontext* definiert als die Beschreibung bestimmter Eigenschaften der Umgebung der Anwendung als auch einzelner Aspekte der Anwendung selbst, auf die sich die Anpassungsmodellierung bezieht. Wie bereits erwähnt, werden diese Eigenschaften in Abhängigkeit vom Abstraktionsniveau, auf das sie sich beziehen, entweder dem *physischen* oder dem *logischen Kontextmodell* zugeordnet.

3.1 Physisches Kontextmodell

Eigenschaften aus dem physischen Kontextmodell entsprechen einem eher niederen Abstraktionsniveau und können meist als objektiv messbar aufgefasst werden. Sie werden entsprechend dem dynamischen Charakter der Umgebung wie auch des Anwendungszustands laufend aktualisiert, wobei das Messen und Aktualisieren der Eigenschaften des physischen Kontextmodells außerhalb des Rahmens unseres Modellierungsframeworks liegen. Dementsprechend betrachten wir diese Eigenschaften auch als von der ubiquitären Web-Anwendung selbst nicht modifizierbar. Sie sollen lediglich eine geeignete Informationsbasis für die Anpassungsmodellierung darstellen. *Bild 3* zeigt das physische Kontextmodell als Klassendiagramm in UML, in dem die betrachteten Kontexteigenschaften als Unterklassen von PhysicalContextProperty modelliert sind. Zukünftigen Anforderungen kann damit durch Angabe weiterer Unterklassen auf einfache und Framework-konsistente Weise entsprochen werden. In Anlehnung an [ScBe99] wird physischer Kontext in die drei Teilbereiche natürlicher, technischer und sozialer Kontext partitioniert.

Der *natürliche Kontext* (NaturalContext) umfasst Lokations- und Zeitinformation. *Lokationsinformation* (Location) dient mobilen und positionsabhängigen Diensten und beschreibt den Standort, von dem aus auf eine Anwendung zugegriffen wird, wobei der Lokationskontext i. A. von so genannten *Location Servern* in Form von Zellenbezeichnern (cellID) zur Verfügung gestellt wird. Die Kontexteigenschaft *Zeit* (Time) erlaubt die Anpassung der Anwendung an zeitbezogene Nebenbedingungen wie Öffnungszeiten von Einrichtungen oder Fahrplänen.

Der *technische Kontext* (TechnicalContext) beinhaltet Hard- und Softwareeigenschaften des Endgerätes (UserAgent), Eigenschaften des Netzwerks (Network) sowie den Zustand der Anwendung selbst (ApplicationState), soweit er für Zwecke der Anpassungsmodellierung von Relevanz ist.

Der *soziale Kontext* (SocialContext) schließlich beschreibt den Benutzer (User) der ubiquitären Web-Anwendung im Sinne einer Personalisierung der Anwendung. Der technische Identifikationsmechanismus hängt stark vom eingesetzten Endgerät ab; bei Mobiltelefonen könnte die Telefonnummer, bei PCs die (statische) IP-Adresse oder eine explizite Benutzerkennung herangezogen werden. Im physischen Kontextmodell wird unabhängig davon von einem abstrahierten Attribut userID ausgegangen.

Der tatsächliche physische Kontext (PhysicalContext) ergibt sich in *Bild 3* als Aggregation der erwähnten Eigenschaften, wobei diese Aggregation als Tupel mit je einer Instanz der jeweiligen Kontext-Unterklasse aufgefasst werden kann.

Die Klasse Session stellt die eigentliche „Ankerklasse" des physischen Kontextmodells dar, sodass im Sinne einer zustandsbasierten Web-Anwendung für jede Session ein eigener Kontext verwaltet wird. Da davon auszugehen ist, dass sich dieser Kontext im Laufe einer Session ändert, und es sich für die Anpassungsmodellierung manchmal als sinnvoll erweist, nicht nur die augenblickliche, sondern auch die historische Kontextinformation zu betrachten (z. B. zur Ermittlung der mittleren Bandbreite bei sich laufend ändernden Bandbreitewerten oder zur Ermittlung der zurückgelegten Wegstrecke aus den vergangenen Positionsangaben), wird diese historische Information im physischen Kontextmodell explizit verwaltet (History). Einem gegebenen Zeitstempel time der Session wird dabei genau ein physischer Kontext zugeordnet. Der History-Mechanismus kann im Rahmen von *proaktiven Anpassungen* auch zur Verwaltung der Vorhersagen über zukünftige Kontexte benützt werden, sofern geeignete Vorhersagemodelle für die interessierenden Eigenschaften existieren.

Sollte die Verfügbarkeit der Ausprägungen einer Kontexteigenschaft nicht dauerhaft gewährleistet sein (Wert liegt nicht vor bzw. ist veraltet – vgl. das Attribut validityPeriod der Basisklasse PhysicalContextProperty), sind entweder Standardwerte oder Mechanismen zur expliziten Ermittlung eines geeigneten Wertes durch den Benutzer vorzusehen. Die Anwendung sollte jedenfalls robust genug sein, um im Extremfall auch ohne physische Kontextinformation operieren zu können [BaFo00]. Analoges gilt für Informationen aus dem im Folgenden besprochenen logischen Kontextmodell.

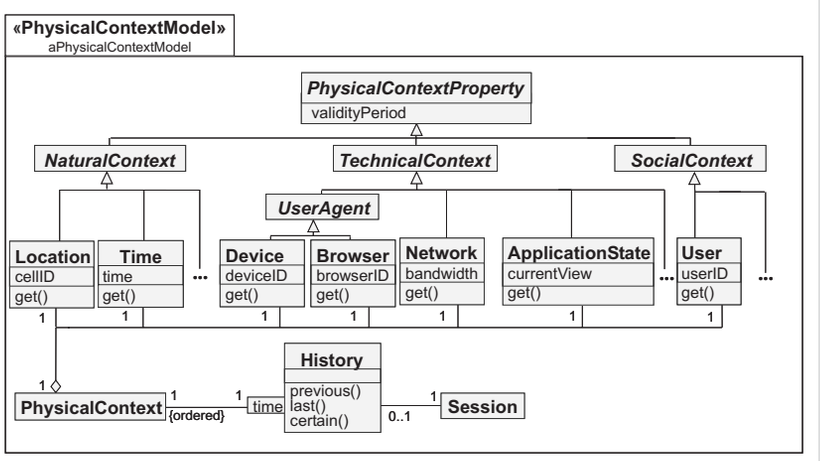

Bild 3 Physisches Kontextmodell [UML: iii, iv, vi, vii]

3.2 Logisches Kontextmodell

Das logische Kontextmodell repräsentiert Kontextinformationen auf höherem Abstraktionsniveau, um physische weiter anzureichern und für die Zwecke der Anpassungsmodellierung brauchbar zu machen. Diese die physischen Kontexteigenschaften erweiternden logischen Informationen werden zu so genannten *Profilen* zusammengefasst und als UML-Pakete modelliert. Dabei finden auch existierende Standardisierungsbestrebungen zur Repräsentation verschiedener Arten von Profilinformationen Berücksichtigung, wie beispielsweise die CC/PP (Composite Capabilities/Preference Profile)-Initiative [W3C01a] und das P3P (Platform for Privacy Preferences)-Projekt [W3C01b] des W3C. Standards wie diese sind von besonderem Interesse, wenn Profilinformationen von Drittanbietern genutzt werden sollen. Zusätzliche Profile können im Zuge einer konkreten Anpassungsmodellierung jederzeit eingeführt werden, um das logische Kontextmodell für eine bestimmte ubiquitäre Web-Anwendung zu erweitern.

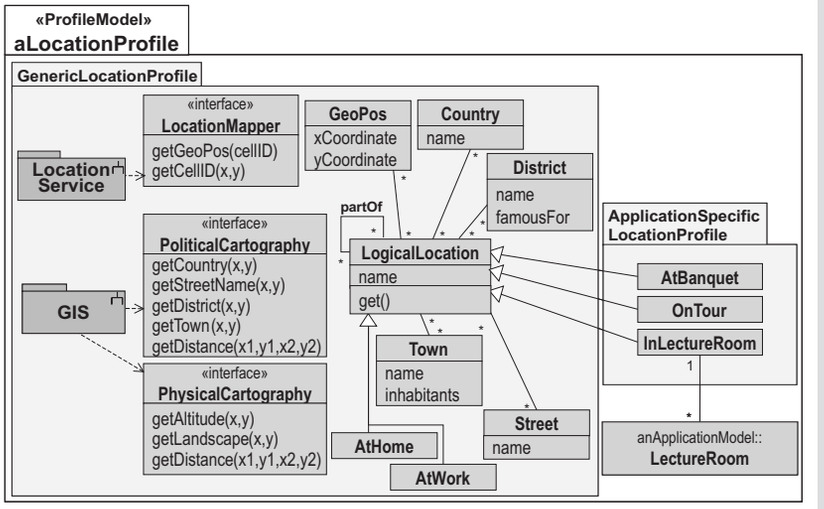

Bild 4 LocationProfile [UML: v]

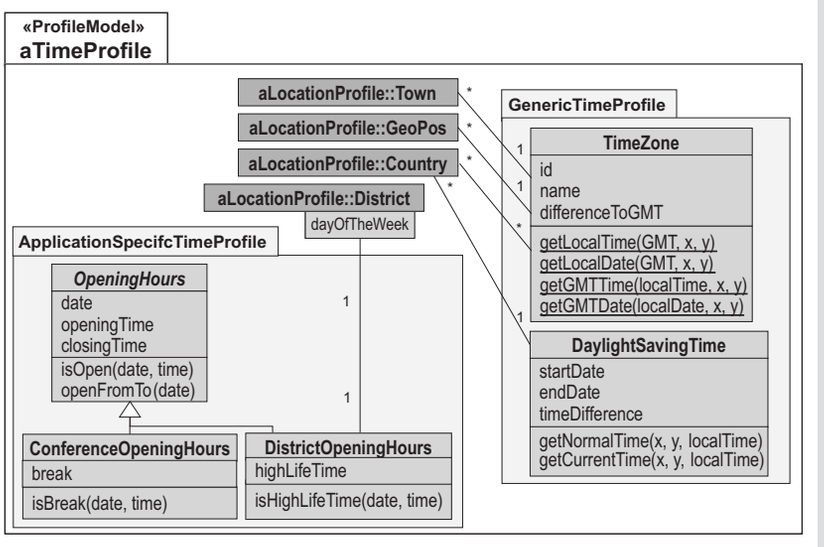

Bild 5 TimeProfile

Im logischen Kontextmodell können typischerweise anwendungsabhängige Teile von generischen, anwendungsunabhängigen und daher wiederverwendbaren Teilen unterschieden werden. In Gegenwart von anwendungsabhängigen Informationen verschwimmt allerdings die Grenze zwischen Anpassungsmodellierung und der klassischen Anwendungsmodellierung, sodass es unklar sein kann, in welchem Bereich ein bestimmter Aspekt modelliert werden sollte. Als Faustregel sollte bei Informationen, die nicht ausschließlich der Anpassungsmodellierung dienen, der Modellierung im Anwendungsmodell der Vorzug gegeben und die entsprechenden Modellelemente im geeigneten Profil über den Import-Mechanismus von UML wiederverwendet werden.

Im Folgenden werden die vordefinierten Profile LocationProfile, TimeProfile, UserAgentProfile, NetworkProfile, ApplicationProfile und UserProfile kurz umrissen.

Das LocationProfile (*Bild 4*) bietet sowohl anwendungsabhängige als auch generische Informationen über die physische und politische Geographie, wobei auf generische Informationen auch über externe Dienste (vgl. etwa das Nexus-Projekt [GrLe01]) zugegriffen werden kann (Pakete LocationService und GIS – Geographisches Informationssystem).

Die zugehörigen Interfaces LocationMapper, PoliticalCartography und PhysicalCartography bieten dazu verschiedene Transformationsmethoden an, um aus der physischen Kontextinformation (cellID) den entsprechenden logischen Kontext herzustellen, der sich allerdings immer noch auf relativ niedrigem Abstraktionsniveau befindet. Für Lokationsinformation auf höherem Abstraktionsniveau sind die Klasse LogicalLocation und ihre Unterklassen zuständig. So können beispielsweise Mechanismen vorgesehen werden, um zwischen den sehr abstrakten „Ortsangaben" AtHome bzw. AtWork zu unterscheiden. Als Beispiel für die Modellierung anwendungsspezifischer Ortsangaben werden im Paket ApplicationSpecificLocationProfile die logischen Lokalisationen angeführt, die für ein Konferenzorganisationssystem relevant sein könnten. Dabei wird auch eine Klasse

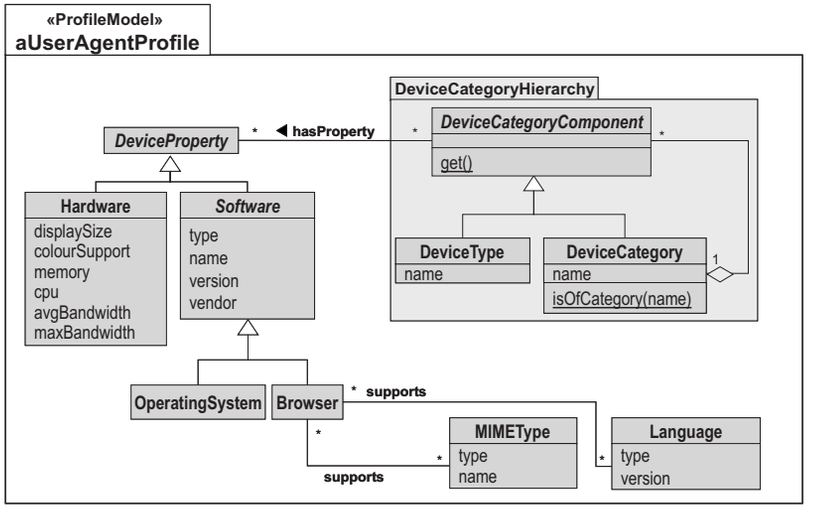

Bild 6 UserAgentProfile

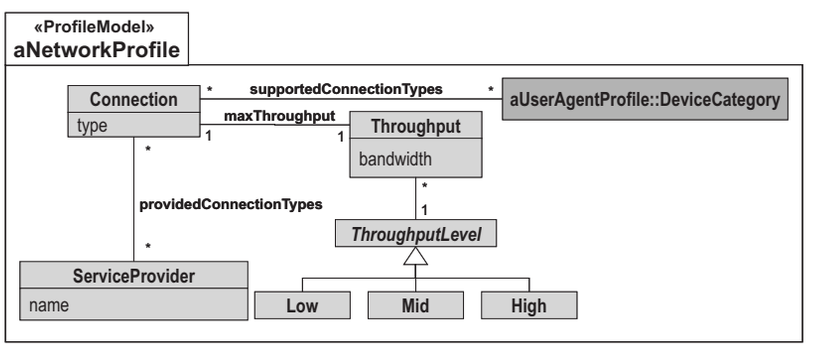

Bild 7 NetworkProfile

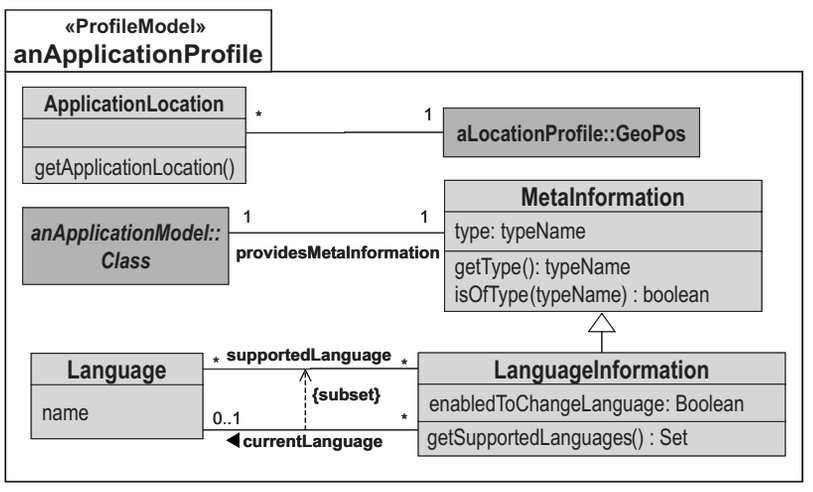

Bild 8 ApplicationProfile

LectureRoom aus dem Anwendungsentwurf importiert [Schw01].

Das TimeProfile (*Bild 5*) umfasst generische wie auch anwendungsabhängige Information zur Interpretation der Zeitangaben aus dem physischen Kontextmodell.

Generische Aspekte sind dabei Zeitzonen und Sommerzeit, während als anwendungsspezifischer Kontext die Zeitangaben aus der Domäne der Konferenzorganisation angeführt sind. Da in beiden Fällen auch Beziehungen zu geographischer Information aufgebaut werden müssen, werden entsprechende Klassen aus dem LocationProfile importiert.

Zu den Informationen, die das physische Kontextmodell über das Endgerät bereitstellt, mit dem auf eine ubiquitären Web-Anwendung zugegriffen wird, werden im UserAgentProfile (*Bild 6*) weitere Daten zu den Eigenschaften der Hard- und Software auf anwendungsunabhängige Weise modelliert. Neben den entsprechenden DeviceProperty-Unterklassen verfügt das Profil über eine dem Composite-Pattern [GaHe94] entsprechende Hierarchie von Gerätekategorien (DeviceCategoryHierarchy). Jede Gerätekategorie wie z. B. „WAP-fähiges Mobiltelefon“ kann eine Reihe von anderen Gerätekategorien als Komponenten enthalten, die ihrerseits Gerätekategorien oder aber Gerätetypen (Blattknoten in der Hierarchie, etwa „Nokia 6120“) darstellen können.

Das NetworkProfile (*Bild 7*) ergänzt das physische Kontextmodell um die Netzwerkverbindungen, die von einzelnen Serviceanbietern zur Verfügung gestellt werden (Klasse Connection, z. B. GMS, HSCSD, GPRS) und ihre Durchsatzinformationen.

Auch hier wird durch Unterklassenbildung ein möglichst hohes Abstraktionsniveau erreicht (Low, Mid, High).

Im ApplicationProfile (*Bild 8*) werden aufbauend auf dem Konzept semantischer Annotationen [NaSh01] die für die Anpassungsmodellierung relevanten Eigenschaften der Anwendung selbst modelliert. Es fokussiert drei generische Aspekte: Die geographische Position des Anwendungsservers (ApplicationLocation), Typinformation über die Objekte der Anwendung (MetaInformation), die vom Anpassungsregelmodell verwendet wird, und die Infor-

mation, welche Sprachen von den einzelnen Anwendungsobjekten unterstützt werden (LanguageInformation).

Das UserProfile umfasst Informationen über den Benutzer der Anwendung, die entweder vom Benutzer explizit zur Verfügung gestellt oder von der Anwendung automatisch aus dem Benutzerverhalten ermittelt werden [Kobs01]. In diesem Profil sind üblicherweise die meisten Informationen anwendungsabhängig. *Bild 9* zeigt oben den generischen Teil, nämlich eine Klasse User mit diversen anpassungsmodellierungsrelevanten Eigenschaften wie z. B. dem Grad der Expertise (Anfänger, Fortgeschrittener, Experte), auf den sich i. A. viele Regeln des Anpassungsregelmodells beziehen, und einer Klassenmethode get(userId), die an Hand einer gegebenen Benutzeridentifikation die entsprechende User-Instanz eruiert.

Darüber hinaus werden in der Generalisierungshierarchie von Role Benutzerrollen modelliert, wobei die abstrakte Unterklasse ActorRole als „Ankerpunkt" für die anwendungsspezifischen Benutzerrollen dient, die u. A. diverse Benutzerpräferenzen (. . . Prefs) darstellen. Besonders hervorzuheben ist die Personalisierung des Layouts einer ubiquitären Web-Anwendung, das durch die ternäre Beziehung zwischen User, PersonalLayout und DeviceCategory auch vom Endgerätetyp abhängig gemacht wird.

Im anwendungsspezifischen Teil werden diverse Rollen für das Konferenzorganisationssystem und persönliche Vorlieben der Konferenzteilnehmer modelliert.

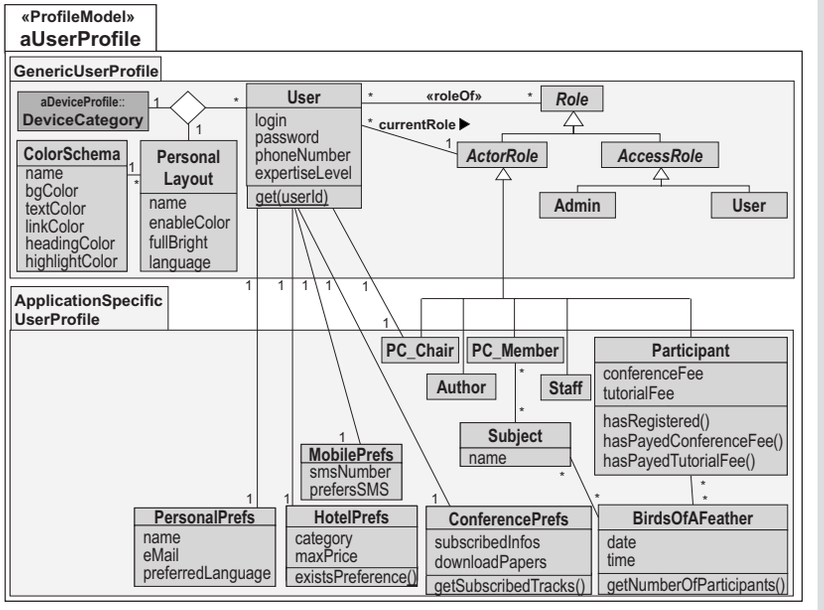

Bild 9 UserProfile

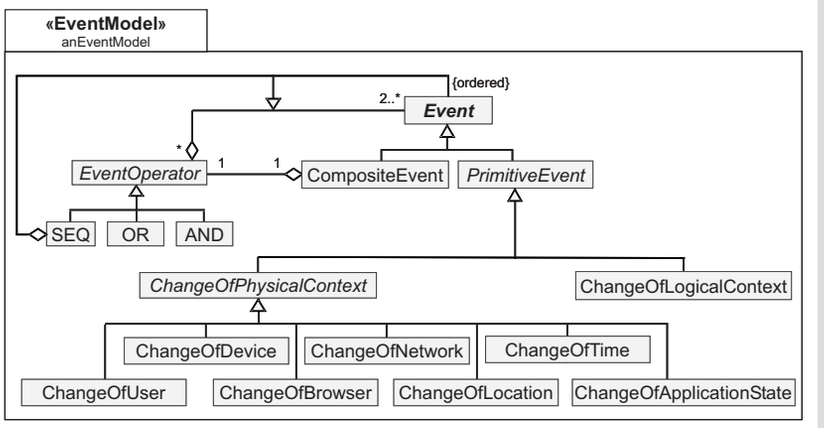

Bild 10 Ereignismodell

4 Anpassungsregelmodell

Die Spezifikation konkreter Anpassungen erfolgt durch so genannte *Event/Condition/Action-Regeln (ECA-Regeln)* [DiGa00], über die Anpassungsinformation zentral verwaltet werden kann (und nicht über die gesamte Anwendung verteilt wird), was die Spezifikation und insbesondere die Wartung wesentlich vereinfacht. Dabei dienen *Ereignisse* zur Erkennung der Kontextänderung und *Bedingungen* zur Prüfung, ob Anpassungen tatsächlich vonnöten sind. Gemeinsam beschreiben sie gewissermaßen die Situation, in der eine Anpassung erfolgen soll. *Welcher Aspekt* der Web-Anwendung konkret *auf welche Weise* anzupassen ist, wird durch die *Aktionen* angegeben.

Das vorgeschlagene Framework ordnet diesen drei Regel-Komponenten eigene Modelle zu, die im Folgenden erläutert werden.

Das in *Bild 10* dargestellte *Ereignismodell* bietet einerseits vordefinierte Ereignistypen, die verschiedene Arten von Kontextänderung modellieren (Generalisierungshierarchie von PrimitiveEvent), andererseits aber auch die Möglichkeit, komplexe Ereignisse (CompositeEvent) zu definieren. Ein komplexes Ereignis setzt sich aus primitiven oder anderen komplexen Ereignissen zusammen, wofür die Operatoren OR, AND und SEQ (Sequenz) zur Verfügung stehen, wie z. B. „ChangeOfLocation OR ChangeOfTime" (vgl. auch die Erläuterung im Anhang UML, Punkt vii).

Darüber hinaus kann das Ereignismodell auch durch Unterklassenbildung um anwendungsspezifische Ereignistypen erweitert werden.

Das *Bedingungsmodell* sieht logische Ausdrücke vor, die in der als Teil von UML definierten Sprache OCL (Object Constraint Language, [RuJa98] [HiKa99]) formuliert

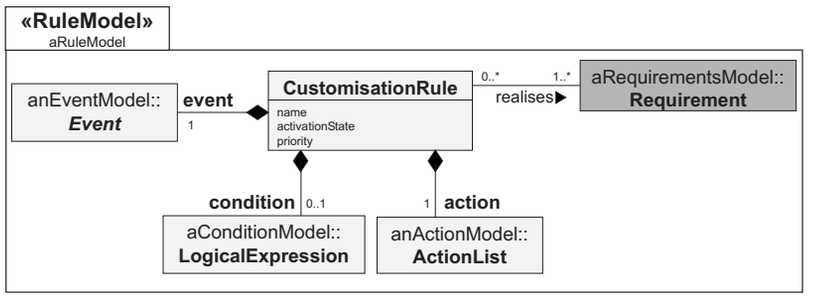

Bild 11 Regelmodell

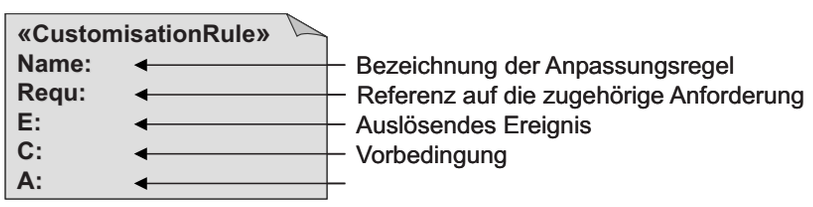

Bild 12 Spezifikation von Anpassungsregeln in Notizen [UML: viii]

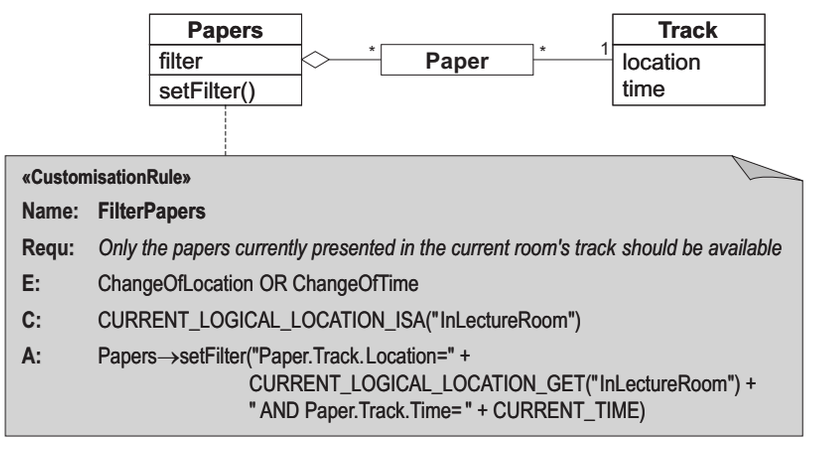

Bild 13 Anpassungsregel zur orts-/zeitabhängigen Einschränkung verfügbarer Artikel

werden. Dabei dürfen nur seiteneffektfreie Ausdrücke verwendet werden, die den Zustand der Anwendung nicht verändern. Da OCL-Ausdrücke durch die notwendigen Navigationspfade innerhalb der referenzierten Modelle zuweilen sehr komplex werden können, wird die Verwendung von *Bedingungsmakros* empfohlen, um einerseits die Spezifikationen leichter lesbar zu gestalten und um andererseits Wiederverwendbarkeit und Änderungslokalität zu erhöhen (vgl. *Bild 13*).

Das *Aktionsmodell* definiert die Aktionen, die innerhalb der ECA-Regeln die eigentlichen Anpassungen bewirken. In jeder Regel kann eine Liste von Aktionen angegeben werden, die ihrerseits entweder atomar (Zuweisung, Aufruf einer Anpassungsoperation) oder zusammengesetzt sein können, um Sequenz, Verzweigung und Iteration auszudrücken. Anpassungsoperationen sind entweder anwendungsspezifisch und müssen daher im Zuge der Modellierung erst definiert und im Anwendungsmodell verankert werden, oder sind generisch, d. h. vordefiniert für die Modellelemente des Anwendungsmodells [Schw01] und somit a priori verfügbar. Diese Anpassungsoperationen bilden somit den variablen Teil einer ubiquitären Web-Anwendung.

Das *Regelmodell* (*Bild 11*) integriert schließlich die einzelnen Regelkomponenten zu Instanzen der Klasse CustomizationRule, wobei Bedingungen optional sind. Das Attribut priority wird zur Auflösung allfälliger Konflikte im Regelsystem herangezogen. Man beachte, dass für jede Regel vermerkt wird, welche Systemanforderungen sie erfüllen soll.

Um Regelinstanzen auf einfache Weise definieren zu können, werden sie in Form einer für diese Zwecke speziell strukturierten UML-Notiz mit dem Stereotyp «CustomisationRule» notiert, in der alle Regelkomponenten textuell referenziert werden (*Bild 12*). Diese Notiz wird jenen Modellelementen des Anwendungsmodells zugeordnet, die von der spezifizierten Aktion betroffen sind, d. h. von ihr modifiziert werden.

Bild 13 zeigt abschließend eine Anpassungsregel aus der Domäne Konferenzorganisation, die bewirken soll, dass nur die zum aktuellen Konferenzsegment (track) gehörigen Artikel verfügbar gemacht werden, wobei „aktuell“ durch Lokation und Zeit definiert ist. Eine diesbezügliche Änderung wird durch eines der vordefinierten Ereignisse ChangeOfLocation und ChangeOfTime signalisiert, während die zugeordnete Bedingung prüft, ob der Aufenthaltsort tatsächlich ein Hörsaal ist. Dazu wird das Bedingungsmakro CURRENT_LOCATION_ISA (*Klassenname*) verwendet, das von dem zugrunde liegenden OCL-Ausdruck

(CURRENT_LOGICAL_LOCATIONS → select (loc | loc → oclType.name=*Klassenname*)) → size > 0

zugunsten der Lesbarkeit abstrahiert. (Anzumerken ist, dass die angegebene Bedingung der Einfachheit halber nicht prüft, ob der Ortswechsel wieder zum ursprünglichen Hörsaal zurückgeführt hat, in welchem Fall eine redundante Anpassung erfolgt.)

Ist diese Bedingung wahr, so bewirkt die angegebene Aktion eine Anpassung der Filterbedingung der Klasse Papers, um die Menge der zur Ansicht verfügbaren Artikel entsprechend einzuschränken.

5 Ausblick

Im vorliegenden Artikel wurde ein UML-basiertes Framework zur Modellierung ubiquitärer Web-Anwendungen vorgestellt, wobei insbesondere der Aspekt der Anpassungsmodellierung behandelt wurde. Zukünftige Arbeiten in diesem Gebiet umfassen schwerpunktmäßig die Bereiche *Werkzeugunterstützung*, *Semantische Annotation* und *Semantische Äquivalenz* und werden im Folgenden näher ausgeführt.

Zur Modellierung ubiquitärer Web-Anwendungen ist eine entsprechende Werkzeugunterstützung unumgänglich. Dazu wird ein Werkzeugkasten in Form eines Regeleditors und eines Regelbrowsers konzipiert, der nicht nur einen integrierten Modellierungsprozess im Hinblick auf Anwendungsmodellierung und Anpassungsmodellierung ermöglicht, sondern auch die Wiederverwendung auf Basis einer Bibliothek von Anpassungsregeln und so genannter (wiederkehrender) Anpassungsmuster erleichtert.

Betrachtet man das Zusammenspiel zwischen dem variablen Teil einer ubiquitären Web-Anwendung (d. h. den Anpassungsoperationen) und dem Anpassungsregelmodell näher, so zeigt sich, dass zwar ein Modellelement eine Reihe von generischen oder anwendungsabhängigen Anpassungsoperationen zur Verfügung stellt, dass jedoch das Wissen, in welcher Form diese Operationen angewandt werden, über eine Reihe von Anpassungsregeln verstreut vorliegt. Im Sinne der Änderungslokalität wäre es wünschenswert, dieses Wissen direkt beim betroffenen Modellelement zu spezifizieren. Dies könnte beispielsweise dadurch erreicht werden, dass das ApplicationProfile gemäß dem Konzept semantischer Annotationen [NaSh01] erweitert wird.

Ein wesentliches Ziel bei der Realisierung ubiquitärer Web-Anwendungen stellt die Sicherstellung semantischer Äquivalenz dar, d. h. die Anpassung einer Web-Anwendung an einen bestimmten Kontext sollte den Nutzen für einen Benutzer nicht schmälern. Auch das Modellierungsframework sollte geeignete Konstrukte anbieten, um semantische Äquivalenz spezifizieren zu können.

Literatur

[Abow99] *Abowd, G. D.:* Software Engineering Issues for Ubiquitous Computing. In: Proc. of the International Conference on Software Engineering (ICSE), Los Angeles 1999.

[BaFo00] *Badrinath, B.; Fox, A. et al.:* A conceptual framework for network and client adaptation. In: Proceedings of the IEEE Mobile Networks and Applications Vol. 5 No. 4. MONET, 2000.

[BaKo99] *Baumeister, H.; Koch, N. et al.:* Towards a UML extension for hypermedia design. UML'99 The Unified Modeling Language – Beyond the Standard, LNCS 1723. Springer, Fort Collins October 1999.

[BaLa01] *Barry, C.; Lang, M.:* A Survey of Multimedia and Web Development Techniques and Methodology Usage. In: IEEE Multimedia Vol. 8, No. 2; April–June 2001.

[Cona99] *Conallen, J.:* Modeling Web Application Architectures with UML. In: Communications of the ACM (CACM), Vol. 42, No. 10, October 1999.

[DiGa00] *Dittrich, K. R.; Gatziu, S.:* Aktive Datenbanksysteme – Konzepte und Mechanismen. 2. Auflage, dpunkt.Verlag, April 2000.

[EhKa97] *Ehmayer, G.; Kappel, G. et al.:* Connecting Databases to the Web – A Taxonomy of Gateways. In: Proceedings of the 8th International Conference on Database and Expert Systems Applications (DEXA 97). Springer LNCS 1308, France September 1997.

[Fisc01] *Fischer, G.:* User Modelling in Human – Computer Interaction. In: User Modelling and User-Adapted Interaction Vol. 11 (2001).

[GaHe94] *Gamma, E.; Helm, R. et al.:* Design Patterns – Elements of Reusable Object-Oriented Software. Addison Wesley, 1994.

[GrLe01] *Großmann, M; Leonhardi, A. et al.:* A World Model for Location-Aware Systems. In: Informatik/Informatique Vol. 8 (2001), 5

[HiKa99] *Hitz, M.; Kappel, G.:* UML @ Work – Von der Analyse zur Realisierung. dpunkt.verlag, 1999.

[KaRe00] *Kappel, G.; Retschitzegger, W. et al.:* Modeling Customizable Web Applications – A Requirement's Perspective. In: Proceedings of the International Conference on Digital Libraries: Research and Practice (ICDL), Kyoto, Japan, November 2000.

[KaRe01] *Kappel, G.; Retschitzegger, W. et al.:* Modeling Ubiquitous Web Applications – The WUML Approach. Proceedings of International Workshop on Data Semantics in Web Information Systems (DASWIS 2001), Yokohama November 2001.

[Kobs01] *Kobsa, A.:* Generic User Modeling Systems. User Modeling and User-Adapted Interaction, Vol. 11, 2001.

[KuBl99] *Kunz, T.; Black, J. P.:* An Architecture for Adaptive Mobile Applications. In: Proc. of Wireless 99, the 11th International Conference on Wireless Communications, Calgary, Alberta, Canada July 1999.

[NaSh01] *Nagao, K.; Shirai, Y. et al.:* Semantic Annotation and Transcoding: Making Web Content More Accessible, IEEE MultiMedia, April–June 2001.

[OpSp99] *Oppermann, R.; Specht, M.:* A Nomadic Information System for Adaptive Exhibition Guidance. In: Proc. of the International Conference on Hypermedia and Interactivity in Museums (ICHIM), D. Bearman and J. Trant (eds.), Washington September 1999.

[RuJa98] *Rumbaugh, J.; Jacobson, I. et al.:* The Unified Modeling Language Reference Manual. Addison-Wesley, 1998.

[ScBe99] *Schmidt, A.; Beigl, M. et al.:* „There is more to Context than Location" Computers & Graphics Journal, Elsevier, Volume 23, No. 6, December 1999.

[Schw01] *Schwinger, W.:* Modelling Ubiquitous Web Applications. Dissertation, Sozial- und Wirtschaftswissenschaftliche Fakultät der Johannes Kepler Universität Linz, 2001.

[Weis91] *M. Weiser:* The Computer for the 21st Century. Scientific American, 265, 3, September 1991.

[W3C01a] World Wide Web Consortium (W3C), Composite Capabilities/Preference Profiles (CC/PP), http://www.w3.org/Mobile, 2001.

[W3C01b] World Wide Web Consortium (W3C), Platform for Privacy Preferences (P3P) Project, http://www.w3.org/P3P, 2001.

Abstract

A UML based framework for modeling ubiquitous Web applications

Electronic commerce (e-commerce) and mobile commerce (m-commerce) have dramatically boosted the demand for services which enable ubiquitous access. Ubiquity offers opportunities in terms of time aware, location aware, device aware, and personalised services. Development of ubiquitous Web applications, however, turns out to be rather complex and thus requires appropriate methodological support. Existing methods for modelling Web applications only partially match the requirements resulting from their ubiquitous nature.
This article aims at filling this gap by presenting a UML based framework for modelling ubiquitous Web applications, focussing on issues of adaptation modelling. It encompasses both, context modelling by providing a physical and a logical context model, and modelling the adaptation process per se. The latter is realised in terms of a rule model enabling monitoring of context changes and activation of corresponding adaptation operations. The separation of a Web application in a stable and context independent part on the one hand and a variable and context dependent part on the other hand supports reusability and locality of change.

Keywords: ubiquitous Web application, object-oriented modelling framework, Unified Modeling Language (UML), event/condition/action rules (ECA rules)

Anhang: UML Vademekum

i. Zur Organisation der einzelnen Diagramme kennt UML den Abstraktions- bzw. Strukturierungsmechanismus des Pakets. Ein *Paket* (package) ermöglicht es, eine beliebige Anzahl von UML-Konstrukten und -Diagrammen zu gruppieren und davon zu abstrahieren. Dabei können Pakete selbst wieder Pakete beinhalten. Ein Paket wird in UML durch ein Rechteck mit aufgesetztem kleinen Rechteck dargestellt („Karteikarte mit Reiter"). Wird der Paketinhalt nicht gezeigt, so steht der Paketname innerhalb des großen Rechtecks. Wird der Paketinhalt dargestellt, so steht der Paketname im Reiter.
Ein gestrichelter Pfeil mit dem Schlüsselwort «access» stellt eine *Genehmigungsabhängigkeit* (permission dependency, vgl. auch ix) zwischen Paketen dar und impliziert, dass die Elemente des ersten, „abhängigen" Pakets alle Elemente mit ausreichender Sichtbarkeit des zweiten, „unabhängigen" Pakets referenzieren können. Referenzen auf Elemente anderer Pakete folgen i. A. der Syntax Paketname::Elementname (Beispiel: anApplicationModel::LectureRoom in *Bild 4*).
Die dabei verwendeten Pakete sind durch Stereotype kategorisiert (vgl. ii).

ii. Ein *Stereotyp* (stereotype) ermöglicht die Einteilung von UML-Elementen in benutzerdefinierte Kategorien, denen eine über den Standard hinausgehende Semantik zugeordnet werden kann und stellt somit einen Erweiterungsmechanismus für UML dar. Ein Stereotyp wird textuell durch ein Schlüsselwort (Bezeichner zwischen «...») oder grafisch durch ein Piktogramm notiert.

iii. Ein *Klassendiagramm* (static structure diagram) zeigt im Wesentlichen Klassen und deren Beziehungen, kann allerdings auch einige andere »klassenähnliche« Konstrukte enthalten (Interfaces, Klassenschablonen, Typen, Referenzen auf Entwurfsmuster u. v. m.). Darüber hinaus können auch konkrete Instanzierungen, also Objekte und Objektbeziehungen repräsentiert werden, um bestimmte Sachverhalte exemplarisch zu illustrieren.

iv. *Klassen* werden als Rechtecke notiert, die in Abschnitte untergliedert sind. Mit Ausnahme des obersten dürfen alle Abschnitte fehlen. Im obersten Abschnitt sind der Name und optional allgemeine Charakteristika der Klasse vermerkt (Stereotyp textuell oder als Piktogramm, Eigenschaftsangaben in geschwungenen Klammern), der zweite Abschnitt enthält die Attribute und der dritte Abschnitt die Operationen. *Abstrakte Klassen* werden durch die Eigenschaftsangabe {abstract} im obersten Abschnitt oder durch kursiv gesetzte Klassennamen charakterisiert.
Ein *Attribut* wird durch seinen Namen definiert. Zusätzliche optionale Angaben umfassen den Datentyp, die Multiplizität (im Standardfall 1), den Initialwert, und die Sichtbarkeit (public oder + für öffentliche Sichtbarkeit, private oder – für private Sichtbarkeit, protected oder # für geschützte Sichtbarkeit). Abgeleitete (berechenbare) Attribute werden durch einen vorangestellten Schrägstrich markiert. *Klassenattribute* werden in UML unterstrichen.
Eine *Operation* aus dem dritten Abschnitt des Klassensymbols wird ähnlich einem Attribut durch Sichtbarkeit und Name spezifiziert, wobei zusätzlich eine (eventuell leere) Parameterliste angegeben werden muss. Je Parameter sind folgende Angaben möglich: Name, Datentyp, Datenflussrichtung (in, out, inout) und Standardwert. Nach der Parameterliste kann der Ergebnistyp der Operation angegeben werden. Klassenoperationen werden wiederum unterstrichen (Beispiel: TimeZone::getLocalTime() in *Bild 5*).

v. Ein *Interface* ähnelt einer Klasse, kann aber lediglich Operationen, Generalisierungsbeziehungen und allenfalls hinführende unidirektionale Assoziationen aufweisen. Es spezifiziert durch die Vereinbarung von Operationen gewünschtes Verhalten, das es aber selbst nicht implementieren kann. Dazu bedarf es „echter" Klassen, die mit dem Interface in einer so genannten Realisierungsbeziehung stehen. Ein Interface wird durch ein Klassensymbol notiert, das i. A. lediglich den Namensabschnitt (mit dem Schlüsselwort «interface» versehen) und Operationssignaturen aufweist. Die Realisierungsbeziehung wird durch einen gestrichelten Generalisierungspfeil dargestellt, der von der realisierenden Klasse zum Interface zeigt.

vi. Zweistellige *Assoziationen* werden durch einfache Verbindungslinien (Kanten) zwischen den beteiligten Klassen dargestellt. Folgende zusätzliche Angaben sind u. A. möglich: Name der Assoziation (eventuell mit Angabe der Leserichtung durch ein schwarzes gleichseitiges Dreieck), Bezeichnung der Rollen, welche die beteiligten Objekte im Rahmen der Beziehung spielen (an den jeweiligen Enden der Assoziationskante notiert), Multiplizitäten der einzelnen Rollen (erlaubte Anzahl von „Partnerobjekten" der beteiligten Klasse, wobei * für „beliebig viele" steht), und allfällige spezielle Eigenschaften dieser Rollen (etwa {ordered} in *Bild 3*).
Assoziationskanten können auch gerichtet sein, um explizit anzugeben, in welche Richtung die Navigation von einem Objekt zu seinem Partnerobjekt erfolgen kann. Ungerichtete Kanten signalisieren „keine Angabe über Navigationsmöglichkeiten".
Bei Vorliegen einer *Aggregation* (aggregation, part-of relationship) als Spezialfall der Assoziation wird jenes Ende der Assoziationskante, das zur Aggregatklasse, also „zum Ganzen" hinführt, durch eine kleine (nicht ausgefüllte) Raute markiert.
Die *Komposition* (composition) stellt eine spezielle, strengere Form der Aggregation dar, die durch eine ausgefüllte Aggregationsraute notiert wird. In der Kompositionsbeziehung gilt für die beteiligten Instanzen, dass ein Teil zu höchstens einem Ganzen gehören darf und dass ein Teil höchstens so lange wie sein Ganzes existiert.
Bei einer *qualifizierten Assoziation* wird die Menge der Objektbeziehungen zwischen den Instanzen durch qualifizierende Attribute der Assoziation (in kleinen Rechtecken an den entsprechenden Assoziationsenden notiert) in disjunkte Teilmengen zerlegt. Dadurch reduziert sich unter Umständen die Multiplizität der Assoziation (Beispiel aus *Bild 3*: Einer History-Instanz können grundsätzlich mehrere PhysicalContext-Instanzen zugeordnet sein, zu einem bestimmten Zeitpunkt time (qualifizierendes Attribut) jedoch nur genau eine).
Eine *mehrstellige Assoziation* wird durch eine Raute dargestellt, die mit allen Klassen, die an der Assoziation teilnehmen, durch eine Kante verbunden ist (Beispiel: *Bild 9*). Ein allfälliger Assoziationsname wird in der Umgebung der Raute notiert.

vii. Die *Generalisierung* (generalization, is-a-Beziehung) wird durch einen Pfeil mit einem gleichseitigen Dreieck als Spitze notiert, der von der Unterklasse zur Basisklasse führt. Mehrere Generalisierungspfeile, die zur selben Basisklasse führen, können zu einem „Mehrfachpfeil" (eine Spitze für mehrere Enden) verschmelzen. In *Bild 3* werden z. B. die einzelnen Kontext-Kategorien in mehreren Stufen zur abstrakten Klasse PhysicalContextProperty generalisiert, deren Attribut validityPeriod an alle Unterklassen „vererbt" wird.

Generalisierungen sind nicht nur bei Klassen, sondern auch bei vielen anderen UML-Modellelementen möglich. So enthält *Bild 10* etwa eine Generalisierungsbeziehung zwischen zwei Aggregationen. Dadurch wird ausgedrückt, dass die Aggregation SEQ-Event eine Spezialisierung der Aggregation Event-Operator-Event ist, wobei erstere alle Eigenschaften der letzteren erbt und durch die Eigenschaft {ordered} zusätzlich verlangt wird, dass die einer SEQ-Instanz zugeordneten Event-Objekte in wohldefinierten Reihenfolge vorliegen.

viii. Eine *Notiz* (note) bietet die Möglichkeit, UML-Modellelemente verbal zu erläutern. Sie wird durch ein Rechteck mit Eselsohr dargestellt und durch eine gestrichelte Linie mit dem beschriebenen Modellelement verbunden. Als Inhalte von Notizen sind z. B. Kommentare, Einschränkungen (meist in OCL) oder Programmcode denkbar.
Im vorliegenden Framework werden die für die Definition von Anpassungsregeln benützten Notizen durch das Stereotyp «CustomizationRule» mit zusätzlicher Semantik versehen (vgl. *Bild 12*).

ix. Eine *Abhängigkeit* (dependency) zwischen zwei Modellelementen stellt eine sehr allgemeine Art der Beziehung zwischen den beiden Modellelementen mit breitem Anwendungsspektrum dar. Ganz allgemein wird eine Abhängigkeit durch einen gestrichelten Pfeil zwischen den Modellelementen angezeigt, wobei Schlüsselwörter wie «access», «use» etc. oder Eigenschaftsangaben wie {subset} die Semantik der Abhängigkeit genauer bestimmen. *Bild 8* enthält eine {subset}-Abhängigkeit zwischen den Assoziationen currentLanguage und supportedLanguage, die angibt, dass die aktuelle Sprache unter den unterstützen Sprachen vorkommen muss.

x. *Aktivitätsdiagramme* ermöglichen die Beschreibung funktionaler Abläufe, wobei spezifiziert werden kann, *was* die einzelnen Schritte des Ablaufs tun (durch Angabe einer Bezeichnung des Schritts und der durch ihn manipulierten Objekte) und *in welcher Reihenfolge* sie ausgeführt werden (durch Transitionspfeile zwischen den Schritten). Die einzelnen Schritte eines Ablaufs werden so genannten Zuständen zugeordnet, in denen die entsprechenden Aktivitäten ausgeführt werden.
Ein *Zustand* (state) im Aktivitätsdiagramm wird durch ein „Rechteck“ mit konvexen Vertikalen dargestellt, das den Namen der auszuführenden Aktivität enthält.
Beginn und Ende eines Ablaufs werden durch einen eindeutigen Start- und möglicherweise mehrere Endzustände markiert (kleiner schwarzer Kreis bzw. kleiner schwarzer Kreis mit Ring).
Aktivitätsdiagramme unterstützen in Anlehnung an Datenflussdiagramme die Modellierung von Objektflüssen (object flows), indem Objekte entweder als Eingabe oder als Ausgabe einer oder mehrerer Aktivitäten modelliert werden. Abhängig davon wird das entsprechende Objektsymbol, ein Rechteck mit dem unterstrichenen Namen des Objekts und seinem aktuellen Verarbeitungszustand in eckigen Klammern, als Eingabe zu oder als Ausgabe von einer Aktivität spezifiziert, indem eine gerichtete gestrichelte Kante vom Objektsymbol zur verarbeitenden Aktivität bzw. umgekehrt gezeichnet wird. Sofern der Objektfluss den Kontrollfluss zwischen zwei Aktivitäten vollständig spezifiziert, wird nur der Objektfluss dargestellt.