M. Herczeg, W. Prinz, H. Oberquelle (Hrsg.): Mensch & Computer 2002: Vom interaktiven Werkzeug zu kooperativen Arbeits- und Lernwelten. Stuttgart: B. G. Teubner, 2002, S. 185-194.

Vesuf, eine modellbasierte User Interface Entwicklungsumgebung für das Ubiquitous Computing

A. Pokahr, L. Braubach, A. Bartelt, D. Moldt und W. Lamersdorf
Universität Hamburg, Fachbereich Informatik

Zusammenfassung

Zur systematischen Erstellung von User Interfaces für Applikationen im Kontext des Ubiquitous Computing wird eine innovative Entwicklungsumgebung vorgestellt. Diese erlaubt es, Applikationen auf einfache deklarative Weise mit verschiedenen User Interface Modalitäten auszustatten und auf Basis abstrakter Modelle für jede Modalität eine optimale Schnittstelle zu konstruieren, ohne die Fachlogik modifizieren zu müssen. Um die Tragfähigkeit der Konzepte und die praktische Einsatzfähigkeit des Systems nachzuweisen, wurden exemplarisch heterogene Dienste in ein Internetportal integriert.

1 Einleitung

Ubiquitous Computing (UbiComp) hat das Ziel, Computer in die reale alltägliche Welt des Menschen zu integrieren (Gallis et al. 2001), was zu einer steigenden Heterogenität der Anwendungskontexte führt. Diese Heterogenität bezieht sich auf die physische Umgebung, die verwendeten Gerätetypen sowie die individuellen Fähigkeiten und Vorlieben der Anwender. Die kombinierten Anforderungen vielfältiger Anwendungskontexte erhöhen die Komplexität für die Erstellung von UbiComp Applikationen.

Um Anwendungen unter den oben genannten Voraussetzungen effizient zu realisieren, ist es notwendig, diese in einer dienstorientierten und dynamisch komponierbaren Form zu entwerfen. Zusätzlich spielen die Geräteunabhängigkeit (device-independence), die Trennung von User Interface und funktionalem Kern (user interface / application separation) und das dynamische Erkennen, Verhandeln und Einbinden von Diensten (service discovery and brokerage) eine grosse Rolle (Banavar et al. 2000).

Derzeitige Ansätze des Anwendungsentwurfs weisen im Kontext des UbiComp erhebliche Defizite auf (Grimm et al. 2000). Ein Teil der Applikationen wird abstrakt entworfen, um unverändert in verschiedenen Kontexten einsetzbar zu sein. Damit können sie nur universell verfügbare Interaktionsmöglichkeiten anbieten, oder sie müssen sehr komplexe Konstruktions- und Ableitungsregeln für diverse Interfacemodalitäten umsetzen. Alternativ werden für jeden Anwendungskontext eigene Anwendungen entworfen, was einen erheblichen Aufwand zur Folge hat.

Zur technischen Lösung dieser Probleme bietet sich eine gemeinsame Plattform an, in der Dienste integriert werden, während eine spezielle Komponente für die Erzeugung von Benutzungsschnittstellen zuständig ist (Grimm et al. 2000). Diese Komponente stellt ein modellbasiertes Werkzeug (Model-Based User Interface Development Environment, MB-UIDE, siehe da Silva 2001) dar, das informationsreiche Modelle auswertet, um dynamisch Benutzungsschnittstellen für den jeweiligen Anwendungskontext zu generieren.

Die Notwendigkeit einer derartigen Komponente wurde beispielsweise im Verlauf der Realisierung des PublicationPORTALs (Bartelt et al. 2001) identifiziert. Im Rahmen des Global Info Projekts (Global-Info 2000) soll das Portal für verschiedene Akteure als Zugangspunkt zu diversen ePublishing Services dienen. Benötigt wurde eine Benutzungsschnittstellenkomponente, mit der heterogene Dienste mit einem einheitlichen Erscheinungsbild versehen und in das webbasierte Portal aufgenommen werden können. Des weiteren soll diese Komponente die Technologie bereitstellen, mit der die Dienste des Portals auch über andere Endgeräte, z. B. ein WAP-Handy erreicht werden können.

Im Folgenden wird ein Überblick über Grundlagen von Entwicklungsumgebungen gegeben (Abschnitt 2). In Abschnitt 3 wird das entwickelte System und in Abschnitt 4 sein Einsatz an Hand eines Beispiels beschrieben. Schließlich folgt eine Zusammenfassung der Ergebnisse und ein Ausblick (Abschnitt 5).

2 Grundlagen

Im Rahmen des hier vorgestellten Projekts wurde eine umfangreiche Untersuchung über Entwurfshilfen für User Interfaces und ihre Eignung in Bezug auf UbiComp durchgeführt, deren Ergebnisse im Folgenden wiedergegeben werden (Braubach & Pokahr 2001). Dabei wird eine grobe Kategorisierung in drei Abstraktionsniveaus vorgeschlagen. Erstens werden unter der Kategorie Werkzeuge alle Artefakte subsumiert, die dem Entwickler ein konkretes Stück Software in die Hand geben, um User Interfaces zu gestalten. Zweitens sind Techniken Grundlage der meisten Werkzeuge und formalisieren gezielt die Beschreibung bestimmter User Interface Aspekte. Drittens werden zur Kategorie der Architekturen alle Artefakte gerechnet, die auf einem sehr hohen Abstraktionsgrad wiederkehrende Designentscheidungen systematisieren. Diese drei Abstraktionsebenen bilden die Basis für die Vision einer Entwicklungsumgebung, die alle drei Ebenen optimal integriert.

2.1 Werkzeuge

Myers klassifiziert Werkzeuge für User-Interfaces danach, wie abstrakt ein Entwickler damit Oberflächen beschreibt und unterscheidet zwischen sprachenbasierter und grafischer Spezifikation und automatischer Generation (Myers 1989). Fähnrich adaptiert dieses Schema (Fähnrich 2000) und fügt als weitere Klassifizierungsdimension eine Unterscheidung nach Funktionalitäten (Präsentation, Dialogsteuerung und Applikationssemantik) hinzu, die auf dem Seeheim Modell (siehe Green 1985) basiert. In (Braubach & Pokahr 2001) wird in dieses Schema ergänzend die Klasse der MB-UIDEs eingeordnet, die ein breiteres Spektrum abdeckt als die von Fähnrich dargestellten 4GL-Werkzeuge. Zusätzlich werden noch CASE Tools für die Anwendungssemantik und Anwendungsgeneratoren in die Darstellung aufgenommen (s. Abb. 1).

Ein Problem bei der Verwendung von verschiedenen Werkzeugen ist die mangelnde Interoperabilität. Daher wird eine höhere Abstraktion benötigt, um Interoperabilität durch den Werkzeugen übergeordnete Standards zu erreichen. Kandidaten für diese Art Standards sind abstrakte Techniken.

2.2 Techniken

Die Klassifizierung der Techniken erfolgt nach dem gleichen Schema, das schon zur Einordnung der Werkzeuge verwendet wurde, wobei sich die Anwendungssemantik zusätzlich in Techniken zur

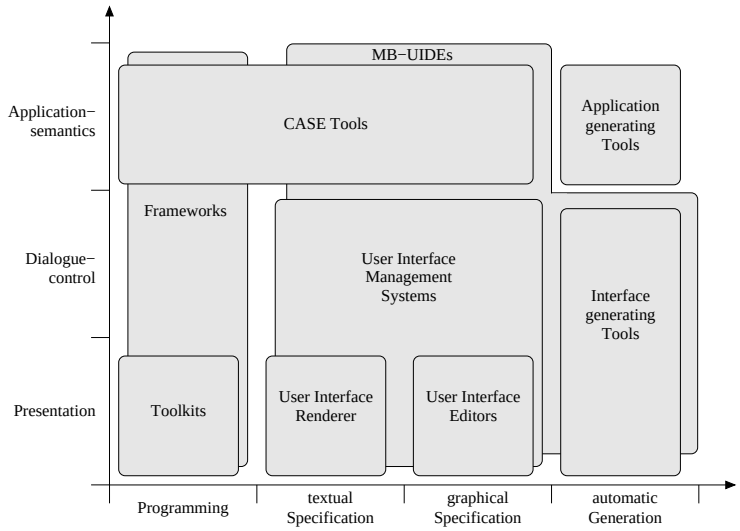

Abbildung 1: Klassifikation von Werkzeugen (in Anlehnung an Fähnrich)

Umsetzung von Aufgaben (*tasks*) und Entitäten (*entities*) differenzieren läßt (s. Abb. 2). Im Unterschied zu den Werkzeugklassen fällt sofort auf, dass sich Techniken meist auf einen Abstraktionsgrad und einen Aspekt der Benutzungsschnittstelle beschränken, während Werkzeuge versuchen, möglichst große Bereiche abzudecken.

Standardisierte Techniken beseitigen nicht die generelle Schwierigkeit, die einzelnen Komponenten der Anwendung zu koppeln. Tatsächlich wird diese durch den Einsatz unterschiedlicher Techniken zur Umsetzung dieser Komponenten eher noch verstärkt. Zur Verbindung der einzelnen Techniken wird eine übergeordnete Architektur benötigt.

2.3 Referenzarchitekturen

In Übereinstimmung zum gewählten Klassifizierungsschema von Techniken und Werkzeugen schlagen alle untersuchten Architekturen in der einen oder anderen Weise eine Trennung in Präsentations-, Dialog- und Applikationskomponente vor, eine Herangehensweise, die laut Kazman & Bass unstrittig ist (Kazman & Bass 1996). Architekturen lassen sich in Schichtenmodelle wie das Seeheim (Green 1985) und das Arch Modell (Bass et al. 1992) und objektorientierte bzw. agentenbasierte Ansätze wie MVC (Burbeck 1992) und PAC (Coutaz 1987) unterteilen. Des weiteren existieren auch Versuche, beide Ansätze zu kombinieren, z.B. PAC-Amodeus (Calvary et al. 1997).

Die Schichtenmodelle erreichen eine gute Modularisierung eines Systems, sind jedoch sehr allgemein und helfen nicht bei der Verbindung verschiedener Techniken, wohingegen die objektorientierten Architekturen ein Zusammenspiel unterschiedlicher Techniken auf die programmiersprachliche Ebene beschränken. Erforderlich ist eine Architektur, die sowohl bei der Kombination von Techniken als auch bei der modularen Gestaltung eines User Interfaces behilflich ist.

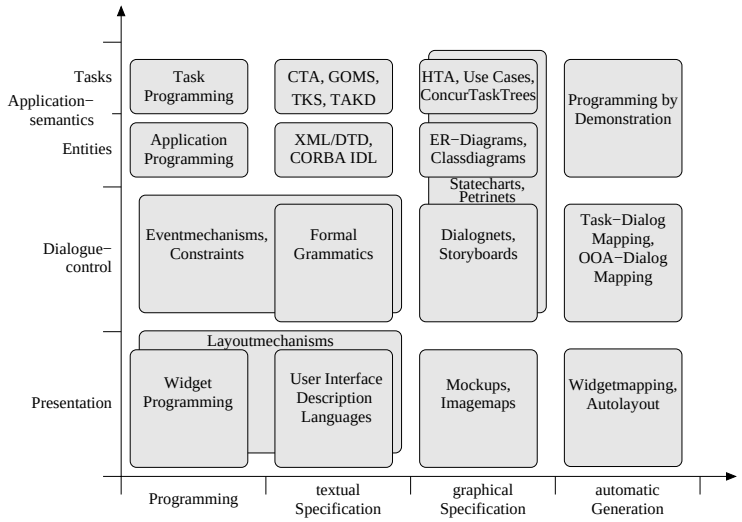

Abbildung 2: Klassifikation von Techniken

2.4 Vision

In der folgenden Vision werden die Fragestellungen betrachtet, auf welcher Basis Applikationen für das UbiComp entworfen werden können, was für eine Werkzeugunterstützung angezeigt ist und auf welche weitergehende Unterstützung ein Entwickler hoffen kann.

Als Grundlage für UbiComp Anwendungen erscheinen Schichtenmodelle besonders interessant, da sie die Modularität einer Anwendung massgeblich unterstützen. Durch Schichtenmodelle kann die Forderung nach Geräte-, Kontext- und Anwenderunabhängigkeit realisiert werden, indem sie bei entsprechender Granularität der Schichten einen Austausch einzelner Teilkomponenten einer Applikation ermöglichen.

Um die Entwicklung von Applikationen für das UbiComp in geeigneter Form zu unterstützen, wird entweder ein Werkzeug benötigt, das viele Kategorien abdeckt oder verschiedene interoperable Werkzeuge, welche sich jeweils für spezifische Aufgaben eignen. Basis dieser Umgebung sollte eine möglichst abstrakte und deklarative Ebene der Spezifikation (grafisch und textuelle Form) sein, die eine hohe Wiederverwendbarkeit, Austauschbarkeit, Komposition und Wartbarkeit ermöglicht. Dadurch wird die mangelnde Flexibilität der Programmiertechniken vermieden und es sind unterschiedliche Interfacemodalitäten für eine Anwendung realisierbar.

Die systembasierte Unterstützung des Entwicklers kann durch die Verwendung des deklarativen und modellbasierten Ansatzes verbessert werden. Analysewerkzeuge können ein spezifiziertes Modell auf Schwachstellen untersuchen und Verbesserungsvorschläge unterbreiten (design critics / advisors). Weiterhin sind Werkzeuge wünschenswert, die kontextsensitive Hilfe, Auswertungen der Benutzbarkeit (usability analyser) und Anpassungen an Benutzervorstellungen (customization tools) anbieten (Szekely 1996)). Für den Entwickler sind weiterhin Automatisierungswerkzeuge von grossem Interesse, die Anteile des User Interfaces aus Modellinformationen deduzieren können. Derartige Werkzeuge sind zwar nicht in der Lage, ein User Interface in

optimaler Qualität zu erstellen, ermöglichen dem Entwickler aber sehr wohl ein bequemes Rapid-Prototyping. So kann mit minimalem Spezifikationsaufwand bereits eine ausführbare Applikation mit Benutzungsschnittstelle erstellt werden, die dann schrittweise mit Informationen ergänzt werden kann und so das Look'n Feel der Anwendung verbessert. Der Prozess der schrittweisen Verfeinerung von deklarativen Modellen wird hier als *slinky automation* bezeichnet.

Weiterhin begünstigt der Einsatz einer modellbasierten Entwicklungsumgebung die Einführung systematischer Entwicklungsmethoden oder sogar die Etablierung einer Methodologie für den Gesamtprozess. Nützlich dafür ist, dass User Interfaces in verschiedenen Abstraktionsgraden modelliert, Modelle inkrementell verfeinert und Interfacespezifikationen wiederverwendet werden können (da Silva 2001).

3 Vesuf

Auf Basis verfügbarer Werkzeuge und Techniken wurde nach einer eingehenden Untersuchung anderer Systeme eine modellbasierte Entwicklungsumgebung entworfen und realisiert, welche die im vorangegangenen Abschnitt vorgestellte Vision mit einigen Einschränkungen umsetzt (siehe Braubach & Pokahr 2001). Die Vesuf Umgebung ermöglicht die Erzeugung von ablauffähigen und vollständig integrierten User Interfaces durch einen Interpreter, der zur Laufzeit Modellinformationen evaluiert. Das System basiert konsequent auf Standards, ist in der plattformunabhängigen Programmiersprache Java implementiert und verwendet zur Modellierung fast ausschließlich den de-facto Industriestandard zur Anwendungsspezifikation – UML (UML 1.4 2001), der um ein neues Metamodell zur Spezifikation von Benutzungsschnittstellen mit UIML (UIML 2.0a 2000) erweitert wurde.

Essentiell im Hinblick auf UbiComp ist die klare Trennung von User Interface und Anwendungskern. Das Vesuf System ermöglicht damit, verschiedene Arten von User Interfaces auf einfache Weise für einen Anwendungskern bereitzustellen, und ist sowohl im Hinblick auf User Interface Modalitäten als auch bezüglich der Anbindung verschiedener Implementationstypen (respektive Altsysteme) erweiterbar. Zusätzlich schafft das System durch die Automatisierung von Aufgaben und durch die vollständige Integration von User Interface und Fachlogik die Grundlage für schnelles Prototyping.

Da das Vesuf System auf UML-Semantik beruht, ist die Lernschwelle für Erstanwender niedrig. Die Spezifikation der Anwendungen in Form von Modellen kann mit beliebigen UML-konformen CASE Tools erfolgen. Oberflächen für Web-Applikationen können mit entsprechenden WYSIWYG Editoren (z. B. für HTML) erstellt werden. Die weitere Bedienschnittstelle des Systems besteht vorläufig aus kompletten durch Textdateien konfigurierbaren Ausführungsumgebungen (z. B. für Standalone und Web-Applikationen), sowie aus kommandozeilenorientierten Werkzeugen zum Import von Modellen aus CASE Tools.

3.1 Architektur der Entwicklungsumgebung

Die Vesuf Entwicklungsumgebung setzt sich aus mehreren Teilkomponenten zusammen (s. Abb. 3). Zentrale Komponente ist ein *Interpreter*, der dynamisch Benutzungsschnittstellen aus *UI-Spezifikationen* erzeugt, und diese nahtlos mit einer *Implementation* des Anwendungskerns integriert. Zur Spezifikation der Modelle dienen *Design-time Tools*, unvollständige Anwendungsmodelle können

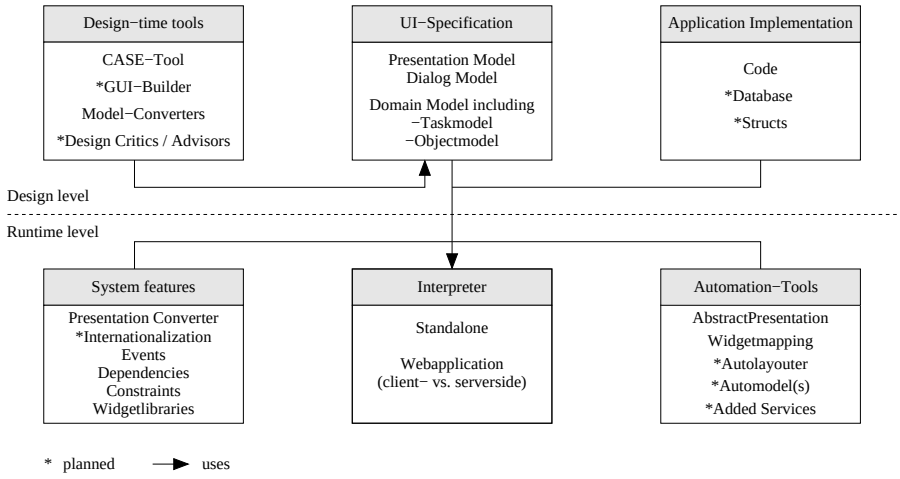

Abbildung 3: Architektur der Entwicklungsumgebung

vom Interpreter zur Laufzeit unter Zuhilfenahme von *Automatisierungswerkzeugen* vervollständigt werden. Zur Integration von Benutzungsschnittstelle und Anwendungskern verwendet der Interpreter auf Programmiersprachenebene frameworkartig implementierte *Systemfeatures*. Wichtigste Eigenschaft dieser Umgebung ist die Offenheit bezüglich weiterer Design-time Tools, Automatisierungswerkzeuge und Systemfeatures, mit deren Hilfe beliebige Techniken und Werkzeuge unterschiedlicher Abstraktionsniveaus (vgl. Abschnitt 2) in die Umgebung integriert werden können.

Vesuf verwendet zur Spezifikation eines User Interfaces zur Zeit vier Modelle: Grundlage des *Objektmodells* (object model) ist das UML Klassendiagramm. Das *Aufgabenmodell* (task model), das zur Beschreibung von Aufgaben dient, die ein Benutzer durch Interaktion mit dem System erledigen möchte, adaptiert in der aktuellen Version direkt die Semantik der UML Use Cases. Mit Hilfe des *Dialogmodells* (dialog model) wird in Vesuf die Abfolgesteuerung der Sichten in der UML Statechart Semantik spezifiziert. Das *Präsentationsmodell* (presentation model) dient zur Beschreibung des Inhalts und Aussehens einzelner Interaktionskomponenten mit UIML. Über einen Applikationsdeskriptor werden die zu verwendenden Modelle einer Anwendung und ihre Verbindungen untereinander festgelegt.

Die Werkzeuge der Vesuf Umgebung lassen sich in zwei Kategorien einteilen: Design-time und Runtime-Tools. In die Gruppe der Design-time Tools fallen alle Werkzeuge, die den Entwickler bei der Spezifikation der Modelle unterstützen. Mit Hilfe von Standard UML CASE Tools lassen sich die Domänenmodelle und das Dialogmodell editieren. Alternativ können Domänen- und Dialogmodell auch direkt als Java-Sourcefile unter Verwendung der Konstruktoren der Metamodellelemente definiert werden. Um ein Präsentationsmodell für das Vesuf System zu erstellen, kann entweder ein UIML-konformes File mit einem Texteditor beschrieben werden oder wiederum auf die Spezifikation mittels eines Java-Sourcefiles zurückgegriffen werden.

In die zweite Kategorie fallen Werkzeuge, die zur Laufzeit verwendet werden. Als Systemfeatures können z. B. Abhängigkeiten (dependencies) zwischen Modellelementen definiert werden, die dann zur Konsistenzwahrung innerhalb des User Interfaces verwendet werden. Ausserdem lassen sich Restriktionen (constraints) einsetzen, die ebenfalls dynamisch verwaltet werden und

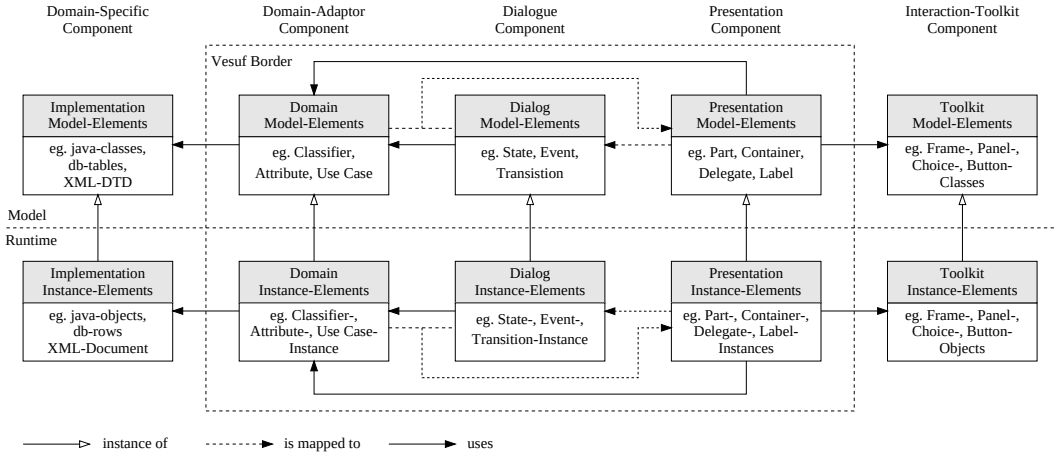

Abbildung 4: Laufzeitarchitektur

z. B. Benutzereingaben validieren. Automatisierungswerkzeuge (automation tools) inferieren einen bestimmten, nicht modellierten Aspekt der Applikation aus vorhandenem Wissen. Basis dieser Schlussfolgerungen sind einerseits in den übrigen Modellen vorhandene Informationen und andererseits vorgegebene Regeln und Wissensbasen. Geht man beispielsweise davon aus, dass kein Präsentationsmodell spezifiziert wurde, verwendet das System zunächst ein Werkzeug, welches die Inhalte einer darzustellenden Domänenentität bestimmt (abstract presentation tool). Im Anschluss daran können die Oberflächenbausteine und ihre Anordnung errechnet werden (widgetmapping, auto layout). Resultat ist eine ablauffähige Oberfläche.

3.2 Laufzeitarchitektur

Die Laufzeitarchitektur ist an das Arch Modell (Bass et al. 1992) angelehnt und besitzt wie dieses fünf separate Schichten (s. Abb. 4). Grundlegendes Konzept ist die strenge Separierung zwischen Modell- und Laufzeitentitäten (*model*, *runtime*) in allen Schichten, die auch von Kent et al. als Basis des UML-Metamodells vorgeschlagen wird (Kent et al. 1999). Modellelemente (*Model-Elements*) der Vesuf Umgebung werden eingesetzt, um die verschiedenen Modelle einer Anwendung zu beschreiben, wohingegen Laufzeitelemente (*Instance-Elements*) konkrete Ausprägungen solcher Modellelemente während der Ausführung einer Applikation darstellen.

Zwischen den einzelnen Elementen in den Arch Schichten bestehen direkte Verbindungen (*uses*), die über Eigenschaften (tagged values) der Modellelemente spezifiziert werden. Damit werden die Adaptionschichten (Domäne und Präsentation) an die konkreten Implementationen (Anwendungskern und Toolkit) angebunden. Für Dialoge werden die korrespondierenden Domänenelemente festgelegt. Die Beziehung von Präsentations- zu Domänenelementen entspricht nicht dem Arch Modell und wird auf Ebene der Laufzeitelemente durch eine Visual-Proxy Architektur (Holub 1999) realisiert. Um größtmögliche Flexibilität und Wiederverwendbarkeit bei der Erstellung von Schnittstellen zu gewährleisten, wird die Auswahl von Präsentationselementen für Dialoge und korrespondierende Domänenelemente nicht in den Modellen, sondern durch extern zu spezifiziernde

Abbildungen (*is mapped to*) festgelegt. Des weiteren können Präsentationselemente (z. B. Buttons) auf Elemente zur Dialogsteuerung (events) abgebildet werden. Diese Architektur ermöglicht die Spezifikation von Benutzungsschnittstellen mit geringem Aufwand und erlaubt dennoch weitgehende Freiheiten bei den verwendeten Werkzeugen und Techniken. Für eine genaue Beschreibung der Umgebung und ihrer Eigenschaften siehe auch (Braubach et al. 2002).

4 Konkreter Einsatz des Systems

Um die Praxistauglichkeit des Vesuf Systems nachzuweisen, wurde es als Präsentationskomponente des PublicationPORTALs[1] (Bartelt et al. 2001) eingesetzt. Der Zugriff auf die zu integrierenden Dienste erfolgt dabei über in Java zu implementierende Proxy-Komponenten, deren Benutzungsschnittstelle durch das Vesuf System erzeugt werden. Die resultierenden Komponenten werden in sogenannten Portlets als einzelne unabhängige Teilbereiche in die Darstellung des Gesamtportals eingebunden.

Zur Konstruktion der Dienste mit Vesuf wird eine erfolgreich eingesetzte Vorgehensweise beschrieben. Zunächst wurde eine elementare Taskanalyse durchgeführt, mit dem Ziel, die für den Benutzer relevanten Aufgaben zu isolieren. Anschließend wurde iterativ ein vollständiges Domänenmodell erarbeitet und parallel dazu bereits Teile der fachlichen Logik in Java implementiert. Im nächsten Schritt wurde die Dialogsteuerung auf Basis der Ergebnisse der Taskanalyse entworfen. Die Spezifikation der im Dialogmodell definierten Sichten erfolgte daraufhin in zwei konzeptionellen Stufen. Zunächst wurde festgelegt, welche Elemente das abstrakte Präsentationsmodell ausmachen. Dann wurde entschieden, durch welche konkreten Interaktionselemente die bereits festgelegten Interaktionsmöglichkeiten präsentiert werden sollen. Im letzten Schritt wurde ein Applikationsdeskriptor für jede Anwendung erstellt, der die zu verwendenden Modelle benennt.

Auf diese Weise konnten äußerst effizient Portlets für Z39.50 und einen Metawörterbuchdienst erstellt werden. Es hat sich herausgestellt, dass die Entwicklung von Portalapplikationen durch die Verwendung einer User Interface Komponente vereinfacht wird und die zusätzliche Einbindung anderer Interfacemodalitäten ohne Schwierigkeiten möglich ist. So wurden innerhalb kurzer Zeit grafisch interaktive (Java AWT), webbasierte (HTML), mobil zugreifbare (WML) und sprachbasierte (VoiceXML) Schnittstellen realisiert. Die gewählte Vorgehensweise läßt sich aufgrund der flexiblen Architektur des Systems für andere Anwendungen beliebig anpassen. Details zur Umsetzung der Dienste finden sich in (Braubach & Pokahr 2001).

5 Diskussion und Ausblick

Vesuf ist ein System zur Entwicklung von Benutzungsschnittstellen für Applikationen des UbiComp. Es ermöglicht die geräteunabhängige Anwendungsentwicklung und stellt universelle Zugriffsmöglichkeiten auf die Anwendungen sicher, was sich im Paxiseinsatz im Rahmen des Global Info Projekts bestätigt hat.

Konzeptionell basiert das Vesuf System auf dem Arch Modell und standardisierten Techniken zur Modellierung der Teilfunktionalitäten (insbesondere UML). Die gewählte Schichtenarchitektur ermöglicht die flexible Umsetzung von Benutzungsschnittstellen losgelöst von konkreten Implementationstechnologien und Zwängen bestimmter Anwendungskontexte. In den einzelnen Schichten

[1] Man findet das PublicationPORTAL unter: `http://pportal.informatik.uni-hamburg.de`

mit klar voneinander abgegrenzten Funktionalitäten können Teile von Anwendungen unabhängig voneinander entwickelt werden. Dem Entwickler bleibt dabei selbst überlassen, welche Schichten einer Anwendung geräteunabhängig entworfen und welche spezifischen Endgeräten angepasst werden.

Das Vesuf System stellt damit eine universelle Plattform zur Verfügung, die die Ausführung beliebiger Anwendungen als Dienste ermöglicht. Weiterhin erleichtert das System durch diverse in Systemfeatures und Automatisierungswerkzeuge integrierte Konzepte die Applikationsentwicklung.

Das Vesuf System kann durch eine Reihe von weiteren Werkzeugen in seinem Komfort und in den Anwendungsmöglichkeiten ausgebaut werden, wie bereits in Abb. 3 dargestellt. Dazu gehört einerseits die Ergänzung der Design-time Tools insbesondere durch ein Werkzeug zur visuellen Gestaltung von Oberflächen (Interface Builder) und andererseits die Komplettierung der Palette von Automatisierungswerkzeugen um den *slinky automation* Gedanken vollständig umsetzen zu können. Ferner ist es möglich weitere Implementationstechniken wie z.B. Datenbanken durch Vesuf zu unterstützen.

Es gibt eine Reihe weiterer Aspekte der User Interface Entwicklung für das UbiComp, für die der modellbasierte Ansatz prädestiniert erscheint. Unter anderem sind dies Adaption, Komposition und Multimodalität. Zur dynamischen Adaption von Benutzungsschnittstellen können z. B. Umgebungs-, Benutzer- und Gerätemodelle in Vesuf integriert werden, um die gezielte Anpassung laufender Anwendungen an aktuelle Gegebenheiten zu ermöglichen. Die deklarative Natur aller Vesuf Modelle erleichert die Entwicklung von Strategien zur dynamischen Komposition von spezifizierten Anwendungen und stellt eine Ausgangsbasis zur Implementation von Multimodalität dar.

Aufgrund seiner offenen, auf Standards basierenden Architektur und der generischen Ausrichtung ist das Vesuf System in idealer Weise als Grundlage für weitere Forschungsprojekte verwendbar. Als Open Source Projekt steht es unter der Adresse `http://vesuf.sourceforge.net` einer breiten Öffentlichkeit zur Verfügung.

Literatur

Banavar, G.; Beck, J.; Gluzberg, E.; Munson, J.; Sussman, J. und Zukowski, D. (2000): An Application Model for Pervasive Computing. In: *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (MOBICOM-00)*, S. 266–274, N. Y. ACM Press.

Bartelt, A.; Faensen, D.; Faulstich, L.; Schallehn, E. und Zirpins, C. (2001): Building Infrastructures for Digital Libraries. In: *DELOS Workshop on Interoperability in Digital Libraries*, volume No. 01/W06. ERCIM Workshop Proceedings.

Bass, L.; Faneuf, R.; Little, R.; Mayer, N.; Pellegrino, B.; Reed, S.; Seacord, R.; Sheppard, S. und Szczur, M. R. (1992): A Metamodel for the Runtime Architecture of an Interactive System. *ACM SIGCHI Bulletin*, 24(1):32–37.

Braubach, L. und Pokahr, A. (2001): Vesuf, eine modellbasierte User Interface Entwicklungsumgebung für das Ubiquitous Computing, vorgestellt anhand der Fallstudie PublicationPORTAL. Diplomarbeit, Universität Hamburg.

Braubach, L.; Pokahr, A.; Moldt, D.; Bartelt, A. und Lamersdorf, W. (2002): Tool-Supported Interpreter-Based User Interface Architecture for Ubiquitous Computing. In: Forbrig, P. und Vanderdonckt, J. (Hrsg.): *Proceedings of DSV-IS'2002*. Springer Verlag. To appear.

Burbeck, S. (1992): Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC). http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html.

Calvary, G.; Coutaz, J. und Nigay, L. (1997): From Single-User Architectural Design to PAC*: A Generic Software Architecture Model for CSCW. In: *Proceedings of ACM CHI 97 Conference on Human Factors in Computing Systems*, volume 1 of *PAPERS: Beyond the Desktop*, S. 242–249.

Coutaz, J. (1987): PAC: An Object Oriented Model for Implementing User Interfaces. *ACM SIGCHI Bulletin*, 19(2):37–41.

da Silva, P. P. (2001): User Interface Declarative Models and Development Environments: A Survey. In: Palanque, P. und Paterno, F. (Hrsg.): *Proceedings of DSV-IS'2000*, S. 207–226. Springer Verlag.

Fähnrich, K.-P. (2000): *Methoden und Werkzeuge zur softwareergonomischen Entwicklung von In-for-mations-sys-te-men*. Jost-Jetter Verlag.

Gallis, H.; Petter, J. und Herstad, J. (2001): The multidevice paradigm in Knowmobile – Does one size fit all? Department of Informatics, University of Oslo.

Global-Info (2001): Globale Elektronische und Multimediale Informationssysteme für Naturwissenschaft und Technik des bmb+f. Bundesministerium für Bildung und Forschung (bmb+f), http://www.global-info.org.

Green, M. (1985): Report on Dialogue Specification Tools. In: Pfaff, G. E. (Hrsg.): *User Interface Management Systems: Proceedings of the Seeheim Workshop*, S. 9–20, Berlin. Springer-Verlag.

Grimm, R.; Davis, J.; Hendrickson, B.; Lemar, E.; MacBeth, A.; Swanson, S.; Anderson, T.; Bershad, B.; Borriello, G.; Gribble, S. und Wetherall, D. (2001): Systems Directions for Pervasive Computing. In: *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, S. 128–132.

Holub, A. (1999): Building user interfaces for object-oriented systems, Part 2: The visual-proxy architecture. *JavaWorld*.

Kazman, R. und Bass, L. (1996): Software Architectures for Human-Computer Interaction: Analysis and Construction. Submitted to ACM Transactions on Human-Computer Interaction.

Kent, S.; Evans, A. und Rumpe, B. (1999): UML Semantics FAQ. In: Moreira, A. und Demeyer, S. (Hrsg.): *ECOOP'99 Workshop Reader*, volume 1743 of *LNCS*, S. 33–56. Springer-Verlag.

Myers, B. A. (1989): User-interface tools: Introduction and survey. *IEEE Software*, 6(1):15–23.

Szekely P. (1996): Retrospective and Challenges for Model-Based Interface Development. In: Bodart, F. und Vanderdonckt, J. (Hrsg.): *Proceedings of DSV-IS'96*, Eurographics, S. 1–27, Wien. Springer-Verlag.

UIML 2.0a (2000): *User Interface Markup Language Specification, version 2.0a*. Harmonia Inc.

UML 1.4 (2001): *Unified Modelling Language Specification, version 1.4*. Object Modeling Group.

Kontakt

Alexander Pokahr, Lars Braubach, Andreas Bartelt, Daniel Moldt, Winfried Lamersdorf
Universität Hamburg, Fachbereich Informatik
Vogt-Kölln-Straße 30, 22527 Hamburg

Email: {pokahr, braubach, bartelt, moldt, lamersd}@informatik.uni-hamburg.de
http://vsis-www.informatik.uni-hamburg.de/