Thomas Schlegel und Alexander Burst

Interaktionsmodelle in der Spezifikation: Von der Datenzentrierung zur Ablauforientierung

Interaction models in the requirements specification: From data centric to flow centric models

Interaktionsmodelle_Requirements Specification_Ablauforientierung_Interaction Modeling Language_IML_Dialog Layer Language_DiLL_User Interface Generation_Interaction Specification

Zusammenfassung. In den frühen Projektphasen der Anwendungsentwicklung steht eine Vielzahl von Informationen von Kunden, Benutzern und Anwendungsdomänen zur Verfügung. Wird dieses Wissen mit Hilfe eines deskriptiven, interakions- und ablauforientierten Modells sedimentiert, kann es zur Generierung von UI-Prototypen und Entwicklungsdokumenten genutzt werden.

Hier soll eine Modellierungsmethode aufgezeigt werden, mit der es möglich ist, Wissen aus der Analyse- und Spezifikationsphase in ein Interaktionsmodell einzubringen und so über die Projektlaufzeit zu konservieren. Die Eignung für eine iterative und inkrementelle Vorgehensweise und die Möglichkeit zur Generierung von Oberflächenprototypen soll zudem eine stärkere Partizipation von Kunden und Benutzern am Entwicklungsprozess ermöglichen.

Summary. A huge amount of information from customers, users and concerning the application domain is present in the early project phases. If this knowledge is being preserved using a descriptive, interaction- and flow-oriented model, it can be used for the generation of UI-prototypes and development documents. In this paper, a method for modelling is presented, that allows stakeholders to integrate the knowledge, that is being acquired in the analyses and specification phase, into an interaction model, that allows to preserve this knowledge over the project runtime. The model is suited for an iterative and incremental proceeding and allows the extraction of information for the generation of user interface prototypes. Therefore it ought to allow a stronger participation of customers and users in the development process.

1. Motivation

Funktionale und fachliche Aspekte von Software standen lange Zeit im Mittelpunkt der Entwicklungen. Dieses Bild hat sich mittlerweile stark gewandelt, so dass nicht-funktionale Anforderungen[1] und Interaktionsaspekte in Forschung und Anwendung große Beachtung finden – meist unter den Stichworten „Softwarequalität" und Usability. Auch eine stärkere Fokussierung auf Benutzer und Kunden ist zu beobachten.

Um hier eine entsprechende Rückkopplung zu ermöglichen, werden in zunehmendem Maße iterative und inkrementelle Entwicklungsmethoden eingesetzt, so dass die erzielten Projekt-Zwischenstände Möglichkeiten zur Evaluation durch die beteiligten Interessengruppen bieten.

Eine zentrale Rolle spielt hier neben agilen Entwicklungsmethoden und Qualitätssicherungsansätzen wie CMM[2] das Oberflächen-Prototyping. Die Ursache hierfür ist, dass alle an der Entwicklung beteiligten Interessengruppen mit Hilfe visuell evaluierbarer Prototypen weit einfacher ein kognitives Modell bilden können – speziell jene, die nicht in die Entwicklung involviert sind. Oberflächenprototypen und anschauliche Modelle führen hier zu einer leichteren Evaluation, besseren Evaluationsergebnissen und – damit verbunden – zu einer geringeren Anzahl von Iterationen mit besserem Ergebnis. Die stark benutzerpartizipativen und agilen Methoden profitieren zudem von anwendungsnahen und visuell erfassbaren Modellen wie Oberflächenprototypen und Use-Cases.

[1] So genannte non-functional requirements (NFR)

[2] Capability Maturity Model, siehe SEI 2002

2. Technische Software und Anforderungen

Zunehmende Komplexität und immer kürzer werdende Entwicklungszyklen haben zu einer Verlagerung von Hardwareorientierung und direkter Programmierung hin zu neuen, abstrahierenden Vorgehensweisen geführt. Moderne Entwicklungswerkzeuge sollen dabei helfen, konstruktiv neue Systeme zu entwickeln und die Handhabung weiter zu vereinfachen. Dies gilt insbesondere für Konfigurationswerkzeuge, mit denen Steuergerätesoftware parametrisiert wird. Am Beispiel dieser Softwaregattung sollen die Entwicklungstendenzen und Anforderungen an die moderne Softwareentwicklung im technischen Bereich verdeutlicht werden.

Entwicklungswerkzeuge werden zumeist von Spezialisten eines Anwendungsgebiets eingesetzt. Diese verfügen teilweise nur über geringe allgemeine Anwenderkenntnisse, jedoch über ein großes Fachwissen in ihrer eigenen Anwendungsdomäne und zum Teil langjährige Erfahrung mit entsprechenden Werkzeugen.

Während Bedürfnisse von Heim- oder Büro-Anwendern von entsprechenden Programmen mit einer meist ausgefeilten Benutzerführung abgedeckt werden, weicht die technische Software hiervon ab: Bei der Software für die Spezialisten eines Anwendungsgebiets führt die starke Konzentration auf technische Aufgabenstellungen und der durch eine kleine Zielgruppe erhöhte Kostendruck oft zu einer Software, die Usability-Aspekte vernachlässigt.

Die Überbetonung der fachlichen Aspekte führt also zu einer Unterschlagung von Qualitätsmerkmalen, speziell im Bereich der Interaktion. Eine von den technischen Aspekten weitgehend losgelöste Behandlung der Interaktionsaspekte führt jedoch unweigerlich zu mehr Aufwand und somit zu höheren Kosten und längerer Entwicklungsdauer.

Aus dem Dreieck „Zeit-Kosten-Qualität" folgt, dass bei Verwendung gleicher Entwicklungsmethoden Projekte länger dauern und höhere Kosten verursachen. Eine positive Beeinflussung der Benutzungsoberflächenqualität darf in einer Gesamtbetrachtung jedoch keine signifikanten Mehrkosten oder eine Verlängerung der Projektlaufzeit auslösen. Deshalb ist die Vermeidung zusätzlichen Aufwands nur möglich, wenn keine gesonderte Betrachtung der Usability-Aspekte stattfindet, sondern eine Integration der technischen und der Interaktionsaspekte erreicht werden kann. Hierzu müssen Methoden und Werkzeuge zur Verfügung gestellt werden, die bei einer Effizienzsteigerung in der Entwicklungsarbeit gleichzeitig die Qualität von Prozess und Produkt erhöhen. Nur so kann bei gleichem oder geringerem Gesamtaufwand eine höhere Qualität erzielt werden.

Ein integriertes, generiertaugliches Modell ermöglicht hier die automatisierte Nachführung von Entwicklungsdokumenten und vermeidet veraltete und inkonsistente Modelle. Speziell bei Wartungs- und Update-Projekten können mit einem aktuellen, konsistenten Modell Aufwände für Einarbeitung und Reengineering eingespart werden.

3. Automatische Ableitung von Artefakten

Um die Qualität durch Benutzerpartizipation zu erhöhen, ist meist eine iterative, inkrementelle Vorgehensweise, gestützt auf einen Oberflächenprototypen, zielführend, d.h. das Interaktionsmodell wird in vielen Zyklen gebildet, von denen jeder Abläufe hinzufügt oder verbessert und als Ergebnis die nächste Version der Spezifikation und des Oberflächenprototyps liefert. Das Interaktionsmodell enthält dabei die Abläufe mit Aktions- und Reaktionsmöglichkeiten von Benutzer und System, so dass mögliche Interaktionen in Bezug auf den Kontext möglichst vollständig beschrieben werden.

Ein derartiges Vorgehen verspricht jedoch nur dann Erfolg, wenn eine Modellierungsmethode eingesetzt wird, die aufgrund ihrer Anwendernähe und Fähigkeit zum Prototyping schon zu Beginn des Projekts eingesetzt werden kann. Für Benutzer und Kunden sind Arbeits- und Interaktionsabläufe besser verständlich als Datenstrukturen und deren Beziehungen, was sich beispielsweise durch den Erfolg von Use-Cases und – in noch stärkerem Maße – von rein textuellen Beschreibungen (vgl. Weber, Weisbrod 2003) in der Analyse und Anforderungsspezifikation zeigt. Ziel des im Folgenden beschriebenen Ansatzes ist deshalb ein interaktions- und ablauforientiertes Modell.

Kann die Modellierungsmethode durch eine generative UI-Prototyping-Funktionalität ergänzt werden, profitieren Benutzer und Kunden sowohl in der Anforderungsanalyse durch das am Arbeitsablauf orientierte Modell als auch bei dessen Evaluierung mit Hilfe des Oberflächenprototyps. Erfolgt die Generierung automatisiert, kann diese Evaluation mit kurzen Zyklen wiederholt werden und bietet somit eine vollständige Integration der Interaktionsspezifikation in einen benutzerzentrierten Entwicklungsprozess.

Als Ausgangsbasis für die Generierung sind viele Meta-Modelle tauglich, so dass eine optimale Methodenentscheidung nur basierend auf der zu generierenden Applikation getroffen werden kann. Interaktionsmodellierungssprachen eignen sich daher meist nur für eine Teilmenge der interaktiven Applikationstypen, da sowohl daten- bzw. strukturorientierte als auch ablauforientierte Modelle nicht jeden Zusammenhang wiedergeben können (vgl. Ziegler 1996). So ist es beispielsweise unmöglich, eine an einen bestimmten Dialogschritt gekoppelte Aktion in einem rein datenzentrierten Modell zu beschreiben. Ein rein ablauforientiertes Modell kann dagegen die Struktur eines Entity-Relationship-Diagramms kaum abbilden.

Für die Entwicklung von datenbank- bzw. datensatzorientierter Systemen wurden bereits zielführende Modellierungsansätze und Werkzeuge zur Generierung ergonomischer Benutzungsoberflächen erarbeitet (vgl. Hofmann 1998; Kruschinski 1999; Janssen 1993). Dabei finden oft auf Entity-Relationship- oder OOA-Diagrammen basierende, datenzentrierte bzw. objektorientierte Meta-Modelle Verwendung.

Bei technischen Entwicklungswerkzeugen, wie Konfiguratoren, stehen einzelne Einstellungen und Abläufe im Vordergrund, die oft eine durch die Anwendungsdomäne festgelegte Reihenfolge und gegenseitige Abhängigkeiten besitzen (vgl. Schlegel 2002). Es existieren viele zwischengeschaltete Kommunikations- und Überprüfungsvorgänge, die häufig eine schrittweise Beschreibung erfordern. Die Datenrepräsentation

kann auch mit objektorientierten Methoden beschrieben werden, allerdings liegt der Anwendung meist keine Datenbank mit Satzstrukturen zugrunde. Der Kontrollfluss ist in datenzentrierten Ansätzen meist nur unter Zuhilfenahme weiterer Modelle zu integrieren. Eine Lösungsmöglichkeit bietet daher die konträre Herangehensweise: eine an Abläufen und enthaltenen Interaktionen orientierte Modellierung.

4. Eine interaktions- und ablauforientierte Modellierungssprache zur Oberflächengenerierung

Für die ergonomische Oberflächengenerierung wird ein deskriptives Ausgangsmodell benötigt, das beispielsweise mit Hilfe einer Modellierungssprache erzeugt werden kann. Interaktionsorientierte Beschreibungsformen für den Systementwurf wurden bereits vorgeschlagen, beispielsweise in Form der UML-Dialogbeschreibung UMLi (vgl. Pinheiro da Silva 2000). Für die Analyse- und Spezifikationsphase stehen allerdings kaum Methoden zur Verfügung, die auch für spätere Phasen verwendbare Modelle liefern.

Da eine Modellierung ohne Iteration meist nicht möglich ist, muss eine Modellierungsmethode Konzepte bereitstellen, die eine hierarchische Strukturierung und stufenweise Vervollständigung ermöglichen, und so eine schritt- und stufenweise Festlegung der Interaktionsschritte und eine inkrementelle Vorgehensweise unterstützen. Nur so kann das Modell gleichermaßen für die initiale Modellierung wie für die nachfolgende Verfeinerung eingesetzt werden.

Für die Interaktionsmodellierung von Automotive-Konfiguratoren wurden bei ETAS folgende Anforderungen an die Modellierungsmethode definiert:

- Möglichkeit zur Erstellung eines initialen, grobstrukturierten Modells, das speziell Interaktionen und Abläufe berücksichtigt
- Verfeinerungsmöglichkeiten für dieses Modell bis hin zur maschinell verarbeitbaren Interaktionsspezifikation
- Eignung für die Generierung von Oberflächenprototypen
- Keine Verwendung mehrerer Notationen oder Modelle
- Integration von Projekt- und Prozessinformationen, wie beispielsweise den Entwicklungsstatus von Komponenten
- Kontextbezogene Sprachunterstützung für mehrere Länder direkt im Modell
- Verwendung in der Praxis verbreiteter Konzepte und Formate

Die zu entwickelnde Modellierungssprache muss neben der Erfüllung dieser Anforderungen die Unterstützung einer inkrementellen und iterativen Vorgehensweise bieten, wie sie für die Modellierung und Umsetzung von interaktiven Anwendungen, wie Entwicklungswerkzeugen, typisch ist.

Dabei ist es wichtig, nicht nur einen bestimmten Aspekt der zu entwickelnden Software zu modellieren. Vielmehr müssen Informationen der funktions-, daten- und ablauforientierten Modellierung ebenso einfließen wie nicht-funktionale Anforderungen.

5. Existierende Konzepte

Für die Modellierung von Anforderungen und Systemverhalten existiert bereits eine Vielfalt von Ansätzen. Diese sollen zur besseren Einordnung der beschriebenen Methode im Folgenden kurz erläutert werden.[3]

Für die Verhaltensmodellierung stehen bereits basale Methoden wie Zustandsautomaten, Statecharts (Harel 1987) und Petrinetze (Petri 1962) zur Verfügung. Durch Weiterentwicklung entstanden dedizierte Spezifikationsmethoden wie ViewNet (Ziegler 1997), die Requirements State Machine Language (RSML), die Software Requirements Engineering Method (SREM), ASTRAL und die Specification and Description Language (SDL). Die Erstellung und Pflege dieser Modelle ist sehr aufwändig, da jeder Schritt exakt modelliert werden muss. Als eigenständige Modellierungsmethode besitzen sie nicht genügend Informationen für die Oberflächenerzeugung und können deshalb zur Gesamtmodellierung nur in Kombination mit anderen Verfahren eingesetzt werden (vgl. Joos 1999). Als Ergänzung zu datenzentrierten Modellen finden sie allerdings in UI-Generatore-Ansätzen, wie GENIUS (vgl. Janssen 1993), Verwendung.

Eine weitere Möglichkeit zur Modellierung bieten die funktionsorientierten Beschreibungssprachen Structured Analysis (SA), Structured Analysis and Design Technique (SADT), STATEMATE und Jackson System Development (JSD). JSD eignet sich kaum für die Anforderungsmodellierung, da es sehr entwurfsnah ist. SA, STATEMATE und SADT konzentrieren sich stark auf den Datenfluss innerhalb der Applikation, was ihren Einsatz als grundlegendes Modell für die initiale Interaktionsmodellierung erschwert. SA und STATEMATE nutzen hierarchisch zerlegbare Datenflussmodelle mit Mini-Spezifikationen und Datenlexikon. Datenmanipulationen und -verknüpfungen lassen sich jedoch gut abbilden. Komplexere Interaktionen sind allerdings nur textuell zu beschreiben. Eine computergestützte Verarbeitung ist somit nur schwer möglich.

Objektorientierte Modellierungsansätze wie OMT, OOAD, OOSA und ROOM basieren in der Verhaltensmodellierung auf Statecharts. UML fügt diesen noch Aktivitätendiagramme hinzu. Die objektorientierte Analyse (OOA) realisiert hingegen die Verhaltensmodellierung mit Zustandsautomaten. Die Open Modelling Language (OML, vgl. Firesmith, Henderson-Sellers, Graham 1998) verfolgt noch stärker als die UML das Konzept der Objekt- und damit Datenorientierung.[4]

Dem objektorientierten Paradigma folgend basiert die Kommunikation in objektorientierten Modellen auf der Versendung von Nachrichten. Entity-Relationship-Diagramme unterstützen keinen Mechanismus zur Verhaltensmodellierung. Funktionen werden in objektorientierten Ansätzen sehr implementierungslastig codiert, das heißt durch entsprechende Methoden im Modell. Für die Analyse eignen sich implementierungsnahe Darstellungen jedoch kaum.[5]

[3] Eine Beschreibung und Verweise zu den ohne Referenzen angegebenen Ansätzen finden sich in Joos 1999.

[4] Ein Vergleich der UML mit der OML findet sich in Henderson-Sellers, Firesmith 1999.

[5] Einen Vergleich der meisten hier genannten Methoden enthält Joos 1999.

Einzig die UML-Use-Cases bieten aufgrund der freien Beschreibung Möglichkeiten zur einfachen Verhaltensmodellierung in der Analysephase. Diese wird allerdings erkauft mit einer vollkommenen Unstrukturiertheit in der Use-Case-Beschreibung, es sei denn eine Strukturierung wird durch zusätzliche Styleguides oder Formalismen vorgeschrieben.

Stark verallgemeinert stehen für die Modellierung des Dialogverhaltens also hauptsächlich zwei Modellierungsansätze zur Verfügung: einerseits die Modellierung von Datenflüssen und andererseits die Modellierung von Zuständen und Zustandsübergängen. Das Konzept der Hierarchisierung und Verfeinerung von Spezifikationsbausteinen findet vielfach Verwendung.

Modellierungsmethoden aus dem Bereich der HCI wie ETIT, GOMS, CLG, TAG und CCT werden in Herczeg 1994 näher erläutert, spielen allerdings eher für eine aufgabenorientierte Modellierung unter kognitiven Aspekten eine Rolle als für Anforderungsspezifikationen in Verbindung mit generativen Ansätzen.

Kombinationen der einzelnen Modellierungsmethoden decken die meisten Anforderungen ab, verletzen jedoch durch die unterschiedlichen Darstellungsarten das Konzept der Modellhomogenität. Aus diesem Grund war es notwendig, basierend auf bereits vorhandenen Konzepten, eine interaktions- und ablauforientierte Sprache zu entwickeln.

6. Entwurf einer Interaktionsmodellierungssprache für die frühen Entwicklungsphasen

Mit diesem Ziel wurde die Interaction *Modeling Language* (IML) entwickelt. Sie verfolgt das Ziel, bestehende Konzepte unter Verwendung einer jederzeit erweiterbaren, auf XML[6] basierenden Sprachplattform zu integrieren und mit neuen Konzepten zu ergänzen. Auf diese Weise können umfangreiche, integrierte Modelle erzeugt werden, die mit Hilfe eines XML-Parsers eingelesen, verarbeitet und in eine Oberfläche mit Applikationsgerüst transformiert werden können.

Um eine frühe Benutzerpartizipation und eine hohe Nutzungsquote der Modelle zu erzielen, muss eine Eignung für die frühen Phasen der Anwendungsentwicklung sichergestellt werden.

Ein hierfür geeignetes und aus diesem Grund in Analyse und Spezifikation weit verbreitetes Hilfsmittel sind die UML-Use-Cases (vgl. Booch, Rumbaugh, Jacobson 1999), die aufgrund der noch sehr informalen Darstellung einfach und übergreifend eingesetzt werden können. Das Konzept der Use-Cases wurde aufgrund der Eignung für die frühen Phasen und aufgrund des hohen Verbreitungsgrades in strukturierter Form in die IML übernommen.

IML-Use-Cases und deren strukturierte Beschreibungskonzepte können dabei sowohl als Ausgangsbasis für die Entwicklung neuer Modelle als auch zur Grobstrukturierung bereits vorhandener, natürlichsprachlicher Ablaufbeschreibungen dienen – beispielsweise im Zuge eines Re-Engineering-Projekts.

Herkömmliche UML-Use-Cases sind weitgehend natürlichsprachlich und unstrukturiert. Sie eignen sich daher kaum für die detaillierte deskriptive Modellierung und die maschinelle Verarbeitung. Die IML kann dank der Basierung auf XML und einem DTD-gestützten[7] Meta-Modell direkt eine Strukturierung mit obligatorischen und fakultativen Blöcken anbieten. Die Unterteilung vereinigt mehrere Strukturierungsvorschläge, unter anderem von Ambler 2000.

Informationen, die in UML als Fließtext eingegeben werden mussten, können in IML partitioniert, geordnet, parametrisiert und sub-strukturiert werden. In nachgelagerten Entwicklungsprozessschritten können mit Generiermodulen weitere Modelle und Darstellungen extrahiert und auch neue Artefakte aus den Modellinformationen erzeugt werden. Die notwendigen Informationen für Ablauf- und Beziehungsdiagramme sind bereits im IML-Modell vorhanden. So kann ein Generator die benötigten Informationen aus dem Modell extrahieren und aufbereiten. Auf eine weitergehende Formalisierung, wie sie beispielsweise von Li 2002 vorgeschlagen wird, wurde mit Blick auf die Partizipation von Benutzern und eine leichte Methodeneinführung (Weber, Weisbrod 2003) verzichtet.

Use-Cases in UML und ebenso in IML enthalten Ablaufbeschreibungen, die das Verhalten von System und Benutzer oder auch zwischen mehreren Systemen widerspiegeln. Wichtig sind dabei zumeist Benutzerinteraktionen und Systemaktivitäten.

In IML enthält jedes Use-Case einen INTERACTION-Block (siehe Tabelle 1), in dem eine dedizierte Ablaufbeschreibung wiedergegeben ist. Diese Ablaufbeschreibung ist für die UI-Generierung maßgeblich. Sie unterteilt sich in den regulären Ablauf und eine beliebige Zahl alternativer Abläufe, die durch Aufrufe und Verzweigungen innerhalb des Use-Case, zum Teil jedoch auch über Use-Case-Grenzen hinweg, verbunden sind.

7. InteractionCases

Bei der Betrachtung herkömmlicher Use-Cases aus praktischen Anwendungsfäl-

Tabelle 1: Schematischer Aufbau des IML-Modells

IML_SPECIFICATION
PROJECT_DEFINITION
DISPLAYED_NAME
STAKEHOLDERS
LANGUAGES
TRANSFORMATION_PARAMETERS
SYSTEM_DEFINITION
DATA_DEFINITION
DATA_OBJECT
DATA_STORED_SIMPLE
DATA_TEMPORARY_SIMPLE
DATA_TEMPORARY_COMPLEX
DATA_STORED_COMPLEX
DATA_TABLE
DATA_CONSTANT
DATA_TEMPORARY_REFERENCE
USE_CASE
OPERATION
DESCRIPTION
UC_RESULT
EFFORT?
INFORMATION_FLOW
USER_GOAL+
ROLE+
DEPENDENCIES
UC_DYNAMICS
UC_STIMULUS
UC_PRECONDITIONS
UC_FAILURES
CONTEXT_MENU
INTERACTION
WORKFLOW
UC_POSTCONDITIONS
BUSINESS_RULES
REQUIREMENTS
IMPLEMENTATION_NOTES
REVIEWS

[6] Die eXtensible Markup Language (XML) ist eine Untermenge von SGML (vgl. http://www.w3.org/TR/REC-xml) und dient zur Datenstrukturierung.

[7] Eine *Document Type Definition* (DTD) definiert eine Menge gültiger XML-Strukturen für eine XML-Datei, die dieser Typ-Definition folgt.

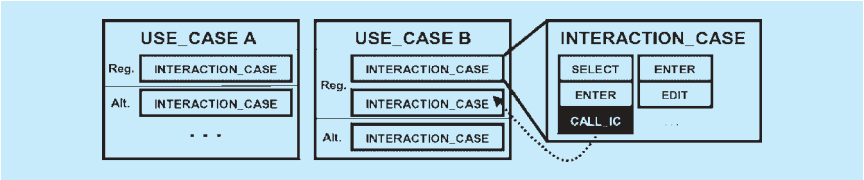

Bild 1: Aggregation von InteractionCases in Use-Cases

len steht nur der einzelne Interaktionsschritt als Strukturierungskriterium unterhalb der Abläufe zur Verfügung. Da größere Use-Cases viele Interaktionssequenzen enthalten können und oft mehrere Dialoge umfassen, fehlt ein geeignetes Mittel zur weiteren Strukturierung und Modularisierung. So ist es beispielsweise nicht möglich, einen bereits vorgesehenen Dialog in ein entsprechendes „Teil-Use-Case" abzubilden. Die IML führt deshalb als neues Konzept so genannte *InteractionCases* ein, mit Hilfe derer sich Use-Cases weiter unterteilen lassen.

Ein *InteractionCase* verkörpert dabei eine selbständige Interaktionssequenz, die Teil eines Use-Case ist und in Beziehung zu anderen Interaktionssequenzen stehen kann. Es umfasst eine oder mehrere zusammengehörige und aufeinanderfolgende Aktionen von System- oder Benutzerseite. Diese Aktionen liegen innerhalb desselben Use-Case und stellen zusammen einen Vorgang dar, der aus ergonomischen und ablauftechnischen Gesichtspunkten nicht aufgespaltet werden sollte.

Ein Use-Case muss dabei innerhalb des regulären Ablaufs mindestens ein InteractionCase enthalten, da ein leeres Use-Case nicht möglich ist. Es kann andererseits aber eine beliebige Anzahl von *InteractionCases* kombinieren (vgl. Bild 1). Eine beliebige Anzahl alternativer Abläufe, die ebenfalls aus InteractionCases aufgebaut sind, komplementiert den regulären Ablauf zu einem vollständigen Interaktionspart des Use-Case.

Die XML-Basierung ermöglicht zu jedem Zeitpunkt eine einfache Synthese von mehreren Teil-*InteractionCases* zu einem größeren oder die Aufspaltung eines größeren in mehrere kleine InteractionCases, so dass ein Use-Case in mehrere weitgehend atomare Interaktionssequenzen zerfällt.

XML-Dokumente können von einem Parser in einem DOM-Baum[8] vorgehalten werden, so dass einem InteractionCase A zugeordnete Einträge bzw. Abläufe automatisch einem anderen InteractionCases B hinzugefügt werden können. Die direkte Lesbarkeit des Plain-Text von XML-Dokumenten ermöglicht Entwicklern mit XML-Erfahrung zudem die direkte Bearbeitung des XML-Dokuments.

8. Gruppen in IML

Auch *InteractionCases* nehmen oft – speziell bei komplexen Dialogen – Dimensionen an, die eine hierarchielose Modellierung aus Interaktionsschritten unmöglich machen und sowohl den Entwickler als auch den späteren Benutzer überfordern würden. In der HCI wird daher beim Dialoglayout oft das Mittel der Gruppierung oder das Gesetz der Nähe eingesetzt.

Das in der IML als Sub-Konstrukt von *InteractionCases* spezifizierte GROUP-Tag[9] definiert die Zusammengehörigkeit von Interaktionsschritten bzw. späteren Interaktionselementen und ermöglicht so deren Gruppierung.

Damit wird es möglich, unterschiedliche Betrachtungsebenen von Gruppen festzulegen. So kann die Reihenfolge von Bearbeitungsschritten variabel oder zwingend gestaltet, Gruppierungen optisch sichtbar gemacht und Gruppen wiederum hierarchisch in Sub-Gruppen untergliedert werden. Eine Gruppe legt in IML nur eine Gruppierungssemantik fest, nicht die exakte Repräsentation. Prozess- und Anwendungswissen fließen so schon früh in die Spezifikation mit ein und können im folgenden UI-Generierschritt ausgewertet werden. Eine vorzeitige Festlegung von Entwurfs- und Implementierungsentscheidungen wird so vermieden. Erst die Umsetzung des Modells erzwingt eine Realisierungsentscheidung anhand von Kriterien wie Zielsystem und Zielgruppe.

Für Entwickler ebenso wie für in die Entwicklungsarbeit einbezogene Anwender wird das Modell so überschaubar und enthält für die Benutzungsschnittstellen-Entwicklung wichtige Zusatzinformationen, die in einer späteren Projektphase oft nicht mehr zur Verfügung stehen würden. Der Generator[10] kann seinerseits aus den zusätzlichen Informationen ein besseres Layout und eine angemessene Dialogsteuerung ableiten. So können Elemente optisch als zusammengehörig gekennzeichnet, zusammen auf neue Reiter ausgelagert oder in Form eines „Wizards" sequentialisiert werden.

Eine Gruppierung ist nicht zwingend vorgeschrieben und schließt einzelne Interaktionsschritte auf derselben Ebene auch nicht aus, so dass Interaktionsschritte und Gruppen von Interaktionsschritten gleichberechtigt nebeneinander stehen können. Die Gruppierung kann wie bereits zuvor erwähnt hierarchisch erfolgen, so dass Gruppen ihrerseits wieder (Sub-)Gruppen und Interaktionschritte enthalten.

Für die Oberflächengenerierung wie auch für die konventionelle Entwicklung von Oberflächen haben *InteractionCases* und Gruppen deshalb so große Bedeutung, weil sie Use-Cases soweit strukturieren, dass entsprechende Blöcke vom Generator erkannt werden können.

Zudem umfassen Gruppen auch Angaben über die Bedeutung der Reihenfolge der in ihnen enthaltenen Interaktionschritte. Auf diese Weise kann der Spielraum des Generators oder des Oberflächengestalters bei der Elementanordnung eingeschränkt oder erweitert sowie Elemente als zusammengehörig gekennzeichnet werden. Dabei reicht das Spektrum von beliebiger Anordnung über Kann-Veränderung bis hin zu zwingender Reihenfolge. Abhängigkeiten zwischen einzelnen Schritten und eine

[8] Beispielsweise unter Verwendung der XML Core Services 4.0 (vgl. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/xmlsdk/htm/sdk_whatsnew_0wts.asp)

[9] Eine nähere Beschreibung des GROUP-Tags ist in Schlegel 2002 nachzulesen.

[10] Ein prototypischer Generator wurde basierend auf der IML-Spezifikation entwickelt. Alle in diesem Paper beschriebenen Schritte und Modelle wurden implementiert (siehe Schlegel 2002).

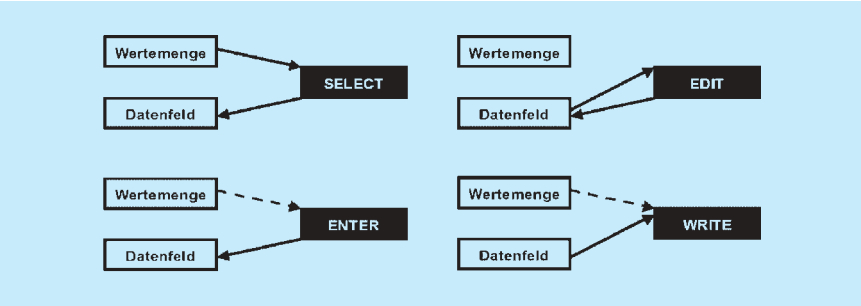

Bild 2: Funktionsweise von ENTER, EDIT, SELECT und WRITE

Einstellung der Wichtigkeit vervollständigen das Instrumentarium.

Dem Spezifizierer stehen damit schon in der Analyse- und Spezifikationsphase die Mittel zur Verfügung, um anwendungsinhärente Zwänge und Flexibilitäten im Ablauf und damit auch in der darauf aufbauenden Dialoggestaltung zu modellieren.

9. Interaktionsschritte

InteractionCases und Gruppen enthalten ihrerseits die eigentlichen Interaktionen. Diese werden in IML mit Hilfe von Interaktionsschritten modelliert. Um Informationen vom Benutzer zu erhalten, sieht die IML drei unterschiedliche Konstrukte vor, die jeweils über einen einzelnen Interaktionsblock Informationen akquirieren (vgl. auch Bild 2):

- Eingabe neuer Informationen (ENTER)
- Veränderungen existierender Informationen (EDIT)
- Auswahl aus existierenden Informationen (SELECT), fallweise ergänzt um eine Erweiterungsmöglichkeit der Auswahloptionen

Je nach Konstrukt sind untergeordnete Mechanismen zur Beschreibung von Mengengerüst, Werten, Datenquellen und dynamischem Verhalten verfügbar. Ein Großteil der Interaktionselemente eines generierten oder klassisch aus der IML-Spezifikation abgeleiteten Oberflächenprototyps geht deshalb aus den Interaktionsschritten und diesen zugeordneten Informationen hervor.

Weitere Konstrukte für allgemeine Ausgaben (WRITE), Funktionen und Ablaufkontrolle sowie ein Konzept für Meldungen ergänzen die Interaktionsschritte in einem IML-Modell. Zur Steuerung des Ablaufs und zur Einbindung UI-unabhängiger Programmteile existieren Ablaufelemente, die den Interaktionsschritten gleichgestellt sind. Deren konzeptionelle Einbindung soll im Folgenden näher beschrieben werden.

10. Ablaufkontrolle und Anbindung der fachlichen Funktionen

Aus einem *InteractionCase* heraus kann bedingungsabhängig, eventabhängig oder bei jedem Durchlauf ein anderes *InteractionCase* aufgerufen werden. Aus diesen Informationen kann vom Generator später eine Dialogsteuerung erzeugt werden. Es ist zudem möglich, aus den Aufrufbeziehungen Abhängigkeiten auf *InteractionCase*- oder Use-Case-Ebene abzuleiten und diese beispielsweise in Form eines Aufrufdiagramms als Entwicklungsdokumentation aufzubereiten.

Interaktionsschritte und Gruppen eines IML-Modells werden in der Ablaufreihenfolge des Modells festgehalten. Sind keine weiteren (Gruppen-)Informationen vorhanden, bestimmt die Reihenfolge des Auftretens auch den späteren Ablauf. Mit bedingten Verzweigungen (IF) und dem Aufruf anderer InteractionCases (CALL_IC) kann der Dialogablauf im IML-Modell direkt vorgegeben werden und ermöglicht so eine Modellierung, die direkt dem Ablauf folgt.

Für die konzeptuellen Diskussionen hinsichtlich der Benutzungsoberfläche reichen anfangs die aus dem Interaktionsmodell generierbaren Oberflächenprototypen aus. Schreitet das Projekt weiter fort, müssen auch Teile der fachlichen Funktionen an die Oberfläche angebunden werden, um in jeder weiteren Iteration eine möglichst vollständige Anwendung erzeugen zu können.

Im Gegensatz zu datenzentrierten Ansätzen ermöglicht ein interaktions- und ablauforientiertes Modell eine einfache Einbindung der fachlichen Funktionen an definierten Punkten im Ablauf. So können Methodenaufrufe bei Erreichen eines bestimmten Interaktionsschritts oder beim Auslösen eines Events vom Generator direkt im Code erzeugt werden, so zum Beispiel Systemfunktionen oder vom Entwickler selbst definierte Methoden.

Die IML ermöglicht den direkten Eintrag von parametrisierbaren Aufrufen der fachlichen Funktionen innerhalb von InteractionCases: Wird beispielsweise eine einfache Berechnung umgesetzt, müssen zwei Werte eingegeben, die Operation ausgewählt und zum Schluss der Aufruf der Berechnungsfunktion getriggert werden. Diese erhält beide Werte und den Operator als Aufrufparameter. Dabei kann die Berechnung unmittelbar oder auf Befehl ausgelöst werden, beispielsweise realisiert durch eine entsprechende Schaltfläche wie in Bild 3.

Für die zu tätigenden Eingaben werden im Modell Datensenken in Form von Datenbankeinträgen, einfachen Varia-

Bild 3: Beispiel für die Einbettung einer Berechnungsfunktion in einen Dialog

blen oder noch unspezifizierten Datencontainern angegeben. Sobald diese im Modell zur Verfügung stehen, können sie zur Parametrisierung von Aufrufen herangezogen und auch als Ziele für die Ergebnisse von Benutzereingaben verwendet werden.

Die Ablauforientierung ermöglicht hier zudem eine Überprüfung, ob die für den Aufruf genutzten Variablenwerte zu diesem Zeitpunkt überhaupt zur Verfügung stehen. Ist dies nicht der Fall, kann in reihenfolgeflexiblen Gruppen eine Umstrukturierung erfolgen oder eine bisher freie Reihenfolge erzwungen werden, so dass das zum Aufruf benötigte Datum nun zum Zeitpunkt des Aufrufs schon zur Verfügung steht. Sieht das Modell eine statische Reihenfolge vor, kann der Generator dem Entwickler mit einer Fehlermeldung Hinweise und Verbesserungsvorschläge liefern.

11. Interaktionsorientierung

Wenngleich in IML auch ein Datenmodell zur Verfügung steht, liegt der Fokus auf der Interaktionsbeschreibung. Datenquellen und -senken werden nur in Ausnahmefällen zentral, meist jedoch dort beschrieben, wo sie im Dialogablauf benötigt werden.

Diese Vorgehensweise hat speziell in den frühen Phasen viele Vorteile, in denen meist organisatorische Abläufe bekannt sind, erste Datenstrukturen jedoch meist noch der Phantasie des Entwicklers oder den Einschränkungen von Vorgängersystemen entspringen.

Während eine datenstrukturorientierte Beschreibungsform kaum Aussagen über sinnvolle Eingabesequenzen machen kann und Auswirkungen semantischer Abhängigkeiten kaum berücksichtigt, spiegelt ein interaktions- und ablauforientiertes Modell alle für Dialogstrukturierung und -ablaufsteuerung notwendigen Vorgaben wider und sammelt – quasi nebenbei – die notwendigen Informationen zum Aufbau der Datenstruktur.

Zudem ist es möglich, an Sequenzen bestimmte Sicherheitsmechanismen zu koppeln, die automatisierte und standardisierte Datensicherung, Undo-Funktionalität oder Zugriffsschutz basierend auf einem einfachen Modell-Attribut definieren. So können die Fehlerrobustheit, die Erwartungskonformität und weitere Eigenschaften der zu entwickelnden Software bezüglich ISO 9241-10 (siehe ISO 1998) positiv beeinflusst werden.

Eine direkte Umsetzung kann dann durch Wiederverwendung eine hohe Kapselung und Konsistenz aufweisen. Es kann mehr Code automatisiert erzeugt werden, da für die genannten Mechanismen keine entsprechenden Code-Fragmente, sondern nur Standard-Funktionsaufrufe nötig sind.

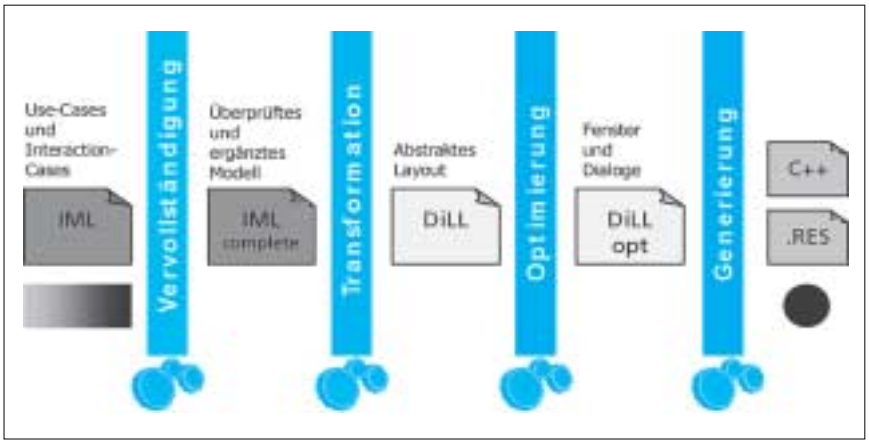

Bild 4: Transformationspipeline des Oberflächengenerators

12. Standardisierung

Das Konzept der Verwendung von standardisierten Mechanismen über einfache Schalter wird in IML generell stark betont. Der Grund dafür liegt auf der Hand: Können Mechanismen zur Unterstützung des Benutzers, wie z. B. Undo, einfach und standardisiert eingebunden werden, entfällt die Notwendigkeit zur Neuentwicklung derartiger Funktionalität. Eine meist sehr heterogene Methodenlandschaft innerhalb einer großen Applikation kann so durch ein Bündel von Standardfunktionen ersetzt werden, die dann alle Vorteile von Standardkomponenten quasi per Mausklick zur Verfügung stellen.

Die Menge der Konstrukte, Attribute und Werte, die dem Modellierer in Form der IML zur Verfügung stehen, wirkt sowohl fokussierend also auch standardisierend: Es wird erschwert, sehr heterogene Mengen von Mechanismen zu spezifizieren. Stattdessen werden Standardmechanismen angeboten, die ohne Programmieraufwand direkt zur Verfügung stehen und somit helfen, die Verständlichkeit und Wartbarkeit zu verbessern und den Aufwand zu verringern.

Steht trotz der Vielzahl an Möglichkeiten ein notwendiges Konstrukt nicht zur Verfügung, kann es nachträglich in der IML Document Type Definition (DTD) ergänzt werden. Da die DTD die Definitionsgrundlage für den UI-Generator bildet, muss dann auch dieser entsprechend erweitert werden – das neue Konstrukt würde andernfalls von ihm ignoriert werden.

13. Transformation

Steht ein erstes IML-Modell der Anwendung zur Verfügung, kann der Generierungsprozess durchlaufen werden, der neben anderen Artefakten auch einen Oberflächenprototyp erzeugt. Wird nur die reine Oberflächengenerierung betrachtet, ähnelt das Konzept einer Pipeline in der Visualisierung, wie in Bild 4 gezeigt.

Die verschiedenen Stufen haben dabei folgende Funktion:

Vervollständigung: Hier wird das deskriptive Modell vervollständigt und überprüft, soweit dies automatisch erfolgen kann. So werden Hilfe-IDs ergänzt und die Texte für jede Zielsprache (Deutsch, Englisch etc.) auf Vollständigkeit überprüft.

Transformation: Dialoglayout und Dialogdynamik werden in Form des Zwischenformats, der so genannten Dialog-Layer-Language (DiLL), aus dem vervollständigten IML-Modell erzeugt. Die XML-basierte Sprache DiLL wurde so

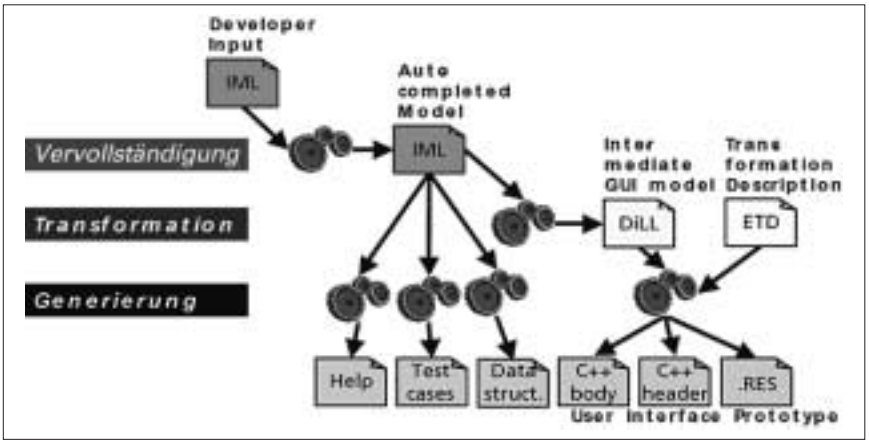

Bild 5: Transformationsansatz - Übersicht

konzipiert, dass sowohl eine abstrakte Repräsentation der Dialogstatik, als auch zur Verfügung stehende Befehlsstrukturen und dynamische Komponenten erzeugt werden können. Für das nun folgende Layout ist eine abstrakte Repräsentation der Dialoge unerlässlich. Das DiLL-Modell enthält im statischen Teil nur „Bildschirme“[11], Gruppen und Interaktionselemente. Aufgrund der Simplizität des zugehörigen Klassenmodells eignet sich diese Repräsentation für die initiale Elementanordnung und die nachfolgende Dialogoptimierung.

Optimierung: Die DiLL-Repräsentation enthält das Dialoglayout in Form von relativen Koordinaten und verfügt zudem über eine Rückkopplung zum IML-Modell. So wird die Optimierung des Dialoglayouts und der Abläufe erleichtert, die in graphischen – speziell mehrspaltigen – Dialogen unabdingbar ist. Kriterien wie Balance, Dialog-Seitenverhältnis, direkte Verfügbarkeit wichtiger Felder, Auslagerung von unwichtigen bzw. Kann-Elementen etc. sind mögliche Kriterien.

Generierung: Der letzte Schritt im Transformationsprozess ist die Generierung der Oberfläche. Die schon im DiLL-Modell vorhandenen Elemente werden durch ihre Definition in der so genannten „Element Type Definition“[12] (ETD) ersetzt, so dass bestimmten Blöcken zugeordnete Codefragmente entstehen. Die Befüllung von Code-Templates[13] mit den erzeugten Code-Blöcken liefert beispielsweise ein vollständiges, direkt compilierbares Visual-Studio-Projekt – den Oberflächenprototyp.

Die gesamte Transformation kann von einem oder mehreren Transformatoren durchgeführt werden. Dabei wird ein IML-Modell schrittweise in eine Oberfläche und weitere Dokumente überführt. Der gesamte Prozess läuft dabei wie in Bild 5 dargestellt ab.

Eine prototypische Implementierung des Generators steht bereits zur Verfügung und erzeugt UI-Prototypen in Form von Visual C++-Projekten. Zur Verarbeitung des vollen IML-Sprachumfangs sind jedoch Erweiterungen notwendig. Die Anpassung auf unterschiedliche Programmiersprachen erfolgt jedoch über externe Templates und ETDs, so dass eine Recompilierung des Generators für neue Ziel-Programmiersprachen nicht notwendig ist.

Aus dem vervollständigten IML-Modell oder auch dem DiLL-Modell können also weitere Resultate mit Hilfe eigenständiger oder integrierter Transformatoren erzeugt werden. Beispiele hierfür sind Testfälle, Hilfen, Diagramme, Statistiken, Verbesserungsvorschläge etc.

14. „National Language Support“

Viele Anwendungen werden in einer bestimmten natürlichen Sprache entwickelt – meist in der Landessprache oder in Englisch. Probleme treten oft erst in den späten Phasen auf, wenn die einsprachige Software „internationalisiert“ werden soll. Ganze Datenbanken von Texten und Meldungen müssen dann – meist losgelöst vom Anwendungskontext – übersetzt werden. Schon am simplen Beispiel des englischen Wortes „no“ (nein, kein, keine, etc.) ist zu ermessen, welche Probleme dabei entstehen. Zudem müssen alle basierend auf den Texten getroffenen Layoutentscheidungen in Frage gestellt werden.

Eine mögliche Lösung bietet das IML-Sprachkonzept mit einer Sprachintegration in die Spezifikation: Jeder Text wird schon in der Spezifikation einer bestimmten Sprache[14] oder allen Zielsprachen des Projekts zugeordnet, so dass zur Vereinfachung Wörter wie „VGA“ oder „PC“, die in allen verwendeten Sprachen gleich lauten, nur einmal angegeben werden. Durch die Einbettung in das IML-Modell bleibt der Kontext erhalten und gewährleistet so eine korrekte Übersetzung.

Anhand der festgelegten Zielsprachen kann das Modell vom Generator auf Vollständigkeit bezüglich dieser Zielsprachen überprüft werden. Die Übersetzung im Modell- bzw. Anwendungskontext erleichtert den Übersetzungsprozess durch Kontextinformationen und die zur Verfügung stehende Modellsemantik.

Sobald ein rechnergestützter Übersetzer IML-Modellinformationen berücksichtigen könnte, wäre auch die Nutzung von automatischen Übersetzern mit verbesserten Ergebnissen möglich. Ein sprachlicher Kontext ist aus dem IML-Modell jedoch ungleich schwerer abzuleiten als die bereits beschriebenen Interaktionsabläufe.

Die Integration der Texte in das IML-Modell ermöglicht eine auf Vollständigkeit überprüfbare Erweiterung mit zusätzlichen Sprachen: Soll Französisch hinzukommen, wird die Sprachmenge um diesen Eintrag erweitert. Der Transformator spürt nun alle Fehlstellen im Modell auf, an denen noch kein französischer Eintrag vorliegt. Die vorhandenen Einträge in anderen

[11] Ein Bildschirm (SCREEN) enthält alle auf einmal sichtbaren Elemente an einer Stelle im Dialogablauf, beispielsweise repräsentiert er ein Dialogfenster oder ein Bildschirmmenü.

[12] Eine ETD enthält für genau eine Zielplattform die programmtechnischen Umsetzungen aller in DiLL erzeugbaren Elementtypen in Form von Codefragmenten und Kompositionsvorschriften.

[13] Ein Code-Template enthält ein Gerüst für eine Code-Datei mit Ankerpunkten, an denen Code eingefügt werden kann.

[14] Beispielsweise „Adresse“ zu „Deutsch“ und „address“ zu „English“

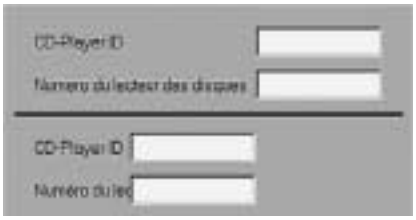

Bild 6: Probleme mit unterschiedlichen Sprachen bei statischem Layout

Sprachen und die Beschreibung des aktuell bearbeiteten Interaktionsschritts erleichtern eine dem Kontext gemäße Übersetzung.

Erzeugt der Transformator das statische Layout abhängig von den Textlängen einer Sprache, ist zwar eine exakte Positionierung möglich, wegen unschöner Überlappungen durch Längenunterschiede kann jedoch eine Sprachumschaltung nicht mehr zur Laufzeit erfolgen, wie aus Bild 6 ersichtlich.

Alternativen sind die Berechnung des Textlängenmaximums für alle enthaltenen Sprachen oder eine dynamische Anpassung zur Laufzeit, z. B. durch die Verwendung unterschiedlicher Dialogressourcen, die für jede Sprache getrennt berechnet wurden, oder durch Layout-Manager.

15. Modellierung und Generierung mit IML

Eine interaktionsorientierte Modellierungssprache mit der beschriebenen Benutzungsoberflächengenerierung bietet Vorteile in Entwicklungsprojekten, die einem iterativen, benutzerorientierten Entwicklungsprozess folgen, der durch intensives Oberflächenprototyping gestützt wird. Die Ablauforientierung eignet sich vor allem für stark verzweigte und entscheidungsorientierte Anwendungen im technischen oder Service-Bereich.

Der Generierung sind engere Grenzen als der Beschreibungssprache gesetzt (vgl. Kruschinski 1999). Während mit der IML viele Ein-/Ausgabeaktionen und Abläufe beschrieben werden können, ist die Generierung auf Oberflächen beschränkt, bei denen zur Informationsakquisition die drei oben genannten Mechanismen ENTER, EDIT und SELECT eingesetzt werden können. Komplexe Eingaben wie bei Zeichen- und CAD-Systemen können zwar rudimentär modelliert werden, müssen in ihrer Funktionalität jedoch direkt implementiert oder vom Meta-Modell als Standardkomponenten zur Verfügung gestellt werden.

Sollen dagegen stärker dialogorientierte Anwendungen, wie Konfiguratoren, entwickelt werden, können weitgehend vollständige UI-Prototypen generiert werden. Für Entwickler und Unternehmen wirkt sich der anfangs in die Modellbildung investierte Aufwand in den späteren Phasen und im Produkt positiv aus und führt bei entsprechendem Ausbau des Transformators zu Zeit- und Kosteneinsparungen, speziell in Folgeprojekten.

In Use-Cases enthaltene Informationen werden häufig in späteren Phasen oder in Folgeprojekten benötigt. Da meist jedoch kein entsprechendes Ausgangsmodell mehr zur Verfügung steht, muss dieses mit Hilfe aufwändiger Reengineering-Verfahren (wieder) abgeleitet werden (vgl. Bojic, Velasevic 2000). Ein integriertes Modell erhöht den Aufwand in der Analyse- und Spezifikationsphase, wie erwähnt, zunächst durch die Modellbildung. Die Konsolidierung der Anforderungen und die Ableitung von UI-Prototypen und anderen Artefakten verringert jedoch den Gesamtaufwand für das Entwicklungsprojekt. Nachfolgende Projekte profitieren in noch größerem Maße von einem bereits existierenden Modell. Können Fehler durch Modellbildung und Prototypevaluation früh erkannt werden, können die Projektkosten beträchtlich gesenkt werden (vgl. z. B. Pressman 1997).

16. Resümee und Ausblick

Mit dem Ziel, die Softwarequalität zu verbessern sowie Anforderungen und Wissen aus den frühen Projektphasen über die gesamte Entwicklungsdauer zur Verfügung zu stellen und auch anpassen zu können, wurde das IML-Meta-Modell entwickelt, das über eine Interaktions- und Ablauforientierung verfügt. Aus einem früh entwickelten IML-Modell können Oberflächenprototypen und weitere Artefakte für die Entwicklung und Dokumentation generiert werden. So erhöht sich die Akzeptanz bei den Entwicklern, da das einmal erstellte Modell zur Generierung genutzt werden kann und so einen in späteren Phasen wesentlich höheren Aufwand einspart.

Für Kunden und Benutzer wird durch kurze Zyklen und die direkte Abbildbarkeit von Modell-Änderungen auf den UI-Prototypen eine enge Partizipation möglich. Realistische Prototypen mit Verbindung zur Anforderungsspezifikation ermöglichen eine frühe Abstimmung der Software auf den Benutzerkreis, so dass die Menge teurer Nachbesserungen verringert werden kann.

Konsequent umgesetzt ermöglicht dieses Konzept ein iteratives, durch Oberflächenprototyping gestütztes Vorgehen, das die Möglichkeiten zur Benutzerpartizipation verbessert, Informationen aus den frühen Projektphasen zur Verfügung stellt und diese zudem aufwandsreduzierend verwertet.

Bei entsprechender Ausstattung mit unterschiedlichen Transformatoren und Transformationsbeschreibungen (vgl. Schlegel 2002) können Oberflächen für unterschiedliche Plattformen aus demselben Modell erzeugt werden. Darüber hinaus ist auch die Unterstützung vollkommen anderer Ziel-Systeme mit geringer zur Verfügung stehender Fläche, wie beispielsweise Telematikanwendungen im Automobil[15] oder PDA-Lösungen, denkbar. Zudem können auch multimodale Anwendungen erzeugt werden, da die IML weitgehend umsetzungsneutral gehalten ist.

Mit dem Entstehen moderner Entwicklungsprozesse und der wachsenden Heterogenität der zu entwickelnden Benutzungsschnittstellen, werden integrative, mehrstufige Modellierungs- und Generierungsmethoden zunehmend Bedeutung erlangen.

Literatur

Ambler, S.W.: Documenting a use case: What to include, and why. (2000) **http://www-106.ibm.com/developerworks/library/tip-docusecase.html (Letzter Zugriff: 9.1.2003).**

Bojic, D.; Velasevic, D.: A Use-Case Driven Method of Architecture Recovery for Program Understanding and Reuse Reengineering.

[15] Telematikanwendungen können beispielsweise zur Navigation oder für Einstellungen im Automobil genutzt werden. Vom DIN-Format bis zu hochwertigen (3D-)TFT-Displays werden viele Lösungen angeboten.

Proceedings of the Conference on Software Maintenance and Reengineering. *IEEE Press* (2000) 23–31.

Booch, G.; Rumbaugh, J.; Jacobson, I.: *The Unified Modeling Language User Guide.* Addison-Wesley object technology series, Addison-Wesley, 1999.

Firesmith, D.; Henderson-Sellers, B.; Graham, I.: *OPEN Modeling Language (OML) Reference Manual.* Cambridge, New York: Cambridge University Press, 1998.

Henderson-Sellers, B.; Firesmith, D.G.: Comparing OPEN and UML: the two third-generation OO development approaches. *Information and Software Technology* **41.3** (1999) 139–156.

Herczeg, M.: Software-Ergonomie: *Grundlagen der Mensch-Computer-Kommunikation.* Reading, Mass.: Addison-Wesley, 1994.

Hofmann, F.: *Grafische Benutzungsoberflächen: Generierung aus OOA-Modellen.* Heidelberg, Berlin: Spektrum Akademischer Verlag, 1998.

ISO: ISO 9241 – Ergonomic requirements for office work with visual display terminals (VDTs). International Standard, International Standards Organization, 1998.

Janssen, C.; Weisbecker, A.; Ziegler, J.: Generating User Interfaces from Data Models and Dialogue Net Specifications. In: *Proceedings of INTERCHI'93* (1993) 418-423.

Joos, S.: ADORA-L - Eine Modellierungssprache zur Spezifikation von Software-Anforderungen. Dissertation, Universität Zürich, 1999.

Kruschinski, V.: *Layoutgestaltung grafischer Benutzungsoberflächen: Generierung aus OOA-Modellen.* Heidelberg, Berlin: Spektrum Akademischer Verlag, 1999.

Li, X.; Liu, Z.; He, J.: Formal and Use-Case Driven Requirement Analysis in UML. Proceedings of the 25th Annual International Computer Software and Applications Conference (COMPSAC'01). *IEEE Press* (2002) 215-224.

Pinheiro da Silva, P.; Paton, N.W.: A UML-Based Design Environment for Interactive Applications. Proceedings of the 2nd International Workshop on User Interfaces to Data Intensive Systems (UIDIS'01) (Hrsg. Kapetanios, E.; Hinterberger, H.), Zürich: *IEEE Computer Society* (2001) 60–71.

Pressman, R.: *Software Engineering: A practitioner's approach.* Fourth Edition, European Adaption, McGraw-Hill, 1997.

Schlegel, T.: Entwurf und Erprobung eines software-gestützten Verfahrens zur Anwendung software-ergonomischer Methoden in den frühen Phasen der Anwendungsentwicklung. Universität Stuttgart/ETAS GmbH, 2002.

Software Engineering Institute (SEI): Capability Maturity Model® for Software (SW-CMM®). **http://www.sei.cmu.edu/cmm/cmm.html, 2002 (Letzter Zugriff: 9.1.2003).**

Weber, M.; Weisbrod, J.: Requirements Engineering in the Automotive Development: Experiences and Challanges. *IEEE Software* **20**, No. 1 (2003).

Ziegler, J.: *Eine Vorgehensweise zum objektorientierten Entwurf graphisch-interaktiver Informationssysteme.* Berlin: Springer Verlag, 1996.

Ziegler, J.: ViewNet – Konzeptionelle Gestaltung und Modellierung von Navigationsstrukturen. In: *Softwareergonomie '97* (Hrsg. Liskowsky, R.; Velichkovsky, B. M.; Wünschmann, W.). Teubner (1997) 343–350.

Ziegler, J.: *Eine Vorgehensweise zum objektorientierten Entwurf graphisch-interaktiver Informationssysteme.* Berlin: Springer Verlag, 1996.

1

2

1 Dipl.-Inf. Thomas Schlegel, wissenschaftlicher Mitarbeiter Competence Center Softwaretechnik Fraunhofer IAO. Hauptarbeitsgebiete: UI-Generierung, Interaktionsmodelle, Computer Aided Cooperative Software Engineering sowie computergestützte Kooperations- und Kreativitässitzungen.
E-Mail: Thomas.Schlegel@iao.fraunhofer.de

2 Dr.-Ing. Alexander Burst, ETAS GmbH, Bereich „Automotive Embedded Control Tools". Hauptarbeitsgebiete: CASE-Methodik, Echtzeitsysteme und Rapid Prototyping.
E-Mail: alexander.burst@etas.de