Alexander Behring, Matthias Heinrich, Matthias Winkler und Waltenegus Dargie

EMODE – Modellgetriebene Entwicklung multimodaler, kontextsensitiver Anwendungen

EMODE – Model-driven Development of Multimodal, Context Sensitive Applications

MDSE_Interactive Systems_MDA_Multimodal_Context Sensitive_User Interface

Zusammenfassung. Die Entwicklung multimodaler, kontextsensitiver Anwendungen gewinnt zunehmend an Interesse. Jedoch stellen diese höhere Anforderungen an den Softwareentwicklungsprozess. In diesem Beitrag werden die Arbeiten und Ergebnisse aus dem EMODE-Projekt vorgestellt, welches sich die Verbesserung der Effizienz der Entwicklung multimodaler, kontextsensitiver Anwendungen zum Ziel gesetzt hat. Dabei nutzt EMODE modellbasierte Entwicklung; wobei insbesondere die Integration der verschiedenen Entwicklungsschritte und eine durchgehende Werkzeugunterstützung betont werden.

Summary. Interest in development of multimodal, context sensitive applications is growing. But these applications have higher demands on the software development process. In this article, the work and results of the EMODE project are introduced. The project aims at improving the efficiency of developing multimodal, context sensitive applications. EMODE builds upon model driven engineering and focuses on the integration of the various development steps, as well as an end-to-end tool chain support.

1. Einleitung

Die Entwicklung der IT-Industrie ist gerade in der letzten Dekade von einem enormen Fortschritt und Wandel geprägt. So gibt es ein verstärktes Interesse an der Entwicklung multimodaler[1] Benutzerschnittstellen (UIs) (Oviatt 2003). Sie besitzen ein breites Spektrum potenzieller Anwendungen, aufgeführt z.B. in W3C (2007) oder Wahlster und Weyrich (2005). Auch die Anpassung von UIs an unterschiedliche Kontexte wird dabei diskutiert.

Kontext kann, wie in Dey, Abowd und Salber (2001) definiert[2], hierbei als „jegliche Information, die genutzt werden kann um die Situation von Entitäten (d.h. Person, Ort oder Objekt) zu charakterisieren und als relevant für die Interaktion zwischen Nutzer und Applikation betrachtet wird, inbegriffen dem Nutzer und der Applikation selbst." (Übersetzung der Autoren) verstanden werden. Gängig ist unter anderem die Nutzung der Kontextinformation zur Weiterverarbeitung im Programm oder auch als Treiber der Adaptierung von Programm und Benutzerschnittstelle (UI) (z. B. die Wahl der Modalität, sodass bei einer lauten Umgebung keine Sprach- sondern grafische Ausgabe genutzt wird).

Multimodalität und Kontextsensitivität führen zu höheren Anforderungen an den Softwareentwicklungsprozess. User Interface Software Tools (UISTs) haben eine lange Geschichte bis zurück in die 70er Jahre. Myers, Hudson und Pausch (2000) geben einen exzellenten Überblick und analysieren, warum verschiedene Konzepte gescheitert sind und andere überlebten. Sie sehen in modellbasierten Ansätzen und Automatisierungstechniken zur UI-Erzeugung eine Möglichkeit der wachsenden Vielfalt an Interaktions-Geräten gerecht zu werden, während sie sie konventionelle GUI Ansätze für nicht geeignet halten. Solche UISTs auf Basis deklarativer Modelle wurden 2000 von da Silva untersucht (da Silva 2000). Der Fokus der dabei untersuchten Systeme lag jedoch auf dem GUI-Bereich. Multimodalität wurde erst in späteren UIST Ansätzen eingehender betrachtet.

Ein weiteres Problem bei der Erstellung interaktiver Systeme ist die Integration von UI- und Anwendungslogik, welche sich gegenseitig stark beeinflussen. Begründet liegt dies vor allem auch in den unterschiedlichen Herangehensweisen zur Bearbeitung der zwei Aspekte. So wird für die Entwicklung der Anwen-

DOI 10.1524/icom.2007.6.3.31

[1] Im Rahmen dieses Beitrags wird damit „Interaktion über verschiedene Geräte (z.B. PDA, Fernseher, PC) und Interaktionsformen (z.B. Gesten, Sprache)" bezeichnet.

[2] Der Begriff des Kontexts ist bei verschiedenen Autoren unterschiedlich definiert.

dungslogik oft ein objekt-orientierter Ansatz genutzt, wohingegen für die UI-Entwicklung eher weniger formale Task-Modelle zum Einsatz kommen. Als Resultat vollzieht sich die Entwicklung dieser beiden „Seiten" bisher weitgehend entkoppelt voneinander. Dies äußert sich wohl am deutlichsten in einer unterbrochenen Werkzeugunterstützung und nicht zusammenhängenden Sprachen für die beiden Aspekte. Der Grad der Integration beeinflusst insbesondere auch die Effizienz der Entwicklung, welche bei der stetig wachsenden Zahl von zu unterstützenden Geräten wichtiger werden wird.

2. Aktuelle Arbeiten

Um die Entwicklung von UI und Anwendungslogik enger zu verzahnen besteht eine Möglichkeit darin, Task-Modelle mehr zu formalisieren und die Modellierungen zu integrieren. ConcurTaskTrees (CTT), vorgestellt in Paternò, Mancini und Meniconi (1997), gilt als formellere Notation und wird daher in vielen aktuellen Arbeiten als Basis genutzt. Es erlaubt die Beschreibung von Aufgaben des Nutzers, des Systems und der Interaktion, sowie deren Abhängigkeiten in einem Task-Modell. Neben dem Task-Modell wird oft ein UML-artiges Klassen-Modell der Anwendung erstellt; des Weiteren werden ein Modell des abstrakten UIs (AUI), und ein Modell des konkreten UIs (CUI) genutzt (da Silva 2000).

Die Unterstützung der verschiedenen Modelle ist bei unterschiedlichen Ansätzen unterschiedlich ausgeprägt. CTT wird durch das Modellierungstool Teresa unterstützt, jedoch liegt der Fokus auf dem Task Modell. Teresa bietet keine Möglichkeit, ein Anwendungsmodell zu erstellen (und die Notation erlaubt dies nur implizit im Task-Modell). Neben CTT stellt Teresa XML eine Notation für AUIs bereit. Verfügbare Elemente zur UI Modellierung sind hierbei fest im Metamodell kodiert. Teresa erlaubt die automatische Erstellung von CUIs und AUIs aus dem Task-Modell. Für das CUI wird ein von Teresa definierter Satz von Zielplattformen angeboten, zusammen mit Regeln, wie die abstrakten in konkrete Elemente überführt werden. Da der Fokus auf der Erstellung von UIs liegt, wird die Generierung von Anwendungscode oder Methodenrümpfen nicht unterstützt. Verknüpfungen zwischen AUI/CUI- und Task-Modell können folglich nicht erstellt werden, sind im CTT Ansatz aber auch nicht nötigt.

Auf CTT aufbauend stellen Limbourg et al (2004) die Sprache usiXML[3] vor. UsiXML integriert CTT und UI Modelle in einem umfangreichen, XML-basierten Metamodell. Die Werkzeugunterstützung ist vor allem in den Bereichen Erstellung des UIs und deren Interpretation sehr umfangreich, aber nicht integriert und durchgängig. Die nutzbaren Elemente für CUI Modelle verschiedenster Art (grafisch, Sprache, etc.) sind auch bei usiXML fest im Metamodell kodiert, wobei usiXML nicht zu detailiert auf die Spezifika der einzelnen Elemente eingehen möchte[4], sondern auf abstrakteren, gemeinsamen Eigenschaften der Elemente seinen Fokus setzt. Für die Verbindung zur Anwendungslogik werden zwar im Metamodell Elemente bereitgestellt, es ist uns aber keine Werkzeugunterstützung zur Nutzung dieser bekannt. Des Weiteren befinden sich diese Elemente auf CUI Ebene und erlauben nur das Aufrufen von Methoden aus dem UI und nicht die Spezifikation von komplexen Abläufen. UsiXML bietet eine gute Integration von Transformationen (Graphgrammatiken), welche direkt im Modell spezifiziert werden können.

Die „andere Seite", also die Entwicklung der Anwendungslogik, adressiert UML. Hierin lässt sich die „System-Seite" aus verschiedenen Blickwinkeln umfangreich beleuchten. Allerdings fehlt die Entwicklung von UIs vollständig. Somit muss auch UML mit anderen Werkzeugen kombiniert werden, um eine vollständige Anwendungsentwicklung durchzuführen. Ansätze dazu sind z. B. UMLi (da Silva und Paton 2003) und Wisdom (Nunes und Cunha 2000). Wisdom integriert CTT und ein abstraktes UI Modell als Profil in UML, jedoch kein CUI-Modell. UMLi dagegen erweitert UML's Aktivitätsdiagramme zur Task-Modellierung und stellt ein AUI sowie CUI Modell bereit. Beide Ansätze legen dabei ihren Schwerpunkt auf GUIs.

Bei der Integration von Kontext unterscheiden wir zwischen Gewinnung und Nutzung. Während die Gewinnung Ableitungskomponenten, wie z. B. Context-Widgets aus (Dey, Abowd und Salber 2001), in den Vordergrund stellt, ist bei der Nutzung die Fragestellung, wie die Reaktion der Anwendung auf eine Kontext-Änderung, oder die Integration von Kontext-Daten formuliert wird. Die Schnittstelle zwischen Gewinnung und Nutzung bildet ein Modell der Elemente, welche den Kontext darstellen (z. B. „Gemütszustand des Fahrers"), wie z.B. beim Bib3R-Dienst (Springer et al 2006). Diese Modellierung kann in einem separaten Kontext-Domänen-Modell geschehen, oder integriert mit dem Anwendungsmodell.

Modell-basierte Ansätze zur Nutzung von Kontext wurden in Van den Bergh und Coninx (2004) studiert. CTT/Teresa bietet dafür das Filtern eines annotierten Ausgangsmodells an. Andere Ansätze (z. B. usiXML) erlauben das Spezifizieren separater Modelle für unterschiedliche Kontexte und Verlinken der Elemente dieser. Die UML-basierten Ansätze UMLi und Wisdom hingegen bieten keinen bzw. nur minimale Unterstützung. Einschränkend wirkt, dass bei den betrachteten Ansätzen keine Modellierung von Kontext-Elementen unterstützt wird. Somit macht auch die Definition von Ableitungskomponenten bei diesen Anwendungen keinen Sinn – und wird folglich nicht explizit unterstützt.

3. EMODE

Wie bisherige Ansätze auch, baut EMODE auf einen modellgetriebenen Softwareentwicklungsansatz. EMODE setzt sich gezielt die Verbesserung der Effizienz der Entwicklung multimodaler, kontextsensitiver Anwendungen zum Ziel. Dabei grenzt sich EMODE insbesondere von anderen Ansätzen durch den Anspruch auf umfangreichere Unterstützung und Integration der einzelnen Entwicklungsschritte ab.

Dafür wurde eine Werkzeugkette entwickelt, welche die Entwicklung von Benutzerschnittstellen mit Hilfe der aus der Literatur bekannten Modelle unterstützt. Doch während in anderen Ansätzen die Entwicklung des CUI-Modells das Ziel ist, verwendet EMODE alle Modelle, inbegriffen Kontext, zur Code-Generierung. Der generierte Code schließlich wird in

[3] http://www.usixml.org
[4] lt. Webseite von usiXML (Letzter Zugriff 30.09.2007)

der EMODE Laufzeit (nicht Fokus dieses Beitrags) ausgeführt.

Transformationensbeschreibungen zur Unterstützung des Entwicklers sind in anderen Ansätzen oft in Werkzeugen kodiert (Teresa, usiXML) oder in nicht standardisierten Sprachen (usiXML) ausgedrückt. Dem wirkt EMODE entgegen, indem alle Transformationen in QVT, einem Standard der OMG, beschrieben sind. Des Weiteren löst EMODE die Dichotomie von CUI und AUI auf, welche in allen anderen Ansätzen besteht. UI Zielsprachen[5] wie D3ML (Göbel et al), XHTML und Java AWT lassen sich unserer Meinung nach schwer auf ein gemeinsames Abstraktionslevel stellen – vielmehr ist es somit adäquat, beliebige Level zuzulassen. Um dabei flexibel zu bleiben, kodiert EMODE keine UI Elemente von UI Zielsprachen im Metamodell, wie in anderen Ansätzen geschehen, vielmehr erlaubt ein Klasse-Instanz Konzept die einfache Erweiterung mit neuen Elementen.

Nachfolgend wird der Ansatz von EMODE anhand der entwickelten Methodik vorgestellt. Des Weiteren wird eine Werkzeugkette beschrieben, die den EMODE-Ansatz implementiert.

3.1 EMODE Methodik

Die EMODE Methodik beschreibt einen Ansatz für die Entwicklung multimodaler, kontextsensitiver Anwendungen. Sie definiert eine Reihe von Entwicklungsphasen, verschiedene Modelle, die während dieser Phasen entwickelt werden, und Transformationen, die den Entwicklungsprozess unterstützen (siehe Bild 1). Ein besonderes Augenmerk liegt hier auf der Einbeziehung von Anwendern, UI Designern und Anwendungsentwicklern, deren Interessen und Fähigkeiten während des Entwicklungsprozesses berücksichtigt und integriert werden müssen.

Während des Entwicklungsprozesses werden vier verschiedene Phasen durchlaufen, in denen unterschiedliche Modelle erstellt oder durch eine Transformation gewonnen und verfeinert werden. In der „Requirements Analysis" Phase wird das Goal-Modell erstellt, welches die funktionalen Anforderungen der Anwendung beschreibt. Alle Anforderungen müssen gemeinsam mit Anwendern, den Experten der Anwendungsdomäne, erfasst werden. Die funktionalen Anforderungen und deren Relationen untereinander bilden die Grundlage für das Task-Modell.

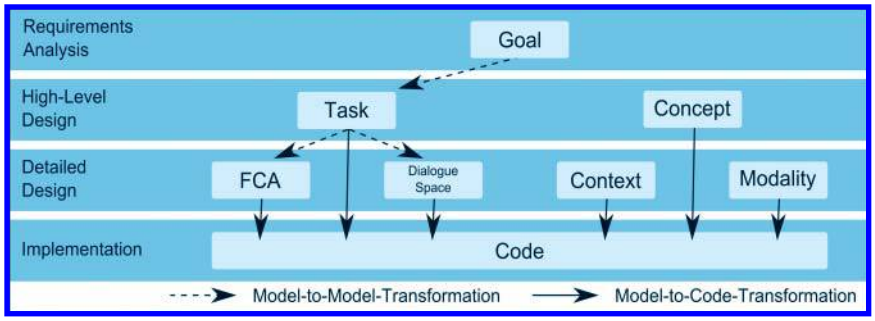

Bild 1: Phasen, Artefakte und Transformationen der EMODE Methodik

In der „High-level Design" Phase wird das Task-Modell bearbeitet. Darin wird der Ablauf der zu modellierenden Anwendung in Form von System-Tasks und Interaction-Tasks erfasst, sowie deren Abhängigkeiten zueinander durch Kontrollflüsse abgebildet. Des Weiteren werden in dieser zweiten Phase die benötigten Anwendungskonzepte im Concept-Modell modelliert und in das Task-Modell integriert. Das fertige Task-Modell stellt den gesamten Ablauf der Anwendung dar.

Aus dem Task-Modell werden in der „Detailed Design" Phase das DialogueSpace-Modell und das FCA-Modell[6] erstellt. Das UI wird von einem UI-Experten modelliert, indem im DialogueSpace-Modell eine initiale, durch Transformation generierte UI Beschreibung schrittweise für verschiedene Modalitäten verfeinert wird (vgl. dazu Abschnitt 4.2). Ein Anwendungsentwickler kann über das FCA-Modell Anwendungslogik anbinden. Besonders wichtig ist hier die Beziehung der Modelle Task, DialogueSpace und FCA. Veränderungen (z. B. das Entfernen eines Tasks) werden an die jeweils anderen Modelle propagiert um für Konsistenz zu sorgen. Weiter werden in dieser Phase Kontextprovider (Dienste, die der Anwendung Kontextinformation verfügbar machen) modelliert, und festgelegt, welche Modalitäten die Anwendung unterstützt.

In der Implementation Phase wird der Quellcode der Anwendung erstellt. Dies geschieht durch eine Modell-zu-Code Transformation bei der fast alle Modelle als Informationsquelle dienen (siehe Bild 1). Die Implementierung der Anwendungslogik in die vorgenerierten Klassen muss von Hand geschehen.

Für den Modellierungsprozess sind vor allem die Modelle Task, DialogueSpace, FCA und Concept wichtig, sie bilden die Kernmodelle mit denen es möglich ist einfache Anwendungen zu erstellen. Um mehr Funktionalität zu erreichen (z. B. Kontextsensitivität) sind allerdings weitere Modelle (z. B. Context-Modell) notwendig. An einem Beispiel wird die Anwendungsmodellierung in Abschnitt 4 näher aufgezeigt.

Auf Basis der EMODE Methodik wurde eine Werkzeugkette entwickelt, die nun kurz beschrieben wird.

3.2 Bereitgestellte Werkzeuge

Die EMODE-Entwicklungsumgebung stellt eine Reihe von Werkzeugen bereit, mit deren Hilfe multimodale, adaptive Anwendungen auf Basis der EMODE-Methodik erstellt werden können.

Das Editieren von Modellen in EMODE erfolgt durch grafische Editoren, in gewohnter Drag'n'Drop-Manier. Die Modelle werden in einem zentralen Modellrepository abgespeichert. Modell-zu-Modell und Modell-zu-Code Transformationen werden von integrierten Transformationsengines ausgeführt. Alle Editoren, das Modellrepository und Transformationen sind in einer Umgebung, Eclipse, integriert, welche auch die Bearbeitung des generierten Codes erlaubt. Somit kann der gesamte Entwicklungsprozess nahtlos durchgeführt werden.

4. Beispielanwendung

Anhand einer Beispielanwendung wird nun die EMODE-Methodik in leicht verkürzter Darstellung erläutert. Die zu entwickelnde Beispielanwendung unterstützt Instandhaltungsmitarbeiter, die mit

[5] D. h. die Beschreibungssprache in welcher das UI am Ende der Entwicklung vorliegt.

[6] FCA – Functional Core Adapter

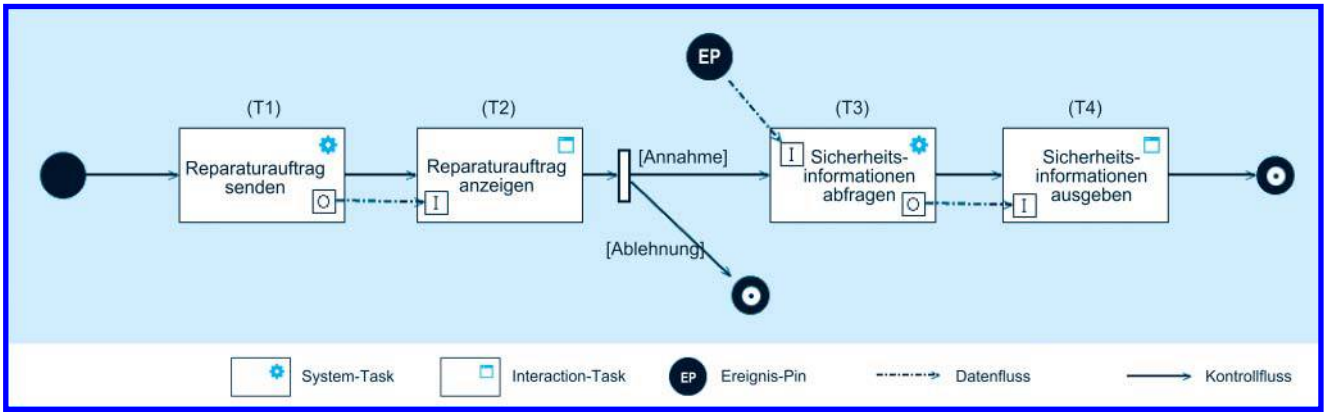

Bild 2: Task-Modell der Beispielanwendung

der Aufgabe betraut sind, den Maschinenpark einer Firma zu warten und anfallende Reparaturen durchzuführen. Im Folgenden werden die vorausgesetzte technische Infrastruktur und die Zielstellung der Beispielanwendung beschrieben.

4.1 Szenariobeschreibung

Eine Aufgabe der Instandhaltung ist die schnellstmögliche Reparatur aufgetretener Störungen. Die genutzte Infrastruktur besteht aus einer Monitoring-Komponente, einem zentralen Server und einer Menge mobiler Endgeräte, welche die Instandhaltungsmitarbeiter bei ihrer Arbeit unterstützen und deren Kommunikation mit dem Server ermöglichen. Sobald die Monitoring-Komponente Störungen in der Produktionshalle feststellt, wird ein Reparaturauftrag an den Server übermittelt. Infolgedessen werden alle Instandhaltungsmitarbeiter über die aufgetretene Störung informiert. Akzeptiert ein Arbeiter den Reparaturauftrag, ändert sich der Auftragsstatus und die Reparaturarbeiten werden begonnen. Während der Reparatur wird der Arbeiter durch die Instandhaltungsanwendung über potentielle Gefahrenquellen, benötigte Utensilien, etc. unterrichtet. Dabei wird die Ausgabemodalität des UIs in Abhängigkeit vom Umfeld (z. B. Lärmpegel, Helligkeit) bestimmt.

4.2 Modellierungsschritte

Den Ausgangspunkt der Modellierung bildet das Anlegen des Task-Modells, welches im Wesentlichen das Verhalten der Applikation abbildet. Für die Task-Modellierung in EMODE wurden neben Elementen aus UML-Aktivitätsdiagrammen (Kontrollflüsse, Objektflüsse, Startknoten, etc.) das Task-Konzept aus dem ConcurTaskTree-Ansatz (Paternò 1997) übernommen. Demzufolge beinhaltet das Task-Modell insbesondere System- und Interaction-Tasks. System-Tasks kapseln Aufgaben, die ausschließlich vom System erbracht werden (z. B. Datenbankabfragen). Im Gegensatz dazu verlangen Interaction-Tasks eine Mensch-Maschine-Kommunikation (z. B. Texteingaben).

Unterschieden wird in EMODE zwischen Kontroll- und Datenabhängigkeiten („Flüssen") zwischen Tasks. Mit Hilfe von Kontrollflüssen wird gesteuert, in welcher Reihenfolge Tasks aktiv sind. Kontrollflüsse können durch Entscheidungs- und Vereinigungsknoten gesteuert werden. Wichtig für die späteren Schritte sind auch Datenflüsse, in Bild 2 gestrichelt dargestellt. Mit ihnen wird modelliert welche Daten zwischen welchen Tasks übergeben werden. So folgt im Szenario auf das Senden eines Reparaturauftrags des Systems (System-Task T1) die Anzeige des Auftrags beim Nutzer (Interaction-Task T2). Dabei wird zum Einen die Kontrolle von T1 an T2 weitergegeben, zum Anderen auch der Reparaturauftrag (modelliert im Concept-Modell). Datenübergabepunkte an Tasks (Pins) sind hierbei typisiert durch modellierte Elemente aus dem Concept-Modell.

Die Integration von Kontextdaten ist an System-Task T3 in Bild 2 zu sehen. Hierbei wird der Kontextdienst nach den aktuellen Umgebungsbedingungen gefragt. Dazu wurde im Context-Modell ein Provider modelliert, der die benötigte Information (z.B. Umgebungslautstärke) ableiten kann. Im Concept-Modell wurde vorher das Konzept der benötigten Information (also die Umgebungslautstärke) modelliert. Die Abfrage an den Kontextdienst ist im Element EP (Ereignis Pin) gekapselt. Die Benennung kommt daher, dass der Kontextdienst Daten auch asynchron (d. h. nicht nur bei Task-Start) an einen Task übermitteln kann. Auch die asynchrone Datenübermittlung zwischen Tasks wird von EMODE unterstützt.

Nicht zu sehen in Bild 2 ist die Möglichkeit Hierarchien zu erstellen – hierfür kann ein Task vom Typ „abstrakt" genutzt und in einem weiteren Modell verfeinert werden. Auch ist die Möglichkeit parallele Flüsse zu erstellen ist aus Platzgründen nicht in Bild 2 illustriert. Mit Hilfe von Verzweigungsknoten kann der Kontrollfluss gespalten und mit Vereinigungsknoten wieder vereint werden.

Im zweiten Entwicklungsschritt wird das FCA-Modell durch Modelltransformationen aus dem Task-Modell abgelei-

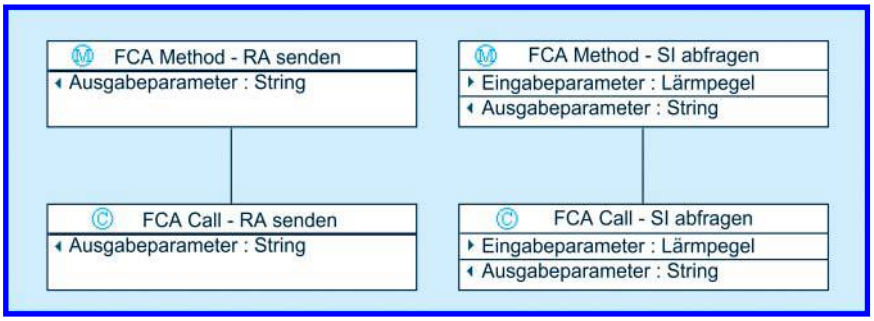

Bild 3: FCA-Modell der Beispielanwendung

tet. Ein FCA-Call wird dabei als Realisierung eines System-Tasks (d. h. Aufruf von Anwendungslogik) verstanden. Die Anwendungslogik liegt in einer Methode (FCA-Method), welche ggf. eine vom Call abweichende Signatur haben kann, z. B. bei Nutzung einer Methode durch mehrere verschiedene Calls. Das FCA-Modell verknüpft somit System-Tasks mit Methoden der Anwendungslogik und beschreibt deren Übersetzung ineinander durch Zuordnung der Parameter.

Das Resultat der Modell-zu-Modell-Transformation ist das generierte FCA-Modell in Bild 3. Dabei werden System-Tasks in FCA-Calls überführt und Task mit Call verknüpft, sowie dazu passende FCA-Methods generiert und mit den Calls verknüpft. Typisierte Pins im Task-Modell dienen als Quelle für Parameter im FCA-Modell. Beispielsweise wird im FCA-Modell in Bild 3 für den System-Task T1 der FCA-Call „RA senden" erzeugt und auf Grund des Ausgabe-Pins an T1 ein passender Parameter an den Call gehängt. Dazu wird eine entsprechende FCA-Method generiert, welche die Grundlage für die Code-Generierung des Methoden-Rumpfes bildet. Im gewählten Beispiel ist jedoch eine weitere Anpassung des FCA-Modells nicht notwendig.

Das ebenfalls aus der Modelltransformation hervorgegangene DialogueSpace-Modell in Bild 4 dient der Spezifikation multimodaler Benutzerschnittstellen. Dafür werden Interactor-Elemente (zu sehen in Bild 4) bereitgestellt, welche über ein Klasse-Instanz-Konzept einen Typ zugewiesen bekommen können (z. B. Container, Textausgabe, …). Bei der Transformation von Task nach DialgoueSpace werden im Wesentlichen Interaction-Tasks auf AUI-Interactors abgebildet und für Pins an den Interaction-Tasks geschachtelte Interaktoren erzeugt. So wird z. B. für Task T2 in Bild 2 der Interactor „Ausgabe von (T2)" erzeugt. Dieser wiederum enthält den Interactor „RA anzeigen", welcher auf Grund des Eingabepins „I" an T2 erzeugt wurde. Im Beispiel in Bild 4 wurden die Namen zur besseren Illustration geändert, sowie die Typen gesetzt.

In Bild 5 ist die Verfeinerung für unterschiedliche Modalitäten illustriert. Dabei wird ein Wurzel UI (dies ist i.A. das initial generierte und mit dem Task-Modell verknüpfte) für Sprache und GUI verfeinert. Bei der Verfeinerung werden die Typen und Eigenschaften der UI Elemente verändert, z. B. wird ein abstraktes „Wähle 1-aus-N" in „Combo-Box" geändert. Der Übergang von „UI für GUIs" zu „UI für PDA-GUI" kann z. B. die Größenangaben der GUI Elemente verfeinern, welche im „UI für GUIs" evtl. noch nicht gemacht wurden. Somit ist die schrittweise Anpassung an die gewünschten Modalitäten möglich. Information, welche für verschiedenen Zielmodelle gleich ist (in Bild 5 z. B. die Hintergrundfarbe für PDA und Desktop) kann auf passendem Abstraktionsniveau (im Beispiel „UI für GUIs") gegeben werden und nicht separat für jedes Zielmodell.

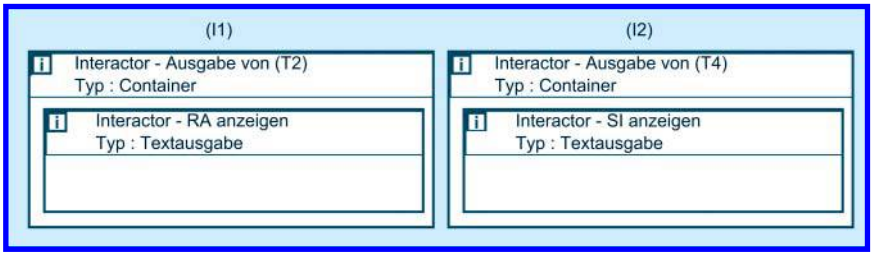

Bild 4: DialogueSpace-Modell der Beispielanwendung

Nach Abschluss der Modellierungsphase, kann aus den Modellen ausführbarer Code erzeugt werden. Die im DialogueSpace-Modell gefassten Benutzerschnittstellen können in verschiedene Zielsprachen (aktuell D3ML (Göbel et al 2006), Java AWT) transformiert werden. Daneben wird das Task-Modell in Java-Klassen überführt, welche durch die EMODE Laufzeitumgebung interpretiert werden. Für die im Context-Modell beschriebenen Kontextprovider zur Ableitung höherwertigen Kontexts werden Methodenrümpfe inkl. Anbindungslogik an den EMODE Kontextdienst erzeugt. Bevor die entworfene Beispielapplikation ausgeführt werden kann, ist ein letzter Schritt notwendig: die Implementierung der Anwendungslogik. Zu diesem Zweck sind die Java-Klassen, welche aus dem FCA- und Context-Modell hervorgegangen sind, mit Platzhaltern für die Implementierung versehen, wie in Bild 6 gezeigt.

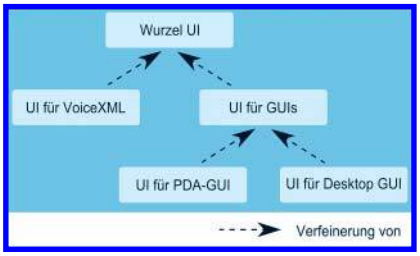

Bild 5: UI Verfeinerungsprozess

Ist die Anwendungslogik und ggf. die Kontextprovider implementiert, erfolgt das Deployment der Anwendung. Das Deployment beschränkt sich dabei auf das Hinzufügen speziell entwickelter Laufzeitbibliotheken zu den kompilierten Java-Klassen.

```
public class EmodeApplicationFca {

    public String getMaintenanceOrder() {

        String returnValue = "default";
        /* begin business logic */

        /* end business logic */
        return returnValue;
    }
}
```

Bild 6: Aus FCA-Method generierte Klasse

5. Ausblick und Fazit

In diesem Beitrag wurden Arbeiten aus dem EMODE-Projekt vorgestellt, das den Entwickler bei der Erstellung multimodaler, kontextsensitiver Anwendungen unterstützt. Zur Evaluation des EMODE-Projektes werden zur Zeit zwei Demonstratoren entwickelt, welche beide die Integration von Kontextwissen, als auch multimodale Interaktionen verlangen. Der erste Demonstrator bildet eine Anwendung aus der Instandhaltung von Industrieanlagen ab, wie bereits als Beispielanwendung in Kapitel 4.1 vorgestellt. Der zweite Demonstrator besteht aus einem Assistenten zur Reiseplanung, -durchführung und -auswertung. Dieser Assistent wird in einem Fahrzeug implementiert und unterstützt Sprachein- und Ausgabe, sowie grafische Ausgabe. GPS Daten aus dem Fahrzeug-Bus werden als Kontextinformation genutzt.

5.1 Fazit

EMODE unterscheidet sich insbesondere durch den hohen Integrationsgrad der Entwicklungsumgebung und im Umfang der Unterstützung von anderen Ansätzen. Im Gegensatz zu anderen Ansätzen

kann mit dieser integrierten Umgebung eine komplette, lauffähige Anwendung, inkl. Kontextanbindung und multimodalen UIs erstellt und ausgeführt werden. Dabei erlaubt EMODE nicht nur die Nutzung einfacher Kontextinformation, sondern auch die Ableitung höherwertigen Kontextwissens.

Mit der Auflösung der strikten Trennung von AUI und CUI in das DialogueSpace-Modell, stellt EMODE eine Möglichkeit bereit, UIs auf beliebigem Abstraktionsniveau zu beschreiben, und diese schrittweise, über mehr als eine Stufe zu verfeinern. Damit wird auch die große Variabilität im Abstraktionslevel von möglichen Sprachen für Zielmodalitäten adressiert. Essentiell ist dabei der Verzicht auf UI Elementtypen im Metamodell. Stattdessen kann EMODE durch ein Klasse-Instanz Konzept mit weiteren UI Elementtypen ohne Metamodelländerung erweitert werden. Zurzeit wird auch an einer Ontologie-Unterstützung der UI-Verfeinerung gearbeitet – sie soll UI Elementtypen bzw. deren Eigenschaften miteinander in Beziehung setzen und einen höheren Automatisierungsgrad für Transformationen ermöglichen.

Verschiedene Test-Anwendungen wurden bereits mit EMODE erfolgreich gebaut. Erste Ergebnisse sind:

- EMODE bietet eine umfangreiche Unterstützung des Entwicklers - von der UI Entwicklung bis hin zur Integration der Anwendungslogik.
- Die Modell-zu-Modell-Transformationen eignen sich um Modellierungs-Aufgaben zu automatisieren. Zu beachten ist jedoch die Komplexität deklarativer Transformations-Beschreibung.
- Die Integration der verschiedenen Editoren wirkt sich positiv auf die Entwicklungsgeschwindigkeit aus. Insbesondere werden Änderungen an einem Modell, welche ein anderes Modell beeinflussen, sofort sichtbar und ermöglichen somit kurze Iterationszyklen.

Verschiedene Aspekte, wie z. B. Verteilung von UIs, Benutzer-Modelle, Gruppen-Modelle, werden in Zukunft stärker in der Softwareentwicklung berücksichtigt werden müssen und erfordern ein entsprechendes methodisches Vorgehen. Modell-getriebene Ansätze sind ein vielversprechender Weg um dieser zunehmenden Komplexität zu begegnen.

Danksagung

Die Autoren und das EMODE-Konsortium bedanken sich für die Förderung durch das BMBF im Rahmen von ITEA. Der Dank der Autoren gilt auch denen, die durch Ihre hilfreichen Kommentare diesen Beitrag verbessert haben.

Literatur

Da Silva, P.P.: User Interface Declarative Models and Development Environments: A Survey. *Lecture Notes in Computer* Science **1946** (2000) 207–226.

Da Silva, P.P.; Paton, N.W.: User Interface Modeling in UMLi. *IEEE Software* **20.4** (2003) 62–69.

Dey, A.K.; Abowd, G.D.; Salber, D.: A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Human-Computer Interaction* **16.2-4** (2001) 97–166.

Göbel, S.; Hartmann, F.; Kadner, K.; Pohl, C.: A Device-Independent Multimodal Mark-up Language. *GI Jahrestagung* **2** (2006) 170–177.

Limbourg, Q.; Vanderdonckt, J.; Michotte, B.; Bouillon, L.; Jaquero, V.L.: USIXML: A Language Supporting Multi-path Development of User Interfaces. *EHCI/DS-VIS* (2004) 200–220.

Myers, B.; Hudson, S.E.; Pausch, R.: Past, Present, and Future of User Interface Software Tools. *ACM Transactions on Computer-Human Interaction* (TOCHI) **7.1** (2000) 3–28.

Nunes, N.J.; e Cunha, J.F.: Towards a UML profile for interaction design: the Wisdom approach. *UML* **2000** (2000) 101–116.

Oviatt, S.: Multimodal interfaces. In: *The human-computer interaction handbook: fundamentals, evolving technologies and emerging applications* (Eds. Jacko, J.A.; Sears, A.). Mahwah: Lawrence Erlbaum Associates, Inc., 2003.

Paternò, F.; Mancini, C.; Meniconi, S.: ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In: *INTERACT '97: Proceedings of the IFIP TC13 International Conference on Human-Computer Interaction*. London: Chapman & Hall, Ltd., 1997.

Puerta, A.; Eisenstein, J.: Towards a general computational framework for model-based interface development systems. In: *IUI '99: Proceedings of the 4th international conference on Intelligent user interfaces*. New York: ACM Press, 1999.

Springer, T.; Kadner, K.; Steuer, F.; Yin, M.: Middleware Support for Context-Awareness in 4G Environments. In: *WoWMoM* **2006** (2006) 203–211.

W3C: Multimodal Interaction Working Group Charter. http://www.w3.org/2006/12/mmi-charter.html (letzter Zugriff 25.09.2007).

Van den Bergh, J.; Coninx, K.: Model-based design of context-sensitive interactive applications: a discussion of notations. *TAMODIA '04* (2004) 43–50.

Wahlster, W.; Weyrich, C. (Hrsg.): Forschen für die Internet-Gesellschaft: Trends, Technologien, Anwendungen, Ergebnisse des Symposiums 2005 des Feldafinger Kreises. 2005.


1


2


3


4

1 Dipl.-Phys. Alexander Behring arbeitet zurzeit an seiner Doktorarbeit in der Gruppe Telekooperation an der TU Darmstadt mit Schwerpunkt auf der Modellierung multimodaler Nutzerschnittstellen. In EMODE zeigte er sich für das Metamodell verantwortlich.
E-Mail: behring@tk.informatik.tu-darmstadt.de
www.tk.informatik.tu-darmstadt.de

2 Matthias Heinrich, Dipl. Medieninformatiker, ist seit 2007 Mitarbeiter bei SAP Research. Als Teil des Software Engineering & Architecture Teams beschäftigt er sich hauptsächlich mit aktuellen Techniken der modellgetriebenen Softwareentwicklung.
E-Mail: matthias.heinrich@sap.com
www.sap.com/research

3 Matthias Winkler (MSc.) ist Mitarbeiter bei SAP Research in Dresden. Seine Forschungsinteressen liegen im Bereich modellgetriebene Softwareentwicklung mit Schwerpunkt auf multimodale Benutzerschnittstellen.
E-Mail: matthias.winkler@sap.com
www.sap.com/research

4 Dr.-Ing. Waltenegus Dargie ist Wissenschaftlicher Mitarbeiter an der Technischen Universität Dresden. Seine Forschungsinteressen liegen im Bereich „Context-Aware Computing", „Self-Managing Systems" und „Wireless Sensor Networks".
E-Mail: waltenegus.dargie@tu-dresden.de
http://z2.inf.tu-dresden.de