Fraunhofer Institut
Experimentelles
Software Engineering

Modellbasierte Entwicklung generischer Benutzerschnittstellen

Autoren:
Sebastian Adam
Christian Bunse
Patrick Kolling

Submitted for publication at
Software Engineering 2006

IESE-Report Nr. 093.05/D
Version 1.0
21. Oktober 2005

Eine Publikation des Fraunhofer IESE

Das Fraunhofer IESE ist ein Institut der Fraunhofer-Gesellschaft.
Das Institut transferiert innovative Software-Entwicklungstechniken, -Methoden und -Werkzeuge in die industrielle Praxis. Es hilft Unternehmen, bedarfsgerechte Software-Kompetenzen aufzubauen und eine wettbewerbsfähige Marktposition zu erlangen.

Das Fraunhofer IESE steht unter der Leitung von
Prof. Dr. Dieter Rombach (geschäftsführend)
Prof. Dr. Peter Liggesmeyer
Fraunhofer-Platz 1
67663 Kaiserslautern

Abstract

Die steigende Dezentralisierung und der wachsende Anteil von Software in vielen Produkten verlangen systematische Entwicklungsmethoden für qualitativ hochwertige und flexibel adaptierbare Software, deren Herstellung gleichzeitig kostengünstig erfolgen soll. Der Erfolg eines Unternehmens hängt dabei direkt von seiner Fähigkeit ab, sich an wechselnde Marktbedürfnisse und Anforderungen anpassen zu können. Hierbei spielt die Software-Wiederverwendung eine besondere Rolle. Je höher der Anteil von bereits existierenden Artefakten in einem neuen Produkt, umso niedriger sind die Kosten einer notwendigen Anpassung. Dies betrifft insbesondere den Bereich der Benutzerschnittstelle oder GUI. Häufig bleibt die Kernfunktionalität eines Systems über längere Zeit konstant während die Benutzerschnittstelle modernisiert oder auf verschiedene Endgeräte (Web, Handy, PDA, etc.) portiert wird. Allerdings ist systematische Wiederverwendung im Bereich der GUI-Entwicklung eher Ausnahme statt Regel. In diesem Beitrag wird eine Methode zur integrierten Modellierung generischer Benutzerschnittstellen – genannt IMRU – vorgestellt und mittels einer empirischen Fallstudie im industriellen Umfeld validiert.

Schlagworte: model-based software development, model-driven development, user interfaces, human-computer interface, Unified Modeling Language (UML), generative programming

Inhaltsverzeichnis

1 Einleitung

Die Softwareindustrie setzt ihre Hoffnungen für zukünftige Produktivitäts- und Qualitätsgewinne verstärkt in moderne Methoden und Techniken des Software Engineering. Ansätze wie komponentenbasierte oder objektorientierte Entwicklung versprechen institutionalisierte Wiederverwendung von Software-Komponenten bzw. Wissen und damit Wettbewerbsvorteile. Allerdings scheint die Entwicklung grafischer Benutzerschnittstellen (GUI) hiervon häufig ausgenommen zu sein. GUIs werden oft ad-hoc entwickelt. Neue Applikationen werden aus kopierten oder modifizierten Teilen bestehender Applikationen zusammengefügt und nur fehlende Teile werden neu, z.B. mittels WYSIWYG Werkzeugen, entwickelt [Mol04]. Dementsprechend fehlt sowohl eine methodische Unterstützung der GUI-Entwicklung als auch eine systematische Anbindung an den eigentlichen Software-Entwicklungsprozess.
Modellgetriebene Ansätze verfolgen die Idee der schrittweisen Verfeinerung der Anforderungen an eine GUI bis hin zum Quellcode. Auf jeder Abstraktionsstufe werden dabei alle wesentlichen Aspekte der GUI mit Hilfe verschiedener Modelltypen beschrieben. Dies erlaubt zum einen eine Verbesserung der Schnittstellenqualität, da notwendige Entscheidungen bereits vor Beginn der eigentlichen Implementierung getroffen und verifiziert werden können, zum anderen aber auch eine erhöhte Produktivität aufgrund automatischer Modelltransformation und Quellcodegenerierung. Aufgrund der aktuellen Problemstellungen, die sich durch den Einsatz heterogener Endgerätetechnologien ergeben, streben neuere Ansätze wie beispielsweise [Nil02] Lösungen an, die den Entwicklungsaufwand von der Anzahl unterschiedlicher Benutzerschnittstellen entkoppeln.

Allerdings konnten sich modellgetriebene GUI-Entwicklungsansätze in der industriellen Praxis bisher nur wenig etablieren, da weder Standards noch geeignete Werkzeuge existieren, welche Methoden und Verfahren zur GUI-Entwicklung in den umgebenden Softwareentwicklungsprozess systematisch integrieren. Eine simple Portierung existierender Software-Entwicklungsansätze lässt zahlreiche Probleme insbesondere im Hinblick auf den Entwicklungsaufwand aber auch die Qualität der Benutzerschnittstelle erwarten.

Dieser Artikel beschreibt einen modellgetriebenen Entwicklungsansatz für grafische Benutzerschnittstellen, der insbesondere die Themen Praxistauglichkeit und Wiederverwendung adressiert. Nach einer kurzen Vorstellung des Stands der Forschung und Praxis in Kapitel 2 wird in Kapitel 3 die Grundlagen des Ansatzes im Kontext moderner Software Engineering Konzepte beleuchtet und in Kapitel 4 der Aufbau des Ansatzes diskutiert. Kapitel 5 beschreibt die Validierung des Ansatzes im industriellen Umfeld, während Kapitel 6 den Artikel zusammenfasst und einen kurzen Ausblick auf zukünftige Arbeiten präsentiert.

2 Stand der Forschung und Praxis

Konzepte zur modellbasierten GUI-Entwicklung werden seit längerem diskutiert. Im Vordergrund stehen dabei die verbesserte Komplexitätsbeherrschung und Wiederverwendung aufgrund einer systematischen Vorgehensweise [Mol04]. Die GUI wird dabei als Brückenfunktion zwischen Anwendungslogik und Anwendungsumgebung aufgefasst. Als Eingabeprodukte zur Modellierung dienen sowohl ein Aufgabenmodell für die Benutzungs- als auch ein Datenmodell für die Systemperspektive [Pin00]. Die GUI-Spezifikation erfolgt mittels abstrakter und konkreter Präsentationsmodelle zur Beschreibung der statischen Struktur und durch Dialogmodelle zur Spezifikation des Interaktionsverhalten [Bom99].

Abstrakte Präsentationsmodelle gruppieren für den Anwender, zur Aufgabenerfüllung, notwendige Daten. Konkrete Präsentationsmodelle beschreiben darauf aufbauend die Abbildung dieser Gruppierungen auf konkrete Fenster, einzusetzende Bildschirmelemente und das konkrete Layout. Die Interaktionsmöglichkeiten zwischen Benutzer und System sowie die dabei aufrufbaren Systemfunktionen werden parallel durch Dialogmodelle beschrieben.

Der Entwicklungsprozess folgt in vielen Ansätzen einem ähnlichen Schema [Bom99] und wird in der Regel durch eine auf den konkreten Ansatz zugeschnittene Entwicklungsumgebung unterstützt. Hierbei werden zunächst abstrakte GUI-Modelle erstellt und anschließend schrittweise verfeinert. Die Möglichkeit einer automatisierten Modelltransformation und Quellcodegenerierung stellt dabei eine hilfreiche Arbeitsunterstützung dar.

Zusammenfassend können als wesentliche Vorteile der modellbasierten GUI-Entwicklung die hohe Abstraktion von technischen Details, Produktivitätssteigerung durch weitestgehende Automatisierung und Qualitätsverbesserung durch Wiederverwendung, Lernschleifen, robuste Codeschablonen aber auch der Möglichkeit eines ingenieurmäßigen Prozesses gesehen werden [Mol04]. Im Hinblick auf Endgeräte- und Plattformunabhängigkeit können modellbasierte Ansätze durch Abstraktion und Automatisierung den Entwicklungsaufwand von der Heterogenität eingesetzter Technologien entkoppeln.

Wesentlicher Nachteil der modellbasierten GUI-Entwicklungsansätze ist das Fehlen einer einheitlichen Notation [Pin00]. Diese ist indirekt auch für mangelnde Werkzeugunterstützung und folglich geringe Akzeptanz in der Praxis ausschlaggebend. Neuere Arbeiten [Pat01] setzen auf den Einsatz der Unified Modeling Language (UML) [OMG05]. Neben der industriellen Akzeptanz und existierender Werkzeuge verspricht der Einsatz der UML eine Verzahnung der

GUI-Modellierung mit dem Softwareentwurf. Allerdings lässt sich das visuelle Erscheinungsbild aufgrund fehlender Diagrammtypen oder vordefinierter Stereotypen in UML nicht modellieren. Hierzu sind eigene Notationen erforderlich, was die standardisierte Entwicklung grafischer Benutzerschnittstellen mit UML erschwert.

Im Kontext der verbesserten Erweiterungsmöglichkeiten der UML 2.0 werden zahlreiche Notationsvorschläge für eine Modellierung des konkreten Erscheinungsbildes einer GUI in UML diskutiert. Auf Basis des „Diagram Interchange"-Standards, welches die Speicherung der Darstellungsinformationen von UML-Diagrammen ermöglicht, definiert [Bla04] ein vollwertiges UML-Profil zur Modellierung des GUI-Layouts. Idee ist hier die Verwendung von UML-Diagrammen als Skizze der erstellten GUI sowie die semantische Behaftung der Anordnung einzelner Stereotypen innerhalb des Diagramms. Der Ansatz ist aber nur schwer in einen modellgetriebenen Entwicklungsprozess integrierbar.

Trotz der zahlreichen Vorteile konnten aufgrund der bisher ungelösten Probleme modellbasierte Entwicklungskonzepte für grafische Benutzerschnittstellen bis dato in der Industrie nicht Fuß fassen und WYSIWYG-Editoren als „Stand der Praxis" ablösen. Erst ein Ansatz, welcher die Praxistauglichkeit im weiten industriellen Umfeld ermöglicht und tatsächliche Effizienz- und Qualitätsvorteile gewährleistet, kann hier ein langfristiges Umdenken erzielen.

3 Modellierung generischer Benutzerschnittstellen

Die Benutzerschnittstelle eines Software-Systems ist, im Gegensatz zur Anwendungslogik, durch einen festen Satz von Bildschirmobjekten, wiederkehrender Standardfunktionen sowie einem zumeist einheitlicher Aufbau nach dem MVC-Muster geprägt [Bus97]. Somit besteht bei der Entwicklung grafischer Benutzerschnittstellen ein hohes Potential an Wiederverwendung und Automatisierung. Die überschaubare Anzahl verschiedener Bildschirmelemente ermöglicht so z.B. die Definition von Abbildungsregeln zur Genierung eines umfassenden Modells zur Beschreibung des visuellen Erscheinungsbildes aus rein fachlichen Datensichtdefinitionen. Weiterhin wird eine „einfache" 1-zu-1-Abbildung auf Konstrukte konkreter Plattformen ermöglicht, da nahezu jedes Bildschirmelement von den gängigen Entwicklungssprachen und Endgeräten unterstützt wird.

Der Zusammenhang zwischen einzelnen Abstraktionsebenen erlaubt somit eine Trennung in plattformneutrale und –spezifische Modelle und einen hohen Grad an automatisierter Transformation zwischen diesen. Somit können Gemeinsamkeiten aller betrachteten Endgeräteschnittstellen unabhängig von der konkreten technologischen und darstellungsrelevanten Ausprägung spezifiziert werden. Die Entwicklung plattformneutraler Modelle erhöht somit langfristig Wiederverwendung.

Während die Modellierung von Gemeinsamkeiten mit Hilfe plattformneutraler Modelle erfolgt, wird deren Verfeinerung auf konkrete Ausprägungen durch präzise Abbildungsvorschriften systemneutral spezifiziert. Allerdings handelt es sich bei diesen Transformationen nicht nur um einfache Abbildungen, da zusätzlich Gestaltungsentscheidungen (z.B. Layout) eingebracht werden. Daher kann auf einzelne Modelltypen, insbesondere zur systemspezifischen Kontextmodellierung, verzichtet werden, da aus Gründen der projektübergreifenden Wiederverwendung die Beschreibung solcher Aspekte in den Abbildungsvorschriften geeigneter erscheint [Pin00]. Somit müssen beispielsweise Benutzer- und Anwendungskontexte, Layouts und Restriktionen hinsichtlich verwendeter Endgeräte nicht in jedem Projekt neu modelliert werden. Diese können durch einfache Auswahl und Verwendung geeigneter Abbildungsschablonen integriert werden.

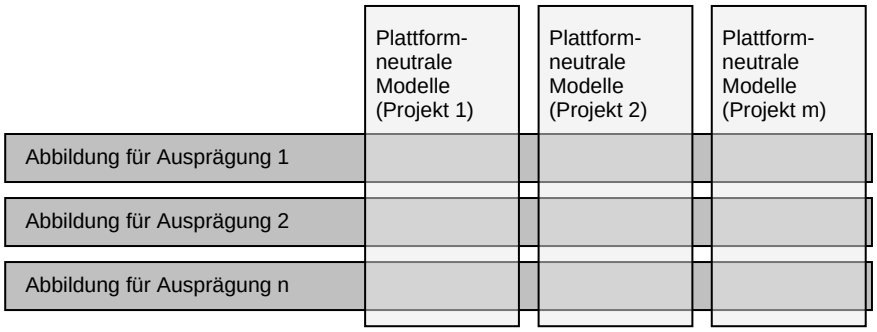

Abbildung 1: Wiederverwendungsmatrix

Die matrixförmige Kombination und Wiederverwendung von Modellen und Abbildungsvorschriften im Umfeld modellgetriebener GUI-Entwicklungskonzepte ermöglicht somit Wiederverwendung, sowohl innerhalb als auch zwischen Projekten (vgl. Abbildung 1). Plattformneutrale Modelle bilden die wiederverwendbare Basis aller Benutzerschnittstellen eines Systems, während Abbildungsvorschriften die wiederverwendbare Basis für spezielle Ausprägungen einzelner Benutzer-Schnittstellen darstellen.

4 Der IMRU Ansatz

Kern des IMRU Ansatzes [Ada05] ist ein modellgetriebener Entwicklungsprozess für grafische Benutzerschnittstellen. Ziel ist insbesondere die Unabhängigkeit von konkreten Technologie- oder Darstellungsausprägungen, wofür IMRU eine Trennung von plattformneutralen und –spezifischen Modellen, analog zur MDA [OMG03], vornimmt. Zusätzlich werden Produktlinienkonzepte bei der Spezifikation der Gemeinsamkeiten aller Schnittstellen verwendet.

Praxistauglichkeit soll zum einen durch einen hohen Grad an Automatisierung, zum anderen durch die konsequente Verwendung existierender Standards wie UML, XHTML [W3C00] order XSLT [W3C99] erreicht werden. Diese ermöglichen unter anderem den Einsatz einer Vielzahl von Werkzeugen und erhöhen somit die industrielle Akzeptanz. Weiterhin verbessern der Einsatz von UML und ein systematischer Prozess die Qualität der Produkte sowie die Integration des Ansatzes in den Softwareentwicklungsprozess. Dies wird im Folgenden näher betrachtet.

4.1 Prozess und Produkte

Ein systematischer Ansatz zur Entwicklung grafischer Benutzerschnittstellen erfordert einen klar definierten Prozess, der Entwicklungsaktivitäten, Dokumente und Rollen beschreibt. Da es sich bei IMRU um einen Teilprozess der Software-Entwicklung handelt, ist die Integrierbarkeit in andere Prozesse (z.B. in den Unified Process) erforderlich. Dies wird in IMRU aufgrund der eindeutig definierter Eingangs- uns Ausgangsschnittstellen gewährleistet (vgl. Abbildung 2**Error! Reference source not found.**).

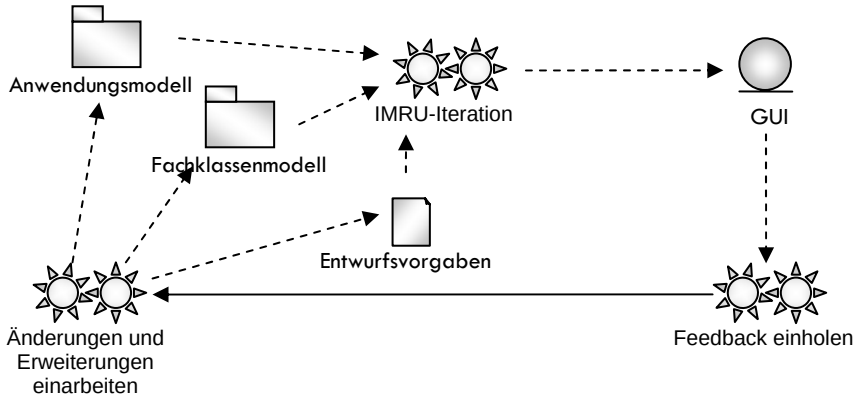

Abbildung 2: Integration einer IMRU-Iteration in Softwareentwicklungsprozesse

Eine IMRU Iteration (vgl. Abbildung 3) nutzt die Ergebnisse der Analysephase in Form von Anwendungsfällen und Domänenobjekten. Hierzu gehören beispielsweise das Fachklassen- und Anwendungsfallmodell aber auch Aktivitätsdiagramme und die Beschreibung der möglichen Systemverwendungen.

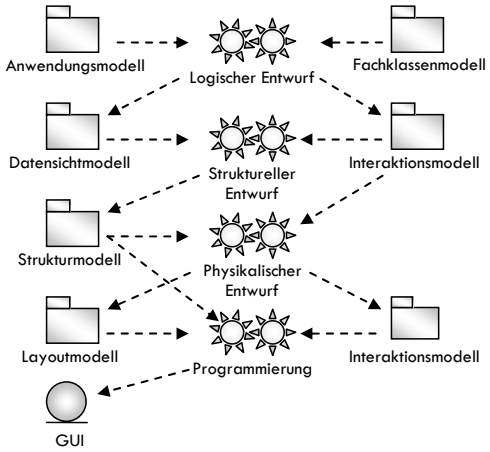

Abbildung 3: Ablauf einer IMRU-Iteration

4.1.1 Logischer Entwurf

Der logische Entwurf befasst sich mit der Entwicklung eines Datensicht- und abstrakten Interaktionsmodells aus den erwähnten Eingabemodellen. Hierbei handelt es sich um eine weitgehend manuelle bzw. kreative Aktivität ohne großes Automatisierungspotential [Pin00].

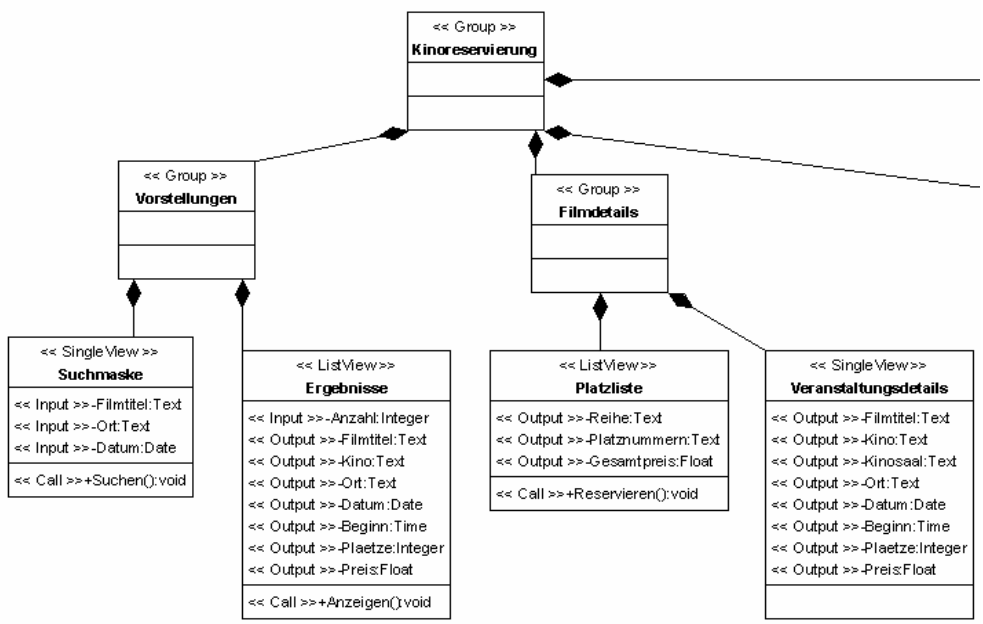

Abbildung 4: Beispiel für ein Datensichtmodell

Das Datensichtmodell, ein mit Stereotypen angereichertes UML-Klassendiagramm, beschreibt die Struktur einer GUI in Form logischer Sichten. Eine logische Sicht ist hierbei ein virtueller Teil des späteren Bildschirms, der alle Daten und mögliche, inhaltlich zusammengehörenden Interaktionen kapselt. Logische Sichten können selbst wiederum zu logischen Gruppen zusammengefasst werden, welche den fachlichen Zusammenhang zwischen verschiedenen Sichten beschreiben. Übertragen auf das Beispiel eines Kinoresevierungssystems (vgl. **Error! Reference source not found.**) spezifizieren die Stereotypen der Klassen, ob es sich um eine logische Sicht (SingleView für Einzeldatensichten, ListView für Listendatensichten, etc.) oder eine Gruppierung (Group) von Sichten handelt. Die mit Stereotypen versehenen Attribute beschreiben die in der jeweiligen Sicht angebotenen Daten und ihre Bearbeitungsrechte (Input, Edit, Output). Analog definieren die modellierten Methoden mögliche Interaktionen, die mit dem System von der jeweiligen Sicht aus möglich sind. Im Beispielmodell stellt somit die Klasse „Suchmaske“ exemplarisch eine Sicht innerhalb der Benutzerschnittstelle dar, in welcher der Anwender die Daten „Filmtitel“, „Ort“ und „Datum“ eingeben und die zugehörige Interaktion „Suchen“ anstoßen kann.

Im Interaktionsmodell, basierend auf einem UML-Zustandsübergangsdiagramm, werden alle Interaktionen des Systems und ihre Effekte beschrieben. Die Zustände repräsentieren hierbei Dialogzustände äquivalent zu den logischen Sichten des Datensichtmodells, während die Übergänge Interaktionen darstellen. Die auslösenden Ereignisse der Übergänge spezifizieren die Benutzeraufrufe, welche entsprechend im Datensichtmodell als Methoden definiert wer-

den. Übergangseffekte beschreiben entsprechend die aufgerufene Systemfunktionalität.

Übertragen auf das Beispiel Kinoreservierung (vgl. **Error! Reference source not found.**) wird so z.B. beim Aufruf der Anwendung (Startzustand) der Anwender direkt auf die Sicht „Suchmaske" umgeleitet. Hier kann er mit dem Start der Interaktion „Suchen" die implementierende Funktion „SearchEvent" innerhalb der Anwendungslogik anstoßen. Nach Abarbeitung bekommt er schließlich die Sicht „Ergebnisse" angezeigt, auf welcher er analoge Interaktionsmöglichkeiten besitzt.

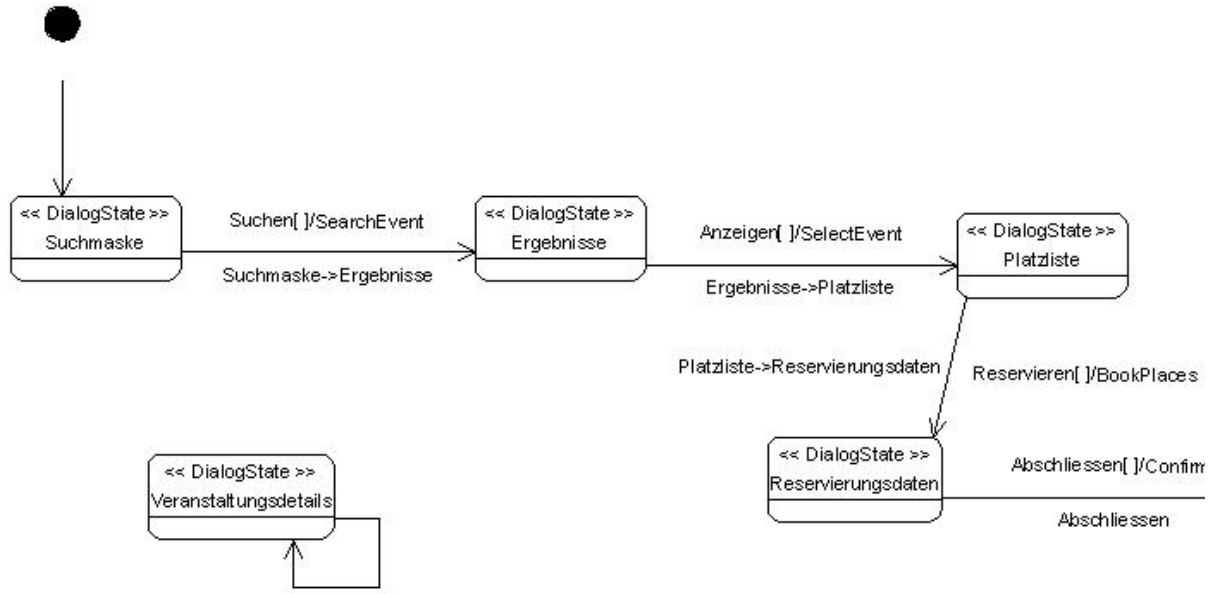

Abbildung 5: Beispiel eines abstrakten Interaktionsmodells

Die Modellierung von freien Navigationsmöglichkeiten zwischen einzelnen Systemsichten wird innerhalb der Interaktionsmodells nicht spezifiziert, da dies von der Aufteilung logischer Sichten auf physikalische Fenster abhängt, die erst auf Ebene des physikalischen Entwurfs (vgl. 4.1.3) betrachtet wird.

Die Modelle des logischen Entwurfs sind unabhängig von späteren Implementierungstechnologien oder Darstellungsaspekten und beschreiben ausschließlich, die fachliche Struktur der Benutzerschnittstelle sowie die Durchführung einzelner Systeminteraktionen.

4.1.2 Struktureller Entwurf

Im strukturellen Entwurf werden die, zur Visualisierung der Datensichten notwendigen Bildschirmelemente durch einen teilweise automatischen Transformationsvorgang ermittelt und in hierarchischer Struktur angeordnet. Die manuelle Aufgabe eines GUI-Entwicklers besteht hierbei in der Definition von Attributwerten, welche auf Basis der Abbildungsvorschriften nicht automatisch gesetzt werden konnten. Beispiele hierfür sind individuelle Beschriftungen eines Textfeldes oder dessen Länge. Ein Beispiel für ein Strukturmodell zeigt Abbildung 6. Deutlich ist hierbei die hierarchische Anordnung der Bildschirmelemente, zu erkennen, welche selbst mit Hilfe von Stereotypen eindeutig definiert und durch Attribute konkret spezifiziert werden können. So beschreibt beispielsweise „Filmtitel" eine 20 Zeichen lange Textbox mit gleichnamiger Beschriftung, in welche der Anwender Eingaben tätigen kann.

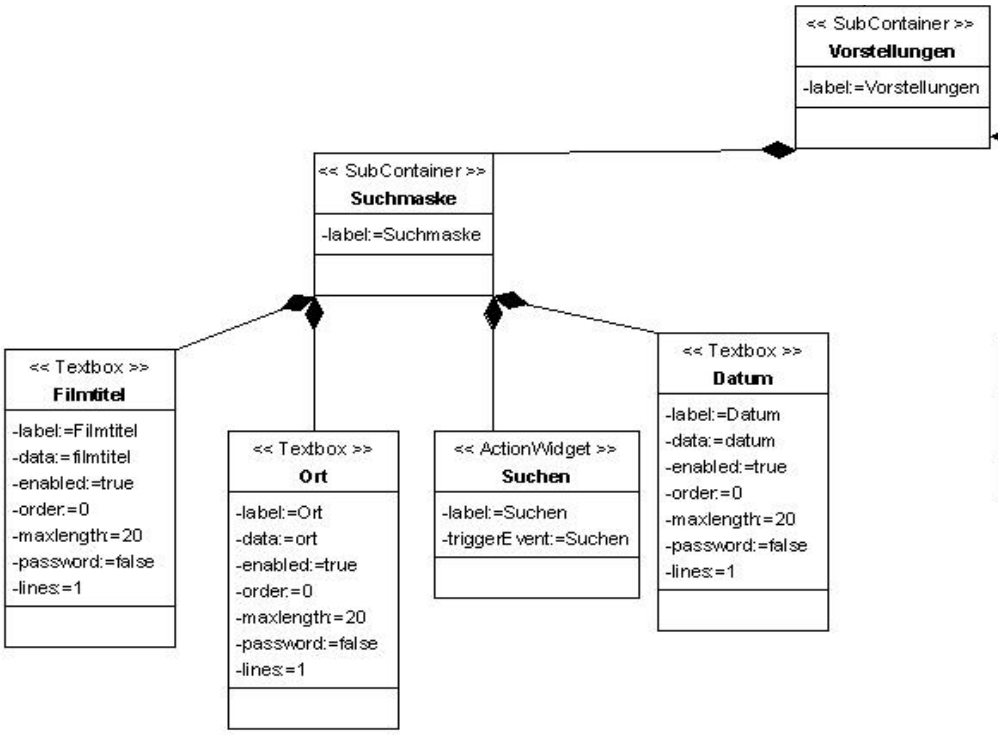

Abbildung 6: Beispiel eines Strukturmodells

4.1.3 Physikalischer Entwurf

Der physikalische Entwurf befasst sich mit der Konkretisierung der existierenden Modelle für ein spezifisches Endgerät und den zugehörigen Darstellungsstil. Hierfür werden ein Layoutmodell und ein konkretes Interaktionsmodell zur Verfeinerung der Interaktionen erstellt. Analog zum Strukturmodell wird ein

Großteil dieser Modelle automatisch auf Basis hinterlegter Abbildungsvorschriften generiert. Lediglich Details müssen manuell spezifiziert werden.

Im Gegensatz zum Interaktionsmodell kann das Layoutmodell nicht mit Hilfe der UML spezifiziert werden. Allerdings können hier standardisierte Layout-Beschreibungssprachen, angepasst auf das jeweilige Endgerät oder den Darstellungsstil, zum Einsatz kommen. Beispiele hierfür sind XHTML und CSS im Umfeld webbasierter Benutzerschnittstellen.

Abbildung 7 zeigt einen Ausschnitt des Layoutmodells für das Beispielsystem. Das Interaktionsmodell ist in diesem Fall eine mit zusätzlichen Modellinformationen angereicherte Version des Modells aus Abbildung 5.

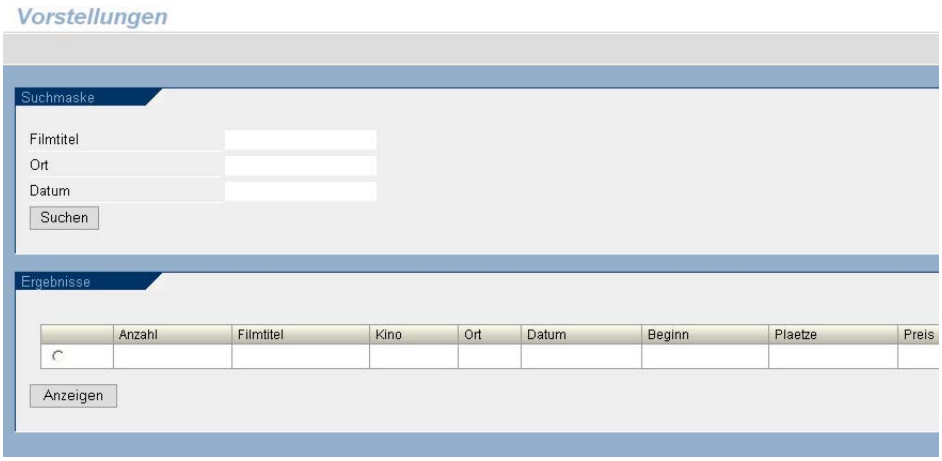

Abbildung 7: Beispiel eines Layoutmodells

4.1.4 Programmierung

In der Programmierung werden die physikalischen Modelle bzgl. eines Endgeräts und eines Darstellungsstil, in ausführbaren Quellcode, nahezu vollständig automatisch, transferiert. Im verwendeten Beispiel wurde so der gesamte Quell- und Konfigurationscode für eine Jakarta-Struts-basierte Webanwendung generiert.

Die Trennung von physikalischem Entwurf und Programmierung dient der Isolation der Implementierungsdetails von der Definition eines konkreten Erscheinungsbildes. So beschreiben z.B. die physikalischen Modelle eine webbasierte GUI, sind aber unabhängig von der verwendeten Technologie wie JSP, ASP, etc. Dies ermöglicht einen hohen Grad an Wiederverwendung, insbesondere aufgrund des im Umfeld grafischer Benutzerschnittstellen raschen Technologiewandels [Kap03].

Die ausführbare Benutzerschnittstelle muss allerdings durch Usability-Experten oder Benutzer validiert werden. Diese Aktivität ist nicht Bestandteil der IMRU-Iteration und schließt somit die Integration von IMRU in den umgebenden Software-Entwicklungsprozess ab.

4.2 Werkzeuge

Durch die konsequente Verwendung von Standards kann bei der Anwendung des IMRU Ansatzes auf eine Reihe existierender Werkzeuge zurückgegriffen werden. Wichtigstes Werkzeug ist dabei ein UML-Editor zur Modellerstellung. Dieser muss, um standardisierten Austausch und Transformation zu ermöglichen, das XMI-Format unterstützen. Zusätzlich werden WYSIWYG-Werkzeuge für die Verfeinerung des Layouts und IDEs zur Programmierung benötigt. Daneben sind spezielle Werkzeuge für das Konfigurations- und Versionsmanagement, sowie ein XSLT-Prozessor für Modelltransformationen und Generierung erforderlich.

4.3 Evolutionäre Aspekte

Aufgrund des generischen Charakter des IMRU Ansatzes sind Optimierungen der Benutzerschnittstelle (z.B. bzgl. Usability) nicht nur innerhalb eines Projektes sondern auch projektübergreifend erforderlich. Dies erfordert auch die Anpassung der Transformationsschablonen an gewonnene Erfahrungen. Ein Beispiel hierfür stellt das Aufkommen neuer Implementierungstechnologien oder Darstellungsstile dar, für die existierenden Schablonen keine adäquate Unterstützung bieten.

Neben der Optimierung der Abbildungsvorschriften erlaubt IMRU weiterhin, wiederverwendbare Komponente einschließlich ihrer Beschreibungen auf verschiedenen Abstraktionsebenen zu extrahieren, abzulegen und in zukünftigen Projekten durch manuelle oder automatische Einbeziehung wieder zu verwenden. So können beispielsweise Gruppen elementarer Bildschirmelemente im Strukturmodell durch eine Komponente ersetzt werden, die die wesentlichen Informationen kapselt.

5 Validierung

Zum Nachweis der Wirksamkeit und Praxistauglichkeit wurde der IMRU-Ansatz im industriellen Kontext bei der T-Systems International GmbH empirisch validiert. Die empirische Untersuchung wurde in Form eines gekreuzten Tests durchgeführt, in dem die Teilnehmer eine web- und eine WAP-basierte Benutzerschnittstelle zum einen mit IMRU und zum anderen auf konventionelle Weise entwickelten. Darüber hinaus wurde die subjektive Meinung der Teilnehmer hinsichtlich der Praxistauglichkeit mit Hilfe eines anonymen Fragebogens erhoben.

Die Analyse der erhobenen Daten zeigte, dass IMRU eine Aufwandsersparnis zwischen 23 und 49% (durchschnittlich 34%) für die Entwicklung einer Schnittstelle, und akkumuliert bis zu durchschnittlich 56% für die Entwicklung beider Schnittstellen (WAP und Web) bei einer Signifikanz <0,01 ermöglichte. Interessant ist auch, dass annähernd 75% des gesamten IMRU-Aufwandes für die plattformneutrale Modellierung aufgewendet wurde. Diese Beobachtung lässt vermuten, dass ein hoher Grad an Wiederverwendung in IMRU-basierten Projekten möglich ist. Insbesondere Systeme die mehr als eine Endgeräteschnittstelle anbieten können demnach Aufwandseinsparungen realisieren.

Aufgrund der geringen Teilnehmerzahl (<10) und der exemplarischen Aufgabenstellung können die Ergebnisse aber nur als Trend interpretiert werden. Generellere Aussagen erfordern daher weitergehende Untersuchungen durch eine Reihe von Experimenten [Gre90].

Die quantitativen Ergebnisse werden aber auch durch die Analyse der qualitativen Ergebnisse (Teilnehmerbefragung) gestützt. Somit lässt sich erwarten, dass der IMRU-Ansatz erfolgreich in der industriellen Praxis eingesetzt werden kann. So wurde der Einarbeitungsaufwand in IMRU von den Teilnehmern als „normal“ empfunden und das vorgeschlagene Entwicklungsvorgehen als „intuitiv und bequem“ bezeichnet. Eine höhere Effizienz hinsichtlich der Entwicklung im Umfeld heterogener Technologie wurden ebenso erkannt, wie ein hoher Grad an Fehlervermeidung und somit Qualitätsverbesserung. Die Trennung in verschiedene Modelltypen und Abstraktionsebenen wurde als „hilfreich“ erachtet und gute Unterstützung für eine Arbeitsteilung zwischen grafischem Design, logischem Entwurf und Programmierung gesehen. Darüber hinaus erklärten die Befragten, dass sie mit IMRU eine Entlastung von Technologie- und Gestaltungskenntnissen während der GUI-Entwicklung erwarten. Zusammenfassend wurde im Rahmen der Befragung der IMRU-Ansatz als „hilfreich im praktischen Umfeld“ bezeichnet.

6 Zusammenfassung & Ausblick

Die steigende Komplexität von Software-Systemen erfordert den Einsatz effizienter und systematischer Verfahren des Software Engineering. Der Bereich der Benutzerschnittstelle wird dabei von existierenden Ansätzen häufig vernachlässigt, sodass Benutzerschnittstellen heute noch oft ad-hoc entwickelt werden. In diesem Artikel wurde ein auf Software Engineering Prinzipien beruhender Ansatz zur modellgetriebenen Entwicklung grafischer Benutzerschnittstellen vorgestellt.

Der IMRU Ansatz basiert auf Ideen der MDA und Prinzipien der Produktlinien-Entwicklung durch die eine Unterteilung in plattformneutrale und –spezifische Modelle, sowie eine Modellierung von Schnittstellengemeinsamkeiten vorgenommen wird. Sowohl innerhalb von Projekten (mit Hilfe der logischen Modelle), als auch über Projektgrenzen hinweg (mittels Abbildungsschablonen) wird somit effiziente Wiederverwendung im Bereich grafischer Benutzerschnittstellen ermöglicht.

Die Akzeptanz eines derartigen Ansatzes in der industriellen Praxis erfordert allerdings eine durchgängige Werkzeugunterstützung die die vorgeschlagenen Konzepte in moderne modellgetriebene Entwicklungsumgebungen integriert sowie einfachere Abbildungsdefinitionen, Modellüberprüfungen und Roundtrip-Funktionalität unterstützt. Die empirische Validierung hat gezeigt, dass der entwickelte Prototyp als Startpunkt ausreichend ist, für den Produktionsbetrieb aber erweitert werden muss.

Ein weiterer offener Punkt ist die tatsächliche Nutzbarkeit generierter Benutzerschnittstellen im Sinne des Usability-Engineering. Konkret ist hierbei zu untersuchen wie die Qualität einer Benutzerschnittstelle durch den systematischen Einsatz von Usability-Patterns gesteigert werden kann.

Referenzen

[Ada05] Adam, S.: Entwicklung eines Ansatzes zur integrierten Modellierung wiederverwendbarer Benutzerschnittstellen. Technische Universität Kaiserslautern, Fachbereich Informatik, 2005

[Bom99] Bomsdorf, B.: Ein kohärenter, integrativer Modellrahmen zur aufgabenbasierten Entwicklung interaktiver Systeme, Dissertation. Universität Paderborn, 1999.

[Bla04] Blankenhorn, K.: A UML Profil for GUI Layout. In: Lecture Notes in Computer Science Volume 3263 / 2004. Springer-Verlag, Berlin, 2004

[Bus97] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. Pattern Oriented Software Architecture: A System of Patterns, John Wiley & Sons, 1997

[Gre90] Green, S., Kouchakdjian, A., Basili, V., and Weidow, D. 1990. The Cleanroom Case Study in the SEL: Project description and early analysis. Technical Report SEL-90-002, NASA-SEL.

[Kap03] Kappel, G., Pröll, B., Reich, S., Retschitzegger, W.: Web Engineering – Systematische Entwicklung von Web-Anwendungen. dpunkt.verlag, 2003

[Mol04] Molina, P.: A Review to Model-Based User Interface Development Technology. In: Proceedings of the first Workshop on Making Model-Based User Interface Design Practical, Island of Madeira, 2004

[Nil02] Nilsson, E.: Combining Compound Conceptual User Interface Components with Modelling Patterns – A Promising Direction for Model-Based Corss-Platform User Interface Development. In: DSV-IS 2002, LNCS 2345, Seiten 104-117. Springer-Verlag, Berlin Heidelberg, 2002

[OMG03] MDA Guide Version 1.0.1, Object Management Group, 2005

[OMG05] Unified Modeling Language , Version 2.0, Object Management Group, 2005

[Pat01] Paterno, F.: Towards a UML for Interactive Systems. In: Engineering for Human-Computer Interaction, LNCS 2254, Seiten 7-18. Springer-Verlag, Berlin Heidelberg, 2001

[Pin00] Pinheiro, P.: User Interface Declarative Models and Development Environments: A Survey. In: DSV-IS 2000, LNCS 1946, Seiten 207-226. Springer-Verlag, Berlin Heidelberg, 2000

[W3C99] XSL Transformations (XSLT) Version 1.0, W3C Recommendation, 1999

[W3C00] XHTML 1.0: Extensible HyperText Markup Language, W3C, 2000

Dokumenten Information

Titel:	Modellbasierte Entwicklung generischer Benutzerschnitt-stellen
Datum:	21. Oktober 2005
Report:	IESE-093.05/D
Status:	Final
Klassifikation:	Öffentlich