Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. Thomas Ertl
Betreuer:	Dr. Matthias Ressel
Betreuer ETAS GmbH:	Dr. Alexander Burst
Beginn am:	01.02.2002
Beendet am:	30.07.2002

CR-Klassifikation (nach ACM CCS98):

D.2.2 [Design Tools and Techniques] -- User interfaces
D.2.1 [Requirements/Specifications] -- Tools
H.5.2 [User Interfaces] -- Ergonomics, Graphical user interfaces (GUI), Screen design, Standardization, User-centered design
I.3.6 [Methodology and Techniques] -- Ergonomics, Languages

Keywords: Interaction Modelling Language, IML, Dialog Layer Language, DiLL, user interface generation, interaction specification, dialog layout, layout optimisation, HERBS

Diplomarbeit Nr. 1985

Entwurf und Erprobung eines software-gestützten Verfahrens zur Anwendung software-ergonomischer Methoden in den frühen Phasen der Anwendungsentwicklung

Thomas Schlegel

Fakultät Informatik
Institut für Informatik (IFI)
Abteilung Visualisierung und interaktive Systeme (VIS)
Universität Stuttgart
Breitwiesenstraße 20-22
D-70565 Stuttgart

Danksagung

Danken möchte ich Prof. Thomas Ertl (VIS), der mir die Bearbeitung dieses Wunschthemas bei ETAS erst ermöglicht hat und meinem Betreuer Dr. Matthias Ressel (VIS), der mich bei Fragen immer unterstützt und mir trotzdem eine eigenverantwortliche Bearbeitung ermöglicht hat, für die angenehme Zusammenarbeit und ihr außergewöhnliches Engagement.

Meinem Betreuer bei ETAS, Dr. Alexander Burst, möchte ich danken für seinen großen Einsatz und seine große Motivation. Er hat mich sehr weit über das übliche Maß hinaus unterstützt und gefördert.

Danken möchte ich auch meinen anderen Kollegen bei ETAS für die Hinweise, die Diskussionen und die Unterstützung.

Inhalt

1 EINLEITUNG 1

1.1 RENAISSANCE DER GENERIERUNG ERGONOMISCHER OBERFLÄCHEN 1

1.2 AUFBAU 2

2 GRUNDLAGEN 3

2.1 NORMEN UND STANDARDS 3

2.1.1 Bildschirmarbeitsverordnung (BildschArbV) 3

2.1.2 DIN EN ISO 9241 4

2.1.3 ISO 13407 und ISO TR 18529 4

2.2 USABILITY UND UTILITY 5

2.3 DIALOGOBERFLÄCHEN UND DIALOGLAYOUT 5

2.3.1 Methoden zur Aufgabenanalyse und -modellierung 6

2.3.2 Vorgänge und Aufgabenabläufe 6

2.3.3 Objektorientierte Analysemethoden 7

2.3.4 Probleme der Methoden und Lösungsansätze 7

2.3.5 Ein Ansatz zur Problembehebung: Task Object Charts 8

2.3.6 UMLi 8

2.4 MODELLIERUNG MIT USE-CASES 9

2.4.1 Use-Cases (UCs) 9

2.4.2 Aktoren und Use-Case-Diagramme 9

2.4.3 Probleme bei der Interaktions-Modellierung mit Use-Cases 10

2.5 MODERNE ANSÄTZE ZUR GUI-ENTWICKLUNG 10

2.5.1 UI-Entwicklungswerkzeuge 11

2.5.2 Modellbasierte UI-Generatoren 11

2.6 MODELLBASIERTE ENTWICKLUNGSUMGEBUNGEN FÜR BENUTZUNGSSCHNITTSTELLEN 11

2.6.1 Ansätze für modellbasierte UI-Generatoren und UI-Entwicklungsumgebungen 12

2.6.2 Vorteile modellbasierter GUI-Generierung 17

2.6.3 Beschränkungen und Nachteile modellbasierter GUI-Generierung 18

2.6.4 Verwendung mehrerer Modelle 18

2.7 XML UND DTD 19

3 AUFGABENSTELLUNG UND ANSATZ 21

3.1 HINTERGRUND 21

3.1.1 ETAS 21

3.1.2 Rahmenprojekt 21

3.2 AUFGABENSTELLUNG 22

3.3 BEGRIFFE 22

3.4 ANSATZ 24

3.5 VORÜBERLEGUNGEN 25
3.5.1 Warum Ablauforientierung? *25*
3.5.2 Wahl des Sprachkonzepts *25*
3.5.3 Benutzerzentrierung *26*
3.5.4 Direkte Generierung von Oberflächen mit Standard-Elementen *27*
3.5.5 Spezialisierungen für technische Systeme *27*
3.5.6 Anforderungen *28*

4 EINE MODELLIERUNGSSPRACHE ZUR INTERAKTIONSBESCHREIBUNG 29
4.1 KONZEPT DER INTERACTION-MODELLING-LANGUAGE (IML) 29
4.1.1 Baumstruktur und Notation der IML *29*
4.1.2 Dreiteilung *30*
4.2 DAS IML-PROJEKTMODELL 30
4.2.1 Sprachdefinitionen *31*
4.2.2 Interessengruppen *31*
4.2.3 Dialog-Layout-Parameter *31*
4.2.4 Systeme *32*
4.3 DAS IML-DATENMODELL 32
4.4 DAS IML-INTERAKTIONSMODELL 32
4.4.1 Use-Cases *32*
4.4.2 Definition von InteractionCase *34*
4.4.3 Das InteractionCase-Konzept *34*
4.4.4 Warum InteractionCases? *35*
4.4.5 Wiederverwendung von InteractionCases *35*
4.4.6 Verzweigungen und explizite Abläufe *36*
4.5 AUFBAU VON INTERACTIONCASES IN DER IML 36
4.5.1 Gruppen und Elementreihenfolge *37*
4.5.2 Interaktionsschritte *38*
4.5.3 Backup-Mechanismus und Undo *44*
4.6 KONSISTENZMECHANISMEN UND ABHÄNGIGKEITEN IN IML 45
4.6.1 Abhängigkeiten auf Systemebene *45*
4.6.2 Abhängigkeiten auf Use-Case-Ebene *45*
4.6.3 Aktoren *45*
4.6.4 Aufruf- und Abhängigkeitsrelationen auf InteractionCase-Ebene *46*
4.6.5 Ergonomie durch Relationsmodellierung? *46*
4.7 FACHKONZEPT-OPERATIONEN 47
4.8 IML VERVOLLSTÄNDIGUNG: IML COMPLETE 48
4.8.1 Feldbezeichnungen *49*
4.8.2 Shortcuts und Hotkeys *49*

5 TRANSFORMATION UND DIALOGMODELLIERUNG 50
5.1 DIE DIALOG LAYER LANGUAGE (DILL) 50
5.1.1 Zwischenformate 50
5.1.2 DiLL: Aufbau der Dialogebenen-Repräsentation 50
5.1.3 Eine Implementierung des DiLL-Objektmodells 54
5.2 FENSTER UND ELEMENTE 55
5.2.1 Fenster 55
5.2.2 Geometrie und Elementeigenschaften von Interaktionselementen 56
5.2.3 Auswahl von Interaktionselementen 63
5.3 FUNKTIONS- UND ABLAUFSTEUERUNG DURCH MENÜS UND BUTTONS 63
5.3.1 Funktionen 63
5.3.2 Buttons und Navigation 66
5.4 LAYOUT-GRUNDSÄTZE 68
5.4.1 Standardisierung 68
5.4.2 Kopplung und Zusammenhalt 69
5.4.3 Lenkung der Aufmerksamkeit 70
5.4.4 Unterscheidbarkeit 70
5.4.5 Groß-/Kleinschreibung 71
5.4.6 Sequenz 71
5.4.7 Simplizität und Klarheit 72
5.4.8 Labels 72
5.5 GRUPPEN 75
5.5.1 Layout-Regeln für Gruppen 75
5.5.2 Bildung und Typisierung von Gruppen 76
5.5.3 Registertypen 76
5.5.4 Layout von Registern 77
5.5.5 Auswahl der passenden Gruppendarstellung 78
5.6 DER IML-TRANSFORMATOR 81
5.6.1 Genereller Ablauf 81
5.6.2 Oberflächengenerierung mit einer Pipelinestruktur 81
5.7 VORGEHENSWEISE ZUR LAYOUTGENERIERUNG 83
5.8 ANWENDUNGSMODELLABHÄNGIGE GRUNDFUNKTIONALITÄT 83
5.8.1 Vordefinierte Anwendungsmodelle und Funktionen 83
5.8.2 Automatisierte Erstellung von Konfigurationsdialogen 85

6 ERGONOMISCHE DIALOGOPTIMIERUNG 86
6.1 LAYOUTGRUNDSÄTZE FÜR DIE OPTIMIERUNG 86
6.1.1 Informationsdichte und anzuzeigende Informationen 87
6.1.2 Proportion 87

6.1.3 Balance ... *88*
6.1.4 Symmetrie ... *88*
6.1.5 Linien, Harmonie und Ordnung ... *88*
6.1.6 Ein Konzept zur Berechnung der optischen Linien ... *89*
6.2 POSITIONIERUNGSVERFAHREN FÜR ELEMENTE ... 90
6.2.1 Sequentielle Positionierung ... *91*
6.2.2 Nicht-sequentielle Positionierung ... *91*
6.2.3 Vertikale Positionierung ... *92*
6.2.4 Das Layoutprinzip von HERBS ... *93*
6.3 GEWICHTUNG VON ELEMENTEN ... 93
6.3.1 Gewichtung generell ... *93*
6.3.2 Gewichte von Spalten ... *93*
6.3.3 Füllgrad von Fenstern ... *94*
6.4 INITIALE POSITIONIERUNG DER ELEMENTE ... 94
6.5 SPALTENBILDUNG ... 95
6.5.1 Unterschiedliche Spaltenhöhen ... *95*
6.5.2 Ausrichtung und Zeilenbildung ... *98*
6.6 IMPLIZITE STYLEGUIDES ... 99
6.7 GRÖßENANPASSUNG ... 100
6.7.1 Registerkarten: Vergrößerung von Elementen bei zuviel Platz ... *100*
6.7.2 Größenanpassung bei Eingabe- und Auswahlfeldern ... *100*
6.8 WEITERE OPTIMIERUNGSMÖGLICHKEITEN ... 103
6.9 METRIKEN ... 104
7 FINALE ARTEFAKTE ... 106
7.1 DAS TEMPLATE-KONZEPT ... 106
7.2 CODE- UND RESSOURCEN-DATEI-GENERIERUNG ... 107
7.2.1 Eine Element-Transformationsbeschreibung ... *107*
7.2.2 Codebasis ... *109*
7.2.3 Manuelle Änderungen und Code-Aktualisierung ... *109*
7.2.4 Windows-Ressource-File-Format : Generierung von RC-Dateien ... *109*
7.3 GENERIERUNG VON HILFE UND DOKUMENTATION ... 110
7.4 TESTFÄLLE ... 112
7.4.1 Übernahme von Testfällen ... *112*
7.4.2 Automatische Generierung von Testfällen aus dem IML-Modell ... *112*
7.4.3 Automatische Generierung von Testprogrammen aus dem IML-Modell ... *113*
7.4.4 Checklisten für Reviews ... *113*
7.5 DATEN-/DOMAINBESCHREIBUNG ... 113
7.5.1 Generierung von Businessobjekten aus DTDs ... *114*

7.5.2 DTD-Generierung 114

8 REALISIERUNG, ANWENDUNG UND EVALUATION DES VERFAHRENS 116

8.1 IMPLEMENTIERUNG EINES TRANSFORMATORS 116

8.2 ANWENDUNG AUF EIN AKTUELLES PROJEKT 117

8.3 AUSBAUSTUFEN VON METHODE UND TRANSFORMATOR 119

8.4 VERGLEICH MIT ANDEREN METHODEN UND AUFWANDSABSCHÄTZUNG 120

8.4.1 Vergleich nach Methoden 120

8.4.2 Vergleich nach Projektphasen 121

8.4.3 Vergleich nach Entwicklungsaufgaben 123

8.5 GRENZEN 125

8.6 EINSATZMÖGLICHKEITEN 126

9 ZUSAMMENFASSUNG UND AUSBLICK 127

1 Einleitung

1.1 Renaissance der Generierung ergonomischer Oberflächen

Immer schneller dreht sich im Softwarebereich das Karussell der neuen Produktversionen. Der immer stärker werdenden Verkürzung von Entwicklungszeiten fallen oft späte Entwicklungsschritte, wie der Test und die Erstellung von Entwicklungsdokumenten, zum Opfer.

Die Software-Ergonomie, schon lange Zeit ein Stiefkind der Softwareentwicklung, leidet zusätzlich unter dem Zeitmangel und der Fülle neuer Technologien und Bedienkonzepte. Kein Wunder also, dass die Entwicklung der GUI (Graphical User Interface) mittlerweile den Löwenanteil von bis zu 60% des Entwicklungsaufwands für sich beansprucht [Kruschinski 1999].

Trotz dieses enorm hohen Aufwands entstehen oft Benutzungsschnittstellen, die nicht einmal den grundlegenden Regeln der Ergonomie entsprechen.

Oft sind software-ergonomische Kenntnisse nicht verfügbar oder werden erst sehr spät angewandt, so dass Architekturentscheidungen und Datenstrukturen schon zu stark sedimentiert sind, um noch konzeptuelle Änderungen durchführen zu können.

Hier sind Anpassungen der Vorgehensweise, d. h. des Prozesses und der Modellierung, notwendig. Um software-ergonomische Überlegungen und Grundlagen schon in den frühen Phasen der Anwendung einzubringen und eine praktische Einsetzbarkeit zu gewährleisten, ist eine softwaregestützte Modellierung der Spezifikation und eine leichte Umsetzung in Entwurf und Implementierung von Nöten.

Diese Notwendigkeit wurde Anfang der 90er-Jahre erkannt – etwa zeitgleich mit der Einführung der objektorientierten Analyse und des objektorientierten Entwurfs, die auf Fachkonzeptebene mittlerweile flächendeckend zu Modellierung eingesetzt werden.

In der Folge wurden Systeme zur softwaregestützten Oberflächengestaltung und bald auch modellbasierte Benutzungsoberflächen-Generatoren entworfen. Triebkräfte der Projekte waren dabei sowohl Software-Engineering- als auch Software-Ergonomie-Aspekte in unterschiedlichen Anteilen.

Gemeinsam ist allen Projekten der Wunsch, Benutzungsoberflächen oder ganze Anwendungsprototypen automatisiert aus deskriptiven Modellen zu generieren.

Die Konzentration auf datenorientierte Modelle oder sehr komplexe, mehrstufige und kaum verbreitete Modellierungsmethoden führte dazu, dass eine praktische Anwendung aus den beschriebenen Zeit- und Aufwandsgründen in der Regel ausblieb.

Mit der zunehmenden Standardisierung und Verbreitung von Modellierungssprachen wie UML und Datenbeschreibungssprachen wie XML besteht heute die Möglichkeit, auf breite Kenntnisse bei Analytikern wie Entwicklern zurückzugreifen, was die Einführung darauf basierender Methoden wesentlich erleichtert.

Zudem legen stärker benutzer- und kundenorientierte Modellierungsmöglichkeiten wie Use-Cases eine Verlagerung weg von datenorientierten hin zu ablauforientierten Konzepten nahe.

Anzustreben ist deshalb die Entwicklung und praktische Erprobung eines software-gestützten Verfahrens, das die oben genannten Konzepte ergänzt und umsetzt.

1.2 Aufbau

Ziel dieser Diplomarbeit ist es deshalb, ein Verfahren zu entwickeln, das es ermöglicht, softwareergonomische Überlegungen und Vorgehensweisen schon in den frühen Phasen der Anwendungsentwicklung mit einzubeziehen und so die Entwicklung ergonomischer Benutzungsoberflächen zu ermöglichen.

Dazu werden in Kapitel 2 zunächst die Grundlagen erarbeitet und aufgezeigt. Um die Weiterentwicklung zu motivieren und die Basis für die Übernahme von Konzepten zu bieten, werden die existierenden Ansätze zur Oberflächengenerierung kritisch betrachtet und mit Bezug zum Anwendungskontext bei der ETAS GmbH näher beleuchtet. Kapitel 3 schließt sich mit einer Vorstellung von Umfeld, Aufgabe und Ansatz an.

Um die aufgezeigten Schwächen zu beheben, wird in Kapitel 4 ein Metamodell zur Beschreibung von Benutzerinteraktion und Dialogverhalten vorgestellt. Es folgt eine Beschreibung der automatisierten Prüfung und Ergänzung eines damit erstellten Modells. Ergebnis ist ein geprüftes und vervollständigtes Interaktionsmodell.

Dieses Modell muss nun in eine direkte Beschreibung von Dialog-Layout (Statik) und Dialogverhalten (Dynamik) transformiert werden. Kapitel 5 beschreibt hierbei das Zielmodell, die notwendigen Transformationen und den zugehörigen Transformator.

Das resultierende Layout der Dialoge kann unter Zuhilfenahme von Modellinformationen und Layoutregeln weiter optimiert werden. Kapitel 6 beschreibt die hierzu verwendeten Regeln und Algorithmen.

Nachfolgend muss das noch weitgehend systemunabhängige optimierte Modell mit Hilfe von Informationen über die Zielumgebung in eine direkt verwendbare oder sogar direkt ausführbare Repräsentation übersetzt werden. Diesen Vorgang zeigt Kapitel 7 anhand der Programmiersprache Visual C++, den zugehörigen Ressourcen-Dateien und weiteren Artefakten wie Testfällen und Hilfen.

Eine Bewertung des Verfahrens (Kapitel 8) und eine Zusammenfassung mit Ausblick (Kapitel 9) schließen die Arbeit ab.

2 Grundlagen

In diesem Kapitel werden die Grundlagen erläutert, die zum Verständnis der entwickelten Konzepte, Ansätze und Verfahren notwendig sind. Zunächst wird dazu ein Überblick über die heute existierenden Normen gegeben. Danach werden die grundlegenden Begriffe erläutert, die für den Aufbau und die Gestaltung von Benutzungsoberflächen, die Beschreibung von Anforderungen und Abläufen sowie für die automatisierte Erzeugung von Benutzungsschnittstellen benötigt werden.

2.1 Normen und Standards

Die Mehrzahl aller Softwareentwicklungsprojekte überschreitet signifikant die geplanten Projektkosten, da Nutzungsprobleme erst in späten Projektphasen erkannt werden und dann mit hohem Aufwand korrigiert werden müssen. [TÜV 2002]

Um Softwareprodukte für die Nutzung an Arbeitsplätzen tauglicher zu machen, wurde eine Reihe internationaler Normen wie ISO 9241 [ISO 1998] und ISO 13407 erarbeitet. Parallel dazu wurden regional und national gesetzliche Vorschriften wie die Bildschirmarbeitsverordnung in Kraft gesetzt, so dass mittlerweile auch rechtsverbindliche Vorschriften existieren.

Einen Überblick über die weltweit bis 1996 verabschiedeten Normen im Computer-Bereich bietet [Smith 1996]. Die für die Software-Ergonomie wichtigsten Normen werden im Folgenden kurz beschrieben.

2.1.1 Bildschirmarbeitsverordnung (BildschArbV)

In der Bildschirmarbeitsverordnung vom 4. Dezember 1996 [BAV 1996] wird im Anhang auch auf „*an Bildschirmarbeitsplätze zu stellende Anforderungen für das Zusammenwirken Mensch – Arbeitsmittel*“ eingegangen (siehe Anhang A: Auszug aus der Bildschirmarbeitsverordnung [BAV 1996]).

Wichtig sind ergonomische Überlegungen somit auch bei der Erstellung jeder Software. Es ist deshalb sinnvoll, diese schon sehr früh im Entwicklungsprozess zu beginnen. Da es sich nicht um Einzelfälle handelt, ist eine Abdeckung mit Werkzeugen sinnvoll und notwendig.

Eine Anpassung an die auszuführende Aufgabe (21.1) kann erreicht werden, wenn die Entwicklung von den Aufgaben und den von diesen implizierten Abläufen ausgeht. Angaben über Dialogabläufe (21.2) können sowohl beschreibend über ein Hilfesystem als auch aktiv von Hilfsmitteln wie Wizards[1] implementiert werden.

Wichtig für die Steuerbarkeit (Beeinflussung der Dialogabläufe, 21.3) ist die Verfügbarkeit einer Abbruchfunktion und evtl. weiterer Navigationselemente, deren Existenz mit Hilfe von Standard-Navigationsmöglichkeiten realisiert werden kann. Eine Fehlerbeschreibung kann sowohl modal[2] in Dialogform als auch nicht-modal in einem Meldungs- oder Statusbereich erfolgen. Wichtig ist dabei, dass diese mit den Abläufen verbunden sind und so vom Entwickler nicht übersehen werden können. Die Navigation zum Ort des Fehlers sollte möglichst verkürzt werden, um den Arbeitsaufwand zur Fehlerbehebung so gering wie möglich zu halten.

Die Anpassung an Kenntnisse und Erfahrung der Benutzer (21.4) kann sowohl passiv über Short-Cuts[3] als auch aktiv über Optionen wie automatische Vervollständigung geschehen. Aktive Einstellmöglichkeiten sind generativ kaum direkt abzudecken.

---

[1] Ein Wizard (engl. Zauberer) unterstützt den Benutzer bei der Bearbeitung oder Eingabe von Optionen. Dabei werden die notwendigen Abläufe und Eingaben schrittweise durchlaufen und zu jedem Schritt Hinweise und Informationen gegeben. Ein bekanntes Beispiel sind Installations-Wizards, die schrittweise durch eine Installation führen.

[2] Ein modaler Dialog muss erst abgeschlossen werden, bevor mit anderen Fenstern im Hintergrund wieder interagiert werden kann. Modale Dialoge wie Fehlermeldungen und Abfragen („Wollen Sie speichern?“) erzwingen eine Antwort vom Benutzer.

[3] Short-Cuts (engl. Abkürzungen) sind Tastenkombinationen, die eine Funktion über die Tastatur schneller zugänglich machen als mehrfaches Klicken in einem Menü bei Mausbedienung. Siehe auch Kapitel 4.8.2.

2.1.2 DIN EN ISO 9241

Die DIN EN ISO 9241 [ISO 1998] wurde jahrelang erweitert und umfasst mittlerweile 17 Teile. Relevant aus Sicht der Software-Ergonomie im engeren Sinne, sind vor allem die Teile 8 und 10 bis 17. Da die ISO 9241 gleichzeitig eine europäische und deutsche Norm ist, ist sie auch in Deutschland richtungsweisend (siehe Anhang A: Überblick ISO 9241: Für diese Arbeit wichtige Teile).

Die Grundsätze der Dialoggestaltung bilden einen allgemeinen Handlungsrahmen für die Erstellung von Benutzungsschnittstellen in Dialogform und stellen wegen ihrer weitgehend allgemeinen Formulierung eine gute Grundlage für weitere Überlegungen dar. Folgende Grundsätze werden durch Teil 10 der ISO-Norm vorgegeben:

- **Aufgabenangemessenheit:** Die Aufgabe muss sowohl effektiv als auch effizient mit Hilfe der Software zu erledigen sein.
- **Selbstbeschreibungsfähigkeit:** Der Benutzer kann sich allein aus der Informationsgestaltung am Bildschirm in der Anwendung zurechtfinden, diese verstehen und den System-Zustand erkennen.
- **Steuerbarkeit:** Der Start, die Ablaufsteuerung und die Ablaufgeschwindigkeit von Dialogabläufen sind durch den Benutzer beeinflussbar.
- **Erwartungskonformität:** Diese wird erreicht, wenn der Dialog konsistent ist und zudem den Erwartungen des Benutzers entspricht, die aus anderen Kenntnissen und Erfahrungen resultieren.
- **Fehlertoleranz:** Das Arbeitsergebnis kann trotz erkennbarer Fehler des Benutzers mit möglichst minimalem Korrekturaufwand erreicht werden.
- **Individualisierbarkeit:** Das Dialogsystem lässt sich an die Aufgabe, die Fähigkeiten und die Vorlieben des Benutzers anpassen.
- **Lernförderlichkeit:** Der Benutzer wird vom Dialogsystem beim Erlernen unterstützt und angeleitet.

Mit der Kenntnis dieser Grundsätze können Anwendungen meist schon entscheidend verbessert werden. Viele Überlegungen zur Vorgehensweise des Transformators und bei der Modellkonzeption in dieser Arbeit beruhen auf den oben genannten sieben Grundsätzen aus ISO 9241.

2.1.3 ISO 13407 und ISO TR 18529

Im Gegensatz zu den bisher beschriebenen Normen und Vorschriften betrachtet die ISO-Norm 13407 [ISO 1999] den Entwicklungsprozess („Human-centered design processes for interactive systems“). Es werden also keine Elemente oder Techniken beschrieben, sondern ein Prozess, der den Menschen, das heißt den Benutzer, in den Mittelpunkt der Entwicklung stellt. [IBM 2002a]

So legt der Standard Aktivitäten fest, die notwendig sind, um „Human-Centered-Design“ sowohl für Software als auch für Hardware zu ermöglichen. Es wird ein Framework definiert, das ein Berichtswesen zur Sicherstellung der notwendigen Aktivitäten enthält und somit das Management des Entwurfsprozesses unterstützt. Dabei werden Punkte wie Prinzipien, Planung und Aktivitäten des „Human-Centered Design Process“ abgedeckt (siehe Anhang A: Vorgehensweise im „Human-Centered Design Process“).

Eine strukturierte und formalisierte Definition des „Human-Centered Design Process“ wurde vom EC INUSE Projekt erstellt und in verbesserter Form unter der Bezeichnung „Usability Maturity Model“ als ISO TR 18529 [ISO 2000] veröffentlicht. Zwar kann das Modell vor allem ermitteln, wie gut die definierten Bereiche abgedeckt werden, es kann aber ebenso als Beschreibung der Vorgehensweise genutzt werden. [TRUMP 2002]

Von den notwendigen strategischen Überlegungen über den Entwurf bis zum Betrieb des Systems werden alle Bereiche in insgesamt sieben Kategorien abgedeckt, die wiederum einer Check-Liste ver-

gleichbare Einträge besitzen. Auf diese Weise können auch Modelle auf ihre Vollständigkeit bezüglich „Human-Centered Design Process“ überprüft werden.

2.2 Usability und Utility

Um die Qualität von Dialogen und Dialogsystemen zu beschreiben wurden im letzten Jahrzehnt unterschiedliche Begriffe eingeführt, von denen viele allerdings synonym verwendet werden [Burmester 2002]:

- Benutzerfreundlichkeit
- Benutzbarkeit
- Gebrauchstauglichkeit
- Bedienungsfreundlichkeit
- Easy to use / Easy to learn
- Usability

Die Usability (Benutzbarkeit) gehört zu den am weitesten verbreiteten und wissenschaftlicheren Begriffen. Sie wird deshalb im Folgenden genauer definiert und als Grundbegriff verwendet.

Usability ist nach [ISO 1998] definiert als

The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use.

Diese Definition basiert auf den Begriffen Effectiveness, Efficiency, Satisfaction, die in Anlehnung an ISO 9241 Teil 11 folgendermaßen definiert sind [USINACTS 2002]:

Effectiveness is the accuracy and completeness with which users achieve specified goals. Efficiency are the resources expended in relation to the accuracy and completeness with which users achieve goals. Satisfaction is the comfort and acceptability of use.

Das ETSI[4], dessen Standards allerdings hauptsächlich an der Telekommunikation orientiert sind, unterscheidet zusätzlich den Begriff “Utility”, der auch wirtschaftliche Aspekte ins Spiel bringt:

Usability is considered as a pure ergonomic concept, independent of the costs of providing the system. Usability, together with the balance between benefit to the user and the financial costs form the concept of utility. An ergonomical highly usable system may have low utility for a particular user who considers the cost to be too high in relation to his or her need for the system. [USINACTS 2002]

Die Bewertung der Usability gestaltet sich sehr schwierig, da zum Teil subjektive Kriterien erfüllt werden müssen und entsprechende Anforderungen nicht wie funktionale Anforderungen exakt werden können. Zudem werden in [USINACTS 2002] zwei unterschiedliche, als komplementär zu betrachtende Arten von Usability-Bewertungen erwähnt, die zudem einen sehr unterschiedlichen Fokus haben: Verhaltens- und Einstellungsbewertungen (siehe Anhang A: Usability-Bewertungen).

2.3 Dialogoberflächen und Dialoglayout

Für die Aufgabenanalyse und -modellierung existieren unterschiedliche Methoden, die als gemeinsames Merkmal Metriken zur Ermittlung von kognitivem Aufwand, Lernaufwand, Ausführungszeit und weiteren benutzerbezogenen Größen verwenden. Zudem existieren objektorientierte Methoden, die mit zunehmendem Einfluss der objektorientierten Programmierung auch im Bereich der Analyse und der Modellierung eingesetzt werden.

[4] ETSI (European Telecommunications Standards Institute); mit 874 Mitglieder aus 54 Ländern, auch außerhalb Europas und führt industrieorientierte Standardisierungen im Telekommunikationssektor durch.

Um Aussagen über zu entwickelnde Anwendungen machen zu können und Spezifikationen zu erstellen ist eine Aufgabenanalyse und -modellierung unerlässlich. Noch größere Bedeutung haben Modelle bei der UI-Entwickler-Unterstützung und der Anwendungsgenerierung, wo sie die Grundlage für weitere Entwicklungsschritte bieten.

2.3.1 Methoden zur Aufgabenanalyse und -modellierung

Für die Aufgabenanalyse und -modellierung existieren in der Software-Ergonomie unterschiedliche Methoden, von denen viele auf psychologische Überlegungen und Modellierungen zurückgehen. Das Ziel dieser Methoden ist eine Modellierung, die Aussagen über Kennzeichen der Abläufe und zu entwickelnden Systeme macht. So sollen beispielsweise Einarbeitungs- und Lernaufwand sowie Bearbeitungszeiten und kognitive Belastung für den zukünftigen Benutzers abgeleitet werden können. Im Folgenden werden die wichtigsten Methoden [Herczeg 1994] näher erläutert.

ETIT-Analyse: Hier wird davon ausgegangen, dass ein Benutzer interne Aufgaben (intern tasks) auf externe Aufgaben (extern tasks) abbilden muss. Mit Hilfe einer Matrix wird dabei der Aufwand und der mögliche Wissenstransfer für ähnliche Aufgaben abgeschätzt.

GOMS-Modellierung: GOMS (Goals, Operators, Methods, Selection Rules) übernimmt aus der allgemeinen Theorie des Problemlösens die Begriffe Ziel, Unterziel, Operator, Methode und Auswahlregeln. Ziele werden in Unterziele zerlegt. Aktivitäten entstehen durch Anwendung von Operatoren, die durch Kontrollstrukturen zu Methoden, d. h. höherwertigen Operatoren, werden. Auswahlregeln bestimmen die zur Erreichung eines bestimmten Ziels eingesetzten Methoden.

CLG-Modellierung: Command-Language-Grammers eignen sich für kommandobasierte Systeme. Es werden dabei die Aufgabenebene (externe Aufgaben), die semantische Ebene (interne Aufgaben), die syntaktische Ebene (Konzepte und Kommandos) und die Interaktionsebene (Aktionen und Darstellung) unterschieden.

TAG-Modellierung: Task-Action-Grammars eignen sich für alle Arten von Benutzungsschnittstellen. TAG stellen eine Meta-Sprache zur Beschreibung von Aufgabenbeschreibungssprachen dar. Ein TAG-Modell besteht aus *Aufgabenelementen* (Features), einer Liste der *Aufgaben* und zugehörigen Elemente und *Ersetzungsregeln* (Grammatik) zur Erzeugung von Aktivitäten.

CCT-Modellierung: Mit Hilfe der Cognitive Complexity Theory kann die kognitive Komplexität eines Systems bestimmt werden, indem eine Beurteilung des benötigten Wissens erfolgt. CCT wurde aus GOMS durch Abbildung der prozeduralen Beschreibung in Bedingungs- und Aktionsregeln entwickelt.

Allen Methoden gemeinsam ist die Ausrichtung auf Metriken zur Ermittlung von kognitivem Aufwand, Lernaufwand, Ausführungszeit und anderen benutzerbezogenen Größen. Für eine Unterstützung der UI-Generierung eignet sich deshalb keine Methode ohne größere Erweiterungen.

2.3.2 Vorgänge und Aufgabenabläufe

Zur Modellierung von Vorgängen und Aufgabenabläufen existiert eine Reihe von unterschiedlichen Ansätzen. Einen Vergleich der Möglichkeiten bietet [Ziegler 1996] für folgende Ansätze:

- Data-Flow Diagrams (DFD, [DeMarco 1979])
- RFA [Oberquelle 1987]
- Petrinetze (vgl. [Reisig 1986])
- State-Charts [Harel 1987]
- Vorgangskettendiagramm (VKD, [Scheer 1991])
- KSA [Krallmann 1990]
- ZÜD (vgl. [Larson 1992])
- Event Schemata [Martin 1992]

Zumeist ist die Ablaufbeschreibung das einzige Ziel dieser Ansätze, so dass andere Modelle zur Verfeinerung oder Ergänzung eingesetzt werden müssen. Aus diesem Grund ergeben sich vollständige Modelle zumeist erst aus der Ergänzung mit anderen Methoden.

2.3.3 Objektorientierte Analysemethoden

Seit Anfang der neunziger Jahre wurden vermehrt objektorientierte Modellierungstechniken entworfen und auch eingesetzt, um dem Trend zu objektorientierten Vorgehensweisen in der Software Entwicklung Rechnung zu tragen, speziell im Bereich der objektorientierten Programmierung (OOP).

Mittlerweile werden sowohl Methoden zur objektorientierten Analyse (object oriented analysis, OOA) als auch zum objektorientierten Entwurf (object oriented design, OOD) in vielen Softwareentwicklungsprojekten eingesetzt, mit dem Ziel, einen Methoden- bzw. Paradigmenbruch während der Entwicklung zu vermeiden.

Einen Vergleich der Einsetzbarkeit für die Entwicklung interaktiver Systeme bietet [Ziegler 1996] für folgende Methoden:

- OMT [Rumbaugh 1991]
- OBA [Rubin 1992]
- OOA [Coad 1991]
- OOA&D [Martin 1992]
- OOSE [Jacobson 1992]

Zusammen mit dem Ansatz von Booch wurden die Methoden OMT und OOSE zur Unified Modelling Language (UML, [Booch 1999]) vereinigt. Die UML hat sich mittlerweile im Bereich von objektorientierter Analyse und objektorientiertem Design als Quasi-Standard etabliert.

Eine Alternative zur UML stellt die OPEN Modelling Language (OML, [Firesmith 1998]) dar. Ein Vergleich der beiden Sprachen aus Sicht der Erfinder von OML ist in [Hend.-S. 1999] nachzulesen. Hier wird der Ansatz von UML als nur hybrid-objektorientiert und C++-orientiert bezeichnet. OPEN bietet einen kompletten Prozess und die Modellierungssprache OML.

Wegen der einfachen und kaum durch Formalismen eingeschränkten Modellierung von Use-Cases und der weiten Verbreitung von Expertenwissen und Werkzeugen zur UML eignet sich diese Sprache jedoch besser als die OML für die in dieser Arbeit genutzte ablauforientierte Modellierung von Interaktionen.

2.3.4 Probleme der Methoden und Lösungsansätze

Es gibt noch weitere Gründe dafür, keine streng objektorientierte Modellierung einzusetzen. Während Objekte als Entitäten existieren und miteinander in nicht zwangsläufig sequentieller Weise kommunizieren, ähneln menschliche Arbeitsabläufe mehr einer prozeduralen Vorgehensweise.

Eine rein objektorientierte Vorgehensweise bringt speziell in der Analyse/Spezifikation Probleme mit sich: So ist bei einer objektorientierten Modellierung keine Nebenläufigkeit möglich. Außer Klassen existieren keine Strukturen, um beispielsweise gröbere Strukturen abzugrenzen. Zudem sehen OOA-Modelle in der Praxis häufig wie frühere ER-Diagramme aus, die eine rein datenzentrierte Sicht enthalten.

Komplexe Konzepte wie Konstruktoren, Destruktoren und Sichtbarkeitsangaben (public, private) sollten nicht in Dokumenten verwendet werden, die für den Anwender verständlich sein sollen. Zudem werden in objektorientierten Spezifikationstechniken oft zwangsläufig Strukturen festgelegt, die nicht notwendig sind und sich für die weitere Entwicklung als hinderlich erweisen. [Broy 2002]

2.3.5 Ein Ansatz zur Problembehebung: Task Object Charts

Ansätze zur Umgehung dieser Probleme finden sich in einer Ablauforientierung durch die Modellierung der Objekt-Zustände mittels Graphen. Eine mögliche Modellierungsform sind Task Object Charts (TOC, [Ziegler 1996]), deren Aufbau demjenigen der in [DeMarco 1979] beschriebenen Datenflussdiagrammen ähnelt.

Allerdings müssen hierbei Zustandsklassen und -subklassen gebildet werden, was nur bei wenigen Anwendungstypen einfach durchführbar ist. Die Ereignis- und Objektflüsse in TOCs sind zudem so komplex, dass sie zwar für den Entwickler eine sinnvolle Ergänzung darstellen, jedoch in der geforderten Kombination mit objektorientierter Modellierung für den Kunden bzw. partizipierenden Anwender keine ausreichende Orientierung bieten.

Kann ein TOC aus existierender Information als Darstellung gewonnen werden, kann es allerdings als sinnvolle, ergänzende Visualisierungsmöglichkeit genutzt werden.

Während rein prozedurale Techniken sich nur am Ablauf orientieren und die Modellierung von Entitäten vernachlässigen, konzentrieren sich rein objektorientierte Techniken zumeist auf die Objekte und benötigen so eine zusätzliche Modellierung des Ablaufs. Wünschenswert ist also eine Darstellungsform, die sowohl Abläufe als auch die bearbeiteten Entitäten modellieren kann.

2.3.6 UMLi

Die UML verfügt auch heute noch nicht über die Möglichkeit, dialogbasierte Interaktionen effizient zu modellieren.

Aus diesem Grund wurde die in [Pinheiro 2000b] vorgestellte UMLi entworfen. UMLi ist eine Erweiterung der UML, die folgende zusätzliche Möglichkeiten zur Modellierung von Benutzungsschnittstellen bereitstellt:

- **User Interface Diagramm** (UID): Einen neuen UML-Diagramm-Typ zur Modellierung der Präsentationsschicht
- Jedes UID besteht aus einem "**FreeContainer**", d. h. einem "top-level"-Fenster
- „FreeContainer" enthalten ihrerseits wieder folgende Elemente:
    - **Displayer** zeigen dem Benutzer Informationen an.
    - **Inputter** empfangen Informationen vom Benutzer.
    - **Editoren** vereinen Displayer und Inputter in sich.
    - "**ActionInvokers**", nehmen Informationen in Form von Ereignissen vom Benutzer entgegen.
    - **Container** können wiederum Container, Editoren, Displayer, Inputter und „ActionInvoker" enthalten.
- Die UID-Konstruktoren sind "**InteractionClasses**", bei denen es sich wiederum um spezialisierte UML-Klassen handelt. Die von diesen Klassen abgeleiteten Interaktionsobjekte sind die Widgets[5] der grafischen Benutzungsschnittstelle.

Grund für die Entwicklung von UMLi war nach [Pinheiro 2000b] die Tatsache, dass mit modellbasierten UIDE zumeist nur Benutzungsschnittstellen und keine kompletten Anwendungen entwickelt werden können.

UIDE konnten sich bisher kaum durchsetzen, da die Modellierung losgelöst von der restlichen Anwendung erfolgt und somit keinen Bezug zur Modellierung des restlichen Systems hat. Zudem wurden oft Modellierungssprachen eingesetzt, die den Entwicklern unbekannt waren. UMLi soll dieses Problem lösen.

[5] Widget setzt sich aus den Begriffen Window und Gadget zusammen und steht für Dialogkomponenten, welche die Eigenschaften von Fenstern (sichtbar, Größe, Position) und von Gadgets (Interaktion, Event-Handling) haben.

UMLi wurde am selben Institut wie das modellbasierte UIDE Teallach entwickelt. Da die UI-Modellierung bisher von UML nicht abgedeckt wurde, setzt [Pinheiro 2000c] in einem Ansatz für die Integration von Benutzungsschnittstellen-Entwicklung und Anwendungsentwicklung die beiden Methoden kombiniert ein.

2.4 Modellierung mit Use-Cases

2.4.1 Use-Cases (UCs)

Lange Zeit wurden Anforderungen als natürlichsprachliche Texte ohne eine Modellgrundlage beschrieben. Seit der Einführung der Use-Cases kann eine Diagramm-Strukturierung auf Aufgabeneben vorgenommen werden.

Use-Cases sind ein Bestandteil der schon erwähnten marktdominierenden Unified Modelling Language. Das Use-Case selbst wird von [Booch 1999] folgendermaßen definiert: Ein Use-Case (UC) ist die Beschreibung einer Menge von Aktions-Sequenzen inklusive Varianten, die ein System ausführt, um ein beobachtbares Ergebnis zu erzielen, das wiederum einen Wert für den Benutzer darstellt.

Use-Cases stellen eine formalisierte Methode zur Spezifikation von im Kontext der zu modellierenden Software auftretenden Anwendungsfällen dar. Sie sind Bestandteil der UML.

Die Anwendungsfälle sind typische Abläufe, die für einen bestimmten Nutzer oder eine bestimmte Nutzergruppe existieren. Charakteristisch ist dabei die sequentielle Ablaufbeschreibung, die den normalen Verlauf schrittweise wiedergibt.

Zudem existieren alternative Abläufe, die Abweichungen vom normalen Ablauf ermöglichen, beispielsweise den vorzeitigen Abbruch eines Dialogs durch den Anwender. Der Name eines Use-Case wird mit einem Verb konstruiert, um den Ablaufcharakter hervorzuheben, z. B. „Passwort eingeben“ oder „Dokument speichern“.

2.4.2 Aktoren und Use-Case-Diagramme

Aktoren stellen Benutzer-Rollen in der Anwendung symbolisch dar. Jeder Aktor ist mit einem oder mehreren Use-Cases assoziiert, d. h. er benutzt diese. Aktoren können sowohl reale Benutzer als auch andere Systeme sein, die mit den Use-Cases interagieren.

Use-Cases werden in einem Use-Case-Diagramm dargestellt, das zudem einen oder mehrere Aktoren enthält, die wiederum die Use-Cases „benutzen“. Dieses Verhältnis wird über eine Assoziationskante dargestellt.

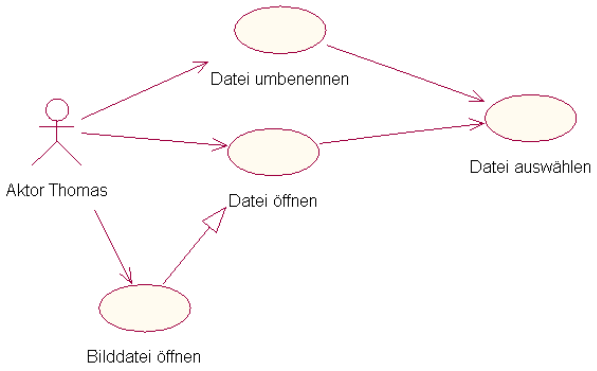

Abbildung 1: Beispiel für ein Use-Case Diagramm in Rational Rose [Rational 1999]

In Abbildung 1 ist ein Beispiel für ein Use-Case-Diagramm gezeigt. Hierbei nutzt der Aktor drei verschiedene Use-Cases. Die Use-Cases „Datei umbenennen“ und „Datei öffnen“ enthalten das Use-Case „Datei auswählen“, da dieser Anwendungsfall für beide identisch ist. „Bilddatei öffnen“ erbt von „Datei öffnen“, da es sich um einen Spezialfall handelt. „Datei öffnen“ generalisiert also „Bilddatei öffnen“. Durch die Generalisierungsrelation enthält selbstverständlich auch „Bilddatei öffnen“ den Use-Case „Datei auswählen“.

Zudem können zwischen Use-Cases Relationen modelliert werden:

- Extends A –extend→B:
  Use-Case A erweitert Use-Case B; B hat damit die Abläufe von A, fügt diesen jedoch weitere hinzu. B baut also auf A auf und ist komplexer.
  Beispiel: A=Lieferung versenden; B=Teillieferung versenden (zusätzliche Selektion möglich)
- Includes A –include→B:
  Use-Case A enthält Use-Case B; B ist hier eine Art Prozedur, die von A verwendet wird. A kann also beispielsweise B durchlaufen, eigene Schritte ausführen und dann weitere Use-Cases C und D aufrufen, die ebenfalls mit „include“-Kante verbunden sind.

Ein Diagramm kann zudem in Systeme unterteilt werden, zu denen jeweils bestimmte Use-Cases gehören, beispielsweise in einen Client und einen Server. Zusätzlich wird ein Generalisierungs-Mechanismus unterstützt, der es ermöglicht, Spezialfälle abzuleiten, z. B. „Kaufen“ und daraus „Auto leasen“, „Lebensmittel kaufen“.

2.4.3 Probleme bei der Interaktions-Modellierung mit Use-Cases

Die Grundidee der Use-Cases beschäftigt sich mit der Modellierung von Interaktionen nur auf Aufgabenebene: Im Use-Case-Diagramm werden die Zusammenhänge zwischen Anwendungsfällen und zwischen Aktoren und Anwendungsfällen beschrieben.

Die jedem einzelnen Use-Case zugeordnete Beschreibung enthält reinen, nicht formal eingeschränkten natürlichsprachlichen Text, wodurch sowohl Probleme durch Ungenauigkeit, Unvollständigkeit und mangelnde Überprüfbarkeit entstehen. Zudem wird eine weitere automatisierte Verwendung im Entwicklungsprozess nahezu unmöglich gemacht.

Die Use-Case-Modellierung eignet sich hervorragend zu einer groben Strukturierung der Anforderungen und zu einer schnellen, formal kaum eingeschränkten Beschreibung. Weitere Schritte zu einer Verfeinerung oder Übernahme in folgende Entwicklungsphasen sind jedoch nicht vorgesehen.

2.5 Moderne Ansätze zur GUI-Entwicklung

Sind Anforderungen und Abläufe modelliert, muss neben dem rein an funktionalen Anforderungen und technischen Gegebenheiten ausgerichteten System auch die Benutzungsoberfläche erstellt werden.

Obwohl das Gros der GUI-Entwicklungen noch durch Programmierung geschieht, existieren heute wesentlich höher entwickelte Methoden, die es ermöglichen, GUI auf höherer Ebene zu spezifizieren. Für die softwaregestützte Erstellung von grafischen Oberflächen gibt es dabei verschiedene Ansätze.

Grundlegende Methoden wie die Programmierung eines API (Application Programming Interface) oder eines davon abstrahierenden User-Interface-Toolkits (UI-Toolkit) sollen hier nicht näher betrachtet werden.

Generell lassen sich bei den höher entwickelten Ansätzen zwei Gruppen unterscheiden:

- UI-Entwicklungswerkzeuge, User Interface Development Environments (UIDE)
- Modellbasierte UI-Generatoren, Model Based Interface Development Environments (MB-IDE)

2.5.1 UI-Entwicklungswerkzeuge

UI-Entwicklungswerkzeuge vereinfachen die Entwicklung von Oberflächen bzw. Dialogen. Während einfache Ansätze nur eine textuelle (z. B. Kommandoergänzung) oder grafische Unterstützung (z. B. grafische Modellierung wie in Ressourcen-Editoren) bieten, unterstützen höher entwickelte Werkzeuge den Anwender mit Regelwerken und Informationen oder verfügen über Kontrollinstrumente.

Diese Kontrollinstrumente unterbreiten dann anhand bestimmter Kriterien (z. B. Dialog-Metriken) Verbesserungsvorschläge für den aktuellen Dialog oder die gesamte Anwendung. Wird nur die Präsentationsschicht (Layout) unterstützt, spricht man von UI-Buildern.

Oft werden diese Werkzeuge allerdings durch ein User Interface Management System (UIMS) ergänzt, das für die Dialogabläufe zuständig ist. Dabei sorgt vom Entwickler geschriebener Code in Form eines Scripts oder in einer Hochsprache für ein animierbares Dialogmodell. Zumeist wird dieser Code interpretiert und auf diesem Weg auf die API eines GUI-Systems bzw. auf ein UI-Toolkit zugegriffen.

2.5.2 Modellbasierte UI-Generatoren

Modellbasierte Generatoren generieren dagegen aus einem Modell direkt die Oberfläche. Ziel ist es dabei, die grafische Benutzungsschnittstelle nicht mehr manuell zu erstellen, sondern sie direkt zu generieren. Der Oberflächen-Entwickler ist dabei nur noch für die Erstellung des Modells und die Kontrolle/Nachbearbeitung zuständig.

Ausgangsbasis für die Generierung können dabei sehr unterschiedliche Modelle sein. Möglich ist beispielsweise die Verwendung des OOA-Modells einer betrieblichen Anwendung [Hofmann 1998] oder eines ER-Modells (Entity Relationship) in Verbindung mit Dialognetzen[6]. [Ziegler 1996]

Der Hauptunterschied zwischen den beiden Gruppen von Ansätzen besteht im Automatisierungsgrad der Oberflächenerstellung und in der Modellbasiertheit.

Während Entwicklungswerkzeuge auf der konkreten Oberfläche arbeiten bzw. zumindest auf deren endgültiger Repräsentation, nutzen modellbasierte Generatoren Daten- oder Anwendungsmodelle, die eine nicht deckungsgleiche Informationsmenge enthalten.

In der Praxis bedeutet dies, dass sowohl Informationen im Modell vorhanden sein können, die für die Benutzungsschnittstelle irrelevant sind, als auch notwendige Informationen fehlen können oder erst mit Hilfe von Regeln generiert werden müssen.

2.6 Modellbasierte Entwicklungsumgebungen für Benutzungsschnittstellen

Modellbasierte Entwicklungsumgebungen werden oft als MB-IDE (model based interface development environment) oder MB-UIDE (model based user interface development environment) bezeichnet.

Bei MB-IDE werden softwaregestützt ein oder mehrere deskriptive Modelle erstellt, die dann mit Hilfe eines Generators oder Transformators in eine Beschreibung von Interface, Abläufen oder Quellcode umgesetzt werden.

Bei dieser Vorgehensweise handelt es sich um eine vergleichsweise neue Technologie, so dass zumeist nur ein Teil des Aufgabenbereichs in konkreten Systemen verwirklicht wird und das Augenmerk eher auf den Modellen oder dem Generierschritt (Software Engineering) als auf der entstehenden Benutzungsschnittstelle (Software-Ergonomie) liegt.

[6] Dialognetze [Janssen 1993] sind weiterentwickelte Petrinetze

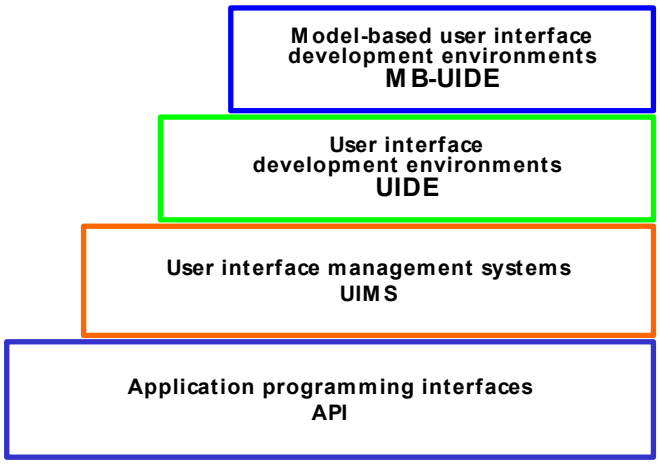

Abbildung 2: Entwicklungsstufen der Erzeugung grafischer Benutzungsschnittstellen

2.6.1 Ansätze für modellbasierte UI-Generatoren und UI-Entwicklungsumgebungen

Es gibt drei unterschiedliche Ansätze, um aus deklarativen Modellen ein User Interface zu generieren (vgl. [Pinheiro 2000a]). Die nachfolgend als Beispiele genannten Ansätze werden im folgenden Kapitel genauer vorgestellt.

- **Interpreter** wie ITS: Diese interpretieren das Modell direkt

  \+ Direkte Einstellung des Laufzeitsystems passend zum Modell

  – Performance-Probleme durch High-Level-Interpreter

- **UIMS-Generatoren** wie HUMANOID, TADEUS und FUSE: Das Modell wird in eine andere, von einem UIMS interpretierbare Darstellung umgesetzt und dann interpretiert.

  \+ UIMS ist flexibel, Interface-Einstellungen können zur Laufzeit geändert werden

  – saubere und effiziente Umsetzung des Modells ist eher fraglich

- **Quellcode-Generatoren** wie JANUS und Mastermind: Quellcode und spezifische Ergänzungen werden direkt generiert

  \+ Volle Funktionalität der Programmiersprache kann genutzt werden

  – Unterstützung der Benutzungsschnittstellen-Funktionalität ist bei der Verwendung einer Programmiersprache geringer als bei einem UIMS

Im Folgenden werden die heute existierenden Generator-Ansätze und deren Besonderheiten vorgestellt.

2.6.1.1 UIDE

Das UIDE (User Interface Design Environment, [Foley 1989]) erlaubt Entwicklern, die Benutzungsoberfläche einer Anwendung zu erstellen, zu verändern und Teile davon zu generieren. Zielanwendungen von UIDE sind sowohl grafische Editoren als auch formularbasierte Anwendungen. Für diese Anwendungsklassen deckt UIDE den gesamten Entwicklungsprozess ab. Der Grad der Automatisierung ist dabei abhängig von der Anwendungsklasse.

Auch bei UIDE wird ein Anwendungsmodell erstellt. Es besteht aus drei Abstraktionsebenen:

1. **Anwendungsmodell** (Application Model)

   Zunächst werden Anwendungsobjekte basierend auf sieben Metaklassen (frames) und Vererbungshierarchien beschrieben. Diese Objekte können mit Anwendungsaktionen manipuliert werden. Die Spezifikation der Anwendungsaktionen (Aufgaben des Benutzers) spielt in dieser Ab-

straktionsebene eine zentrale Rolle. Sie werden mit Hilfe von Parametern, Parameter-Restriktionen und Vor- und Nachbedingungen (pre-conditions, post-conditions) beschrieben.

2. **Benutzungsoberfläche** (Interface Model)

   In dieser Abstraktionsebene werden Oberflächenobjekte und die zugehörigen Oberflächenaktionen anwendungsunabhängig beschrieben. Durch Vererbung können spezialisierte Oberflächenobjekte abgeleitet werden. Auf dieser Ebene werden die Komponenten der Benutzungsoberfläche und die Verbindung zum Anwendungsmodell und zu den anwendungsmodellunabhängigen Aufgaben der Software beschrieben. Die Aktionen des Anwendungs- und des Benutzermodells müssen dabei allerdings manuell verbunden werden.

3. Auf der untersten Ebene werden **Interaktionstechniken** spezifiziert. Sind keine speziellen Interaktionstechniken vorgesehen, kann die Modellierung dieser Ebene entfallen.

Die entwickelten Modelle sind die Grundlage für verschiedene Systeme zur Unterstützung des Entwicklers während des Entwurfs und während der Ausführung. Neben Vorgaben zur Anwendung und deren Benutzungsoberfläche wird auch ein Hilfesystem erzeugt. [Sukaviriya 1990]

Die Trennung von Anwendungsmodell und Benutzungsoberfläche in Modell und Anwendung ist vorteilhaft bezüglich der Konsistenz. Hilfe und Entwicklerunterstützung aus dem Modell zu erzeugen vereinfacht zudem die Entwicklung. Eine direkte Generierung der Benutzungsoberfläche ist allerdings nicht möglich. UIDE ist also kein rein modellbasierter Generator, sondern eine Entwicklungsumgebung mit generativen Ansätzen.

2.6.1.2 ITS

Für die Weltausstellung 1992 in Sevilla wurde das ITS (Interactive Transaction System) von IBM entwickelt [Wiecha 1990]. Ziel war dabei die Unterstützung der Besucher-Informationssystem-Entwicklung.

Trotz dieser Ausrichtung wurde das Werkzeug so offen entworfen, dass es auch bei der Entwicklung von datenbankorientierten Anwendungen, speziell betrieblichen und Kiosk-Anwendungssystemen, eingesetzt werden kann.

Um die Anwendung zu definieren wird ein sogenannter „**Datenpool**“ (data pool) und eine Menge von **Aktionen** (actions) modelliert. Diese enthalten den Kontrollfluss, den Datenaustausch zwischen „Datenpool“ und Oberfläche und die Operationen. Die Struktur von Anwendungsobjekten und Objektsammlungen wird mit Hilfe einer speziellen Definitionssprache beschrieben.

Die weitere Vorgehensweise wird in Anhang A („Vorgehensweise bei ITS“) beschrieben.

Ein UIMS (dialog manager) verbindet zur Laufzeit die Benutzungsoberfläche mit den zuvor implementierten fachlichen Aktionen. Am Ende erfolgt eine Anpassung durch den Oberflächenprogrammierer.

Praktisch genutzt wurde ITS auf der EXPO 1992. Eine entsprechende Video-Präsentation [Kelley 1993] wurde auf dem „Human Factors and Ergonomics Society Annual Meeting“ 1993 gezeigt.

Kritisiert wird an ITS vor allem die schwierige und umfangreiche formale Notation. Über den tatsächlichen Anwendungsgrad auf der EXPO 1992 und den Einsatz in weiteren Softwareentwicklungen liegen keine Quellen vor.

2.6.1.3 DON

DON ist ein Werkzeug, das den Entwickler direkt beim Entwurf von formularbasierten Benutzungsoberflächen unterstützt [Kim 1993]. Es basiert auf dem oben beschriebenen wissensbasierten UIDE.

DON unterstützt den Entwurf von formularbasierten Anwendungen mit dem „Organization Manager“ und dem „Presentation Manager“, die allgemeine und spezielle Gestaltungsregeln enthalten. Die Verbindung dieser Regeln mit dem Anwendungs- und Interface-Modell von UIDE ist die Hauptaufgabe dieser beiden Module.

Der „**Organization Manager**“ besteht aus Organisations- und den Selektionsregeln. Mit Hilfe der Organisationsregeln werden die Operationen und die Attribute des Modells in Menüs und Dialoge umgewandelt. Mit Hilfe der Selektionsregeln wählt DON basierend auf den Beschreibungen der Attribute die passenden Interaktionselemente aus, die als Repräsentation der Attribute dienen sollen.

Der „**Presentation Manager**“ erstellt Teile der Präsentationsschicht: Der zugehörige Assistent erledigt in Zusammenarbeit mit dem Entwickler die Zusammenstellung der Menüs und erzeugt das Layout der Dialogfenster. Grundlage dafür sind die Ergebnisse des „Organization Managers“.

Neben dem automatisierten Anordnen der Interaktionselemente und der Zusammenstellung von Menüs fasst DON die Gestaltungsregeln des Layout-Prozesses zusammen und ermöglicht benutzerdefinierte Gestaltungsregeln. Zudem ist DON in der Lage, Bewertungen von Layout-Alternativen unter Zuhilfenahme von Evaluationsmetriken zu erstellen. Das JANUS-System [Hofmann 1998] übernimmt viele Prinzipien von DON/UIDE.

Wie UIDE hat auch DON den Nachteil, dass keine lauffähigen Prototypen oder Anwendungen erzeugt werden können. Zwar verfügt DON über einen interaktiven Mechanismus um Teile der Anwendung zu erstellen, allerdings ist der Aufwand bei vielen Inkrementen hierfür zu hoch. Da DON auf UIDE aufbaut, kommt auch hier eher das Konzept der Entwicklerunterstützung zum Tragen. Einbindung des Anwenders durch häufige Prototypengenerierung ist kaum möglich.

2.6.1.4 MASTERMIND

Das Projekt HUMANOID wurde mit UIDE vereinigt und wird als MASTERMIND-Projekt weiterentwickelt und an neue Techniken angepasst [Szekely 1995].

Bei MASTERMIND ist ein verteiltes Arbeiten und eine Anbindung neuer Werkzeuge durch CORBA möglich, auf das die MASTERMIND-Werkzeuge sowie die MASTERMIND Prototyping-Unterstützung aufsetzen. Aufbau und Modell werden im Anhang A („MASTERMIND: Entwicklungswerkzeuge und Modell“) näher erläutert.

Sowohl den Modellen, als auch den grundlegenden Konzepten, ist die Herkunft von UIDE und HUMANOID anzumerken. Neu ist vor allem die Generierungsmöglichkeit für C++ und die CORBA-Basierung[7], einschließlich einer Interface Definition Language (IDL).

Allerdings ist auch MASTERMIND nicht in der Lage, aus den Modellen eine Anwendung zu generieren. So können nur Teile generiert werden, und es entstehen ähnliche Probleme wie bei den Vorgänger-Methoden. Jedoch konzentriert sich MASTERMIND auf gängigere Sprachen (C++, CORBA-IDL) und unterstützt inkrementelles Prototyping und eine teilweise Generierung.

2.6.1.5 JANUS

Das JANUS-System [Balzert 1994] wurde an der Ruhr-Universität Bochum entwickelt und erlangte durch die Ausgründung der Firma Otris Bedeutung in der praktischen Anwendung. Ziel des JANUS-Systems ist die Generierung einer vollständigen Anwendung aus einem OOA-Modell, das mit zusätzlichen Informationen erweitert wurde [Hofmann 1998].

Für die Beschreibung werden spezielle Klassen für das OOA-Modell und eine auf UML basierende Beschreibungssprache JDL bereitgestellt. Diese ermöglicht die Angabe von wichtigen Parametern innerhalb des OOA-Modells.

Über die Erweiterungs-Schnittstelle können die JANUS-spezifischen Erweiterungen in CASE[8]-Werkzeuge wie Rational Rose [Rational 1999] oder MID Innovator [MID 1997] eingebunden und so bei der Modellierung verwendet werden.

Während andere Ansätze zumeist auf einen reinen Oberflächenprototypen abzielen, kann das JANUS-System komplette Anwendungsprototypen erstellen. Allerdings beschränkt sich JANUS auf betriebli-

[7] Die „Common Object Request Broker Architecture“ (CORBA) wurde von der Object Management Group (OMG) standardisiert.
[8] Computer Aided Software Engineering, d. h. Softwareentwicklungsunterstützung durch einen Rechner

che Anwendungssysteme, so dass eine sinnvolle Umsetzung nur für diese Anwendungstypen weitgehend ohne Kenntnis der Abläufe möglich ist.

Für die im OOA-Modell definierten Klassen wird je eine Ansicht generiert. Zudem werden Relationen von Klassen berücksichtigt und finden ebenfalls Eingang in die Dialoge. Die fertigen Dialoge und Abläufe werden auf C++-Code und Windows-Resource-Dateien abgebildet. Eine weitere Dissertation im Bereich JANUS [Hofmann 1998] beschäftigt sich zudem mit der automatischen Client-Server-Verteilung der so entstehenden Anwendung.

Das JANUS-Generatorensystem wurde in realen Projekten bereits erfolgreich zur Prototypen-Generierung eingesetzt. Zudem vertreibt die Firma Otris mittlerweile mehrere Versionen des JANUS-Generators. Eine breite praktische Einsetzbarkeit ist somit nachgewiesen.

Allerdings bieten die zugrundeliegenden OOA-Modelle nicht ausreichende Informationen und Optionen, um mehr als die Generierung von Prototypen und speziellen Benutzungsschnittstellen zu ermöglichen.

Die Code-Generierung für C++ und Windows liefert wesentlich effizientere Anwendungen als die auf UIMS basierenden Systeme. Zudem entsteht kein Bruch zwischen einem in C++ implementierten Fachkonzept und der Benutzungsschnittstelle.

Speziell für die bei ETAS entwickelten technischen Anwendungen eignet sich allerdings die an Datenbanktabellen orientierte Vorgehensweise nicht. Technische Anwendungen verfügen selten über homogene Datensatzgruppen. Vielmehr stehen hier einzelne Einstellungen und Abläufe im Vordergrund, speziell bei der für diese Arbeit als Zielsystem ausgewählten Konfigurations-Software.

2.6.1.6 Teallach

Die Beschreibung der zu erzeugenden Benutzungsschnittstelle erfolgt bei Teallach [Barclay 1999] gestützt auf drei Modelle:

- **Task Model** (TM): Das TM beschreibt hierarchisch Aufgaben und Unteraufgaben sowie deren Interaktionen auf Basis von Elementen des Presentation Model (PM).
- **Domain Model** (DM): Im DM werden die Daten und das Applikationskonzept als Klassenbeschreibung in einem ODMG[9]-Datenmodell spezifiziert. Der Zugriff auf Datenbanken oder Java-Klassen erfolgt ebenfalls über das DM.
- **Presentation Model** (PM): Das PM beschreibt die visuellen Aspekte der Benutzungsschnittstelle.

Da die einzelnen Modelle zum Teil gegenseitige Abhängigkeiten aufweisen, erfolgt die Modellkonstruktion reihenfolgeunabhängig und zudem interaktiv.

Eine Generierung kann zu unterschiedlichen Zeitpunkten ausgelöst werden. Das Ist-Ergebnis kann mit dem Soll-Ergebnis verglichen und das Modell auf dieser Grundlage weiter verfeinert werden.

Das Presentation Model besteht aus zwei Objekttypen:

- AIO (Abstract Interaction Object)
- CIO (Concrete Interaction Object)

Diese stellen unterschiedlich konkrete Realisierungen von Interaktionsobjekten dar. Ein CIO beschreibt Objekte der real generierten Oberfläche.

Ein AIO wird dagegen als Inputter, Displayer oder Editor klassifiziert, was eine Unterscheidung in reine Eingaben, reine Ausgaben und Mischformen bedeutet. Die Umsetzung verbleibender AIO auf für die Generierung notwendige CIO wird automatisch oder durch Spezifikation als CIO erledigt.

[9] Object Database Management Group

Bei der Generierung werden die Modelle auf eine MVC-Architektur (model-view-controller, [Krasner 1988]) abgebildet. AIO werden abhängig von Ihrem Typ auf bestimmte Elementklassen abgebildet: Inputter auf Controller, Displayer auf reine Views und Editoren auf beide, d. h. Controller und Views.

Der Abbildung liegt dabei eine eigene Klassenbibliothek zugrunde, so dass Tasks auf steuernde Klassen abgebildet werden können. Ein Überblick über den Aufbau des Code-Generators befindet sich in Anhang A (Aufbau des Teallach-Code-Genreators).

Der Ansatz ist allerdings erst zwei bis drei Jahre alt und wurde bisher kaum in der Praxis erprobt. Das Modell ist relativ einfach und kann somit gut eingesetzt werden. Problematisch ist, dass der Code nicht als „production code“ verwendet werden soll [Pinheiro 2000a]. Der Ansatz eignet sich daher nur für reines Prototyping.

Die Struktur des derzeitigen Marktes für Softwareentwicklungswerkzeuge und das Vorgehen in Industrieprojekten zeigt allerdings, dass reine Prototyping-Lösungen ohne Verwendbarkeit für das Produkt kaum eingesetzt werden. Für eine reine Prototyp-Studie ist die Modellierung zu aufwändig.

2.6.1.7 GENIUS

Mit GENIUS (GENerator for Interfaces Using Software ergonomic rules) können Prototypen von Benutzungsoberflächen für datenbankorientierte Anwendungen erzeugt werden ([Weisbecker 1993], [Janssen 1993], [Janssen 1995]).

GENIUS versucht, den Entwurf ergonomischer Benutzungsoberflächen in existierende Methoden der Softwaretechnik zu integrieren. Ausgangspunkt der Benutzungsoberflächen-Generierung ist hier deshalb das Datenmodell der zu entwickelnden Anwendung in Form eines **Entity-Relationship-Modells** (ERM).

Da das ERM nicht beschreibt, welche Daten der Benutzer für seine Aufgaben benötigt, wird ausgehend vom ERM ein weiteres Modell in Form von logischen Sichten auf die Daten definiert. Unter Zuhilfenahme eines grafischen Editors können so aus den Daten des Modells Fenster zusammengestellt werden, während die enthaltenen Attribute mit Hilfe von Auswahlregeln in Interaktionselemente umgesetzt werden. GENIUS positioniert diese wiederum mit Hilfe von Layoutregeln automatisch im Fenster.

Während damit die Präsentationsschicht (statisches Layout) generiert werden kann, müssen für die Erzeugung eines animierbaren Prototypen die groben Dialogabläufe erst noch mit einem grafischen Werkzeug definiert werden.

Für die Code-Generierung ist deshalb eine Definition der groben Dialogabläufe mit Hilfe von **Dialognetzen** [Janssen 1995] notwendig. Dazu werden Dialogübergänge zwischen den einzelnen Fenstern und der Aufruf von Anwendungsfunktionen als Reaktion auf Benutzeraktionen spezifiziert. Die Dialogabläufe enthalten Dialogübergänge, Reaktionen auf Benutzeraktionen und Aufrufe von Fachkonzept-Operationen.

Aus beiden Modellen wird Code für ein UIMS generiert, was eine Evaluation und Nachbearbeitung des Benutzungsoberflächen-Prototyps ermöglicht. Schon bei der Generierung wird durch Einsatz von ergonomischen Regeln die Benutzbarkeit verbessert. Die fachlichen Operationen werden mit dem für das UIMS generierten Code angebunden.

GENIUS geht vom Datenmodell aus und leitet daraus die Benutzungsoberfläche ab. Ähnlich wie das JANUS-System können daher auf Datenbanken basierende Anwendungen einfach modelliert und eine passende Benutzungsschnittstelle erzeugt werden.

Für technische Anwendungsgebiete eignet sich GENIUS daher weniger als ein ablauforientierter Ansatz. Zudem ist die Verwendung eines UIMS als Generierungsziel eher für reine Prototypen geeignet.

2.6.1.8 Weitere Ansätze

Die im Folgenden aufgezählten Ansätze haben ebenfalls Bedeutung für diese Diplomarbeit. Eine Ausführliche Beschreibung befindet sich in Anhang A („Ansätze zu MB-IDE“).

- ADEPT: Bei ADEPT [Wilson 1996] werden Aufgaben (Tasks), die der Benutzer mit dem System lösen soll, identifiziert und schrittweise verfeinert. Daraus entsteht das Aufgabenmodell, das in der abstrakten Spezifikationssprache TKS (Task Knowledge Structure, [Johnson 1992]) definiert wird.
- TRIDENT: Bei TRIDENT (Tools foR an Interactive Development ENvironmenT) handelt es sich um eine werkzeuggestützte Methode zur automatisierten Entwicklung von interaktiven, betriebswirtschaftlich orientierten Applikationen [Bodart 1993]. TRIDENT basiert auf der in [Bodart 1985] beschriebenen Methodik IDA zur Formalisierung von interaktiven Applikationen.
- HUMANOID: Bei der Modellierung mit HUMANOID stehen Applikationen mit grafischen Benutzungsoberflächen im Vordergrund [Szekely 1992]. Prinzipiell besteht aber keine Einschränkung auf bestimmte Anwendungsklassen. Die Entwurfskonzepte werden in unterschiedliche, voneinander unabhängige Dimensionen aufgeteilt. HUMANIOD unterstützt zudem mehrere Abstraktionsebenen bei der Regel-Spezifikation. Aus diesen Regeln wird wiederum die Transformation des Anwendungsmodells in eine konkrete Benutzungsoberfläche hergeleitet.
- FUSE: Auch FUSE (Formal User Interface Specification Envoironment, [Lonczewski 1996]) stellt verschiedene Werkzeuge mit dem Ziel der automatisierten Generierung von Benutzungsoberflächen zur Verfügung. Im Gegensatz zu den meisten Ansätzen ist FUSE auf keinen bestimmten Anwendungsbereich spezialisiert.
- TADEUS: Das TADEUS-Projekt [Schlungbaum 1995] ist vom Ansatz her mit GENIUS vergleichbar. Es werden zunächst ein Aufgabenmodell, ein Fachkonzept und ein Benutzermodell erstellt. Der Entwickler definiert daraus ein abstraktes Modell der Benutzungsoberfläche. Aus diesem wird UIMS-Quellcode generiert.
- MECANO: Die modellbasierte UI-Entwicklungsumgebung MECANO [Puerta 1996] ist auf formular- und graphenbasierte Benutzungsoberflächen ausgerichtet. MECANO benötigt ein Modell des Fachkonzepts (domain model) und ein Modell der Benutzungsoberfläche (interface model).
- SCON: SCON wurde als spezialisierter Generator-Ansatz bei der Robert Bosch GmbH entwickelt [Beutter 1999]. Um Konfigurationen von vielen verschiedenen Systemen durchführen zu können, ohne für jedes System manuell eine Konfigurationsoberfläche erstellen zu müssen, ermöglicht SCON eine automatisierte Erstellung von JAVA-Benutzungsoberflächen zur Konfiguration.

2.6.2 Vorteile modellbasierter GUI-Generierung

Grund für die Entwicklung der oben vorgestellten Ansätze zur GUI-Generierung ist die Notwendigkeit einer Modellierung von Systemen in der Analyse/Spezifikationsphase verbunden mit dem hohen Bedarf an für den Kunden/Anwender verständlichen Darstellungsformen.

So bieten Oberflächenprototypen immer noch die beste Grundlage für eine Evaluierung durch den Benutzer. Können Prototypen schon früh im Entwicklungsprozess generiert werden, verringern sich die Kosten durch falsche Entwurfsentscheidungen. Im Gegenzug wird die Benutzerpartizipation vereinfacht und gefördert.

Ein weiterer Vorteil einer Generierung von grafischen Benutzungsschnittstellen liegt im einfachen Übergang vom Modell zur Realisierung. Zudem wird eine Portierung auf andere Systeme trotz Verwendung von proprietären Aufrufen erleichtert.

Die bei der Generierung verwendeten Regeln erzeugen eine höhere Konsistenz der Benutzungsoberfläche als dies bei manueller Eingabe, zumal durch mehrere Entwickler, der Fall ist, was schon allein die Usability verbessert.

Zumeist scheitert der Einsatz von Modellen, die der ergonomischen Verbesserung der Benutungsschnittstelle dienen, am Mehraufwand und der beschränkten Verwendbarkeit im weiteren Entwicklungsprozess. Durch den Einsatz von GUI-Generatoren steigt die Akzeptanz von Modellen der Benut-

zungsschnittstelle sowohl auf Entwickler- als auch auf Anwenderseite, da der einmal investierte Aufwand durch die Generierung wieder zu einer Zeiteinsparung führt.

2.6.3 Beschränkungen und Nachteile modellbasierter GUI-Generierung

Mit dem Einsatz von GUI-Generatoren gehen außer den schon genannten Vorteilen auch Nachteile und Beschränkungen einher. So legt der GUI-Generator die zu erstellenden Modelle fest, was den alternativen Einsatz auch von eventuell besseren Modellierungen nicht gestattet. Zudem setzen fast alle bekannten GUI-Generatoren unterschiedliche Modelle ein, die nicht portabel sind, so dass ein Umstieg gleichbedeutend mit der Notwendigkeit einer neuen Modellierung ist.

GUI-Generatoren beruhen auf Algorithmen und haben deshalb speziell in Bereichen Probleme, die auch Menschen unterschiedlich beurteilen. In das Ergebnis fließen deshalb auch immer Entscheidungen mit ein, die vom Entwickler des GUI-Generators getroffen wurden und oft nicht im Nachhinein verändert werden können.

Aufgrund der Beschränkungen der Modelle und des „Weltwissens“ der GUI-Generatoren, sind Generator-Ansätze immer zur Beschränkung auf bestimmte Anwendungstypen oder zur Eingabe zusätzlicher Einstellungen gezwungen.

Die meisten der hier untersuchten Ansätze sind vorwiegend für grafische Benutzungsschnittstellen konzipiert, da hier mehr Optimierungsmöglichkeiten bestehen und diese Systeme mittlerweile die Mehrheit der Neuenwicklungen darstellen. Durch Modifikationen im letzten Generierschritt sind allerdings viele Ansätze auch für andere Benutzungsschnittstellen verwendbar.

2.6.4 Verwendung mehrerer Modelle

Die vorgestellten Ansätze versuchen auf unterschiedliche Art, mehrere Darstellungen zu einem Modell zusammenzufassen. Durch diese Vorgehensweise sind detaillierte Beschreibungen sowohl von dynamischen, prozeduralen Vorgängen also auch von entitätenorientierten Datenstruktur-Darstellungen möglich.

Problematisch sind diese mehrteiligen Modelle aus drei Gründen:

1. **Höhere Komplexität:** Müssen mehrere Teilmodelle erzeugt werden, die zusammen das gesamte Modell ergeben, so sind Zusammenhänge oft nur schwer vermittelbar, da sich sowohl die Inhalte als auch die Darstellungsweisen (z. B. Ablaufdiagramm und Entity-Relationship-Modell) erheblich unterscheiden. Zudem müssen die Berührungspunkte der Teilmodelle zusätzlich modelliert werden. Aus Sicht der Software-Ergonomie können sich so inkonsistente Benutzungsschnittstellen ergeben, wenn die Modellierung der Abläufe nicht der Umsetzung des Objekt- oder Datenmodells entspricht. Der Anwender bemerkt derartige Inkonsistenzen beispielsweise, wenn Attribute desselben Objekts verteilt auf unterschiedliche Dialogsequenzen auftreten.
2. **Höherer Verwaltungsaufwand, Inkonsistenzen und fehlende Aktualisierung in der Entwicklung:** Aus Sicht des Software Engineering fehlt in solchen Modellen ein sogenanntes Master-Dokument. Die Wahrscheinlichkeit, dass bei unterschiedlichen Modellen für dieselbe Benutzungsschnittstelle Inkonsistenzen auftreten, ist sehr hoch. Zudem erschwert sich so auch die Aktualisierung des Gesamtmodells bei Änderungen in späteren Entwicklungsschritten. So eignen sich die meisten Modelle nicht zur Verwendung über den kompletten Entwicklungsprozess, sondern bieten wie FUSE eine mehrstufige Entwicklung eines Oberflächeprototypen, der nachfolgend mit Anwendungsfunktionalität ausgestattet werden kann.
3. **Inkonsistente Entwicklungsdokumente:** Konsistente Entwicklungsdokumente sind die Grundlage für eine konsistente Software. Basieren Spezifikation, Hilfe, Anwender-Dokumentation, Testfälle und Entwicklungsdokumentation auf unterschiedlichen Grundlagen, leidet auch die Ergonomie der Anwendungssoftware. Falsche Beschreibungen und Hilfen verwirren den Anwender. Anwendungs-Fehlreaktionen, die durch Tests nicht entdeckt werden, können zu Datenverlust und mangelnder Steuerbarkeit führen.

Eine Möglichkeit, diese Probleme zu umgehen, bietet die Nutzung eines Basismodells, aus dem bei Bedarf vertikale Modelle (gleicher Zeitpunkt, nur andere Sicht) generiert werden können. Ein derartiges Modell unterstützt zudem eine Möglichkeit des Reverse-Engineerings, um Änderungen in nachgelagerten horizontalen (anderer Zeitpunkt) Modellen auf das Ausgangsmodell abzubilden.

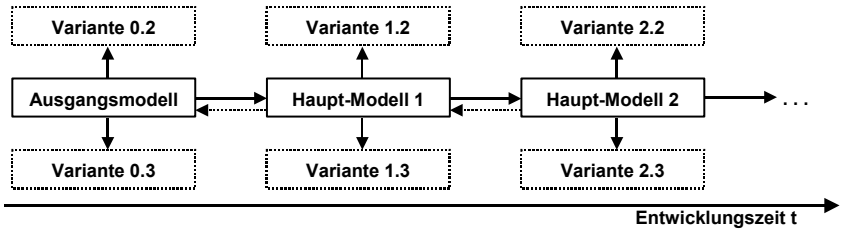

Abbildung 3: Variantenbildung und Reverese-Engineering über Hauptmodelle.

2.7 XML und DTD

Noch vor einigen Jahren wurden strukturierte Daten in fast allen Fällen in einem proprietären Format abgespeichert, das zudem eine für Menschen nicht lesbare Repräsentation hatte.

Mit dem Erfolg von HTML hielten auch andere Markup-Sprachen Einzug in den Markt. Für Datenbeschreibungssprachen war eine schnelle Verarbeitung, eine klare Syntax und Erweiterbarkeit wichtig. Alle diese Merkmale sind in der eXtensible Markup Language (XML) zu finden. Zudem ist XML direkt von Menschen lesbar.

Das Prinzip von XML ist, wie bei HTML, der Einsatz geschachtelter Tags[10], die im Gegensatz zu HTML allerdings fast beliebig definiert werden können. Sie setzten sich aus Buchstaben, Unterstrich („_“) und Ziffern zusammen und können über Attribute verfügen.

```
<MEIN_NAME>
    <VORNAME>Thomas</VORNAME>
    <NACHNAME>Schlegel</NACHNAME>
</MEIN_NAME>
```

Abbildung 4: Beispiel für ein XML-Fragments

Anders als bei HTML werden nur „wohlgeformte“ (well-formed) XML-Dokumente akzeptiert. Während die Syntax bei HTML teilweise aufgeweicht wurde, müssen wohlgeformte XML-Dokumente von einem Parser linear zu verarbeiten sein. Eingeschachtelt zwischen den Tags stehen die Werte. Es ist auch möglich, leere Elemente zu definieren, die dann keinen Wert enthalten.

Eine Definition der legal möglichen Dokument-Strukturen, d. h. eine Meta-Beschreibung der Struktur, ist nicht zwangsläufig notwendig, allerdings sehr hilfreich. Auf diese Weise können erlaubte Strukturen festgelegt werden, die auch semantisch sinnvoll sind.

Mit Hilfe von Document Type Definitions (DTD) oder XML-Schemata können diese Angaben zur Struktur erstellt werden. DTD stellen eine etablierte Beschreibungsform dar, mit der die möglichen Hierarchien festgelegt werden können. Allerdings sind nicht alle Angaben möglich. So kann zum Beispiel keine Einschränkung auf 3 bis 5 Sohnknoten angegeben werden, sondern nur von 0/1 bis unendlich. Auch ist die Angabe von Wertebereichen und genaueren Typen nicht möglich.

[10] Ein (öffnendes) Tag hat die Form <Tagname>. Ein schließendes Tag unterscheidet sich durch einen Schrägstrich „/“ vor dem Namen: </Tagname>. Um in HTML beispielsweise eine Zelle zu definieren müssen die Zeilen (Table Row, TR) und Spalten-Tags (TD) den Eintrag geschachtelt umschließen: <TR><TD>Der Eintrag</TD></TR>. Tags werden mit dem Schlüsselzeichen (Token) „<“ begonnen und enden mit einem „>“.

DTD stellen also für die Strukturbeschreibung die einfachere – im Vergleich zu XML-Schemata besser unterstützte – Alternative dar. Allerdings sind XML-Schemata mächtiger und somit für manche Aufgabenstellungen unumgänglich, gerade wenn es um Typenbeschreibungen und Bereiche geht.

Mit Hilfe von bestimmten Operatoren können Vorgaben für die untergeordneten Tags gemacht werden:

Notation	**Menge (untergeordneter Elemente)**
A	1
A+	1..n
A*	0..n
A?	0..1
A, B	A und danach B
A | B	Entweder A oder B (XOR, exklusives oder)
#PCDATA	Unformatierter Text, d. h. kein untergeordnetes Tag, sondern Inhalt

Tabelle 1: Tabelle der Notationen und damit beschriebenen Mengen in einer DTD. Sei dabei A dabei das untergeordnete Tag bzw. seien A und B dabei die untergeordneten Tags.

XML stellt in Verbindung mit DTD eine sehr gut unterstützte und leistungsfähige Methode zur Informationsstrukturierung dar. XML-Schemata erlauben darüber hinaus eine noch genauere Spezifikation mit Hilfe von Definitionsbereichen und differenzierten Datentypen, sind jedoch noch nicht vollständig standardisiert.

3 Aufgabenstellung und Ansatz

Die Kenntnis von Hintergrund, Aufgabe, Begriffen und Ansatz der Arbeit ist Grundvoraussetzung für das Verständnis der nachfolgend präsentierten Konzepte. Diese Übersicht soll deshalb in diesem Kapitel vermittelt werden.

3.1 Hintergrund

In diesem Kapitel wird die Firma ETAS GmbH vorgestellt, bei der die Diplomarbeit erstellt wurde. Zudem wird zum besseren Verständnis des Umfelds auch das Rahmenprojekt vorgestellt.

3.1.1 ETAS

Die ETAS[11] GmbH mit Hauptsitz in Stuttgart wurde 1994 als Tochtergesellschaft der Robert Bosch GmbH gegründet und beschäftigt über 500 Mitarbeiter weltweit bei einem Umsatz von 100 Millionen Euro im Jahr 2001.

Eigene Tochtergesellschaften in Ann Arbor (USA), Yokohama (Japan) und Paris (Frankreich), Vertriebspartner in Brasilien und Korea sowie ein Sales Office in der Nähe von Birmingham (Großbritannien) machen den Fokus der ETAS auf die Automobilindustrie weltweit deutlich.

Das Unternehmen liefert ein komplettes Spektrum an Engineering-Werkzeugen für die Entwicklung eingebetteter Systeme, die auf den Automobilbereich abzielen, und bietet auch entsprechende Engineering-Dienstleistungen sowie Auftragsentwicklungen, Beratung, Support und Schulungen an.

CASE-Werkzeuge (Computer Aided Software Engineering) für eingebettete Systeme im Automobilbereich stellen das Gros der ETAS-Toolkette. Mit dem Werkzeug ASCET-SD steht eine durchgängige Entwicklungsumgebung für den gesamten Entwurfsprozess zur Verfügung, der von der Funktionsentwicklung und Simulation über die Erzeugung von Prototypen mit Echtzeitverhalten bis zur automatischen Seriencodegenerierung reicht.

Die offene Architektur, der modulare Aufbau und die Unterstützung von Industrie- und Automotive-Standards ermöglichen die flexible Anpassung an unterschiedliche Entwicklungspotenziale und bestehende Infrastrukturen der Anwenderfirmen.

3.1.2 Rahmenprojekt

Für die Bewältigung von Entwicklungsaufgaben stellen ASCET-SD und die in Entwicklung befindliche Toolsuite YUKON eine Vielzahl von Funktionen zur Verfügung, die für den Anwender möglichst leicht erlernbar und benutzbar sein sollte. Aus diesem Grund soll die Benutzerinteraktion in den nächsten Versionen weiter verbessert werden.

Die Oberfläche wird bisher von den Entwicklern direkt programmiert. Das Entwicklungsprozessmodell im Unternehmen sieht keine spezielle Vorgehensweise zur Benutzungsschnittstellengestaltung im YUKON-Projekt vor.

Eine entsprechende Vorgehensweise, gekoppelt mit einem Werkzeug zur Unterstützung bei der Entwicklung von Benutzungsschnittstellen oder zur Erzeugung entsprechender Anwendungsteile, könnte jedoch in den flexiblen Entwicklungsprozess eingebunden werden. Durch den Prototypenstatus von YUKON könnte diese auch zur Evaluierung von Alternativen eingesetzt werden.

Ein Generator zu Erzeugung der grundlegenden Businessobjekte aus DTDs existiert bereits. Ist es mit Hilfe eines Transformators möglich neben Teilen der Benutzungsoberfläche auch die entsprechenden DTDs zur Verfügung zu stellen, wäre eine generative Anbindung der Benutzungsoberfläche an die Datenhaltung möglich.

[11] Engineering Tools Applications and Services for the development of automotive embedded software

Als Beispiel in dieser Arbeit wird der Experimental Target Configurator (ETC) verwendet. Dieser wurde als Teil einer Tool-Suite konzipiert. Dabei wird mit dem ETC eine Experimentalplattform auf VME-Basis (ETAS ES 1000, ETAS ES 4100 o.ä.) angesteuert, die weitere I/O-Hardware in Form von Einschubkarten enthält [ETAS 2001].

Die Konfiguration dieser Einschubkarten und der zu diesen Bauteilen gehörenden Signale ist die zentrale Aufgabe des ETC. Der ursprüngliche ETC dient dabei einer komfortablen und einfacheren Einrichtung neuer Karten in einem ASCET-SD-Modell.

3.2 Aufgabenstellung

Die Nutzung funktional hochwertiger Software wird häufig durch eine grafische Benutzungsoberfläche entscheidend geprägt. Speziell bei komplexen Systemen hat die Gestaltung der Benutzungsschnittstelle eine zentrale Bedeutung für den Erfolg einer Anwendung.

Bei der Entwicklung von Benutzungsschnittstellen werden jedoch bisher kaum standardisierte Methoden eingesetzt. Häufig werden erst sehr spät in der Anwendungsentwicklung oder am fertigen Produkt Änderungen vorgenommen, um die als ungünstig erkannten Teile nachzubessern. Dies bringt neben hohem Zusatzaufwand meist eine suboptimale Lösung mit sich, da eine Änderung des Gesamtkonzepts nicht mehr möglich ist.

Wünschenswert ist deshalb die Entwicklung und Erprobung einer systematischen Vorgehensweise für die frühen Phasen des Anwendungsentwurfs, mit der die Software-Ergonomie verbessert werden kann.

Aufgabe der Diplomarbeit ist es, moderne Vorgehensweisen und Methodiken auf dem Gebiet der Software-Ergonomie zu erschließen. Der Schwerpunkt liegt dabei auf der Erweiterung und Verbesserung der frühen Phasen des Entwicklungszyklus unter dem Gesichtspunkt der Software-Ergonomie.

Zur Einordnung dieser Methodik in das Themengebiet Software-Ergonomie soll zunächst ein Überblick über den aktuellen Stand der Technik in diesem Teilgebiet gegeben werden.

Mit Hilfe der Vorgaben für das Design von Oberflächen kann Einfluss auf die Benutzbarkeit des Endproduktes genommen werden. Möglich wäre eine Ausgabe von Hinweisen. Allerdings bietet die – in der Aufgabenstellung vorgeschlagene – prototypische Anwendung der erarbeiteten Methodik in Form eines Transformators die besseren Möglichkeiten für eine Weiterentwicklung.

Es soll eine Methodik entwickelt werden, die es erlaubt, aus einer formalen Notation von Use-Cases Designvorgaben, Vorschläge für Oberflächentests, Oberflächenprototypen und weitere Artefakte zu generieren. Mit Hilfe dieser Ergebnisse kann die Benutzbarkeit des Endproduktes verbessert werden.

Um die genannten Resultate erzeugen zu können, muss allerdings erst ein mächtiges Interaktionsmodell zur Verfügung gestellt werden. Das zu diesem Zweck entworfene Metamodell wird in dieser Arbeit vorgestellt.

Mit Hilfe einer automatisierten Umsetzung der Vorgaben in eine Oberfläche stellt der Transformator die Einhaltung bestimmter software-ergonomischer Grundsätze für die Benutzungsschnittstelle sicher.

Um die Praxistauglichkeit dieses Vorgehens zu untersuchen, soll die Methodik mit Hilfe des Transformators beispielhaft auf die Anforderungsbeschreibung des Experimental Target Configurators aus einem aktuellen ETAS-Entwicklungsprojekt angewandt. Durch noch notwendige manuelle Änderungen und Ergänzungen soll die Erstellung eines funktionsfähigen Oberflächenprototyps für das Entwicklungsprojekt möglich werden.

Durch einen Vergleich mit bisher eingesetzten Methoden soll abschließend dargelegt werden, wie sich der Aufwand auf die einzelnen Projektphasen qualitativ verteilt.

3.3 Begriffe

Im Zuge der Erarbeitung des in dieser Arbeit entwickelten Ansatzes werden verschiedene Begriffe benutzt oder geprägt. Die wichtigsten werden in diesem Abschnitt kurz erläutert, alle weiteren bei Ihrer Verwendung definiert.

Label: Ein Label ist die Beschriftung eines Interaktionselements, die außerhalb des Elements steht.

Abbildung 5: Eingabeelement mit Label

Register bestehend aus Registerkarten mit Reitern (Tab 1 bis Tab 5):

Abbildung 6: Ein Register mit seinen Registerkarten und den Reitern der Karten

Hotkey: Um Menüs oder Dialoge auch per Tastatur bequem ansteuern zu können, kann für jedes Element, das den Fokus erhalten kann, eine Taste oder Tastenkombination definiert werden, die zum Anspringen dieses Elements führt. Unter Windows ist dies meist die Kombination <Alt>-<in_der_Beschriftung_markierter Buchstabe>, z.B. <Alt>-<D> für „Datei“.

Shortcut: Häufig benötigte Funktionen sollen oft per Tastatur direkt ausgelöst werden. Hier können Tastenkombinationen einem Befehl zugeordnet werden, z.B. <Strg>-<X> dem Befehl [Bearbeiten]-[Ausschneiden], so dass dieser per „Abkürzung“ angewählt werden kann.

Artefakte: Im Zusammenhang dieser Arbeit sind Artefakte immer Zwischen- oder Endresultate eines Softwareentwicklungsprojekts bzw. eines Generierschritts. Artefakte haben dabei immer Dokumentenform oder eine entsprechende nicht-flüchtige Repräsentation.

Metamodell: Ein Metamodell stellt in diesem Zusammenhang Mechanismen zur Beschreibung eines Modells zur Verfügung, d. h. das Metamodell bietet die Grundlage und enthält die Vorschriften zur Erstellung eines korrekten Modells.

IML: Die Interaction Modelling Language (IML) ist eine in dieser Arbeit entwickelte Sprache zur Interaktionsbeschreibung.

IML-Metamodell: Das IML-Metamodell ist die Definition der Sprache IML und liegt in Form einer IML-DTD und weiterer Vorschriften vor. Mit Hilfe dieser Informationen können IML-Modelle erstellt werden.

IML-Modell: Ein IML-Modell ist die konkrete Instanz eines Ablaufmodells in Form einer XML-Datei. Jedes IML-Modell basiert auf dem IML-Metamodell und nutzt deshalb die oben genannte DTD, nach der es auch auf syntaktische Korrektheit überprüft werden kann.

DiLL: Die Dialog Layer Language (DiLL) ist eine, ebenfalls in dieser Arbeit entwickelte, Sprache zur Beschreibung von statischem Aufbau und dynamischem Ablauf von Dialogen.

DiLL-Metamodell: Das DiLL-Metamodell liegt in Form einer DiLL-DTD vor, ergänzt um weitere Vorschriften. Es bildet die Grundlage zur Erstellung von Dialogrepräsentationen und -abläufen mit Hilfe der in der DiLL-DTD definierten Sprache DiLL.

DiLL-Modell: Ein DiLL-Modell enthält Dialogrepräsentationen und -abläufe eines konkreten Projekts. Ein DiLL-Modell liegt in Form einer XML-Datei vor und basiert auf der oben genannten DiLL-DTD, nach der es auch auf syntaktische Korrektheit überprüft werden kann.

ETD: Die Element Transformation Description (ETD) ist eine sehr einfache Sprache zur Umsetzung von Elementen eines DiLL-Modells in Codefragmente. Auch die ETD wurde im Rahmen dieser Arbeit entwickelt.

ETD-Metamodell: Das ETD-Metamodell ist eine DTD. Diese enthält die Syntaxangaben für den Aufbau von Elementbeschreibungen.

Elementbeschreibungen: Elementbeschreibungen werden in einer XML-Datei zusammengefasst, die eine Instanz des ETD-Metamodells bildet. Für jeden Elementtyp sind darin Codefragmente abgelegt.

3.4 Ansatz

Um software-ergonomische Überlegungen anstellen zu können, bedarf es einer Kenntnis der Zusammenhänge im jeweiligen Anwendungsbereich. Zumindest aber müssen die von Benutzerseite sichtbaren Abläufe vorliegen. Nur Anhand dieser Informationen können Entwurfsentscheidungen getroffen werden.

Eine rein datenzentrierte Repräsentation ermöglicht diese Sicht nicht, da nur Zustände und nicht die für den Benutzer wichtigen Interaktionen wiedergegeben werden. Ziel ist es also, diese Interaktionen zu modellieren.

Sollen aus einer derartigen Beschreibung Oberflächen generiert werden, so ist es wichtig, dem Entwickler trotzdem die Möglichkeit zur Behebung von Transformator-Unzulänglichkeiten in seinem Sinne zu geben. Hierzu ist eine Zwischenrepräsentation der Oberfläche notwendig, die Anpassungen ermöglicht.

Die Modellierung einer benutzerzentrierten, interaktions- und ablauforientierten Sicht und deren Umsetzung in eine layoutorientierte Sicht sind die Basis für die letztendliche Generierung eines Oberflächenprototypen und weiterer software-ergonomiebezogener Artefakte wie Hilfen und Oberflächentestfälle.

Hierzu wird im Folgenden der Ansatz *HERBS* entwickelt: **H**uman-centered **E**rgonomic **R**equirements specification and **B**uilding of **S**oftware.

HERBS soll dabei ein durchgängiges Konzept zur Verfügung stellen, das folgende Teile enthält:

- Eine Eingabesprache zur Ablaufmodellierung
- Eine Dialogmodellierungssprache
- Regeln zur Transformation der Eingabesprache in die Dialogmodellierungssprache
- Regeln zur Layoutgestaltung mit Hilfe des Dialogmodells
- Eine Vorgehensweise zur Erzeugung von ablauffähigen Oberflächenprototypen
- Einen beispielhaften Transformator zur Erzeugung von Dialogmodell und weiteren Artefakten aus der Eingabesprache

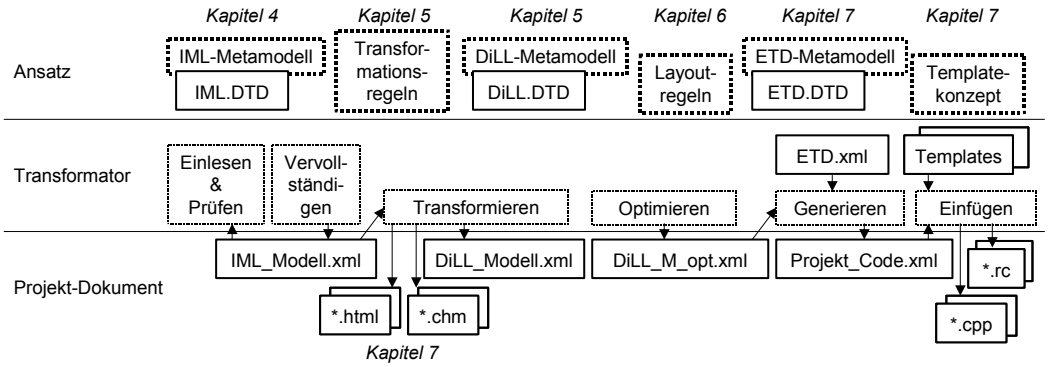

Abbildung 7: Überblick über Ansätze, Transformatorschritte und entstehende Dokumente von *HERBS*

3.5 Vorüberlegungen

3.5.1 Warum Ablauforientierung?

Viele der aktuellen Forschungsergebnisse zum Thema modellbasierte Oberflächengenerierung beruhen auf Datenmodellen. Hierbei ist eine Konzentration auf das Layout und die einfache Vernetzung der Dialoge zu beobachten.

Oft wird dabei in den veröffentlichten Schriften an diesen Ansätzen die Kritik geübt, dass Abläufe nicht sinnvoll berücksichtigt oder generiert werden können, es sei denn, dass zusätzliche Modelle zur Verfügung gestellt werden, die aber den Entwicklungsaufwand wesentlich vergrößern.

Im Falle von JANUS ist beispielsweise noch nicht klar, wie Interaktionsabläufe überhaupt integriert werden könnten – ein Manko der datenzentrierten Modellierung: „*Mit der hier vorgestellten JDL-Beschreibung ist es nicht möglich, Sichten auf das OOA-Modell zu beschreiben. Weiterhin können keine Arbeitsabläufe beschrieben werden.*“ [Kruschinski 1999]

Seinen Grund hat dieses Problem in der reinen Datenzentrierung dieser Ansätze. Sowohl OOA-Beschreibung als Entity-Relationship-Beschreibungen stellen nur Relationen zwischen Daten und keine Abläufe dar.

Die Grundlage einer ergonomischen Gestaltung von Dialogen ist jedoch die Konzentration auf die Arbeitsabläufe und Ziele des Benutzers.

In realen Softwareentwicklungsprojekten wird eine Interaktionsablaufbeschreibung spätestens in der Entwurfs-Phase aus dem Projekt entfernt, da in der OOD-basierten Architektur mit UML nur noch Daten und deren Kommunikation beschrieben werden. Eher prozedurale Vorgehensweisen wie bei der sequentiellen Eingabe von Daten können hier nur noch mit Hilfe zusätzlicher Diagramme modelliert werden.

Eine Ablauforientierung bietet also die Grundlage für eine benutzerinteraktionsorientierte Modellierung.

3.5.2 Wahl des Sprachkonzepts

Stehen geeignete Entwicklungsumgebungen oder Editoren zur Verfügung, kann das Format für Modelle, d. h. die syntaktische Repräsentation, prinzipiell beliebig gewählt werden.

Fehlen diese, oder soll eine gute Verständlichkeit und Erweiterbarkeit des Metamodells gesichert werden, muss die Repräsentation vom Menschen direkt lesbar sein.

Eine Darstellungsform, die diesen Kriterien entspricht, stellt die mittlerweile weit verbreitete XML dar. Zudem ist sie für die Modellierung baumartiger, hierarchischer Strukturen, wie denen von Dialogbeschreibungen mehrschichtigen Ablaufspezifikationen, bestens geeignet. Use-Cases und Gruppen können so intuitiv angeordnet werden.

Positiv wirkt sich zudem die weite Verbreitung von Werkzeugen für XML aus. Während für proprietäre Formate zunächst immer Editoren und Viewer erstellt werden müssen, kann ein XML-basiertes Modell direkt in einem gängigen XML-Editor eingegeben werden, sobald eine Document Type Definition (DTD) des Metamodells existiert.

Ein weiterer Vorteil ist die leichtere Verarbeitung durch die Verwendung bereits existierender XML-Parser, wie den Microsoft XML Core Services, die eine Klassenhierarchie für den Aufbau und den Zugriff auf das Document Object Model (DOM)[12] zur Verfügung stellen. Zudem erleichtern diese eine Transformation in eine ebenfalls XML-basierte Zwischenrepräsentationen.

[12] baumförmige Repräsentation des XML-Dokuments, auf die mit einer Klassenbibliothek zugegriffen werden kann

Durch die Realisierung in XML können zunächst grobe Beschreibungen zur Generierung herangezogen und in den folgenden Schritten verfeinert werden. Das Metamodell stellt dazu die passenden Tag-Definitionen[13] mit Hilfe von DTDs oder XML-Schemata zur Verfügung.

Ein wichtiger Punkt ist die Realisierbarkeit eines entsprechenden Modell-Editors. Hier bietet eine DTD sehr gute Ansatzpunkte, da für einen ersten Entwurf eines Editors die meisten Einstellungen und die gesamte Struktur direkt aus dieser übernommen werden können.

Auf diese Weise kann sichergestellt werden, dass die konzeptionellen Überlegungen in dieser Arbeit auf einfache Weise umgesetzt und für die Bildung von Interaktionsmodellen relativ leicht unterstützende Anwendungen erstellt werden können.

3.5.3 Benutzerzentrierung

Während früher der Benutzer gezwungen wurde, sequentiell die angeforderten Daten einzugeben, geht man heute von einer stärker benutzerorientierten Sicht aus. Der Anwender kann selbst auswählen was und in welcher Reihenfolge er eingeben bzw. editieren will.

Ausgehend von der Sichtweise, dass jeder Benutzer, zumindest aber jede Benutzergruppe, spezielle Anforderungen und Eigenschaften hat, werden bei der Anforderungsanalyse Szenarien eingesetzt. [Rechenberg 1997]

Der Trend in der Softwaretechnik geht ebenfalls in Richtung Modellierung von Anwendungsszenarien. In der UML wird diese mit den Use-Cases abgedeckt. Gängige Werkzeuge wie Rational Rose [Rational 1999] sehen keinerlei Formalisierung vor, so dass eine einfache, an wenige Regeln gebundene, Beschreibung möglich ist, die zudem über eine grafische Repräsentation verfügt.

Nachteile der fehlenden Formalisierung sind die Inkompatibilität zwischen Spezifikationen unterschiedlicher Institutionen und der fast zwangsläufige Bruch zwischen der Spezifikation mit Use-Cases und dem Entwurf.

Notwendig ist also eine für alle verständliche, gemeinsame und formale Darstellung, die als Ausgangsbasis für die weiteren Entwicklungsschritte dient und bis hin zu Test und Dokumentation durchgängig Verwendung finden kann.

Trotz Fortschritten in Richtung Automatisierung mit Oberflächengeneratoren und Werkzeug-Unterstützung der Oberflächenentwicklung bereitet allein die Integration von funktionalen Anforderungen mit nicht-funktionalen Anforderungen (NFR, non-functional requirements) immer noch Probleme.

Mengengerüst, Abläufe, Relationen zwischen Anwendungsfällen, Antwortzeiten und andere Teile der Spezifikation können bisher kaum in ein Modell integriert werden.

Mit existierenden MB-DIE können Datenmodelle wie Entity-Relationship-Modelle [Ziegler 1999] oder OOA-Modelle [Kruschinski 1999] zu Masken verarbeitet werden. Die ergonomischen Aspekte werden dabei allerdings meist vernachlässigt oder führen mangels semantischer Informationen zu einer falschen Abbildung der realen Arbeitsabläufe, da in den statischen Datenmodellen keine Reihenfolgen oder dynamischen Abläufe beschrieben werden können.

Dies führt in letzter Konsequenz zu Applikationen, die für den Benutzer nicht nachvollziehbare, nur auf die Implementierung gestützte, Abläufe enthalten. Um die Benutzbarkeit zu verbessern, muss also schon in den frühen Phasen der Anwendungsentwicklung eine Orientierung am Vorgehen des Benutzer stattfinden, und dieses entsprechend modelliert werden.

[13] Erläuterung des Begriffs „Tag“ in XML siehe Kapitel 2.7.

3.5.4 Direkte Generierung von Oberflächen mit Standard-Elementen

Viele Ansätze zur direkten Generierung von Oberflächen arbeiten mit Interpretern oder zusätzlichen Bibliotheken. Gerade die Generierung mit spezialisierten Elementen (SCON, JANUS) koppelt die Software von der weiteren Entwicklung ab. Es können keine neuen Komponenten und nicht einmal die Standardkomponenten des gewählten Zielsystems eingesetzt werden.

Oft bieten die spezialisierten Elemente zudem eine sehr eingeschränkte Funktionalität und verfügen über ein Zugriffsprotokoll, das sich nur schlecht mit dem Entwurfskonzept verknüpfen lässt.

Wenngleich hier noch Probleme bestehen, zeigen diese Arbeiten jedoch, dass die Generierung einer Benutzungsoberfläche aus Modellen prinzipiell auch ohne Berücksichtigung eines getrennten UIMS oder einer Zusatzbibliothek möglich ist.

3.5.5 Spezialisierungen für technische Systeme

Wie schon in Kapitel 2 ersichtlich ist, existieren viele Ansätze, die datenbankorientierte Anwendungssysteme unterstützen. Aufgrund der sehr homogenen Oberflächen und leicht umzusetzenden Strukturen ist eine Implementierung dort mit datenzentrierten Modellen weitgehend möglich.

In technischen Systemen, wie der in dieser Arbeit als Beispiel verwendeten Software „Experimental Target Configurator“ (ETC), genügen flache Datenstrukturen und Relationen in der Regeln nicht. Vielmehr sind hier exakt quantifizierende hierarchische Modelle notwendig, die zudem oft Abhängigkeiten im Ablauf wiedergeben müssen.

Der Grund liegt in der exakten Repräsentation einer technischen Umwelt, die nur in bestimmten Grenzen parametrisierbar ist. So dürfen nur elektronische Bausteine wiedergegeben werden, die auch tatsächlich vorhanden sind oder Werte erst dann geladen werden, nachdem andere Einstellungen gesendet wurden.

Die schon durch die Benutzerzentrierung begründete Ablauforientierung wird durch die Abhängigkeiten in technischen Systemen nahezu unumgänglich.

Der als Konfigurationsprogramm ausgelegte „Experimental Target Configurator“ soll Einstellungen von Parametern bei der Experimental-Hardware gesteuert von Software vornehmen. Die hierzu notwendige Vielzahl an Einstellungen und Eingaben lässt eine frühzeitige, am Benutzer orientierte, Spezifikation der Vorgehensweise als notwendig erscheinen.

Der Hauptunterschied der technischen Systeme zu datenbankorientierten und betrieblichen Anwendungssystemen liegt also in der geringen Datensatzorientierung. Es werden nicht hauptsächlich Datensätze ausgefüllt sondern viele Einzeleinstellungen getätigt.

Empfehlungen zur Interaktionsgestaltung müssen immer den Anwendungskontext berücksichtigen. Deshalb sind die in dieser Arbeit gegebene Empfehlungen nicht als allgemeingültig zu verstehen. Für den gelegentlichen Heimanwendungsbereich oder manche Web-Seiten sind ästhetische Gesichtspunkte oft wichtiger als eine einfache Bedienung.

Dagegen spielt für die komplexen technischen Systeme, ebenso wie für die betrieblichen Anwendungssysteme [Kruschinski 1999], die Möglichkeit zur effektiven und effizienten Aufgabenbearbeitung eine entscheidende Rolle.

3.5.6 Anforderungen

Aus dem technischen Anwendungsumfeld ergibt sich eine Reihe allgemeiner und spezieller Anforderungen an die zu entwickelnde Methode. Forderungen, die in der Analysephase der Diplomarbeit aufkamen, waren folgende:

- Weitgehende Interaktionsorientierung bei gleichzeitiger Integrationsmöglichkeit für Daten
- Möglichst weit gehende Plattformunabhängigkeit
- Frühzeitige Sicherstellung der Einhaltung ergonomischer Regeln
- Unterstützung entsprechender Modelle bzw. Formalismen
- Erzeugung von Native-Code (C++) als Ergebnis
- Unterstützung mehrsprachiger Entwicklungen
- Umfangreiche und zugleich unkomplizierte Beschreibungssprache
- Zwischenrepräsentation der Dialogstruktur, die eine Nachbearbeitung ermöglicht
- Templatefähige Endstufe, austauschbare Applikationstemplates
- Integration aller Spezifikationsdetails in einem Modell möglich
- Zugleich auch reine Interaktionsspezifikation möglich
- Beschreibungstexte für die Generierung einer integrierten Hilfe, von Tooltips etc.
- Generierung von Software, Testfällen, Hilfe etc. ohne Zusatzinformationen aus dem Modell
- Oberflächenprototypen mit Interaktionsfähigkeit sollen User-Centered-Design ermöglichen
- Ergonomie für Abläufe nach ISO 9241 und Styleguide (z. B. „Windowskonformität“) über entsprechende Tools möglich

Die Probleme, die existierende Methoden mit diesen Anforderungen haben, gehen aus den Methodenbeschreibungen im Grundlagen-Kapitel hervor.

Ziel ist also die Entwicklung einer Methode, die diese Unzulänglichkeiten nicht oder zumindest in wesentlich geringerem Umfang aufweist.

4 Eine Modellierungssprache zur Interaktionsbeschreibung

Der erste Schritt zur Erstellung eines entsprechenden Ansatzes ist der Entwurf eines Metamodells zur Interaktionsbeschreibung. Mit Hilfe dieses Metamodells soll es ermöglicht werden, ein Modell der Anwendung zu erstellen, das eine möglichst vollständige Beschreibung der Interaktionsschritte beinhaltet.

Gleichzeitig soll das Metamodell sicherstellen, dass schon bei der Erstellung des Modells softwareergonomische Grundsätze, beispielsweise in Form einer Standardisierung, einfließen.

Im Folgenden wird deshalb eine XML-basierte Sprache entworfen, die den genannten Anforderungen in möglichst hohem Grad gerecht werden soll. Die Struktur ist dabei baumartig, orientiert sich aber am Ablauf der modellierten Interaktionen.

4.1 Konzept der Interaction-Modelling-Language (IML)

Die Interaction-Modelling-Language (IML) verfolgt das Ziel einer für Benutzer und Kunden verständlichen und zudem strukturierten und formalisierten Darstellung der Interaktionsspezifikation.

Die IML soll die Tauglichkeit des Modells zur transformationellen und generativen Weiterverarbeitung sicherstellen. Das Modell soll zudem als Diskussionsgrundlage für Gespräche mit Benutzern dienen. Aus diesem Grund findet eine starke Anlehnung an das Konzept der Use-Cases statt, das sich in der Praxis auf diesem Gebiet bereits bewährt hat.

Die IML ist eine auf XML basierende Beschreibungssprache, die in dieser Diplomarbeit neu entwickelt wurde. Mit ihr ist es möglich, Use-Cases zu formalisieren, diese in einzelne Interaktionsblöcke aufzuteilen und mit Hilfe weiterer Konstrukte sogar einzelne Interaktionsschritte weiter zu detaillieren.

Das formalisierte Modell eignet sich zudem zur Generierung unterschiedlicher Artefakte.

4.1.1 Baumstruktur und Notation der IML

Schon an der Grobstruktur eines IML-Modells lässt sich die einem XML-basierten Modell zueigene Baumstruktur erkennen. Die Baumstruktur wird in der Notation durch Verschachtelung der Tags erzeugt. Dabei sind Child-Elemente immer zwischen öffnendem (<XYZ>) und schließendem (</XYZ>) Wurzel- bzw. Eltern-Tag einbeschrieben

Für die IML gelten zusätzlich folgende Notations-Vorschriften:

- Tags werden komplett in Großbuchstaben und mit trennenden Unterstrichen („_“) geschrieben: <TEST_TAG>
- Attributnamen werden in gemischter Schreibweise, groß am Anfang jedes Teilworts und zusammen, d. h. ohne Unterstriche geschrieben: TestAttribute
- Einträge aus Aufzählungstypen werden komplett klein und mit Unterstrichen getrennt geschrieben: (test_enum_1 | test_enum_2 | other_test_enum)
- Tags ohne Child-Elemente erhalten in der DTD immer ein #PCDATA

4.1.2 Dreiteilung

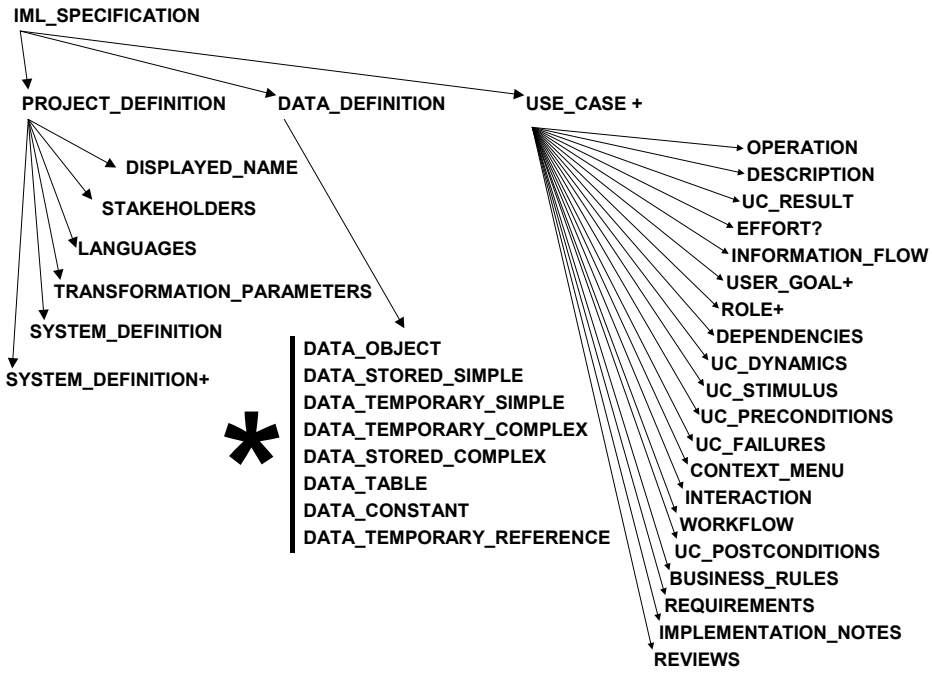

Abbildung 8: Überblick über das IML-Metamodell

Die vollständige Struktur des IML-Metamodells ist für eine Gesamtabbildung zu umfangreich (siehe IML-DTD im Anhang E).

Wie im Schaubild sichtbar, unterteilt sich ein IML-Modell in drei Teile:

- Einen **Projekt**teil, der alle für das Gesamtprojekt verfügbaren Informationen und eine Unterteilung in Systeme enthält.
- Eine **Daten**beschreibung, die einen leichteren und gemeinsamen Zugriff aus den Interaktionsabläufen der Use-Cases heraus auf Datenstrukturen und -elemente ermöglicht.
- Die **Use-Cases**, die sowohl die Ablauf- und Interaktionsbeschreibungen als auch weitere Modellinformationen enthalten.

Im Folgenden werden diese drei Teile näher beschrieben.

4.2 Das IML-Projektmodell

Ein komplettes IML-Modell spiegelt ein Entwicklungsprojekt wider. Zu diesem Zweck könnten prinzipiell alle für das Entwicklungsprojekt relevanten Informationen im IML-Modell gespeichert werden. Allerdings ist dies aus Effizienzgründen nicht sinnvoll, speziell dann, wenn das IML-Modell hauptsächlich zur Interaktionsmodellierung dient.

Aus diesem Grund ist eine Erweiterung mit Hilfe von fakultativen Angaben oder über eine Änderung des IML-Metamodells, möglich. Im existierenden IML-Metamodell sind deshalb hauptsächlich Definitionen enthalten, die für die Interaktionsgenerierung wichtig sind.

Der Projektname ist in zwei Formen vorhanden: In der ersten Form existiert er als eindeutiger, nur für den Entwickler sichtbarer Name (**ProjectName**), der das Projekt gegen andere abgrenzt und identifiziert, und deshalb bei der Generierung auch zur Konstruktion von Datei- und Verzeichnisnamen des Entwicklungsprojektes genutzt werden kann. Letztendlich in der Applikation sichtbar ist die zweite Form des Projektnamens vom Typ DISPLAYED_NAME, der unter dem gleichnamigen Tag gespeichert wird.

4.2.1 Sprachdefinitionen

Je nach Spracheinstellung kann dabei ein anderer Name verwendet werden, sofern für diese Einstellung eine passende sprachabhängige Definition des Namens existiert.

Dabei kann dieselbe Definition für alle Sprachen genutzt werden („all“) oder für eine Sprache bzw. eine Nation eine separate Namensdefinition erstellt werden. Motivation und Umsetzung der mehrsprachigen Modellierung wird in Anhang B („NLS – National Language Support“) näher erläutert. Die bei der Generierung verwendete Zielsprache bzw. -nation (ActualGenerationLanguage) wird, ebenso wie Versionsinformationen, im Projektmodell bestimmt.

Unterstützung von Mehrsprachigkeit macht vor allem dann Sinn, wenn sichergestellt werden kann, dass für alle Zielsprachen schon frühzeitig vollständige Übersetzungen existieren, die den Kontext berücksichtigen. Die Übersetzung des englischen „no“ ist beispielsweise ohne Kontextinformation nicht eindeutig möglich (nein, kein, keine, keiner, keines, etc.).

Um die Texte für verschiedene Sprachen auf Vollständigkeit testen zu können, ist ein Sprachdefinitionsblock (LANGUAGES) in der Projektdefinition vorgesehen, in dem alle gewünschten Zielsprachen definiert werden können.

Das oben schon erwähnte Sprach-Schlüsselwort „all“ gehört zu einem Mechanismus, der es ermöglicht, mehrsprachige Anwendungen effizient zu gestalten: Kann eine Zeichenfolge für mehrere Sprachen verwendet werden, zeigt „all“ als Sprachidentifikation (LanguageID) das an. Weitere mögliche Kürzel sind „en“ für englisch und beispielsweise „de“ für deutsch.

4.2.2 Interessengruppen

An einem Entwicklungsprojekt sind unterschiedliche Interessengruppen beteiligt. Um Anforderungen von Gruppen, wie beispielsweise den zukünftigen Benutzern, zu berücksichtigen, die häufig im Entwicklungsprozess zu wenig Beachtung finden, ist es wichtig, diese zu Projektbeginn als separate Gruppe aufzuführen und deren Anforderungen und Beiträge ebenfalls in das IML-Modell aufzunehmen.

Um diese Leistung des Requirements-Engineerings zu unterstützen, enthält jedes IML-Projekt eine Definition der beteiligten Interessengruppen (STAKEHOLDERS). Den hier definierten Interessengruppen können Anforderungen und andere Teile der Use-Case-Definition zugeordnet werden. Die rollenbasierte Aufteilung ermöglicht zudem eine einfachere Überprüfung (siehe Anhang B).

4.2.3 Dialog-Layout-Parameter

Das Ziel eines Transformators bzw. Generators, der Interaktionsmodelle in Oberflächenprototypen übersetzen soll, ist eine möglichst geringe Interaktion während des Generiervorgangs. Nur so kann eine so hohe Iterationszahl erreicht werden, wie dies für eine auf Prototyping gestützte Einbeziehung von Benutzern in den Entwicklungsprozess nötig ist.

Ein IML-Modell weist eine weitgehende Plattformunabhängigkeit auf, die an dieser Stelle mit dem Ziel einer Transformator-Parametrisierung durchbrochen wird. Die Layout-Parameter könnten auch in projektspezifischen Initialisierungsdateien gespeichert werden, was allerdings dem Single-Source-Prinzip („generate everything from one model“) zuwider laufen würde, das darauf abzielt, alle Informationen möglichst in einem Modell anzulegen.

Zudem wäre es nicht sinnvoll, projektspezifische Parameter außerhalb des IML-Modells zu speichern, da die entsprechenden Dateien für jedes Projekt neu erstellt werden müssten.

Die im IML-Modell enthaltenen „Layoutparameter“ regeln die Darstellung, d. h. das Layout der erzeugten Interaktionsstrukturen. Die Parameter und Vorgaben des aktuellen IML-Metamodells beziehen sich auf die MS-Windows-Plattform ab MS-Windows 95, sind aber, abgesehen von einer Skalierung der Vorgabewerte, komplett auf andere fensterbasierte Umgebungen anwendbar.

Größere Änderungen sind hier nur nötig, wenn andere Zielsprachen oder andere Systeme, z. B. im Telematik- oder sprachgeführten Bereich generiert werden sollen.

4.2.4 Systeme

Oft beinhalten Software-Projektmodelle mehrere Einzelsysteme, wenn es sich beispielsweise um Client-Server-Systeme, Plug-in-Verbünde oder eine aus mehreren Komponenten bestehende Software handelt.

Das Konzept der Systeme wurde von der UML übernommen, da Systeme eine gute Strukturierungsmöglichkeit auf der Ebene einzeln zu implementierender Anwendungen bilden. Jedes System kann daher meist in eine eigene ausführbare Datei umgewandelt werden.

4.3 Das IML-Datenmodell

Nach der Projektdefinition stellt das Datenmodell (DATA_DEFINITION) die zweite Säule eines IML-Modells dar. Die Idee dabei ist, in IML sowohl eine Möglichkeit zur Modellierung von Abläufen anzubieten, als auch die ergänzende Modellierung von Datenstrukturen mit Hilfe des IML-Datenmodells zu ermöglichen.

Handelt es sich beispielsweise um ein betriebliches Anwendungssystem, können so zugrundeliegende Tabellen modelliert werden. Die Interaktionsbeschreibung kann dann auf das Modell der Datenstruktur Bezug nehmen und somit auch schon implementierte oder entworfene Datenhaltungsmodelle unterstützen und an die Oberfläche anbinden.

Eine Beschreibung der verfügbaren Typen und Ausprägungen findet sich in Anhang B.

Das so spezifizierte Datenmodell dient zur Anbindung der Benutzungsschnittstelle und als Grundlage für das Interaktionsmodell, beispielsweise bei Zugriffen auf Daten, die aus verschiedenen Use-Cases heraus erfolgen.

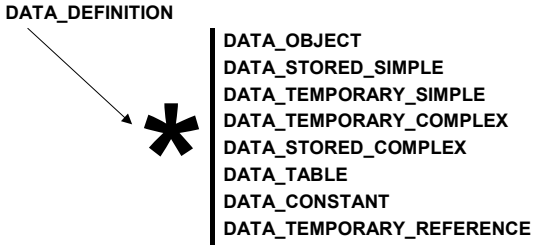

Abbildung 9: Möglichkeiten zur Datenbeschreibung

4.4 Das IML-Interaktionsmodell

Wie schon erwähnt, stellt das Interaktionsmodell die größte der drei Säulen eines IML-Modells dar. Sind das Projekt und wichtige Teile des Datenmodells spezifiziert, können die Use-Cases erzeugt werden, die in Ihrer Gesamtheit das IML-Interaktionsmodell bilden.

4.4.1 Use-Cases

Die Use-Cases (USE_CASE) in IML entsprechen in ihrer Bedeutung den schon aus der UML bekannten Use-Cases. Allerdings steht nach Abschluss der Modellierung wesentlich mehr Information zur Verfügung als in UML-Use-Cases, trotzdem können von diesen alle Informationen übernommen werden. Zudem ist es durch die Verwendung von XML möglich, weitere Relationstypen hinzuzufügen.

Werden Use-Cases zur Analyse und Spezifikation verwendet, stehen schon früh im Entwicklungsprojekt spezifizierte Abläufe zur Verfügung. Da Use-Cases eine sehr einfache und freie Beschreibungsform darstellen und in vielen modernen CASE-Werkzeugen integriert sind, werden diese gern genutzt – speziell auch als Grundlage für die Kommunikation mit Kunden und zukünftigen Anwendern.

Ein Vergleich der objektorientierten Modellierungsmethoden liefert zudem als Ergebnis, dass von allen Aufgabenbeschreibungstechniken[14] Use-Cases die einzigen sehr gut geeigneten sind. [Ziegler 1996].

Grund genug also, dass gute Basiskonzept der Use-Cases für einen erweiterten Einsatzbereich zu formalisieren.

Der formale Aufbau von Use-Cases: Die ursprüngliche Definition der Use-Cases sieht außer den im Use-Case-Diagramm widergespiegelten Angaben nur rein textuelle Informationen vor.

Vom Software-Engineering her gibt es daher Bestrebungen, Use-Cases stärker zu strukturieren. Dies geschieht vor allem durch eine Aufteilung in unterschiedlich überschriebene Teile, die bestimmte Informationen enthalten. Für die Interaction Modelling Language (IML) findet die im Folgenden beschriebene Aufteilung Verwendung, die sich an [IBM 2002b] anlehnt.

Der *Name* des Use-Cases steht an vorderster Stelle und entspricht der im Diagramm angezeigten Use-Case-Bezeichnung, die meistens eine Tätigkeit beschreibt.

Mit Hilfe der nur wenige Zeilen umfassenden (Kurz-)Beschreibung (*Description*) wird ein kurzer Überblick über das Use-Case insgesamt gegeben, während im nächsten Abschnitt nur das gewünschte Ergebnis (*desired outcome*) näher beschrieben wird, das der spätere Benutzer mit dem Use-Case erreichen möchte.

Während dabei eher Artefakte und konkrete Ergebnisse im Vordergrund stehen, werden im Abschnitt über die Ziele des Benutzers (*User goals*) mehr übergeordnete Ziele bzw. die Motivation zur Ausführung erläutert.

Die Beschreibung der Rollen (*Participants* / *Roles*) im Use-Case ermöglicht die Unterscheidung zwischen unterschiedlichen Rollen desselben Anwenders oder auch unterschiedlicher Benutzer, beispielsweise Administrator und Sachbearbeiter.

Die im Kapitel 4.6.2 beschriebenen Relationen und Abhängigkeiten (*Dependencies*) werden wie die Vorbedingungen (*Preconditions*) und Nachbedingungen (*Postconditions*) des Use-Cases ebenfalls gesondert aufgeführt.

Die Angabe bestimmter Szenarien (*Scenarios*) ist mangels Informationsgehalt für die Modellierung und die Interaktions-Generierung nicht im IML-Metamodell enthalten stattdessen werden die Interaktionsabläufe gesondert beschrieben.

Anforderungen (*Requirements*) und der umgebende Arbeitsablauf (*Workflow*) bzw. Prozess sind ebenso im IML-Modell enthalten wie bestimmte Regeln, die nicht durch Kundenwünsche oder direkte Anforderungen abgedeckt werden, sondern vielmehr die Spielregeln des Anwendungsbereichs (*Business rules*) wiederspiegeln, die Einfluss auf das modellierte Use-Case haben.

Oft werden schon während der Analyse und Spezifikation mit Hilfe von Use-Cases bestimmte Details klar, die sich auf die Implementierung auswirken. Zumeist gehen diese Informationen im Laufe des Entwicklungsprozesses teilweise verloren. Um dies zu verhindern, ist eine Eingabe diesbezüglicher Informationen schon im Use-Cases vorgesehen: Die Implementierungshinweise (*Implementation notes*).

Der Informationsgehalt von Use-Cases steigt mit dieser Formalisierung erheblich an. Zudem können die identifizierten Teile nun weiter untergliedert werden und Relationen zwischen diesen Teilen modelliert werden.

Das IML-Metamodell sieht für Use-Cases alle wichtigen Einträge vor und detailliert sie weiter, so dass jedes IML-Modell in jedem Use-Case über die entsprechenden Einträge verfügt.

[14] Verglichen wurden OMT: Szenarien/DFD, OBA: Szenarien, OOA: Methoden, OOA&D: O.-Flußdiagramme, OOSE: Use-Cases

4.4.2 Definition von InteractionCase

Ein InteractionCase (IC) ist eine weitgehend selbständige Interaktionssequenz, die Teil eines Use-Cases ist. Ein IC umfasst eine oder mehrere zusammengehörige Aktionen von System- oder Benutzerseite, die innerhalb desselben Use-Case liegen. Diese Aktionen stellen zusammen einen Vorgang dar, der nicht aufgespaltet werden sollte.

Der Reihenfolge der Aktionen kommt dabei eine semantische Bedeutung zu, die in der Dialogmodellierung verwendet werden kann.

Ein InteractionCase lässt sich oft als Bildschirmmaske, Dialogfenster, Wizard-Schritt oder ähnliche abgeschlossene Einheit interpretieren. Das Konstrukt InteractionCase wurde in dieser Diplomarbeit entwickelt.

Während ein Use-Case einen vollständigen „Anwendungsfall“ abdeckt, stellt ein InteractionCase also einen Bruchteil davon dar: Eine in einem Use-Case enthaltene Interaktionssequenz.

4.4.3 Das InteractionCase-Konzept

Jedes IML-Use-Case enthält eine Interaktionsbeschreibung (INTERACTION). Diese ist vergleichbar mit der textuellen Beschreibung in UML-Use-Cases, wurde jedoch weiter formalisiert.

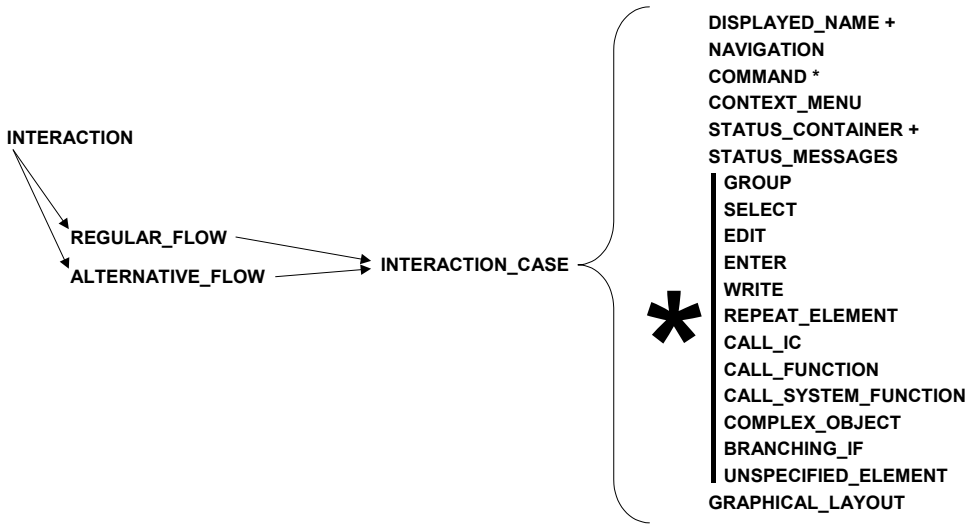

Abbildung 10: InteractionCases strukturieren Use-Cases

Schon in der informalen UML-Use-Case-Beschreibung existiert ein regulärer Ablauf, der durch alternative Abläufe ergänzt wird, wie sie durch Fehlerfälle oder unterschiedliche Selektionen hervorgerufen werden. Seinen Grund hat diese Verzweigung in der szenarienartigen Darstellung von Abläufen, die mangels Linearität auch alternative Abläufe beinhalten.

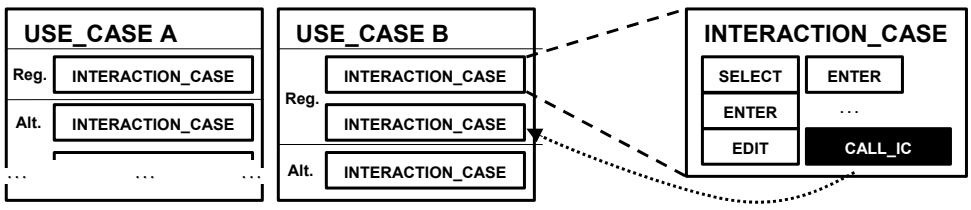

Abbildung 11: Reguläre und alternative Abläufe in IML

Auch innerhalb der Interaktionsbeschreibung in IML existiert deshalb eine Aufteilung in den regulären Ablauf (REGULAR_FLOW) und eine beliebige Anzahl von alternativen Abläufen (ALTERNATIVE_FLOW). Der reguläre Ablauf muss allerdings mindestens ein InteractionCase enthalten.

Jede eigenständige Interaktionssequenz wird in Form eines InteractionCase dem regulären oder einem alternativen Ablauf zugeordnet. Zu beachten ist dabei, dass gegenüber der natürlichsprachlichen, schrittweisen Beschreibung eine gröbere Strukturierung zum Zuge kommt: Nur vollständige InteractionCases können den einzelnen Abläufen zugeordnet werden.

Eine schrittweise Zuordnung ist damit nicht mehr möglich. Dies hat den Nachteil, dass kleine alternative Ablaufsequenzen wie Fehlermeldungen innerhalb des gerade aktiven Ablaufs beschrieben werden und nicht in einen eigenständigen alternativen Ablauf ausgelagert werden sollten.

Trotzdem ist eine derartige Modellierung in der Praxis klar im Vorteil gegenüber einer schrittweisen Modellierung: Zum einen ist auf diese Weise die Erstellung eines wesentlich kompakteren und durchgängigeren Modells möglich; zum anderen können die alternativen Schritte wahlweise trotzdem in ein eigenes InteractionCase ausgelagert, und so in einen alternativen Ablauf umgewandelt werden.

4.4.4 Warum InteractionCases?

Das Use-Case-Konzept hat sich in den letzten Jahren aufgrund seiner geringen Komplexität und einfachen Verständlichkeit auch für Nicht-Entwickler als Grundlage für die Spezifikation funktionaler Anforderungen durchgesetzt.

Ohne einen geeigneten Formalismus sind jedoch automatisiert kaum Informationen aus den Modellen zu gewinnen. Zudem erlaubt die uneingeschränkte Verwendung der natürlichen Sprache als Medium keine Weiterverarbeitung mit vertretbarem Aufwand.

Für den Einsatz zur Feinspezifikation und als Vorlage für Oberflächenprototypen ist die Granularität der Use-Cases zu grob.

Um einzelne Interaktionsblöcke separieren und diese wiederum zu Abläufen und Use-Cases aggregieren zu können, wurde das Konzept der InteractionCases entwickelt.

4.4.5 Wiederverwendung von InteractionCases

InteractionCases bilden weitgehend geschlossene Einheiten, so dass es möglich ist, diese sowohl innerhalb der Anwendung[15] oder des Projekts mehrfach zu verwenden, als auch in Form eines Templates unternehmensweit als Standardkomponente zu etablieren.

Ein einmal spezifiziertes InteractionCase kann so mit geringen Anpassungen innerhalb sehr unterschiedlicher Projekte verwendet werden und sorgt so für Erwartungskonformität [ISO 1998] durch standardisierte Interaktionssequenzen.

Das Konzept der Wiederverwendung hat sich mit den Klassenbibliotheken in der objektorientierten Programmierung etabliert. Im Bereich des Entwurfs kommt dem Entwurfsmuster [Gamma 1994] mittlerweile eine große Bedeutung zu. Gerade für die Verbesserung der Benutzbarkeit von Anwendungen ist eine Ausdehnung des Wiederverwendungskonzepts auf die frühen Phasen – vor allem die Spezifikationsphase – nötig.

Die Wiederverwendung einer natürlichsprachlichen Beschreibung[16] kann durch die Verwendung von Textbausteinen geschehen. Ein Erfolg kann nur dann erzielt werden, wenn diesen Bausteinen auch entsprechende Umsetzungen zugeordnet sind.

Dagegen bringt die Wiederverwendung von InteractionCases kaum Probleme durch nachgelagerte Abhängigkeiten mit sich. Sie liefert schon die Basis für eine zumindest prototypische Oberfläche, da nur vorgegebene Konstrukte für eine vollständige Ablaufbeschreibung verwendet werden müssen.

---

[15] über CALL_IC oder SELECTION_BY
[16] beispielsweise in Form einer MS-Word-Datei

4.4.6 Verzweigungen und explizite Abläufe

Abhängig von Eingaben während des Ablaufs eines InteractionCases kann sich der Ablauf verzweigen. Geschieht dies binär[17] und rekursiv[18], können die Verzweigungen sehr einfach im Ablauf angegeben werden. Wird abhängig von einer bestimmten Auswahl verzweigt, so muss die Verzweigung explizit als solche beschrieben werden. Dabei gibt es drei unterschiedliche Typen:

- Binäre Verzweigung: Ein zu einer Bedingung (BRANCHING_IF) gehörender Sub-Dialog-Aufruf wird direkt beim Element beschrieben.
- Verzweigung im Dialog: Soll innerhalb des Dialogs auf eine bestimmte Aktion des Benutzers hin (z. B. Drücken auf einen Kommando-Knopf) ein Sub-Dialog aufgerufen werden, so wird der Aufruf direkt im zugehörigen Interaktionsschritt beschrieben oder dort ein Link auf die Beschreibung eingetragen.
- Verzweigung am Dialog-Ende: Hier findet die Verzweigung des Dialogablaufs erst statt, wenn der Dialog beendet wurde. Es wird nicht ein Sub-Dialog aufgerufen, sondern ein gleichgestellter Folgedialog. Alternativ besteht auch die Möglichkeit keinen Folge-Dialog aufzurufen. In diesem Fall ist der Dialogablauf beendet. Vor der Weiterführung des Dialogablaufs besteht außerdem die Möglichkeit zu einem Funktionsaufruf, so dass in den Businessobjekten oder anderen, unterhalb der GUI liegenden, Anwendungsteilen Aktionen durchgeführt werden können, bevor der Dialog weiterläuft.

4.5 Aufbau von InteractionCases in der IML

Innerhalb der IML stellen die InteractionCases ein zentrales Konzept dar. Während Use-Cases für eine übersichtliche Darstellung meist zu groß sind, enthalten InteractionCases in der Regel nur so viele Elemente, dass diese sich zusammen auf einem Bildschirm darstellen lassen.

Auf dieser Granularitätsebene ist es dem Entwickler möglich, das Verhalten der Anwendung sehr detailliert zu beschreiben, so dass nicht nur generelle Ablaufmöglichkeiten, sondern auch weitergehende Spezifikationen wie Anzeigeelemente für Nachrichten (STATUS_CONTAINER), verfügbare Befehle (COMMAND) oder die Definition eines Kontextmenüs (CONTEXT_MENU) möglich sind.

Eine Besonderheit stellt die Beschreibung der Navigation bzw. der Navigationsknöpfe (NAVIGATION) dar, die für jedes InteractionCase angegeben werden muss. Vom einfachen OK-Cancel bis hin zur Datensatznavigation sind Navigationsmuster (NAVIGATION_PATTERN) einstellbar (siehe Kapitel 5.3.2).

Durch die Ablauforientierung der InteractionCases sind die Beschreibungen für Interaktionselemente implizit enthalten. Eine Auswahl-Aktion als Schritt innerhalb des Ablaufs hat beispielsweise immer die Generierung eines oder mehrerer Auswahl-Interaktionselemente zur Folge.

Hinweise zur Implementierungsneutralität von InteractionCases: Die Beispiele, die sich an den Benutzungsoberflächencharakteristiken existierender Betriebssysteme orientieren, legen den Schluss nahe, dass ein InteractionCase genau beschreibt, wie die zu generierende Benutzungsschnittstelle auszusehen hat. Dies ist allerdings nur in wenigen Punkten der Fall.

Ziel des IML-Metamodells ist eine möglichst große Plattformunabhängigkeit. Es soll beschrieben werden, was geschieht, nicht wie dies geschieht. Entwurfs- und Implementierungsdetails sind damit von nachrangiger Bedeutung.

Allerdings lassen sich, mit dem Ziel einer direkten Generierung aus dem IML-Modell ohne zusätzliche Informationen, nicht alle Bestandteile eines IML-Modells plattformneutral gestalten. Aufgrund der Heterogenität heutiger Benutzungsschnittstellen ist eine parameterlose Generierung aus rein abstrakten Modellen nicht möglich.[19]

---

[17] d. h. es gibt nur eine Verzweigungsmöglichkeit. Es wird entweder verzweigt oder fortgefahren

[18] d. h. es wird ein Sub-Dialog aufgerufen, während der aufrufende geöffnet bleibt. Der Dialogablauf kehrt daher nach den folgenden Sub-Dialogen automatisch wieder zum aufrufenden Dialog zurück

[19] Die aktuelle IML-Version enthält beispielsweise Transformationsparameter, die Abstände plattformspezifisch Regeln.

4.5.1 Gruppen und Elementreihenfolge

Dialoge unterschiedlicher Komplexität können oft nicht mit denselben Mechanismen strukturiert werden. Während bei Dialogen mit geringer Komplexität oft InteractionCases als Strukturierungsmechanismus ausreichen, müssen Dialoge hoher Komplexität oft mehrstufig unterteilt und zum Teil auch auf mehrere Seiten aufgeteilt werden.

Ähnlich wie bei der in [Pinheiro 2000a] beschriebenen MB-IDE Teallach wird auch bei den InteractionCases zwischen kompletten Interaktionssequenzen und Gruppen unterschieden. InteractionCases entsprechen meist Top-Level-Fenstern[20], Second-Level-Fenstern oder abgeschlossenen Elementen wie „Registern".

InteractionCases können wiederum über eingebettete InteractionCases verfügen, wie es beispielsweise mehrere variable Unterfenster in einem Hauptfenster erfordern können.

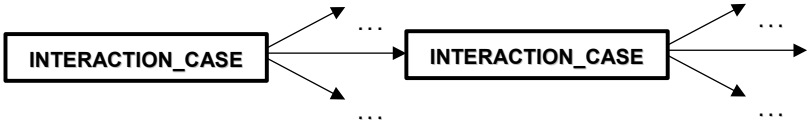

Abbildung 12: InteractionCases können ihrerseits wiederum (Sub-)InteractionCases enthalten

Den InteractionCases untergeordnet existiert deshalb ein weiteres Gruppierungselement: Die Gruppe (GROUP). Alle Typen von IML-Gruppen dienen der Gruppierung von Interaktionselementen, wenngleich diese Gruppierung im generierten Endergebnis nicht zwangsläufig sichtbar sein muss.

Gruppen stellen meist Unterfenster oder Element-Gruppierungen innerhalb eines Dialogfensters dar.[21]

```
<!ELEMENT GROUP (ELEMENT_SPECIFICATION, DISPLAYED_NAME*, (((SELECT | EDIT |
ENTER | GROUP | WRITE | COMPLEX_OBJECT | ABSTRACT_OBJECT)*) |
PROTOTYPE_GROUP | TABLE_GROUP))>
```

Abbildung 13: Definition des Gruppenelements in der IML-DTD

Zudem enthalten Gruppen ebenso wie InteractionCases Interaktionselemente und teilweise auch weitere (Unter-)Gruppen.

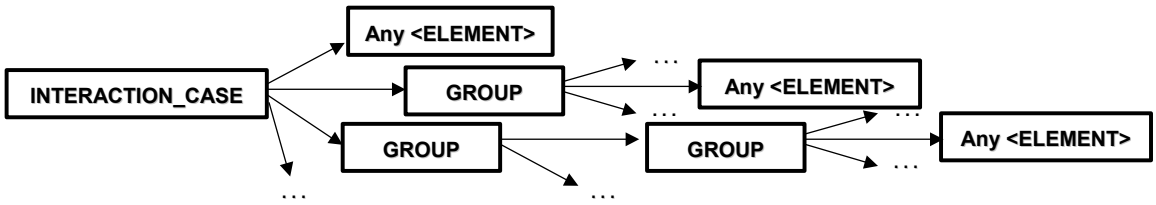

Abbildung 14: Aggregation von Gruppen und Elementen in InteractionCases und Gruppen

Elemente von Gruppen haben je nach Definition der Gruppe eine unterschiedlich starke Bindung aneinander. Zudem hängt ihr Erscheinungsbild wesentlich von den Einstellungen der Gruppe ab, der sie zugeordnet sind. Welche Auswirkungen die Gruppierung auf den Anordnungsprozess hat, regeln die drei Attribute Type, Order und Appearance.

Oft muss der Benutzer eine bestimmte Gruppe von Elementen – bezogen auf den Ablauf – vor anderen Elementen bearbeiten. Innerhalb der Gruppe kann aber weitgehende Reihenfolgenunabhängigkeit

[20] vergleichbar mit den „FreeContainers" in Teallach bei [Pinheiro 2000a]
[21] vergleichbar mit den „Containers" aus in Teallach bei [Pinheiro 2000a]

vorherrschen. Um die Vielzahl dieser Möglichkeiten abzudecken, teilt das Gruppen-Attribut Order dem Transformator mit, wie stark die Reihenfolge innerhalb der Gruppe einzuhalten ist.

Werden die Elemente, wie bei den meisten Type/Order-Kombinationen, innerhalb eines gemeinsamen Blocks dargestellt, kann über das Appearance-Tag entschieden werden, ob und wie die Gruppierung visualisiert wird, beispielsweise durch einen Rahmen (siehe Anhang B: Reihenfolge und Erscheinungsbild von Gruppen).

4.5.2 Interaktionsschritte

Interaktionsschritte stellen innerhalb der InteractionCases die atomaren Bausteine für Interaktionen dar. Werden Daten angezeigt[22], eingegeben, verändert, ausgewählt oder Befehle gegeben – immer ist ein Interaktionselement beteiligt.

Für die Strukturierung der Elementtypen gibt es unterschiedliche Ansätze. Diese reichen von einer Spezifikation jedes möglichen Interaktionselements als separaten Typ bis hin zu einem einzigen Interaktionselement-Typ, der über Attribute genauer spezifiziert wird.

Metamodell mit vielen Elementtypen: Für die Generierung von Oberflächen ist es prinzipiell einfacher jedes Interaktionselement als separaten Typ zu spezifizieren. Zum einen fällt die 1:1-Umsetzung algorithmisch wesentlich leichter, zum anderen kann der Entwickler genau vorgeben, was im gerade aktuellen Anwendungskontext sinnvoll ist – eine aufwändige Herleitung im Generator wird somit unnötig.

Schwerwiegender sind allerdings die Gründe, die gegen eine derartigen Ansatz sprechen. Die hohe Anzahl Typen macht ein Erlernen und Verstehen der einzelnen Typen nahezu unmöglich. Zudem ändert sich das gesamte Modell mit jeder Neuerung auf Seiten des Ziel-Betriebssytems, was eine weitgehende Plattformunabhängigkeit von vorneherein ausschließt.

Metamodell mit nur einem Elementtyp und interner Typisierung: Die Spezifikation nur eines Interaktionselements mit interner Typisierung hat für alle iterativen Prozesse, wie beispielsweise die Layoutoptimierung, große Vorteile, da eine sehr homogene Objektstruktur zur Verfügung steht. Zudem können Angaben über die Umsetzung in fertige Oberflächenelemente so sukzessive hinzugefügt werden.

Aus diesem Grund wurde für die Zwischenrepräsentation im Bereich Layout-Gestaltung ein homogenes Modell mit nur einem Interaktionselement-Typ gewählt (siehe Dialog Layer Language (DiLL) in Kapitel 5).

Während also auf Layout-Seite die Vorteile überwiegen, ist für den Analysten bzw. den Entwickler, der die Spezifikation erstellt, ein derart homogenes Modell zu unübersichtlich. Speziell sind Datenflüsse vom System und zum System kaum sichtbar.

Für das IML-Metamodell fand daher eine Mischform Verwendung. Die Interaktionsschritte zur Ein- und Ausgabe sind dabei im IML-Metamodell in vier Typen unterteilt:

- Auswahlelemente (SELECT)
- Eingabeelemente (ENTER)
- Bearbeitungselemente (EDIT)
- Ausgabeelemente (WRITE)

Navigationselemente, Kontext-Menüs, komplexe bzw. entwicklerdefinierte Objekte und andere ergänzende Interaktionsbeschreibungen sind zusätzlich verfügbar.

Zudem können abstrakte Objekte (ABSTRACT_OBJECT) definiert werden, die in den späteren Entwicklungsphasen Entwurf und Implementierung außerhalb des Generierungsprozesses mit Leben gefüllt werden können. Alle Elemente, die konkrete Werte zurückliefern, verfügen über eine Definition

[22] Reine Ausgabeelemente sind hier eingeschlossen, auch wenn dies beispielsweise bei kommandozeilenbasierten System fragwürdig ist.

des Eingabetyps, der auch als Rückgabetyp genutzt wird. Die einzelnen Typen sind in Anhang B beschrieben.

4.5.2.1 Eingabe (ENTER)

Eine grundlegende Technik zur Akquisition von Informationen durch Anwendungsprogramme sind direkte Eingaben des Benutzers. In diesem Fall sind hier nicht beliebige, vom Benutzer ausgehende, Interaktionstechniken gemeint. Vielmehr bezieht sich „Eingabe“ auf die Eingabe einer Zeichenkette.

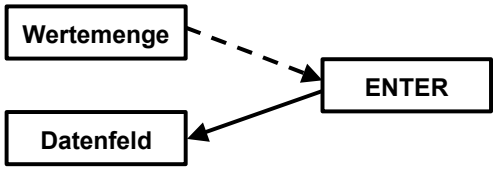

Abbildung 15: Datenfluss bei ENTER

Wie die Eingabe durch Interaktion mit dem Benutzer realisiert wird ist in diesem Fall noch nicht von Belang. Beispielsweise könnte die Eingabe auch über eine eingeblendete Tastatur auf einem Touch-Screen[23] erfolgen, was syntaktisch einer Selektionsfolge über Tasten gleichkommen würde. Trotzdem ist das Resultat eine neu geschaffene Information, die in dieser Form noch nicht im Rechner zur Verfügung stand und stellt damit eine Eingabe dar.

Standardvorgaben: Aufgeweicht wird dieses Prinzip nur durch die mögliche Vorgabe von Standardwerten oder früheren Eingabewerten. Die Vorgabe wird dabei über das Tag DEFAULT_VALUE abgewickelt.

Dabei stehen unterschiedliche Möglichkeiten zur Bestimmung des Vorgabewerttyps (Type) von der Konstanten bis hin zur History oder vordefinierbaren Liste zur Verfügung. Zudem kann ein Wert direkt vorgegeben oder nur abrufbar sein.

Überlegungen bezüglich der Zuweisung von Änderungsmöglichkeiten bzw. Änderungskompetenzen sowie zur automatisierten Generierung entsprechender Konfigurationsdialoge enthält Kapitel 5.8.2.

Überprüfungen: Vorgaben müssen – sofern überhaupt vorhanden – nicht zwangsläufig übernommen werden. Dem Benutzer wird eine weitgehend freie Eingabe gestattet. Allerdings ist es für den Benutzer extrem wichtig, schon frühzeitig auf fehlerhafte Eingaben hingewiesen zu werden, oder besser nicht erst in die Lage versetzt zu werden, solche Eingaben tätigen zu können.

Von Seiten der IML existiert deshalb der DOMAIN_CHECK-Mechanismus, der helfen soll, diese Sicherheit schon weitgehend im Modell, d. h. möglichst generativ, sicherzustellen. Dabei soll sowohl die Überprüfung als auch die Fehlerbehandlung möglichst weitgehend abgedeckt werden.

Über das Sub-Tag DOMAIN_CHARACTERS wird deshalb fakultativ zunächst ein Alphabet zur Verfügung gestellt, aus dem die Eingabe bestehen muss.[24] Eine Nachricht wie „Zeichen nicht erlaubt!“ kann dann auf Wunsch vom Generator beispielsweise direkt in die Statuszeile geschrieben werden.

Um auch komplexere Einschränkungen formulieren zu können, gibt es die Möglichkeit, Bedingungen (DOMAIN_CONDITION) zu formulieren. Werden diese Bedingungen nicht eingehalten, können die im übergeordneten DOMAIN_CHECK definierten Aktionen ausgeführt werden. Zudem verfügt jede Bedingung auch über eine entsprechende Fehlermeldung für den Fall der Nicht-Einhaltung.

---

[23] berührungsempfindlicher Bildschirm, der durch Druck auf eine beliebige Stelle einen „Mausklick“ an dieser Position auslöst

[24] Der zu generierende Mechanismus ist dabei relativ einfach (hier am Beispiel von C++ und Strings als Eingabetyp). Wurde das Alphabet einem CString zugewiesen, muss nur noch das zugehörige Event abgefangen werden, das einen Tastendruck oder eine Eingabe kennzeichnet. Dann kann für jedes neu eingegebene Zeichen eine Suche im Alphabet ausgeführt werden, die bei Nicht-Auffinden zum Ignorieren des Zeichens führt.

```
CString AlphabetString = „ABCDEFG“;
If (AlphabetString.Find(0, NewChar) < 0)
  //delete Input <fixme:>
```

Für die Angabe der Bedingung ist bisher keine Syntax definiert. Eine im Modell angegebene Bedingung kann deshalb vom Transformator nur zusammen mit dem bekannten Elementnamen in den Bedingungsteil einer If-Abfrage innerhalb des generierten Programmcodes übertragen werden.

Mit Hilfe von DOMAIN_RANGE kann zudem für Zahlenwerte ein Definitionsbereich angegeben werden. Wird dieser unter- oder überschritten, steht auch hier eine entsprechende Fehlermeldung zur Verfügung.[25]

Um sicherzustellen, dass die Überprüfung zum richtigen Zeitpunkt erfolgt, kann der Zeitpunkt der Überprüfung angegeben werden (siehe Anhang B). Unterschiedliche Anzeigemöglichkeiten und Systemreaktionen ermöglichen eine differenzierte Hilfestellung und Korrektur (siehe Anhang B: Reaktionen auf fehlerhafte Eingaben im Dialog).

Definitionen in XML-Schemata: Auch innerhalb von XML-Schemata können Bedingungen für Werte spezifiziert werden. Es liegt jedoch nur das IML-Metamodell, als DTD oder XML-Schema vor. Ein IML-Modell existiert nur als XML-Datei, die einer DTD gehorcht. Zudem müssen die Bedingungen im generierten Quellcode eingefügt werden, da sie erst zur Laufzeit zu überprüfen sind.

Eine Definition im XML-Schema müsste also ebenfalls umgesetzt werden. Um die Erzeugung mehrerer Dateien und ein unterschiedliche Modellierungsweise zu vermeiden werden die Bedingungen in der genannten Form in IML-Modelle integriert.

Die hier nicht beschriebenen, mit EDIT und SELECT gemeinsamen genutzten Tags und Konzepte innerhalb von ENTER, werden in Anhang B („ENTER, EDIT und SELECT: Gemeinsame Definitionen“) näher erläutert.

4.5.2.2 Bearbeitungsvorgang (EDIT)

Oft sind schon Daten vorhanden, die vom Benutzer bei Bedarf geändert werden sollen. Beispielsweise muss der Inhalt eines Dateinamenfeldes änderbar sein, d. h. es existiert schon ein gültiger Wert, der geändert wird. Ein Editierfeld (EDIT) hat deshalb die Besonderheit, schon vorhandene Daten aus dem angegebenen Datenfeld bzw. der angegebenen Variable zu lesen, dem Benutzer zu Änderung anzubieten und auch wieder in der Variablen zu speichern.

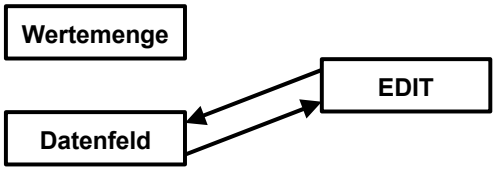

Abbildung 16: Datenfluss bei EDIT

Für den Benutzer ist der Vorgang soweit gekapselt, dass dabei der Eindruck entsteht, er nehme alle Änderungen direkt in dem betreffenden Feld vor. Deshalb muss sich das System auch entsprechend verhalten. Eine Ausnahme bildet der Commit-Mechanismus: Alle Änderungen werden erst übernommen, wenn ein entsprechender „Übernehmen“-Knopf gedrückt wird. Dieser Mechanismus kommt beispielsweise bei direkt auf einer Simulation oder Hardware arbeitenden Systemen und maskenbasierten Anwendungen zum Einsatz.

Ähnlich der Eingabe stellt auch das Editierfeld eine Reihe von Beschreibungsmechanismen zur Verfügung. Im Gegensatz zur Auswahl existiert bei Eingabe und Bearbeitung die Möglichkeit zur Eingabe beliebiger Werte. Während also bei der Auswahl die Überprüfung auf Korrektheit implizit geschieht, müssen für EDIT und ENTER Regeln festgelegt werden. Dies geschieht über die Angabe von Überprüfungen (DOMAIN_CHECKS) wie sie schon im Kapitel 4.5.2.1 für ENTER beschrieben wurden.

[25] Die Handhabung von Meldungen und damit auch Fehlermeldungen wird im Kapitel 4.5.2.4 beschrieben.

Wie auch bei ENTER wäre es hier möglich DOMAIN_CHECKs schon der bei IML-Vervollständigung passend zum Typ des Rückgabewerts zu definieren, z. B. {'1'..'9', '.'} für „IP". Eine solche Funktionalität kann auf Zeichenebene mit den Werten aus Anhang B (Eingabetypen) leicht implementiert und in die Vervollständigung als eigenständiger Schritt integriert werden (siehe Kapitel 4.8).

Voreinstellungen des Inhalts, wie z. B. Standardwerte, sind bei EDIT-Feldern nicht sinnvoll, da immer der aus der Datenquelle gelesene Wert gesetzt wird.

Die überwiegende Mehrzahl der EDIT-Felder in den angedachten Anwendungsgebieten sind Textfelder. Aus diesem Grund wurde in das IML-Modell eine Angabe über die Mehrzeiligkeit (MultiLine) aufgenommen, die sicherstellen soll, dass textuelle Bearbeitungsfelder in der korrekten Weise dargestellt werden (siehe Anhang B: Mehrzeilen-Einstellung bei EDIT).

Da das Mapping von IML-Elementen auf konkrete Oberflächenelemente erst in der Zwischenrepräsentation DiLL erfolgt und die Implementierung sogar erst im letzten Schritt festgelegt wird, ist es möglich an dieser Stelle sehr komplexe Felder zur Textbearbeitung einzusetzen. Allerdings ist dabei zu Bedenken, dass die Mehrzahl der Felder wie z. B. Namenseingaben nur sehr geringe Zusatzfunktionen benötigen und Features wie Rechtschreibprüfung hier kontraproduktiv sind.

Aus diesem Grund erstellt der Transformator hier Standard-MFC-Komponenten (CEditBox).

4.5.2.3 Auswahl (SELECT)

Die beiden bisher beschriebenen Elemente bezogen sich immer auf direkte Veränderungen (Eingaben und Bearbeitungen) durch den Benutzer. Das Element SELECT dient dagegen der Auswahl eines Wertes. Eine Eingabe freier Werte ist nicht möglich.

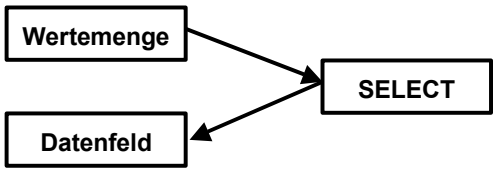

Abbildung 17: Datenfluss bei SELECT

Eine Ähnlichkeit zu ENTER tritt auf, wenn durch eine entsprechende Definition die Angabe von zusätzlichen Werten möglich wird, mit Hilfe derer die Liste verfügbarer Werte direkt erweitert werden kann.

Zur Spezifikation der auswählbaren Menge und der benutzerseitigen Listenerweiterungsmöglichkeiten stehen verschiedene Optionen zur Verfügung (siehe Anhang B: Mehrfachselektion und Listenerweiterung).

Wertelisten und Wertebäume: Neben den auch für Eingaben und Bearbeitungen vorhandenen Feldern stehen umfangreiche Funktionen zur Eingabe der Auswahl-Elemente zur Verfügung (siehe Anhang B : ENTER, EDIT und SELECT: Gemeinsame Definitionen).

Die Modellierung ist hier ungleich komplexer, da je nach Quantität und Struktur der Wertemenge sehr unterschiedliche Auswahlelemente erzeugt werden müssen und die Werte in einer formalen Struktur eingegeben werden.

```
<!ELEMENT VALUE_LIST ((VALUE_ITEM | VALUE_DATASOURCE | VALUE_SPAN |
VALUE_LIST_SAME_AS)*, SELECT_EXPANSION?)>
```

Abbildung 18: Definition der Werteliste in der IML-DTD

Der einfachste Fall ist, wie schon erwähnt, die 1:N-Auswahl. Hier genügt eine lineare Liste von Werten. Die allgemeine Selektion ist aber vielschichtiger. So kann eine hierarchische Auswahl, wie sie für

Bäume (Tree-Views) charakteristisch ist, nicht mehr mit linearen Mitteln repräsentiert werden. Zudem kann es notwendig sein, bestimmte Elemente generisch zu halten, um dem Benutzer die Möglichkeit des Hinzufügens bestimmter Elementtypen zu bestimmten (Teil-)Listen zu erhalten.

Das grundlegende Konzept für Wertemengen ist in IML die lineare Werteliste VALUE_LIST. Sie liegt jeder Auswahl zugrunde und aggregiert eines oder mehrere Elemente, die zusammen eine Hierarchieebene bilden.[26]

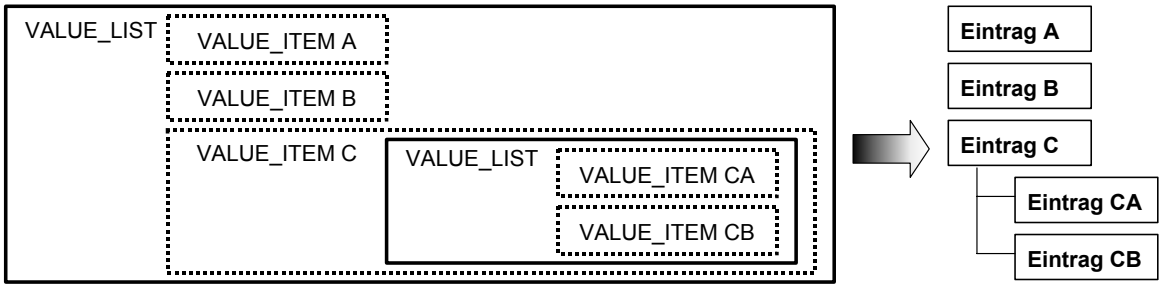

Abbildung 19: Struktur einer hierarchischen Werteliste

Werden Daten dynamisch geladen, oder sind Einträge durch den Benutzer einzufügen und zu entfernen, so ist es oft sinnvoll kapazitive Ober- und Untergrenzen für Wertelisten einzuführen, die sicherstellen, dass die im modellinhärente Logik nicht zerstört wird. Ein Beispiel hierfür ist die Ansteuerung einer CAN-Controller-Karte[27], die über eine bestimmte Anzahl Controller verfügt. Das Einfügen eines zusätzlichen Controllers brächte eine Verletzung des Weltmodells mit sich.

Definition einzelner Werte: Eine Werteliste kann mit Hilfe einzelner Einträge (VALUE_ITEM) gefüllt werden. Für wenige oder spezielle Einträge ist dies die sinnvollste Vorgehensweise.

Nicht immer müssen diese Elemente auswählbar sein. Gerade in baumartigen Auswahlfeldern können Kopfelemente (header) oder inaktive Kommentare (remark) eingestreut werden. Ist der Inhalt vom Benutzer zu bearbeiten, kann das Element mehrfach vorkommen (RepeatMin, RepeatMax, RepeatInit).

Um dem Benutzer bei komplexen Elementen Bearbeitungsmöglichkeiten zu bieten, können zu jedem Eintrag Reaktionen auf den Fokuserhalt und ein Kontextmenü definiert werden.

Der Rückgabewert kann sich vom angezeigten Wert unterscheiden. Dies ist wichtig, um sprechende Einträge[28] erzeugen zu können und dem Fachkonzept trotzdem effizient verarbeitbare Werte anzubieten (z. B. true / false).

Untergeordnete Listen: Zur hierarchischen Anordnung von Einträgen kann jeder Eintrag über eine oder mehrere untergeordnete Listen verfügen. Listen sind für den Benutzer nur in Form der enthaltenen Einträge sichtbar. Es handelt sich also um ein Aggregationshilfsmittel.

Datenquellen einbinden: Eine effizientere Vorgehensweise zur Befüllung einer Werteliste ist es, die Liste oder Teile der Liste aus einer oder mehreren Datenquellen zu laden (VALUE_DATASOURCE). Hierbei macht der Zeitpunkt der Einbindung einen erheblichen Unterschied aus, da er bestimmt, ob eine statische oder dynamische Variante zum Einsatz kommt (siehe Anhang B: Externe Wertelisten).

Wertebereiche: Sollen lineare Wertebereiche angegeben werden, die sich über ein Minimum, ein Maximum und eine Schrittgröße bestimmen lassen, kann dies mit VALUE_SPAN geschehen. Ist der Transformator in der Lage Formeln zu berechnen, können über eine angegebene Formel die auswählbaren Werte aus den linearen Basiswerten zwischen Minimum und Maximum berechnet werden. Die Anzahl der Werte ändert sich dadurch nicht: {1,2,3,4} —[n²]–> {1,4,9,16}

[26] Aufgrund der Seltenheit und Komplexität wurde das Konzept der VALUE_DIMENSION entfernt, da die hier mögliche mehrdimensionale Auswahl aus im n-dimensionalen Würfel beliebig gestreuten Daten zu komplex wäre, um für ein Generatorenkonzept sinnvoll zu sein. Eine direkte Implementierung liefert hier vermutlich die besseren Visualisierungskonzepte.
[27] CAN ist ein Bussystem-Standard, der in der Automobilindustrie zur fahrzeuginternen Kommunikation von Steuergeräten genutzt wird.
[28] z. B. bestehend aus mehreren Wörtern

Referenzen und Pool-Konzept: Soll eine an anderer Stelle schon definierte Liste genutzt werden, ist dies ebenfalls möglich (VALUE_LIST_SAME_AS). Hierzu sind keine weiteren Definitionen notwendig. Die Liste kann dabei kopiert werden oder als sogenannter „Pool“ Verwendung finden.

Die Entwicklung des Pool-Konstrukts geht auf die Notwendigkeit zurück, eine vorhandene Menge von Einträgen auf entsprechende Auswahlfelder zu verteilen. Wichtig dabei ist, dass ein in Feld1 ausgewählter Eintrag nicht mehr in der Auswahlmenge von Feld2 enthalten sein darf: Das Element wurde aus dem Pool herausgenommen und kann nur durch Selektion eines anderen Eintrags in Feld1 wieder in Feld2 zur Auswahl stehen.

Die genaue Wirkweise eines solchen Pools wird bei seiner Definition festgelegt (siehe Anhang B: Definition von Werte-Pools).

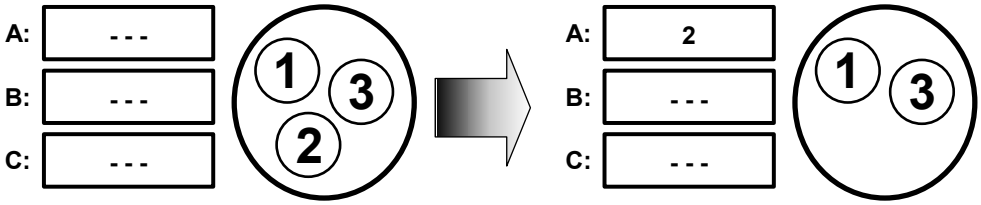

Abbildung 20: Das Poolkonzept. Im Beispiel enthält der Pool die drei Werte {1,2,3}, drei Felder greifen zu.

Mit einem SELECT werden meist lineare Listen von Elementen beschrieben, wie sie Listboxen, Drop-Down-Elementen oder Radiobutton-Gruppen entsprechen. Trotzdem ist auch die Modellierung kompletter applikationssteuernder hierarchischer Elemente möglich.

4.5.2.4 Texte und Fehlermeldungen

Nachrichten (WRITE) können im Dialogbereich eingeblendet werden, permanent verfügbar sein oder direkt in eine Datei geschrieben werden. Wichtig ist, dass der Generator anhand des Typs bereits erkennt, wie er den Text zu behandeln hat.

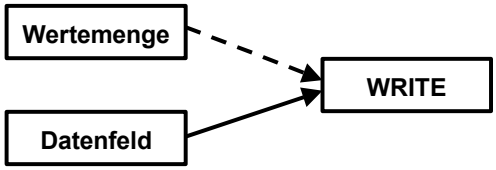

Abbildung 21: Datenfluss bei WRITE

Externes Ziel oder externe Quelle können bei Bedarf angegeben werden. Die standardisierte Zuweisung bestimmter Meldungsklassen ist ein grundlegendes Konzept von IML. Ausgaben werden nicht positioniert, sondern bestimmten Containern zugeordnet, die ihren Ausgabebereich kennen.

```
Type (information | tip | error | multiple_errors | critical_error | fa-
tal_error | warning | help | status_info | normal_text | text_element |
to_file | userdef) #REQUIRED
```

Abbildung 22: Meldungstypen der IML (aus der IML-DTD)

Ein großes Problem moderner Systeme ist die Flut schwer differenzierbarer Meldungen. Oft haben Meldungen einen falschen Text oder sind mit den falschen Piktogrammen oder Titeln ausgestattet, die dem Typ der Meldung nicht gerecht werden.

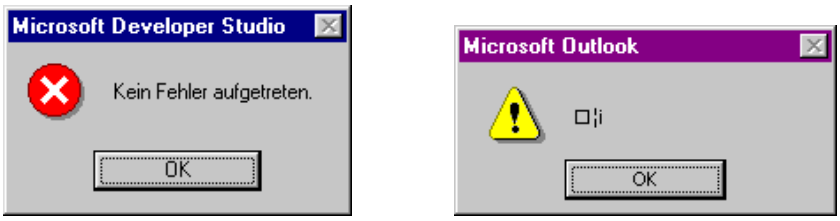

Abbildung 23: Beispiele für durch falsche Typisierung oder fehlerhafte Einbettung entstandene Meldungen (Quelle WWW.SWR3.DE)

Um derartige Problem schon bei der Modellierung auszuschließen und die Anwendungen so ergonomischer zu gestalten werden in der IML Meldungen nicht direkt als Dialoge modelliert, sondern der Gestaltungsspielraum bei den standardisierten Meldungstypen eingeschränkt.

Hierzu wird für jeden Meldungstyp ein „Container“ (STATUS_CONTAINER) definiert, der über einen frei definierbaren Namen identifiziert wird. Über einen Ansichtstyp (View) wird das Ziel der Ausgaben des Containers definiert. Verkleinerte Ansichten, Statuszeilen oder ganze Fenster sind möglich.

Um dem Entwickler die Möglichkeit zu geben, dem Benutzer eine Art Logbuch zur Verfügung zu stellen, können die Ausgaben temporär oder permanent mitgeschrieben werden. Das entstehende Logbuch kann in einer vom Entwickler festzulegenden Ansicht (ExtendedView) bis hin zum eigenständigen Editorfenster geöffnet werden.

Der Generator sollte bei zu versteckt gewählten Ansichten für wichtige Meldungen eine Warnung ausgeben, um unsichtbare Fehlermeldungen etc. zu vermeiden. Jeder Meldungstyp stellt eine eigene Kategorie dar.

Für jedes InteractionCase wird die Zuordnung der Nachrichten zu den Containern definiert (STATUS_MESSAGES). Jede Nachricht wird dann in dem ihrem Typ zugeordneten Containerbereich ausgegeben. Zusätzlich kann eine Liste der Short-Cuts angezeigt werden. Auch die mit ERROR_MESSAGE erzeugten Nachrichtentexte sind mehrsprachenfähig.

4.5.3 Backup-Mechanismus und Undo

Eine wichtige Möglichkeit zur Verbesserung der Bedienbarkeit ist ein Sicherheitskonzept, das Aktionen direkt oder zumindest indirekt rückgängig machen kann, vergleichbar einem Transaktionskonzept.

Als direktes Verfahren bietet sich Undo an, als indirektes ein permanent gespeichertes Backup. Eine Möglichkeit Datenverluste durch Systemversagen und auch manche Fälle von Fehlbedienungen abzufangen oder zumindest zu mildern ist ein Failsave-Mechanismus.

Der Benutzbarkeit und Wartbarkeit abträglich ist bei allen drei Verfahren gleichermaßen, der Einsatz vieler unterschiedlicher Methoden und eine stellenweise unterschiedliche Realisierung der einzelnen Methode.

Eine Lösung, die direkt aus Einstellungen im IML-Modell generiert werden kann, sorgt dagegen für eine optimale Integration in das Interaktionskonzept und für Homogenität in der Implementierung. Trotzdem lässt sie eine Adaption an die jeweilige Dialog-Situation zu. Dies ist wichtig, um bestimmte Funktionen fallweise zu- und abschalten zu können, sofern sich das als notwendig erweist.

So ist es beispielsweise manchmal unnötig bzw. im Fall des Passwortdialogs sogar gefährlich, eine Undo- oder Backup-Funktionalität zur Verfügung zu stellen. Eine solche Funktionalität würde in diesem Fall die Sicherheit des Systems gefährden, ohne einen Nutzen für den Benutzer zu erzielen.

Gruppen-Undo: In [Ressel 1995] wird kooperatives Arbeiten, speziell Gruppen-Undo, am Beispiel von Texteingaben gezeigt. Anhand der Transformationsregeln für Texteditoren ließen sich auch kooperative Anwendungen beschreiben, wenn ein entsprechender Mechanismus generiert werden kann.

Aus Sicht der Entwickler käme dabei nur ein weiterer Undo-Typ im IML-Metamodell hinzu. Für die Transformatorentwicklung stellt dieser Schritt allerdings ein weitaus größeres Problem dar, da die Mechanismen für jedes Feld generiert oder alternativ passende Eingabefeldklassen definiert werden müssen, die der Transformator im Anschluss nur noch verwenden muss.

Backup: Die Form des Backups wird sowohl für das gesamte InteractionCase[29] als auch individuell für die untergeordneten Eingabe- und Selektionselemente eingestellt. Die einzelnen Modi für Backup und Undo können Anhang B entnommen werden.

Die Häufigkeit bei einem zeitabhängigen Sicherungs-Mechanismus sollte im Code durch Verstellen einer vom Generator vorgesehenen Konstante durch den Entwickler regelbar sein. Noch besser wäre die Generierung eines passenden Einstellungsfelds im Options-Dialog.

Allerdings sollte der Übersichtlichkeit wegen eine Einstellung des Speicherungsintervalls mindestens auf System- oder sogar Use-Case-Ebene und nicht unterhalb dieser Grenze erfolgen.

4.6 Konsistenzmechanismen und Abhängigkeiten in IML

Für rechnergestützte Modellierungsmethoden ist eine Konsistenzprüfung unerlässlich. Um schon auf Ebene der Use-Cases starke Mechanismen zur Verfügung stellen zu können, existiert innerhalb der IML eine Vielzahl von Relationstypen. Auch die Systemabhängigkeiten auf höherer Ebene und implizite Abhängigkeiten zwischen Use-Cases gehören zum IML-Metamodell.

4.6.1 Abhängigkeiten auf Systemebene

Use-Case-Diagramme enthalten Systeme, die zumeist getrennte Anwendungsteile oder auch komplette Anwendungen modellieren. Über einen einfachen, eindeutigen Zugehörigkeits-Mechanismus wird sichergestellt, dass jedes Use-Case zu genau einem System gehört.

Eine Definition von weiteren Systemabhängigkeiten, wie vorgeschriebene Reihenfolge, ist denkbar und einfach zu realisieren, wurde mangels praktischer Relevanz aber nicht ins IML-Metamodell integriert.

4.6.2 Abhängigkeiten auf Use-Case-Ebene

Auf der Ebene der Use-Cases existiert ein detailliertes Abhängigkeitsmodell. Der Grund hierfür liegt in ihrer hohen Bedeutung für die Strukturierung einer Anwendung und der schlechten Überschaubarkeit von umfangreichen Use-Case-Diagrammen.

Während direkte Uses-Relationen (Use-Case A nutzt Use-Case B) schon im Ablauf leicht erkannt werden können, sind Vererbungen, Erweiterungen oder sogar Importe von Teilfunktionen nur sehr schwer erkennbar. Um eine dahingehende Modellierung schon früh zu unterstützen, sieht das IML-Metamodell eine Vielzahl von Relationskonzepten vor, die in Anhang B beschrieben sind.

Dabei gilt immer: Die in Use-Case UC-A eingetragene Abhängigkeit von Use-Case UC-B ist vom genannten Typ <XYZ>: UC-A <XYZ> UC-B.

4.6.3 Aktoren

Die schon in Kapitel 2.4.2 beschriebenen Aktoren in UML können in der IML ebenfalls modelliert werden. Dabei sind Aktoren immer einem Projekt zugeordnet, d. h. sie werden in der Projekt-Definition angelegt (siehe Kapitel 4.2).

Neben dem angezeigten Namen (DISPLAYED_NAME), einer eindeutigen Kennung (ActorID) und einer textuellen Beschreibung (TEXTUAL_DESCRIPTION) enthält die Definition eines Aktors die – im UML-Diagramm ebenfalls vorhandenen – Uses-Kanten.

[29] Quasi als übergreifende Einstellung für das InteractionCase

Wie in der UML auch, zeigen diese Kanten an, welche Rolle bzw. welcher Aktor, welche Interaktionsabläufe bzw. Use-Cases nutzt. Im Zuge der Generierung können diese Informationen dazu verwendet werden, jedem Anwender nur die Optionen anzuzeigen, die zu den über Uses-Kanten erreichbaren Use-Cases gehören.

Auf diese Weise ist es möglich, eine automatisierte rollenspezifische Anpassung der Anwendung zur Laufzeit vorzunehmen oder für jede Rolle eine eigenständige Applikation zu generieren.

Beide Vorgehensweisen stellen sicher, dass generativ ein möglichst hohes Maß an Aufgabenangemessenheit (vgl. [ISO 1998]) in der Menüsteuerung, d. h. dem Ausgangspunkt der Applikation, erreicht wird. Dem Benutzer stehen somit in der entsprechenden Rolle genau die Funktionen zur Verfügung, die er zur Bearbeitung benötigt.

Um einen direkten Zusammenhang mit der graphischen Repräsentation im Diagramm herzustellen, stehen sowohl bei den Aktoren als auch bei den Systemen 2D-Koordinaten-Attribute zur Verfügung.

4.6.4 Aufruf- und Abhängigkeitsrelationen auf InteractionCase-Ebene

Werden exakt gleiche Interaktionsabläufe mehrfach eingesetzt, ist es nicht sinnvoll, diese immer komplett neu zu beschreiben. Ähnlich den Uses-Kanten zwischen einzelnen Use-Cases in UML-Diagrammen, können auch InteractionCases Uses-Relationen aufbauen.

Während bei Use-Cases die Relationen sehr ausführlich und exakt angegeben werden können, wäre dies für InteractionCases zu aufwändig. Zudem enthalten InteractionCases Aufrufrelationen implizit durch die Angabe von InteractionCase-Aufrufen (<CALL_IC> / <SELECTION_BY>). Diese Informationen reichen aus, um eine Diagrammdarstellung auch für InteractionCases zu berechnen.

Durch die Einbettung in Use-Cases stehen zudem alle Modellierungsmöglichkeiten auf Übergeordneter Ebene zur Verfügung. Zudem können wichtige und weitgehend eigenständige InteractionCases in separate Use-Cases ausgelagert werden, so dass sich alle Relationen des Use-Case unmittelbar auf das enthaltene InteractionCase auswirken.

Werden InteractionCases über Use-Case-Grenzen hinweg aufgerufen, können sich allerdings Einstellungen aus dem Use-Case-Kontext widersprechen, da das genutzte InteractionCase in einem anderen Use-Case, das andere Relationen hat, zu finden ist.

Sinnvoll ist eine Einbettung in ein eigenes Use-Case deshalb nur, wenn ein InteractionCase mehrfach und aus verschiedenen Use-Cases heraus benutzt werden soll.

4.6.5 Ergonomie durch Relationsmodellierung?

Die oben beschriebenen Möglichkeiten zur Modellierung von Relationen und Abhängigkeiten sind zunächst für das Software-Engineering von großer Bedeutung. Bei der Entwicklung kann durch das rechtzeitige Erkennen von Relationen oft Aufwand eingespart werden. Erkannte Äquivalenzen spiegeln sich dann bei rechtzeitiger Berücksichtigung direkt in Entwurf und Implementierung wieder. Durch die Überprüfung von Abhängigkeiten können zudem Fehler vermieden werden.

Doch auch die Ergonomie wird entscheidend durch die Modellierung von Relationen beeinflusst. Wird der Entwurf direkt aus einer Liste von – zumeist funktionalen – Anforderungen erstellt, spiegelt sich zunächst die Gedankenwelt des Entwicklers im Konzept der Software wider.

Die Benutzungsschnittstelle entwickelt sich dann aus den von der Anwendung benötigten und den zurückgegebenen Informationen und verhält sich dann aus Sicht des Benutzers – mangels dessen Kenntnis der Systemarchitektur – meist völlig unerwartet.

Wurde jedoch das Ablaufmodell in der Domäne des Anwendungsgebiets mit seinen Relationen und Abhängigkeiten schon früh modelliert, folgt die Ablauflogik einem Schema, das dem Modell des Benutzers vom Anwendungsbereich B(A) konzeptuell wesentlich näher kommt. Damit wird eine kohä-

rente Ausbildung der Modelle des Benutzers vom System B(S(A)) und der Sicht des Systems S(S(A)) bzw. S(A) ermöglicht. [30]

Die Ablaufstrukturierung erfolgt dann mit Hilfe der vom Kunden bzw. der zukünftigen Benutzergruppe vorgegebenen Abläufe und ist für diese nachvollziehbar.

4.7 Fachkonzept-Operationen

Ein Großteil der rein funktionalen Anforderungen bezieht sich nicht auf die Benutzungsschnittstelle, sondern das Fachkonzept (siehe [Hofmann 1998]).

Beim Fachkonzept handelt es sich um diejenigen Teile, die nicht direkt zur Oberfläche gehören, sondern die eigentliche Funktionalität bereitstellen. Dabei kann es sich um Berechnungen, das Laden oder Speichern von Daten, den Aufbau eines Modells oder andere Anwendungsfunktionen handeln, die zusammen mit der Oberfläche und eingebundenen Bibliotheken die fertige Applikation bilden.

Ihrer nicht-benutzerinteraktiven Eigenschaften und Entwurfsnähe wegen, können Fachkonzept-Operationen nicht mit Hilfe der IML erstellt werden. Hierfür eignen sich OOD-Diagramme in UML oder andere entwurfs- und funktionsorientierte und Modellierungsmethoden.

Auch wenn keine inhaltliche Modellierung des Fachkonzepts in IML-Modell stattfindet, besteht doch eine enge Verbindung zum Fachkonzept: Ist ein Entwurf oder sogar eine Implementierung der Basisfunktionen wie Datenbankzugriff und Ablaufsteuerung oder höherer Applikationsbestandteile wie Businessobjekte vorhanden, müssen diese mit dem IML-Modell verbunden werden, um die Funktionalität in die Benutzungsschnittstelle zu integrieren.

Sind schon Methoden oder Funktionen vorhanden, können diese mit einem passenden Tag an die Beschreibung angekoppelt bzw. eingebunden werden. CALL_FUNCTION ist dabei in der Lage, anhand von Angaben über die Ausführungszeit (EXECUTION_TIME) eine Bestimmung der Zeit bis zum Timeout zu ermöglichen. Für längere Ausführungszeiten steht zudem ein Fortschrittsbalken (PROGRESSBAR) zur Verfügung.

Durch die Einstufung der „Gefährlichkeit“ (Danger) wird es dem Transformator möglich gemacht, Sicherheitsabfragen (CALL_QUESTION) einzufordern, wo eine hohe Gefährdung besteht. Zudem ist es so möglich Backups und andere Sicherungsfunktionen einzuplanen oder sogar zu generieren. Auf diese Weise soll der Benutzer vor unerwünschten Effekten bewahrt werden.

```
<!ELEMENT CALL_FUNCTION (CALL_PARAMETER*, EXECUTION_TIME, PROGRESSBAR?,
CALL_QUESTION?)>
```

Abbildung 24: Definition eines Funktionsaufrufs in der IML-DTD

Durch die Definition von benötigten Parametern (CALL_PARAMETER) ist es möglich, Werte aus dem IML-Modell (zumeist Eingaben des Benutzers bzw. Feldinhalte) direkt an die Funktion zu übergeben.

Die Aufrufsyntax kann vom Generator mit Hilfe einer Elementbeschreibung erzeugt und dort auch jederzeit geändert werden, so dass die übergebenen Parameter direkt in den Programmcode geschrieben werden können. Bei vereinbarter Methodensignatur kann also das Fachkonzept generativ an die Benutzungsschnittstelle angeschlossen werden.

Wichtig ist dabei die Regelung von Ein- und Ausgabe. Eingabeparameter müssen aus den passenden Feldern ausgelesen, evtl. transformiert und übergeben werden. Als Eingabeparameter kann das Modell alle enthaltenen und im Kontext auch verfügbaren Felder verwenden. Diese sind durch Angabe des Namens genau selektierbar.

[30] Hierbei handelt es sich um das mentale Modell des Benutzers vom Anwendungsbereich B(A), das dem Modell des Systems vom Anwendungsbereich S(A) ähnlich sein sollte. Sind die Strukturen verwandt, kann das Modell des Benutzers vom System B(S(A)) leicht (falls nötig) dem Modell des Systems von sich selbst S(S(A)) angepasst werden. Das System wird dann intuitiv bedienbar. Siehe [Herczeg 1994], Kapitel 2.2

Alternativ sollte noch ein Mechanismus zur Verfügung stehen, der es ermöglicht, Konstanten und extern zu belegende Variablen mit einzubinden. Dieser Mechanismus ist vor allem im Hinblick auf eine umfangreiche Codegenerierung wichtig.

Ausgabeparameter müssen von der Benutzungsoberfläche übernommen, evtl. transformiert und dann in die passenden Elemente geschrieben werden. Zusätzlich kann es nötig werden, bestimmte Änderungen an der Oberfläche durchzuführen. Werden hierzu nicht spezialisierte Klassen-Bibliotheken eingesetzt, muss ein Codefragment generiert werden, das die Statusänderungen vornimmt, beispielsweise in Form einer Farbänderung.

Je nach eingestellter Übergaberichtung (Direction: in, out, inOut) können Werte übergeben oder auch wieder an die Benutzungsschnittstelle zurückgegeben werden. Eine genaue Umsetzungsempfehlung für den Transformator kann mangels praktischer Erprobung an dieser Stelle nicht gegeben werden. Das IML-Metamodell kann allerdings leicht um zusätzlich benötigte Attribute erweitert werden.

Ein Konzept zur Verwendung interner, vom Generator bereitgestellter, Funktionen ist die Verwendung von CALL_SYSTEM_FUNCTION. Auch hier können Parameter und eine Sicherheitsabfrage modelliert werden. Im Unterschied zum normalen Funktionsaufruf stehen hier nur die im Attribut Function-Name auswählbaren, für den Transformator vordefinierten Funktionen zur Verfügung, die jedoch für jeden Transformator erweitert werden können.

4.8 IML Vervollständigung: IML complete

Die Vervollständigung des IML-Modells kann in nahezu beliebigem Umfang geschehen. Einfache Algorithmen überschreiben generische Inhalte mit den konkreten Werten oder ergänzen fehlende Attribute.

Allerdings ist es auch möglich, schon an dieser Stelle bestimmte Dialoge hinzuzufügen, Makros in konkrete Tags umzuwandeln oder bestimmte Teile herauszunehmen. Im Folgenden werden einige Funktionen beispielhaft erklärt.

Sprachüberprüfung: In der Projektdefinition werden bereits die zu unterstützenden Sprachen festgelegt. Anhand dieser Information können alle Knoten mit einem Spracheintrag darauf überprüft werden, ob im entsprechenden Block für jede Sprache eine Definition vorliegt.

Je nach Transformator-Einstellung werden anstelle einer konkreten Sprache auch „all“-Elemente akzeptiert. Wird eine Sprache nicht gefunden, gibt der Vervollständigungsteil des Transformators eine Fehlermeldung aus (siehe Anhang B: Sprachdefinitionen).

Konsistenzprüfung: In einem IML-Modell existieren zum Teil Relationen, die auf beiden Seiten vermerkt sind. Zudem dürfen manche Felder keine Duplikate enthalten oder nur mit Zeichen aus einem bestimmten Alphabet gefüllt sein.

Modellbestandteile, die derartig definiert sind, können in dieser Stufe auf Konsistenz geprüft werden. Duplikatprüfung bei IDs, die „Wohlgeformtheit“ des XML-Dokuments und die Prüfung gegen die DTD werden schon beim Einlesen vom XML-Parser durchgeführt.

Entfernung von Elementen anderer Konfigurationen: Use-Cases und InteractionCases können über BELONGS_TO_CONFIGURATION einer bestehenden Konfiguration zugeordnet werden.

Gehört ein Element nicht zur gerade aktiven Konfiguration (CONFIGURATION_DEFINITION), kann es entfernt werden. Dieser Mechanismus ist allerdings nicht sicher, da er Aufrufketten zerstören kann.

Hilfe-IDs: Zum Aufbau einer kontextsensitiven Hilfefunktion werden Konstanten benötigt, die den jeweiligen Hilfe-Kontext identifizieren. Hierzu ist es nötig, eindeutige Konstanten in den Code bzw. die Ressourcen-Definition zu integrieren.

Um diese Arbeit nicht dem Entwickler zu überlassen und Inkonsistenzen zu vermeiden, können Hilfe-Konstanten (HelpID) im Modell mit dem Wert „system“ versehen werden. Diese Attribute werden dann vom Transformator aufsteigend mit noch freien Werten ab dem im Modell vorgegebenen HelpIDStart belegt.

4.8.1 Feldbezeichnungen

Die UseCase-Spezifikation wird in der Sprache des Kunden bzw. Anwenderkreises definiert. Die daraus abgeleiteten InteractionCases enthalten ebenfalls diese Sprache.

Spätestens zu Beginn des Entwurfs wird in entwicklerzentrierten Projekten eine Art "Entwicklersprache" eingeführt. Basierend auf dieser wird die Software entwickelt. Für den Anwender verständliche Bezeichnungen werden dann meist erst nachträglich in die Benutzungsoberfläche eingebracht.

HERBS verfolgt eine andere Zielrichtung. Hier soll die Benutzungsoberfläche direkt aus den im Use-Case-Modell verwendeten Bezeichnungen generiert werden. Ein Bruch in der Kommunikation mit dem Anwender kann so vermieden werden.

Wie aus entsprechenden Realisierungen von Oberflächengeneratoren hervorgeht, kann ohne entsprechende Zusatzinformationen keine adäquate ergonomische Benutzeroberfläche generiert werden.

Als Beispiel kann hier der in [Hofmann 1998] verwendete "ergonomische Bezeichner" verwendet werden: Datenfeldbezeichner im Modell des Systems *S(D)* bzw. im Modell des Entwicklers vom System *E(S(D)))* dürfen keine Leerzeichen enthalten und werden nach anderen Kriterien ausgewählt, wie dies im Benutzermodell *B(D)* der Fall ist.

Aus diesem Grund wird in [Hofmann 1998] ein "ergonomischer Bezeichner" eingeführt. Mit dessen Hilfe wird das Problem durch die Eingabe eines für den Benutzer verständlichen Synonyms gelöst. Auf diese Weise wird die Sprache des Benutzers nachträglich synthetisiert.

In der IML wird sowohl ein Bezeichner als auch eine Beschreibung sprachenabhängig eingegeben. Wegen der starken Benutzerorientierung findet dieser Bezeichner auch in der Entwicklung Verwendung. Nur wo eindeutige Namen gefragt sind werden die Bezeichner aus dem späteren System (S(D)) verwendet, z. B. „RichtigerBezeichner“ statt „richtiger Bezeichner“.

Existieren entsprechende Umwandlungsvorschriften, können eindeutige Variablennamen auch erzeugt werden, beispielsweise durch die Entfernung von Leerzeichen und die Erweiterung mit einer eindeutigen Zahl. Allerdings muss hierzu immer eine Sprache, gewöhnlich das Englische, genutzt werden, da sonst die Anbindung des Fachkonzepts ebenfalls sprachabhängig würde.

4.8.2 Shortcuts und Hotkeys

Speziell in Dialogen, die vorwiegend textuelle Eingaben erfordern, ist die Bindung beider Hände zur Tastatur stark. [Hofmann 1998] schlägt deshalb vor, in solchen Dialogen die tastaturgebundene Benutzbarkeit durch Tastaturkürzel zu forcieren.

Eine automatische Generierung hat den Vorteil, dass keine Doppelbelegungen von Shortcuts vorkommen und sichergestellt wird, dass alle Einträge, sofern erwünscht, einen Hotkey zugewiesen bekommen.

Wichtig ist zudem eine Überprüfung auf Shortcut-Belegungen, die nach gängigem Standard anderen Funktionen zugeordnet sind, z. B. <Strg><Alt>-<Entf> oder <Alt>-<F4>. Eine Überprüfung durch den Transformator kann hier mit Warnungen und Hinweisen helfen, bessere Belegungen zu erstellen.

Generierung: Sind keine Hotkeys angegeben, kann mit Hilfe eines relativ einfachen Algorithmus der passende Hotkey generiert werden. Eine Beschreibung für einen möglichen Algorithmus ist in Anhang B (Hotkey-Generator) angegeben.

Shortcuts: Während Hotkeys für möglichst viele Elemente zur Verfügung stehen sollten und zudem aus dem (Label-)Text generiert werden können, sind Shortcuts nicht automatisch zuweisbar. Der Transformator kann deshalb nur auf Duplikate überprüfen.

Shortcuts eignen sich zudem nur für Befehle und nur selten zum Anspringen bestimmter Stellen. Die sinnvollste Vorgehensweise für einen IML-Editor wäre ein Wizard oder „Assistent“, der das IML-Modell für den Entwickler durchläuft und zu jedem Befehl die Möglichkeit zur Auswahl von Standard-Shortcuts, wie <Strg>-C, <Strg>-V etc., und selbst definierten Kürzeln bietet.

5 Transformation und Dialogmodellierung

Steht ein vervollständigtes und überprüftes IML-Modell zur Verfügung, kann daraus eine Reihe verschiedener Artefakte generiert werden, die für den weiteren Entwicklungsprozess, das Produkt oder den Benutzer benötigt werden.

Hauptziel ist dabei zunächst die Generierung der Benutzungsschnittstelle. Zu diesem Zweck wird ein Transformator benötigt, der das IML-Modell anhand von im Folgenden beschriebenen Regeln in eine ausführbare Repräsentation der Oberfläche umsetzt.

Dieses Kapitel beschreibt die hierzu als Zwischenrepräsentation verwendete Dialog Layer Language (DiLL), Ablaufsteuerungsmechanismen, Layout-Grundsätze und Strukturierungsmöglichkeiten sowie den generellen Ablauf des Transformationsprozesses und eine mögliche Umsetzung.

5.1 Die Dialog Layer Language (DiLL)

In diesem Abschnitt wird die Dialog Layer Language (DiLL) beschrieben. Sie dient als weitgehend plattformunabhängige Zwischenrepräsentation der Dialogebene. Ein Abschnitt zur Motivation ergänzt dabei die Beschreibung der verwendeten Konstrukte und Vorgehensweisen.

5.1.1 Zwischenformate

Bei JANUS [Hofmann 1998] wird ein Zwischenformat generiert, das noch semantische Informationen und mehrere Darstellungsmöglichkeiten enthält.

Diese Vorgehensweise ist aus den in [Kruschinski 1999] genannten Gründen auch für den IML-Transformator sinnvoll. Auch ist der Bruch zwischen ablauforientierter Repräsentation und grafischen Benutzungsoberflächenstrukturen zu groß, als dass ein direkter Übergang sinnvoll erschiene.

Ein Generator der aus einem IML-Modell immer exakt die gewünschte Lösung erzeugt ist mangels eines vollständigen Weltwissens im Anwendungs- bzw. Ergonomiebereich und der subjektiven Einschätzung der am Projekt Beteiligten nicht möglich. Ein Eingriff in den Ablauf bietet sich dort an, wo Elementtypen und -positionen festgelegt werden. Ebenso ist für eine Optimierung des Layouts in jeder denkbaren Weise eine layoutorientierte Repräsentation erforderlich.

Die bewährte baumartige Repräsentation kommt auch hier zum Einsatz. Sie ermöglicht die Generierung einer XML-basierten Zwischenrepräsentation, die mit den schon in Kapitel 3 genannten Vorzügen einhergeht. Fenster, verschachtelte Gruppen und darin enthaltene Elemente lassen sich auf diese Weise gut repräsentieren.

5.1.2 DiLL: Aufbau der Dialogebenen-Repräsentation

Die Dialog Layer Language DiLL ist eine Beschreibungssprache für alle direkt den Dialog betreffenden Bestandteile einer zu entwickelnden Applikation. Sie wurde als Zwischenrepräsentation des Transformationsprozesses entworfen, welcher ein IML-Modell in einen Oberflächenprototyp überführt.

Auch die DiLL ist eine XML-basierte Sprache, für die eine DTD existiert. Alle Überlegungen für die IML hinsichtlich Notation, Verarbeitung etc. sind daher für die DiLL in gleicher Weise gültig.

Während bei der Konzeption von IML das Ziel war, eine möglichst vielseitige, plattformunabhängige und mächtige Sprache zu entwerfen, soll die DiLL möglichst kompakt alle für Dialogaufbau und -optimierung notwendigen Informationen bereitstellen. Zudem soll sie eine Struktur aufweisen, die den Aufbau eines Objektmodells zur Layoutoptimierung ermöglicht.

Die DiLL beschränkt sich deshalb auf eine sehr geringe Anzahl von Tags, die die späteren Bestandteile der Oberfläche repräsentieren. SCREEN, GROUP und ELEMENT bilden dabei die statische Komponente. Menüstrukturen und dynamische Abläufe ergänzen das Konzept.

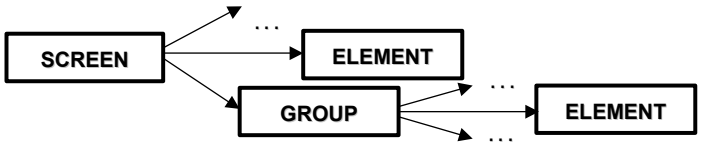

Abbildung 25: SCREEN, GROUP und ELEMENT in einem DiLL-Modell

5.1.2.1 SCREENs

Direkt dem Wurzelknoten (DILL_SPECIFICATION) untergeordnet existiert eine beliebige Anzahl von SCREEN-Knoten.

Ein Screen repräsentiert einen Block innerhalb des Dialogablaufs, der gleichzeitig aktiv ist und mit dem der Benutzer interagieren kann. Zumeist entspricht ein SCREEN auf der IML-Seite einem InteractionCase und auf der Code-Seite einem Dialog oder auch einer „Bildschirmseite“ im Infotainment.

Verhalten, Typ und Beschriftung werden vom Transformator oder manuell von einem Entwickler im DiLL-Modell angegeben. Sind bestimmen später das Ergebnis des Generierschritts. Ein besonderer SCREEN ist der initiale SCREEN, der durch Angabe seiner ID im Attribut InitialScreen identifiziert wird. Der spätere Programmablauf startet mit diesem SCREEN und verzweigt von dort aus weiter.

Jeder SCREEN kann weitere SCREENS sowie Gruppen und Elemente enthalten. Zudem ist die Menüstruktur und eine Repräsentation der Ablaufdynamik enthalten. Eine Beschreibung der untergeordneten Tags erfolgt in den nächsten Kapiteln.

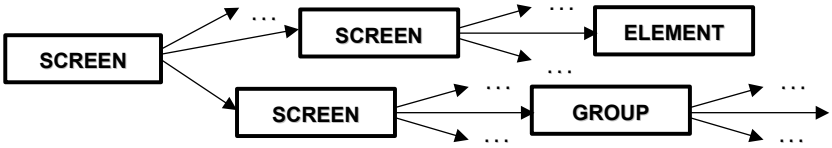

Abbildung 26: Untergeordnete SCREENs

5.1.2.2 Gruppen

Gruppen dienen, wie ihr Name schon sagt, als Rahmen für eine Menge von Elementen.

```
<!ELEMENT GROUP (GEOMETRY, (ELEMENT | GROUP)*, CONTENT*)>
```

Abbildung 27: Definition einer DiLL-Gruppe in der DiLL-DTD

Dabei können sie, vom Benutzer weitgehend unbemerkt, als Gruppierungselement wirken, oder auch mit Hilfe eines Rahmens sichtbar gemacht werden.

Gruppen verfügen ebenso wie Elemente über eine Angabe, aus welchem IML-Element sie hervorgegangen sind. So wird sichergestellt, dass benötigte Informationen aus dem IML-Modell auch für Entscheidungen im Rahmen der Anordnung und Optimierung weiterhin verfügbar sind.

Viele Informationen aus dem IML-Modell existieren im DiLL-Modell in kopierter oder kondensierter Form, so dass die layoutrelevanten Daten verfügbar sind.

Welche Art von Gruppe generiert wird, legt das ElementType-Attribut fest. Mit Hilfe dieses Eintrags werden aus der ElementSpecification zugehörige Ressourcen und Code generiert.

5.1.2.3 Geometrie-Informationen in Gruppen und Elementen

Gruppen verfügen ebenso wie Elemente über Geometrie-Informationen (GEOMETRY), die Dimensionen und Position festlegt.

Zudem kann über die Geometrie eine Verteilung der Gruppen und Elemente auf optische Zeilen und Spalten erfolgen und der Fill-/Full-Mechanismus aktiviert werden.

Fill-/Full-Mechanismus: Für jedes Element kann definiert werden, ob es die noch verfügbare Breite bzw. Höhe ausfüllen soll und ob es sich sogar über die komplette Breite oder Höhe erstrecken muss. Das Element kann dann entsprechend vergrößert und positioniert werden. Die Werte für Fill und Full werden berechnet oder aus dem IML-Modell übertragen.

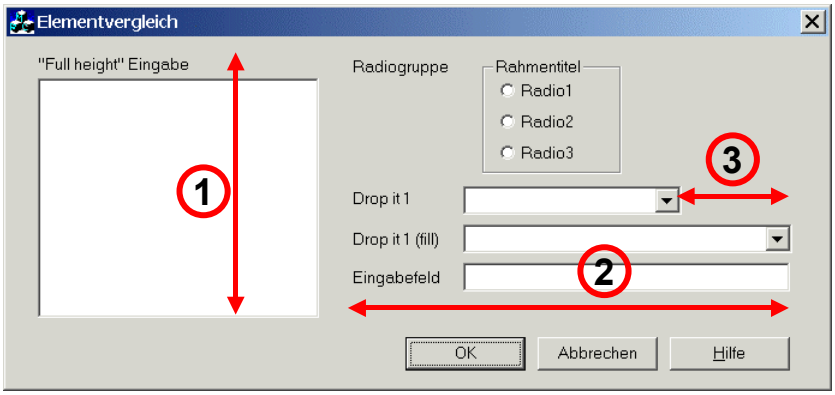

Abbildung 28: Beispiel für full height und fill width. Full height wird auf volle Höhe vergrößert (1). Das Auffüllen auf die verfügbare Breite (2) vergrößert das Element um die Differenz (3) zum größten Element.

Spaltenübergreifende Darstellung: Durch die Angabe von „full width“ entsteht eine spaltenübergreifende Darstellung wie auch bei Tabellen oder manchen mehrzeiligen Eingabefeldern. Diese macht allerdings nur dann Sinn, wenn der Arbeitsablauf noch sinnvoll abzubilden ist, d. h. wenn das Feld ganz am Anfang oder am Schluss Gesamtablaufs, in Ausnahmefällen auch bei einem Spaltenumbruch in diesem Dialog steht.

Das Feld kann dann oben oder unten im Dialog platziert werden. Bei dominierenden Tabellen ist eine Positionierung oben, bei allen anderen Elementen eine Positionierung unten vorzuziehen, soweit nicht die Abläufe eine andere Position nahe legen.

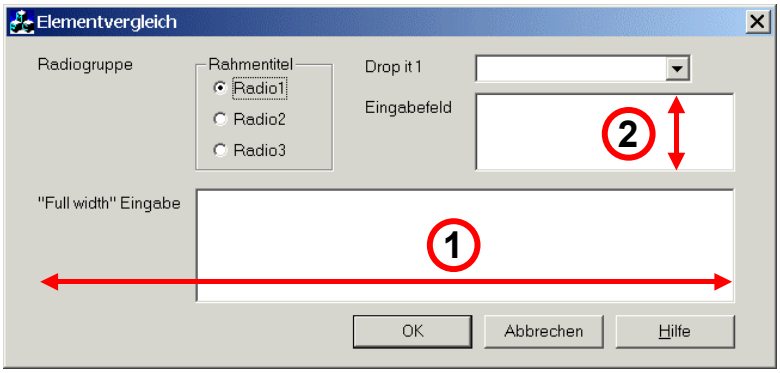

Abbildung 29: Beispiel für full width und ein fill height. Full width nutzt die gesamte Dialogbreite (1). Fill height füllt die verfügbare Höhe auf.

In manchen Fällen ist es auch möglich, den Dialog durch ein spaltenübergreifendes Element in einen oberen und einen unteren Teil zu trennen. Der Layout-Algorithmus wird dann allerdings komplexer,

da eine Anordnung für den oberen und den unteren Teil gefunden muss. Sind andere Elemente sehr hoch, kann dies die genannte Auftrennung unmöglich machen.

Optische Zeilen und Spalten sorgen für eine regelmäßige Anordnung, indem sie als pixelunabhängiges Raster[31] dienen. Über dieses Konzept können die optischen Linien erfasst werden, die sich aus beginnenden und endenden Elementen sowie anderen markanten Linien ergeben. Ziel ist eine Minimierung der Anzahl optischer Linien.

Jedes Element enthält eine optische Spalte und eine optische Zeile. Da Labels auch als Elemente beschrieben werden, steht die optische Spalte bei jedem sichtbaren Element zur Verfügung und kann ausgewertet werden. Die Anzahl der optischen Spalten erhöht sich immer dann, wenn ein Element wegen der relativen Position zu anderen nicht mehr auf den vorhandenen Spalten angeordnet werden kann.

5.1.2.4 DiLL-Elemente

Alle später auf der Oberfläche sichtbaren Elemente, mit Ausnahme der Gruppenrahmen, sind als DiLL-Elemente definiert. Position und Größe werden über die Geometrie festgelegt. Das letztendlich generierte Element wird wie bei den Gruppen durch das ElementType-Attribut festgelegt.

Da der Elementtyp schon festliegt, kann aus dem Mengengerüst zumindest initial die Größe bestimmt werden, so dass ein grundlegendes Layout generiert werden kann.

Zudem verfügt jedes Element über einen Tooltip-Eintrag, der aus den Beschreibungen im IML-Modell generiert wird.

5.1.2.5 Das Content-Konzept

Je nach festgelegtem Elementtyp eines SCREENs, einer Gruppe oder eines Elements werden unterschiedliche Parameter benötigt. Es ist nicht sinnvoll, diese alle fakultativ vorzusehen. Zudem wäre es beispielsweise nicht möglich, einer Liste Einträge zu übergeben, da Attribute für erweiterbare Listen nicht in Frage kommen.

Aus diesem Grund wurde das Content-Konzept eingeführt. Jeder Eintrag der über den Generierungsmechanismus umgesetzt werden soll, enthält eine Liste von Contents (CONTENT*).

Jeder Content verfügt über einen Namen (ContentName) – vergleichbar einem Parameternamen – über den der Zugriff erfolgt. Die eigentliche Information wird im Attribut ContentText gespeichert. Das Auslesen der Einträge eines Listenelements gestaltet sich mit diesem Prinzip denkbar einfach: Alle Contents mit dem Namen „ValueItemEntry“ werden eingelesen und als Einträge in die Liste geschrieben.

		ContentName	ContentText	ContentInfo
SELECT	CONTENT	ValueListEntry	Thomas	
	CONTENT	ValueListEntry	Michael	
	CONTENT	ProjectName	ETC	V 1.0.1

Abbildung 30: Fiktives Beispiel für einen CONTENT-Block

5.1.2.6 Menüstrukturen

Jeder SCREEN kann eine Menge von Befehlsgruppen (COMMAND_GROUP) enthalten. Jede dieser Gruppen stellt einen Eintrag in einer Menüzeile oder einem Bildschirmmenü dar. Untermenüs innerhalb einer Befehlsgruppe werden genau wie Befehlsgruppen definiert. Eine Unterscheidung erfolgt nur über den Elementtyp. Der mit einer Markierung des Hotkeys versehene Name (HotKeyString) wird zur Anzeige verwendet.

[31] Notwendigkeit und Anwendung von pixelabhängigen Rastern in Dialogen werden in [Schweizer 1993] näher beschrieben.

Die Menüeinträge (COMMAND) enthalten eine auch später im Quellcode verfügbare eindeutige ID (CallCommandID), die ein Auslösen des mit dem Eintrag verbundenen Befehls ermöglicht. Auch die Menüeinträge verfügen über einen hotkeyfähigen Namen, enthalten aber zusätzlich eine Definitionsmöglichkeit für einen Shortcut.

5.1.2.7 Dialogabläufe

Die Ablaufsteuerung (DYNAMICS) ermöglicht die Ergänzung der bisher beschriebenen statischen Teile und Menüstrukturen durch modellierbare Aktionen. Eine Erprobung im Generator liegt nicht vor. Das Konzept kann aber beispielsweise innerhalb der Ereignisbehandlungsroutinen mit entsprechenden Nachrichtencodes integriert werden.

Die Dynamik ist in fünf Blöcke unterteilt:

- Laden von Werten und Initialisierung zu Beginn (INIT)
- Ereignisverarbeitung und Ablaufsteuerung während des Ablaufs (RUN)
- Werte überprüfen beim Beendigungssignal (LEAVE)
- Überprüfen, speichern und Folgedialoge aufrufen bei erfolgter Beendigung (FINALIZE)
- Beim Dialog-Abbruch Folgebildschirme öffnen (ABORT)

Alle notwendigen Überprüfungen und Aufrufe können so zum passenden Zeitpunkt ausgeführt werden. Zudem kann eine weitgehende Anbindung an das zu implementierende Fachkonzept ermöglicht werden: Ein Aufruf, beziehungsweise eine Ausführung, von Codefragmenten ist in jedem der fünf Blöcke möglich.

5.1.3 Eine Implementierung des DiLL-Objektmodells

Für die Generierung und Layoutoptimierung des DiLL-Modells kommt im „Referenz-IML-Oberflächengenerator“ ein Objektmodell zum Zug, das die Struktur der DiLL weitgehend wiedergibt.

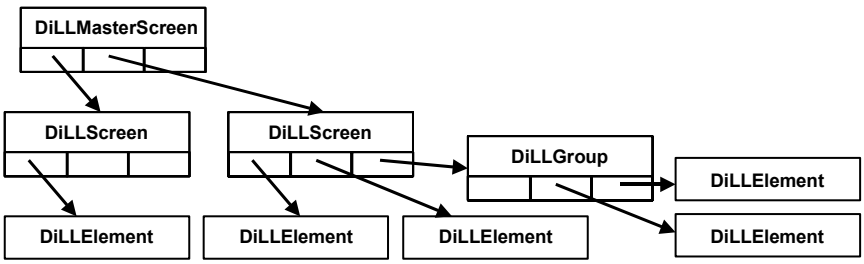

Abbildung 31: Beispiel für einen Zustand der DiLL-Objekte während des Transformationsprozesses

Es existiert für jedes System bzw. jede Applikation ein DiLLMasterScreen-Objekt, das seinerseits eine beliebige Anzahl von DiLLScreen-Objekten verwaltet.

Screenobjekte können wiederum DiLLScreen-Objekte aber auch DiLLGroup-Objekte und DiLLElement-Objekte enthalten. DiLLGroup-Objekte enthalten ihrerseits DiLLGroup-Objekte und DiLLElement-Objekte.

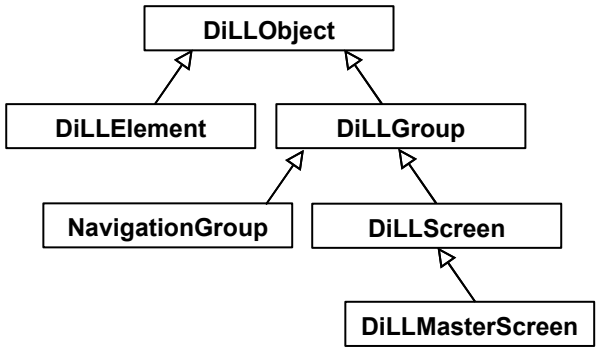

Abbildung 32: Die Vererbungshierarchie der Klassen für die DiLL-Repräsentation

Durch die gemeinsame Abstammung von DiLLObject verfügen alle Objekte der DiLL-Klassen über Methoden zu Positionierung. Zudem können alle Listen, Gruppen und Elemente gleichberechtigt aufnehmen.

Durch das Einfügen von Methoden zur Errechnung von Metriken in die DiLL-Klassen, können mittels Tiefensuche beliebige elementabhängige Metriken berechnet werden; beispielsweise das Verhältnis von Leerraum zu gefülltem Platz.

5.2 Fenster und Elemente

5.2.1 Fenster

Jedes InteractionCase wird mit Hilfe eines Transformationsschritts auf ein Dialogfenster oder ein Register abgebildet. Dabei wird ein Fenster oder Register in der DiLL durch einen SCREEN repräsentiert.

Bevor ein Fenster vollständig konstruiert ist, müssen allerdings, weitgehend unabhängig vom Inhalt des InteractionCase, noch einige Merkmale erzeugt werden.

5.2.1.1 Fenstertitel

Aussagekräftige und nur einmal vorhandene Fenstertitel sorgen für eine leichte und eindeutige Erkennung durch den Benutzer.

Der Fenstertitel des Applikationsfensters kann generativ aus folgenden Bestandteilen zusammengesetzt werden.

- Firmenname
- Projektname (aus der IML-Projektdefinition)
- Systemname
- Version

Die Version setzt sich zusammen aus:

- MainVersion
- SubVersion
- SubSubVersion
- BuildNumber

Meist genügt aber eine Angabe der ersten zwei oder drei Teile der Version.

Abbildung 33: Beispiel: Titelzeile des ETC im YUKON-Projekt

Bei Unterfenstern wäre es möglich, diese Informationen ebenfalls darzustellen. Dies macht aber meist keinen Sinn, da zuviel Information angezeigt wird.

Besser ist es, den Titel wie folgt zusammenzusetzen:

- Systemname
- „ - - “
- Bezeichnung des InteractionCase

Abbildung 34: Beispiel: Titelzeile des ETC im YUKON-Projekt

Bei mehreren gleichen Fenstern sollte eine Nummerierung oder eine eindeutige Kennung hinzugefügt werden, z. B. die ID des dargestellten Datensatzes. Es sei denn der Inhalt ist derselbe; dann muss eine fortlaufende Nummerierung in Klammern erzeugt werden, um die Fenster zu unterscheiden.

5.2.1.2 Inhärente Funktionen des Hauptfensters

Die Funktionen zum Starten und Beenden sind in jedem Hauptfenster vorhanden. Das Schließen des Fensters muss zur Folge haben, dass

- eine Sicherungsabfrage erfolgt
- alle untergeordneten Fenster und alle Dateien geschlossen werden
- die Anwendung beendet wird

Beim Start muss die Anwendung initialisiert werden. Bei Mehrbenutzersystemen und manchen anderen speziellen Anwendungsfällen ist ein vorgeschalteter Dialog zum Login oder ähnlichen Eingaben notwendig, bevor das Hauptfenster angezeigt wird.

5.2.1.3 Inhärente Funktionen aller Fenster

Jedes Fenster verfügt über bestimmte Optionen, die nicht explizit modelliert werden müssen. Unter Windows stehen im Window-Menü die Befehle „schließen“, „minimieren“, „maximieren“, „verschieben“ und „Größe ändern“ zur Verfügung. In Dialogfenstern nur die Funktion „schließen“.

5.2.2 Geometrie und Elementeigenschaften von Interaktionselementen

5.2.2.1 Abstände

Für die möglichen Ziel-Plattformen existieren meist Styleguides, die die Abstände von Elementen regeln und so das für repräsentative Anwendungen dieser Plattform typische Aussehen und Verhalten bestimmen.

Eine Regelung ist dabei aber nur auf der Ebene von Elementarten sinnvoll. So sollten beispielsweise für Eingabefelder und Drop-Down-Elemente keine separaten Werte vorgesehen werden.

Die in dieser Arbeit verwendeten Werte können der IML-DTD entnommen werden. Sie sind in den TRANSFORMATION_PARAMETERS als Voreinstellungen enthalten.

5.2.2.2 Eingabefelder

Während die Höhe eines Eingabefeldes sich direkt aus der Schriftgröße ableiten lässt, sind bei der Festlegung der Breite einige Vorüberlegungen anzustellen. Die Höhe entspricht der Höhe des verwendeten Schriftsatzes zuzüglich des doppelten Elementrandes. Der Elementrand kann als Konstante festgelegt werden, da er sich bei gleichbleibendem Elementtyp nicht ändert.

Die Breite des Elements könnte pessimistisch aus der maximalen Anzahl Zeichen multipliziert mit der maximalen Zeichenbreite plus dem doppelten Elementrand festgelegt werden. Es zeigt sich aber, dass diese Festlegung die Elemente zu groß gestaltet (vgl. [Kruschinski 1999]).

Im praktischen Gebrauch werden zumeist weniger Zeichen eingegeben als maximal möglich wäre. Zudem kompensieren kleinere Zeichen zumeist den Einsatz größerer, so dass die mittlere Zeichenbreite und auch eine unterhalb des theoretischen Maximums liegende Anzahl sichtbarer Zeichen ausreicht. Beispielsweise würde für ein Nachnamen-Feld eine Breite von 25 Zeichen ausreichen, auch wenn Doppelnamen im Extremfall länger werden können. Hier wird also die erschwerte Bearbeitung des Extremfalls der Verständlichkeit im Normalfall untergeordnet.

Das Breitenproblem tritt vor allem deshalb auf, weil mittlerweile meist Proportionalschriften eingesetzt werden und auch die Systemschriften keine einheitliche Zeichenbreite mehr aufweisen. Für den Benutzer verbessert sich die Lesbarkeit. Das Layout wird im Gegenzug allerdings schwieriger.

Die Breite eines Eingabefelds wird daher am besten aus der mittleren Zeichenbreite und einem Wert zwischen der durchschnittlichen und der maximalen Länge bestimmt.

- Die Elementlänge E wird für textuelle Eingaben über die durchschnittliche Länge L, die maximale Länge M und die Randbreite R bestimmt:

$$E = Min(L \cdot 1{,}5; L + 0{,}65 \cdot (M - L)) + 2 \cdot R$$

Gleichung 1: Bestimmung der Elementlänge E

- Bewegt sich der Wert M unterhalb des verfügbaren Platzes, wird M eingesetzt.
- Für Zahleneingaben wird immer die maximale Länge der Eingabe plus 2 * R reserviert.

5.2.2.3 Mehrzeilige Texteingabefelder und Tabellen

Eine Darstellung der Maximalgröße des Inhalts ist bei mehrzeiligen Feldern nur in den wenigsten Fällen sinnvoll, da oft mehr als 1000 Zeichen in ein Textfeld eingegeben werden können, die Felder im praktischen Gebrauch aber trotzdem oft leer oder wenig gefüllt sind.

Stehen mehrere mehrzeilige Textfelder oder Tabellen in einem Dialog, wirkt dieser überladen. Dann ist zu prüfen, ob Textfelder oder Tabellen ausgelagert oder der gesamt Dialog aufgeteilt werden kann.

In einem Dialog mit mehr als zwei Spalten ist ein Textfeld kaum einspaltig unterzubringen, da die erforderliche Breite dann nicht mehr erreicht werden kann. Auch bei zweispaltigen Dialogen sollte zumindest bei größeren Textfeldern oder Tabellen geprüft werden, ob nicht die gesamte Breite des Fensters genutzt werden kann, und nur die kleineren Elemente spaltenweise organisiert werden. Um dies anzuzeigen, kann das Element-Attribut „full“ genutzt werden.

Scrollbars können bei Überschreitung der Anzeigebereichsgröße eingesetzt werden. Während vertikale Scrollbars immer nach Bedarf dynamisch verfügbar sein sollten, sind horizontale Scrollbars meist kontraproduktiv und sollten daher von einem Transformator nur bei expliziter Vorgabe zugelassen werden.

Aus kognitiver Sicht reicht eine Darstellung von vier bis sechs Zeilen bei 80 Zeichen aus, da der Benutzer nicht mehr Information auf einmal wahrnehmen kann. Ist mehr Platz im Dialog verfügbar, kann das Feld vergrößert werden, darf dabei aber die maximale Zeilenzahl nicht überschreiten.

Um sicher zu gehen, wird zunächst die minimale Anzahl Zeichen pro Zeile als Grundlage genommen. Ist ein größerer horizontaler Platzverbrauch möglich, kann das Feld bis zur maximalen Zeichenzahl pro Zeile vergrößert werden.

Regeln:

- Gesamtbreite des Dialogs für Tabellen und Textfelder nutzen, wenn sie vorne, an einem möglichen Spaltenumbruch, oder hinten im Dialogablauf stehen
- Horizontaler Scrollbar nur wenn explizit gefordert (HorizontalExpandable), vertikale bei Bedarf einblendbar
- Horizontale Größe auf minimale Zeilenlänge plus Scrollbarbreite und Rand setzen. Soweit möglich vergrößern und durch gegenseitige Anpassung von Fenster und Textfeld an die durchschnittliche oder, sofern möglich, maximale Breite annähern
- Vertikale Größe auf die durchschnittliche Anzahl Zeilen setzen. Im Fall eines einfachen Algorithmus genügt es, die Zeilenzahl auf die durchschnittliche Zeichenzahl, geteilt durch die durchschnittliche Zeilenlänge, zu setzen.
- Ist ein entsprechendes Attribut angegeben, können Zeilenbreite und Zeilenzahl als Ausgangswert übernommen und bei Bedarf durch Vergrößerung an die Layoutsituation angepasst werden.

Tabellen können nur bei homogenen Objekten eingesetzt werden, d. h. wenn alle Objekte über dieselben Attribute verfügen und diese sinnvoll zeilenweise angezeigt werden können.

Ist dies nicht der Fall, so können Objekte mit einer Art Dateisicht, vergleichbar mit dem Windows-FileOpen-Dialog, visualisiert werden [Hofmann 1998]. Tabellen sollten immer auf volle Fensterbreite gestreckt werden, es sei denn es handelt sich nur um wenige und kleine Spalten.

Regel:

- Berechnung der benötigten Anzeigegröße entsprechend den enthaltenen Elementtypen. Kann diese innerhalb einer Dialog-Spalte untergebracht werden, erfolgt die Darstellung in dieser Größe, sonst als spaltenübergreifendes Element.

5.2.2.4 Listen

Anders als Textfelder können Listen die enthaltenen Texte nicht umbrechen und meist zu lange auch nicht Zeilen notfalls mit einem horizontalen Scrollbar komplett zugänglich machen[32]. Deshalb muss für die Breitenbestimmung bei Listen der Maximalwert für die Zeilenlänge zugrundegelegt werden.

Die Höhe sollte bei einem normalen Listenfeld, d. h. einem Feld mit normaler Priorität, 9 Zeilen nicht überschreiten. Allerdings können sehr wichtige Felder auch größer dargestellt werden.

Im IML-Modell kann auch eine Spalte mit der entsprechenden full- oder fill-Option als komplett genutzt oder als aufzufüllend angegeben werden.

Die Werte für Klapplisten werden im nächsten Abschnitt beschrieben.

Regeln:

- Die Breite einer Listbox setzt sich somit zusammen aus der maximalen Länge eines Eintrags, der Summe von zwei Randbreiten und der Breite eines vertikalen Scrollbars. Dabei kann der Scrollbar vernachlässigt werden, falls die Anzahl der Einträge die Höhe nicht überschreiten kann.

[32] Ausnahmen bilden hier Listen mit mehrzeiligen Einträgen, die allerdings nur im Falle weniger Einträge und auch dann sparsam verwendet werden sollten, da ausreichend Platz zur Verfügung stehen muss und die Übersichtlichkeit leidet.

- Wird die Höhe von 9 Zeilen nicht überschritten, kann die Maximalzahl, durchschnittliche Anzahl oder minimale Anzahl (in dieser Reihenfolge) überprüft und falls möglich übernommen werden.
- Die Höhe ergibt sich aus der Anzahl der sichtbaren Zeilen multipliziert mit der Zeilenhöhe. Hinzu kommt noch die doppelte Rahmenbreite.

5.2.2.5 Kombinationsfelder

Bei Kombinationsfeldern kommt zu den Eigenschaften der Listen noch ein einzeiliges Eingabefeld hinzu. Es gelten die gleichen Regeln wie für Listen. Allerdings muss bei der Dimensionierung das an der Oberseite ohne Abstand angeschlossene Eingabefeld berücksichtigt werden.

Für das Eingabefeld gilt der Grundsatz, dass es dieselbe Breite hat wie die darunter angeordnete Liste inklusive aller Elemente, d. h. auch inklusive des vertikalen Scrollbars. Maßgeblich für die Gesamtbreite ist also die maximale Breite eines Listeneintrags. Das Eingabefeld wird einfach in passender Breite hinzugefügt.

Für Drop-Down-Kombinationsfelder gilt die gleiche Regelung. Allerdings ist der Platzbedarf eines solchen Feldes in der Höhe nur der einer Eingabezeile. In der Breite muss ein Klapp-Knopf (Drop-Down-Button) hinzugerechnet werden. Die Liste wird nur bei Aktivierung mittels Klapp-Knopf ausgeklappt und sichtbar. Klapplisten sind gleich zu behandeln – ein Unterschied wird nur im Verhalten sichtbar.

Die Höhe der ausklappenden Liste kann wie beim Listenfeld berechnet werden. Der Wert wird jedoch beim Layout nicht berücksichtigt. Allerdings besteht beim Ausklappen längerer Listen von im Dialog weit unten stehenden Elementen die Gefahr, dass der Bildschirmrand in zu geringer Entfernung liegt.

Gelöst wird das Problem meist durch ein wahlweises Aufklappen nach oben statt nach unten, wenngleich die englische Bezeichnung „drop down combo box“ ein anderes Verhalten suggeriert. Über den Vorteil einer dynamischen Größenanpassung im Vergleich zur Änderung der Klapp-Richtung liegen keine Studien vor.

Regeln:

- Kombinationsfeld: Dimensionierung wie Listenfeld mit zusätzlichem einzeiligen Textfeld gleicher Breite
- Klappliste und Drop-Down-Kombinationsfeld: Höhendimensionierung wie bei der Eingabezeile, Breitendimensionierung aus Eingabezeile plus Breite für den Klapp-Knopf. Die ausklappende Liste wird in ihrer Zeilenzahl wie das Listenelement dimensioniert.

5.2.2.6 Auswahlfelder

Einfachauswahlfelder (Radiobuttons) kommen immer zu mehreren vor. Sie werden dabei zu einer Gruppe zusammengefasst, innerhalb der immer genau ein Element ausgewählt sein muss. Mehrfachauswahlfelder (Checkboxes) können einzeln vorkommen, werden aber zumeist ebenfalls in Gruppen zusammengefasst.

Listen, Drop-Down-Elemente und ähnliche Auswahl-Elemente werden in diesem Abschnitt nicht betrachtet.

Da es sich um spezielle Gruppen handelt, gelten auch andere Layoutregeln. Definiert werden müssen:

- Abstand zwischen den Elementen horizontal bei waagrechter Ausrichtung
- Abstand zwischen den Elementen vertikal bei senkrechter Ausrichtung
- Abstände zum Rand
  - Links
  - Rechts
  - Unten
  - Oben (dabei muss das Label auf dem Rand der Gruppe mit berücksichtigt werden)
- Distanz zwischen Knopf und Text

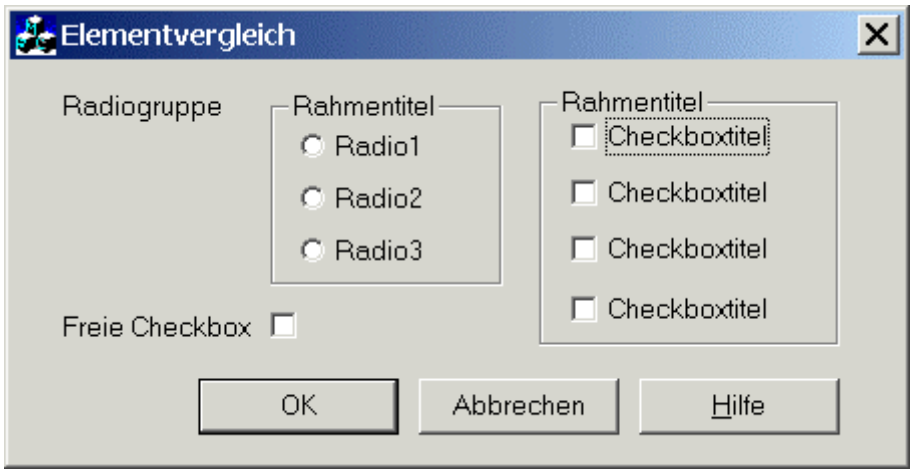

Abbildung 35: Radio-Buttons und Checkboxes

Nach Möglichkeit sollten diese Definitionen für Checkboxes und Radiobuttons übereinstimmen, um einen homogenen Eindruck zur schaffen.

Gehört keine Gruppe zu einer Checkbox, d. h. es handelt sich um eine alleinstehende Checkbox wie z. B. bei booleschen Werten, so ist die Breite nur durch das Kästchen, den Abstand zur Beschriftung und die Beschriftung selbst definiert. In diesem Fall ist es sinnvoll, andere Vorgaben zu wählen und die Checkbox wie ein Eingabefeld zu behandeln, d. h.

- Die Beschriftung wird, wie alle Labels, links vom Element dargestellt.
- Die Checkbox wird mit dem linken Rand am linken Rand der anderen Elemente in der Spalte ausgerichtet.
- Die Höhe entspricht der Zeilenhöhe des Beschriftungstextes.
- Der untere Rand der Checkbox sitzt nach Möglichkeit auf gleicher Höhe mit der Grundlinie von Großbuchstaben, d. h. nicht am unteren Ende.

Bei Auswahlfeldern kann zum Teil eine horizontale Anordnung statt der gewohnten vertikalen Anordnung sinnvoll sein. Sind Labels und Optionszahl klein, so können die Auswahlfelder in einer Zeile angeordnet werden.

Für den Fall einer Längenüberschreitung wird in [Kruschinski 1999] eine mehrzeilige Darstellung ermöglicht. Das Gesetz der Nähe, die Gefahr einer Überladung und die verstärkte Bildung von Linien widersprechen allerdings einer solchen Vorgehensweise, so dass eine vertikale Anordnung ist in diesem Fall vorzuziehen ist – speziell bei einer mehrspaltigen Anordnung der anderen Elemente im Dialog.

Die Entscheidung, ob eine horizontale oder vertikale Anordnung bevorzugt wird, kann bei einem IML-Modell mit Hilfe einer entsprechenden Gruppendefinition geschehen. Grundsätzlich sollte die endgültige Entscheidung abhängig vom verfügbaren Platz in der Breite gefällt werden, d. h. abhängig davon, ob ein Umbruch notwendig wäre.

5.2.2.7 Schieberegler (Slider)

Mit Hilfe eines Schiebereglers wird ein Wert innerhalb eines diskreten Wertebereichs ausgewählt. Die Dimension der Schieberegler richtet sich nach der durchschnittlichen Elementbreite einer Dialog-Spalte und der Anzahl auswählbarer Werte A.

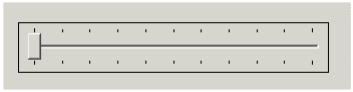

Abbildung 36: Schieberegler (Slider)

Steht mindestens ein Pixel, besser eine mittlere Zeichenbreite Platz pro Wert zur Verfügung, kann ein Vielfaches von A, das etwa im Bereich der mittleren Spaltenbreite S liegt, gewählt werden (n beliebig):

$$S = A \cdot n$$

Gleichung 2: Wahl eines Vielfachen von A

Steht nicht genügend Platz zur Verfügung, muss ein Bruchteil von A gewählt werden, um die maximale Breite M nicht zu überschreiten (n minimal):

$$M \approx \frac{A}{n}$$

Gleichung 3: M als Bruchteil von A

5.2.2.8 Spinbuttons

Sollen größere ganzzahlige Zahlenbereiche[33] auswählbar sein, können statt Schiebereglern Spinbuttons eingesetzt werden.

Abbildung 37: Ein Spinbutton in Verbindung mit einem Textfeld

Die Gesamtbreite G berechnet sich dabei mit Hilfe von K folgendermaßen (Eingabefeld plus Spinbutton):

K = [1 für Vorzeichen falls möglich, 0 sonst]
+ [Anzahl Vorkommastellen]
+ [1 für Komma/Punkt,
falls Anzahl Nachkommastellen > 0, 0 sonst]
+ [Anzahl Nachkommastellen]

Gleichung 4: Berechnung der Zeichenzahl K

[33] Bei bekannter, kleiner (< 3) Anzahl Nachkommastellen können ebenfalls Spinbuttons verwendet werden.

$$G = K \cdot [\text{mittlere Zeichenbreite}] + 2 \cdot [\text{Randbreite Eingabefeld}] + [\text{Breite Spinbutton}]$$

Gleichung 5: Bestimmung der Gesamtbreite G

5.2.2.9 Aktivierungsknöpfe (Toggle-Buttons)

Neben den „normalen“ Buttons, die zum Aufruf von Funktionen oder zur Navigation dienen, werden Buttons oft für das Ein- und Ausschalten bestimmter Optionen oder Funktionen genutzt.

Können Checkboxes nicht sinnvoll dafür eingesetzt werden, sollten entweder spezielle Druckknopf-Text-Kombinationen eingesetzt werden oder beispielsweise Radiobuttons zum Einsatz kommen, wenn Richtungen ausgewählt werden sollen. Auch spezielle Elemente können eingesetzt werden.

Der Einsatz Navigationsknöpfen ähnelnder Buttons für einen solchen Mechanismus ist fraglich, da aus Benutzersicht keine Unterscheidungsmöglichkeit besteht.

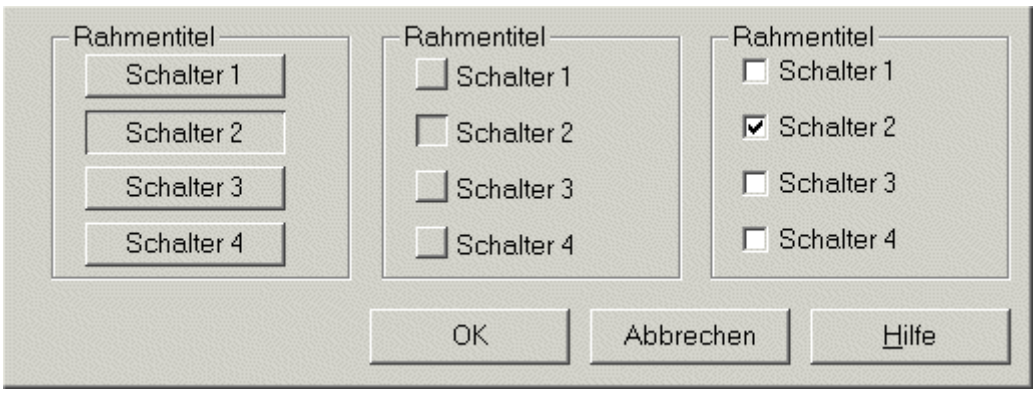

Abbildung 38: Zwei Umschaltflächentypen (Togglebuttons) im Vergleich mit der Checkbox

Regeln:

- Ein-/Ausschalten von Optionen soweit möglich mit Checkboxes modellieren
- Alternativ spezielle Buttons nutzen
- Eine Verwechslung mit Navigations- und Funktionsknöpfen darf nicht möglich sein

5.2.2.10 Dialogknöpfe zum Aufruf untergeordneter Dialoge und Funktionen

Untergeordnete Dialoge sind meist modale Dialoge, die von einem Dialog aufgerufen werden und auch die Dialogsteuerung wieder an diesen zurückgeben. Diese Aufrufe werden entweder aufgrund eines expliziten Aufrufs im Modell (CALL_IC) angelegt oder generiert, wenn eine Auslagerung zu großer und selten benötigter Interaktionselemente in einen untergeordneten Dialog notwendig wird.

Der untergeordnete Dialog wird dann mit Hilfe eines Knopfes aufgerufen, der in seinem Aussehen den Navigations- und Funktionsknöpfen entspricht. Um den Aufruf eines Dialogs zu zeigen, werden drei Punkte an die Beschriftung angefügt. Es ist zudem möglich ein entsprechendes Verb hinzuzufügen: Aus „Adresse“ wird dann „Adresse bearbeiten...“.

5.2.3 Auswahl von Interaktionselementen

Abhängig von der Art der Interaktion, dem Datentyp und dem Mengengerüst müssen für Eingabetypen die zughörigen Interaktionselemente bestimmt werden.

Oft werden bei anderen Ansätzen zur Oberflächengenerierung die auszugebenden Elemente direkt festgelegt, d. h. es ist nicht möglich, basierend auf dem Kontext zu entscheiden. Für eine Auswahl aus einer Liste steht dann beispielsweise nur ein Drop-Down-Element zur Verfügung [Kruschinski 1999]. Stehen keine Kontextinformationen zur Verfügung ist das die richtige Vorgehensweise.

Im Falle der IML steht jedoch eine genauere Klassifikation des modellierten Interaktionsschritts zur Verfügung. Somit kann hier die Auswahl des passenden Elements abhängig von den im Modell enthaltenen Informationen getroffen werden.

Abhängig von der Art der Benutzungsoberfläche und der gewählten Interaktionsform sind unterschiedliche Elemente zu generieren.

Genaue Werte können an dieser Stelle daher nur exemplarisch angegeben werden. So kann ein linearer Auswahlbereich für natürliche Zahlen, beispielsweise in einem Windows-Dialog, bei einer maximalen Elementbreite von 100 Pixeln nicht mehr als 50 Werte sinnvoll enthalten. Bei Überschreiten dieser Grenze muss ein Textfeld mit Spinbutton erstellt werden.

Die Grenze für Radiobutton-Darstellungen liegt bei einer Auswahl von unterschiedlichen Einträgen in einem wichtigen Element beispielsweise bei maximal 9. Je nach Erweiterbarkeit müssen dann Listen- oder Combobox-Elemente erstellt werden. Bei weniger wichtigen Auswahlelementen ist eine Drow-Down-Darstellung vorzuziehen.

5.3 Funktions- und Ablaufsteuerung durch Menüs und Buttons

Die Ablaufsteuerung bzw. Navigation wird meist mit Hilfe von Menüs und Buttons gestaltet. Während bei umfangreichen Wahlmöglichkeiten einer Menüdarstellung der Vorzug zu geben ist, sollte bei weniger als 5-7 Optionen eine Darstellung mit Hilfe einer Toolbar oder mit Navigationsknöpfen erfolgen.

Die IML unterschiedet Navigationsmuster und Kommandos, so dass hier eine direkte Differenzierung möglich ist. Das DiLL-Modell kann sowohl Kommandos als auch explizit Navigationselemente in Form einer Navigationsgruppe mit entsprechenden Buttons aufnehmen.

Im Folgenden werden die Navigationskonzepte der Dialog Layer Language näher beschrieben.

5.3.1 Funktionen

Der folgende Abschnitt beschreibt den Zugriff auf notwendige Funktionen der Anwendung mit Hilfe unterschiedlicher Konzepte.

5.3.1.1 Globale Funktionen

Globale Funktionen haben die Eigenschaft, über das Hauptfenster immer verfügbar zu sein. Ein Reihe globaler Funktionen ist für GUI-Anwendungen oft schon definiert, beispielsweise aufgrund des Template-Konzepts auch für den Referenz-IML-Oberflächengenerator.

Zudem können vom Entwickler im IML-Modell weitere globale Funktionen definiert werden. Durch eine unterschiedliche Interpretation der InteractionCases und des Haupt-InteractionCase ist es möglich, die globalen Funktionen als lokale Funktionen des Haupt-InteractionCase zu definieren.

Ein Transformator hat beispielsweise die Möglichkeit Definitionen von Gruppen und Funktionsknöpfen mit entsprechenden Schlüsselwörtern auf diese Weise auszuwerten.

Globale Funktionen werden im Hauptmenü der Anwendung dargestellt. Bei einer Windows-Applikation können wichtige Funktionen zudem als Icons in der Toolbar definiert werden, was einen schnelleren Zugriff ermöglicht.

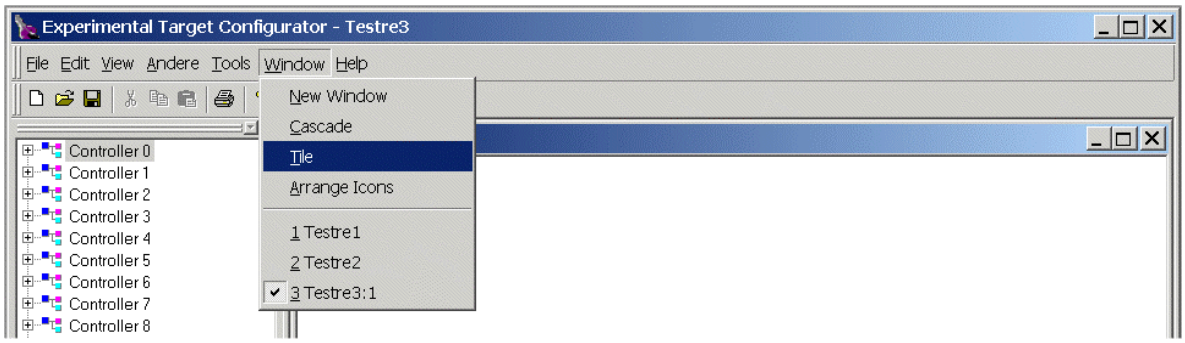

Abbildung 39: Globale Funktionen zur Fensterverwaltung

5.3.1.2 Lokale Funktionen

Während die globalen Funktionen zentral über das Hauptfenster verfügbar gemacht werden, sind lokale Funktionen nur in einem bestimmten Kontext aktiv. Sie müssen zu diesem Zweck in Form von

- Context-Menüs (CONTEXT_MENU), meist als Pop-Up-Menü dargestellt und gültig nur im Bereich unterhalb der Definition, oder
- Funktions-Elementen (NEW_FUNCTION_ELEMENT), meist als Knopf am rechten Dialograndargestellt und gültig für den gesamten aktuellen Dialog,

definiert werden.

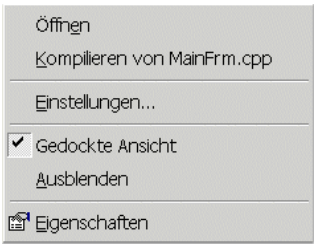

Abbildung 40: Lokale Funktionen auf einer Datei-Ansicht (Quelle: Microsoft Visual Studio 6.0)

5.3.1.3 Menüeinträge für explizit angelegte lokale Funktionen

Statt einer Darstellung als Funktionsknöpfe können explizite Funktionen auch innerhalb einer Menüstruktur eingeordnet und dem Anwender zur Verfügung gestellt werden (vgl. [Hofmann 1998]). Dabei kann es sich sowohl um ein Drop-Down-Menü als auch um ein Pop-Up-Menü handeln. Ist ein Drop-Down-Menü im Unterfenster vorhanden, können Funktionen vom Transformator dort eingefügt werden, sonst bleiben nur Pop-Up-Menüs, Funktionsknöpfe und das Hauptmenü.

Im Fall des Pop-Up-Menüs kann die Anwendung auch auf eine Gruppe oder ein Element beschränkt bleiben – das Menü ist dann im gewählten Kontext immer gültig. Soll diese Beschränkung auch für ein Drop-Down-Menü möglich sein, muss die Bibliothek einen Mechanismus zur Verfügung stellen, der die Menüoption im Bedarfsfall ausblenden oder besser inaktivieren kann.

Die Informationen im IML-Modell genügen für kontextabhängige Funktionen, da die Angaben im CONTEXT_MENU sowohl für ein Pop-Up-Menü als auch für ein Drop-Down-Menü gelten können.

Zudem sieht die IML vor, dass übergeordnete Menüs referenziert werden können, so dass bestimmte Funktionen auch gruppen-, dialog- oder use-case-weise abgeschaltet werden können. Die An- und Abschaltung kann dabei vom Transformator als Systemfunktion definiert und im Modell eingesetzt werden.

Werden nur wenige Funktionen angeboten, sind Funktionsknöpfe gegenüber Menüs aus softwareergonomischer Sicht im Vorteil. Für die Ausführung ist dann nur ein Schritt notwendig, der zudem

Bezug auf ein permanent sichtbares Objekt nimmt [Hofmann 1998]. Bei häufiger und dialogübergreifender Verwendung können die entsprechenden Funktionen auch in die Toolbar integriert werden.

5.3.1.4 Anordnung von Funktionen

Prinzipiell könnten alle Funktionen einer untergeordneten Ebene im Hauptmenü aufgenommen werden. Dass sich dies in der Praxis als wenig sinnvoll erweist, kann schon an einem einfachen Beispiel nachvollzogen werden.

Beispiel: Eine Funktion „Namen aus Liste hinzufügen" könnte sich bei zwei offenen Fenstern einer Lieferantenverwaltung einmal auf die Firma und einmal auf den Ansprechpartner beziehen. Eine Unterscheidung ist deshalb bei Mehrfachverwendbarkeit in unterschiedlichen Unterfenstern niemals gewährleistet.

Mit Hilfe von lokalen oder globalen Einstellungen muss deshalb gewählt werden können, welche Teile wirklich übernommen werden. Per Definition sind dies zumindest die global verfügbaren Funktionen.

Die am häufigsten verwendeten oder wichtigsten Funktionen sollten zudem in eine Werkzeugleiste (Toolbar) integriert werden. Wichtig ist dabei, dass auch Informationen zur Verfügung stehen, die zu einem Tooltip[34] verarbeitet werden können.

Zumindest für die wichtigen Funktionen sollte möglichst ein Shortcut angelegt werden, es sei denn dass sich hierdurch die Gefahr der ungewollten Aktivierung einer irreversiblen Operation erhöht.

Wenn bei Objektoperationen nicht klar ist, auf welches Objekt die Funktion angewendet werden soll, kann eine Auswahl in Form eines der Funktion vorgeschalteten modalen Auswahldialogs, oder bei kleinen Mengen über ein Sub-Menü, erfolgen.

Im IML-Modell werden vom Aktor aufrufbare InteractionCases auf die Zuordnung zu einem vordefinierten Menü überprüft.

Die schon im IML-Modell vorhandene Zuordnung zu einer Befehlsgruppe (CommandGroupSection) innerhalb des Menüs führt zu einer Einordnung der Gruppe in ein Untermenü oder zur Abtrennung von den anderen Gruppen mit Hilfe eines Separators. Für die Wahl des Verfahrens ist die Größe der Befehlsgruppe und der noch verfügbare Platz im Menü ausschlaggebend.

Die Anordnung der Befehle innerhalb einer Befehlsgruppe erfolgt in derselben Reihenfolge, die im Modell vorliegt.

Grenzen: Wie in vielen anderen Bereichen, ist es auch in Menüs ergonomisch ungünstig, mehr als 7 Einträge zusammen anzuzeigen. Dies gilt sowohl für die Anzahl Menüs in der Menüleiste, als auch für die Punkte innerhalb eines Menüs oder Sub-Menüs. Werden die Befehlsgruppen durch Separatoren abgetrennt, dürfen diese theoretisch wiederum bis zu 7 Elemente enthalten.

Ausnahmen von der 7er-Regel, wie es die Monatsnamen sind, müssen vom Entwickler manuell definiert werden, da der Generator nicht über ausreichende Informationen für eine Entscheidung verfügt. Eine Definition von mehr als 10 bis 12 Eintragen innerhalb eines Menüs ist trotz Unterteilung nicht sinnvoll.

Die sich ergebende Breite aller Menüs sollte die Fensterbreite möglichst nicht überschreiten.

Bezeichnungen in der Menüleiste sollten möglichst aus einem Wort, Elementbezeichnungen innerhalb des Menüs maximal aus zwei Wörtern bestehen. Drei Punkte hinter einer Elementbezeichnung sollten auf einen folgenden Dialog hinweisen.

5.3.1.5 Standardeinträge

Mit dem Anwendungstyp sind meist automatisch schon bestimmte Menüeinträge bekannt, die immer vorkommen. Bei datenbankbasierten, betrieblichen Anwendungssystemen sind dies beispielsweise Funktionen zum Aufruf von Sichten und zur Dateneingabe vorhanden. [Hofmann 1998] beschreibt

[34] Hinweis, der unter Windows z. B. in Form eines kleinen gelben Feldes eingeblendet wird, wenn sich die Maus über dem zugehörigen Element befindet.

außer diesen speziellen Funktionen zudem auch allgemeinere Bestandteile, die sich auf viele GUI-basierte Softwaresysteme übertragen lassen. Beispiele sind die Verwaltung von Fenstern und Benutzereinstellungen sowie die Hilfefunktionen.

Diese Standardeinträge können direkt unterstützt werden. Soll der Transformator allerdings auf möglichst allgemeine Fälle, z. B. auch One-Window-Applikationen, anwendbar sein, dürfen Funktionen wie die Fensterverwaltung nicht automatisch generiert werden. Vielmehr sollte innerhalb der Beschreibung des Hauptfensters und aller weiteren Fenster die Möglichkeit zum Überschreiben der Standardeinstellungen bestehen und der Anwendungstyp (ApplicationType) im IML-Modell eingestellt werden.

Alternativ können auch nur explizit angegebene Funktionen zur Verfügung gestellt werden, was allerdings den Aufwand für das Oberflächenprototyping stark erhöht.

Mit dem Template-Konzept der Endstufe des Referenz-IML-Oberflächengenerators ist die zielsystemabhängige Generierung von Funktionen allerdings möglich.

5.3.2 Buttons und Navigation

Druckknöpfe (Buttons) haben immer applikationsweit die gleiche, per Konstante definierte Höhe, es sei denn, die Schriftgröße ändert sich. Allerdings bleibt der Abstand vom Beschriftungstext zum Rand auch bei einer Änderung der Schriftgröße gleich. Denkbar wäre hier allerdings bei gravierenden Größenunterschieden auch eine skalierbare Anpassung.[35]

Die Breite der Buttons muss für jeden Dialog neu berechnet werden. Sie geht auf die Breite der längsten Beschriftung im Dialog zurück. Hinzu kommt noch ein Abstand, der ebenso der Vertikalen wie auch der Horizontalen konstant hinzuaddiert wird.

Die Breite der Buttons im Dialog sollte möglichst die gleiche sein. Unterschiede dürfen höchstens auftreten, wenn unabhängige Buttons zum Einsatz kommen. Folgende Button-Typen existieren:

- Dialogknöpfe: Aufruf zusätzlicher Sub-Dialoge wie „Datum auswählen...“; diese Buttons stehen nicht am Rand, sondern sind über das Dialogfenster verteilt.
- Funktionsknöpfe: Z. B. rechts untereinander angeordnet, wie „bearbeiten“.
- Navigationsknöpfe: „OK“, „Abbrechen“, etc. ; zumeist unten, nebeneinander angeordnet.

5.3.2.1 Dialogknöpfe

Im Fall der Dialogknöpfe sind unterschiedliche Größen kein Problem, da die Knöpfe nicht gruppiert stehen. Dialogknöpfe werden zudem mit drei Punkten am Ende („...“) versehen, um anzuzeigen, dass ein Subdialog folgt. Knöpfe ohne „...“ lösen Aktionen ohne Folge-Dialog aus.

Am günstigsten ist es, den Dialogknopf an der Stelle im Ablauf zu positionieren, an der die Funktion beschrieben ist, d. h. direkt nach dem vorausgehenden Element. Handelt es sich um Gruppenfunktionen, ist die Positionierung innerhalb der Gruppe prinzipiell frei wählbar.[36] Gruppen-Dialogknöpfe können am unteren, oberen oder rechten Rand der Gruppe oder auch gemischt positioniert werden.

5.3.2.2 Funktions- und Navigationsknöpfe

Die beste Position für Navigationsknöpfe ist der untere Rand, für Funktionsknöpfe der rechte oder obere Rand des Dialogs. Wird die Breite bzw. Höhe des restlichen Fensters gravierend überschritten, sollte eine mehrzeilige bzw. mehrspaltige Anordnung erfolgen. Sind mehrere Funktions- und Navigationsknöpfe vorhanden, so können diese beiden Typen getrennt werden.

Eine weit verbreitete Strategie ist hier eine rechtsseitige Anordnung der Funktionsknöpfe und eine Ausrichtung der Navigationsknöpfe am unteren Rand.

---

[35] Da die Transformation im Zuge dieser Diplomarbeit allerdings Microsoft Ressource-Dateien liefert, wird die Größe in Dialogeinheiten angegeben, die mit der Systemschrift skalieren.

[36] dazu wäre allerdings eine Unterscheidung in Prä-, Post- und Auswahlfunktionen notwendig, die über die jeweilige Position entscheidet

Für die links, rechts oder zentriert ausgerichtete Druckknopf-Gruppe gilt der minimale Abstand zum Fensterrand ebenso wie für andere Interaktionselemente. Im zentrierten Fall liegt zudem der Mittelpunkt der Gruppe auf der Mittellinie des Fensters. Möglich ist auch eine Ausrichtung nach dem Blocksatz. Allerdings variieren hier die Abstände von Fenster zu Fenster ohne dass dies einen Mehrwert bringt.

Bei der Positionierung am rechten Fensterrand stehen nur die Möglichkeiten oben, unten und Blocksatz zur Verfügung, da eine zentrierte Ausrichtung unüblich ist.

Eine einmal gewählte Rechts-, Oben- oder Untenausrichtung und eine gewählte Positionierungsstrategie[37] sollte beibehalten werden. Ein häufiger Wechsel der Strategie führt zur Irritation des Benutzers.

Eine Ausnahme hiervon bilden Listen- oder Auswahlfenster, die – in kompakter Form und deutlich als Sub-Dialoge gekennzeichnet – durchaus absichtlich eine andere Optik aufweisen können.

Zumeist sind Wert-Zuweisungen oder Aktionen mit den Navigationsknöpfen verbunden. Bestimmte Grundfunktionen können von einer Beschreibungssprache bzw. dem zugehörigen Transformator zur Verfügung gestellt werden. In der IML sind die folgenden Buttons vordefiniert:

```
Type (yes | no | ok | cancel | assign | reset | undo_restore | close | next
| previous | to_next | to_previous | fast_forward | fast_backward | to_end
| to_start | filter | finish | button_space | button_show_position)
#REQUIRED
```

Abbildung 41: Navigationsknopftypen in der IML

Die Beschreibungen für Navigationsknöpfe von transaktionsorientierten Anwendungen (vgl. [Hofmann 1998]) lassen sich gut auf den allgemeinen Windows-Dialog-Fall übertragen (siehe Anhang C: Navigationsknöpfe und ihre Bedeutung).

Um sicherzustellen, dass der Entwickler zeitsparend und effektiv eine passende Navigation zur Verfügung stellen kann, wurde als Normalfall die Definition eines Navigationsmusters eingeführt.

Navigationsmuster gruppieren die oben erwähnten Navigationsknöpfe, so dass sie zusammenpassen und in einer korrekten und konsistenten Anordnung auftreten.

```
Type (none | close | edit | multiEdit | standard | decision | partialDeci-
sion | message | wizard | wizard_without_intermediate_finish | wi-
zard_first_page | wizard_last_page | yes_no_cancel | yes_no | ok_cancel |
dataset_full_navigation | dataset_small_navigation | dataset_full_edit |
dataset_full_edit_filter | dataset_small_edit | position_tape |
big_position_tape | big_filter_tape) „edit“
```

Abbildung 42: Vordefinierte Navigationsmuster

Für den Benutzer hat dies eine bessere Erlernbarkeit zur Folge, da immer bekannte Buttons in gleichbleibender Kombination und Reihenfolge auftreten.

Abbildung 43: Beispiel für die Umsetzung der Angabe „dataset_full_edit_filter“

Der Entwickler spart Aufwand und erzeugt standardisierte Buttons mit standardisierten Reaktionen.

[37] Positionierungsstrategien können sein: Linksbündig, rechtsbündig, zentriert, oben, unten, Blocksatz

5.3.2.3 Standardknopf

Der OK-Knopf oder ein vergleichbarer Knopf ist normalerweise der vorselektierte Knopf, es sei denn, dass Löschungen oder andere destruktive Abläufe die Folge sind. In einem solchen Fall sollte der „Abbrechen"-Knopf als Standard selektiert werden. Ein Abschalten der Vorselektierung ist in größeren Dialogen oft sinnvoll, um eine versehentliche Aktivierung, z. B. durch „Return", zu vermeiden.

Die für den Standardknopf auszuführende Aktion stellt den regulären Ablauf dar. Trotzdem ist es zumeist nicht sinnvoll allein für die Reaktionen auf andere Buttons alternative Abläufe anzulegen, es sei denn, dass nicht nur ein Funktionsaufruf, ein InteractionCase-Start oder ein Abbruch erfolgt, sondern mehrere Schritte ausgeführt werden. In diesem Fall ist das Anlegen eines alternativen Ablaufs ratsam.

Regeln:

- Eine einmal gewählte Knopfausrichtungsstrategie wird beibehalten
- Ordne Navigationsknöpfe unten horizontal rechtsbündig, Funktionsknöpfe rechts vertikal nach oben bündig und Dialogknöpfe an ihrer Definitionsstelle an.
- Dialogknöpfe tragen immer eine Beschriftung, die den Titel des folgenden Sub-Dialogs, gefolgt von drei Punkten („..."), enthält.
- Navigationsmuster stellen den Standardfall dar und sollten, um Konsistenz zu wahren, nur wo unbedingt nötig ersetzt oder ergänzt werden.
- Navigationsknöpfe werden für die Einträge NAVIGATION_ELEMENT und NEW_NAVIGATION_ELEMENT generiert.
- Funktionsknöpfe werden für die NEW_FUNCTION_ELEMENT-Einträge generiert, in bestimmten Anwendungsfällen auch für CONTEXT_MENU-Einträge auf INTERACTION_CASE-Ebene.

5.4 Layout-Grundsätze

Bei der Transformation des IML-Modells in ein DiLL-Modell erfolgt die Erstellung eines grundlegenden Layouts. Die hierzu notwendigen Vorüberlegungen sind in diesem Abschnitt wiedergegeben.

Spaltenumbrüche, layoutgetriebene Vergrößerungen und andere für die Layoutoptimierung wichtige Teile sind einschließlich der hierzu notwendigen Vorüberlegungen in Kapitel 6 wiedergegeben.

5.4.1 Standardisierung

Mit der Lernförderlichkeit, der Erwartungskonformität und der Selbstbeschreibungsfähigkeit verbessert sich die Anwendung bei zunehmender Standardisierung gleichzeitig in drei Kategorien der ISO 9241 [ISO 1998].

Standardisierung ist also ein wichtiges Konzept für die Benutzbarkeit von interaktiven Anwendungen. Der Benutzer muss hinter dem Dialog ein Konzept und damit ein Muster erkennen können, um zu vermeiden, dass für jeden Dialog ein Konzept neu verstanden und gelernt werden muss.

Abstände und Linen, d. h. auch Zeilen und Spalten, müssen nach einem möglichst regelmäßigen und wiederkehrenden Muster aufgebaut und erkennbar sein. Werden Zeilenabstände nicht eingehalten, wirkt der Dialog verwirrend und es kommt zu Fehlinterpretationen.

Regularität und Vorhersagbarkeit [Kruschinski 1999] sind direkte Folgen einer Standardisierung und haben ihrerseits wiederum Einfluss auf die oben genannten Größen Lernförderlichkeit, Erwartungskonformität und Selbstbeschreibungsfähigkeit.

Gerade in der Standardisierung liegen auch die Stärken von modellbasierten Oberflächengeneratoren, da ein hohes Maß an Standardisierung ohne erheblichen Mehraufwand für die Entwicklung erreicht werden kann und rein manuelle UI-Entwicklung in diesem Punkt selten bessere Ergebnisse erzielt.

Allerdings gibt es auch semantische Implikationen mit Bezug auf die Erwartungskonformität, die schwieriger mit einem Transformator oder Generator abzubilden sind als das rein syntaktische Layout mit Hilfe von Abständen und Gittern.

Beispielsweise sollte ein Feld mit Zugriff auf denselben Wert in einem Folgedialog möglichst an der gleichen oder zumindest an einer ableitbaren Stelle positioniert werden. Diese Positionierung kann nur mit zusätzlichen (semantischen) Informationen aus dem Modell durchgeführt werden.

Regeln:

- Generelle Linksausrichtung der Eingabeelemente und der Labels.
- Ausrichtung an wenigen, starken optischen Linien.
- Kontrolle in vorangehenden InteractionCases auf Variablen, die im aktuellen InteractionCase angezeigt werden.
- Verwendung weniger, passend eingesetzter Standardelemente der jeweiligen GUI-Bibliothek.
- Wiederverwendung / Mehrfachverwendung von InteractionCases unterstützen.

5.4.2 Kopplung und Zusammenhalt

Im Software Engineering sind Kopplung und Zusammenhalt wichtige Kriterien für die Unterteilung in Module und Pakete. Ähnliche Zusammenhänge lassen sich auch bei der Aufteilung von Informationen in Gruppen erkennen.

Informationsgruppen sind Blöcke von Informationseinheiten, die als zusammengehörig zu betrachten sind. Dies kann sowohl syntaktisch, z. B. bei Elementen einer Liste, als auch semantisch, z. B. in Form eines Anschrift-Blocks, der Fall sein.

Während syntaktische Zusammengehörigkeit anhand der Definition erkennbar ist, ist die rechnergestützte Erkennung einer semantischen Zusammengehörigkeit ohne weitere Informationen nicht möglich.

Erkannte Gruppen müssen vom Entwickler als solche modelliert und vom Transformator zusammen dargestellt werden. Je nach Größe muss dies in unterschiedlichen Interaktionsbausteinen geschehen, z. B. in einem Rahmen, einer Registerkarte oder einem Fenster.

Über die Größe einzelner Gruppen und die Gruppenanzahl in einem Fenster gibt es unterschiedliche Auffassungen in der Literatur. Die in der Theorie vorgeschlagenen 19 bis 40 Gruppen pro Fenster erweisen sich in der Praxis als zu großer Richtwert.[38]

Der Benutzer sollte die Elemente einer Gruppe mit einem Blick erfassen können. Als Richtwert bietet sich daher ein Wert von 7±2 an. Basis für diese Zählung sind dabei Einzelelemente wie Radio-Buttons oder Eingabefelder. Untergruppen müssen etwa doppelt gewichtet werden, eine genauere Gewichtung müsste sich an der Anzahl der Elemente und an deren Größe orientieren.

Regeln:

- Elemente in Gruppen werden zusammen in einem Block dargestellt.
- Semantische Zusammengehörigkeit muss vom Entwickler definiert werden.
- Die Anzahl der Elemente in einer Gruppe sollte 7 (maximal 9) nicht übersteigen.

[38] vgl. [Galitz 1993], [Weisbecker 1995], [Kruschinski 1999]

5.4.3 Lenkung der Aufmerksamkeit

Um den Benutzer zu entlasten, ist eine Lenkung seiner Aufmerksamkeit unerlässlich. Das beste Mittel ist eine geeignete Anordnung und die Hervorhebung. Akustische Mittel oder Ausblenden sind innerhalb einer technischen oder kaufmännischen Anwendung nur in wenigen Fällen sinnvoll. Zu häufiger Einsatz wirkt störend oder verwirrend. In speziellen Umgebungen (z. B. Cockpit-Anwendungen) kann ihr Einsatz allerdings sinnvoll sein.

Negative Wirkung von Hervorhebung in Dialogen: Größenunterschiede lassen sich eher bei geometrischen Objekten einsetzen. Kontrast geht zumeist mit farblichen Anpassungen einher. Die Umrandung eignet sich eher für die Gruppierung als für die pure Hervorhebung. Isolierung lässt sich ebenso wie die abweichende Orientierung nur begrenzt zur Hervorhebung einsetzen.

Blinken lenkt und stört die Aufmerksamkeit erheblich, so dass es nur modal eingesetzt werden kann und bei Bestätigung des Benutzers sofort verschwinden muss. Die unaufdringlichste und im Dauereinsatz beste Möglichkeit ist die Anordnung in Leserichtung: Von links oben nach rechts unten.[39] Am sinnvollsten ist also zumeist die Nutzung von Farbe/Kontrast in Form des „Highlighting“ und die Positionierung.

Für Fehlermeldungen, also vom Normalverlauf abweichende Abläufe, sind allerdings Änderungen nötig. Für diese Meldungen erfolgt eine Definition von Containern, die die Nachricht dem Nachrichttyp entsprechend visualisieren.[40]

Der Weg des Auges orientiert sich an bestimmten Merkmalen. Er verläuft [Kruschinski 1999]:

- Von stark gesättigten Farben zu schwach gesättigten,
- von dunklen Bereichen zu hellen,
- von großen Objekten zu kleinen und
- von ungewöhnlichen zu gewohnten Formen.

Regeln:

- Ausblendung ermöglichen.
- Anordnung von links oben nach rechts unten: Spaltenweise; Elemente in voller Breite sorgen für eine Trennung in „oberer Teil“ und „unterer Teil“.
- Zuordnung von Status, Hinweis, Fehler etc. zu Containern.
- Container visualisieren Meldungen auf eine festgelegte Art.
- History in Containern für entgangene Meldungen möglich.
- Stehen sehr wichtige Elemente nicht im Ablauf vorne, erhalten sie einen Rahmen oder eine andere Markierung.

5.4.4 Unterscheidbarkeit

Symbole sind die am besten unterscheidbare Codierungsform. Bildliche Form, Position, Winkel, Farbton und geometrische Form ermöglichen noch etwa 5 bis 10 Stufen [Herczeg 1994].

Regeln:

- Toolbar nutzen.
- Icons unterstützen.
- Bildhafte Ergänzung zulassen.
- Farben nur zur Markierung, nur bei wenigen Stufen zur Unterscheidung einsetzen.

[39] bei europäischem Gebrauch, jedoch z. B. nicht arabisch wegen anderer Leserichtung
[40] in Kapitel 4.5.2.4 wurde das Container-Konzept schon näher beschrieben

5.4.5 Groß-/Kleinschreibung

Die gemischte Schreibweise von Wörtern, d. h. z. B. Adjektive klein, Substantive am Anfang groß, ist in Büchern selbstverständlich. Oft werden allerdings in Anwendungen andere Schreibweisen verwendet.

Untersuchungen in [Rehe 1974] und [Moskel 1984] haben jedoch ergeben, dass Benutzer die gemischte Schreibweise über 13% schneller lesen können als dies bei reinem Einsatz von Großbuchstaben der Fall ist. Auch die Fehlerrate wird geringer.

Daraus lässt sich schließen, dass der kognitive Aufwand für Wörter aus Großbuchstaben höher liegt als für Wörter in gemischter Schreibweise. Es sollten deshalb nicht nur Großbuchstaben verwendet werden.

Regeln:

- Gemischte Schreibweise.
- Untersuchung auf erhöhte Mengen von Großbuchstaben, die nur in Abkürzungen nicht zu vermeiden sind. Mögliche Grenze: Mehr als 60% Großbuchstaben bei einer Länge von mindestens fünf Buchstaben.

5.4.6 Sequenz

Mit Hilfe der Sequenz wird das Auge des Benutzers durch den Dialog geführt. Maßgebliche Informationen müssen dabei so positioniert werden, dass der natürliche Weg zum richtigen Zeitpunkt an Ihnen vorbeiführt.

Vergleichbar ist diese Forderung mit dem Layout einer Zeitung: Hier werden ebenfalls die wichtigsten Informationen (im Falle von Interaktionen auch Eingaben) links oben platziert bzw. durch Mittel wie Größe, Schriftmerkmale, Rahmen etc. hervorgehoben. Hat ein Zeitungsleser den grundlegenden Aufbau verinnerlicht, kann er auf einen Blick fast die ganze Seite grob klassifizieren. Während der obere, linke Bereich die grundlegenden Informationen enthält, werden im unteren bzw. rechten Bereich nachgeordnete und detailliertere Informationen gegeben.

Ähnlich sollte auch der Aufbau von Dialogen gestaltet werden: Links oben beginnend gibt der Benutzer wichtige Informationen ein. Mit zunehmendem Weg nach rechts oder unten nimmt die Wichtigkeit ab oder es handelt sich um Detailinformationen.

Pflichteingaben sollten in einem Block stehen. Fakultative Eingaben (sogenannte Kann-Eingaben) sollten diesen nachgeordnet sein.

Das Mittel der Sequenz bezieht sich sowohl auf die physische Anordnung im Dialog als auch auf logische Anordnung (Reihenfolge), beispielsweise per Tab-Stop.

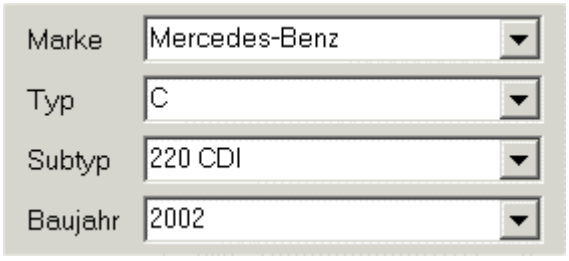

Abbildung 44: Beispiel für den Einsatz des Sequenz-Konzepts

Regeln:

- Untersuchen, ob im IML-Modell als wichtig eingestufte Angaben im Dialogablauf nach vorne gesetzt wurden.
- Die Elemente werden möglichst untereinander in Spalten oder nebeneinander in Zeilen angeordnet, so dass sie einer Linie folgen.

- Gruppen werden nicht umgebrochen, so dass der Blick nicht bei der Optionssuche oder Bearbeitung springt.
- Die Tab-Reihenfolge folgt der spaltenweisen Anordnung und vermeidet Sprünge (kürzester Weg).
- Der Navigationsbereich (Navigationsknöpfe) wird erst nach dem Eingabebereich fokussiert.
- Ist ein Element A von einem Element B abhängig, muss B vor A bearbeitet werden und somit auch physisch und logisch vor A angeordnet werden.
- Abhängigkeiten durch ELEMENT_DEPENDS_ON und ELEMENT_ACTIVE_WHEN werden bei der DiLL-Generierung berücksichtigt oder überprüft.

5.4.7 Simplizität und Klarheit

Der Grundsatz „so einfach wie möglich, so komplex wie nötig“ ist in Bezug auf Simplizität das Leitmotiv.

Wichtig hierbei ist: Es besteht ein gravierender Unterschied zwischen Komplexität und Kompliziertheit. Während die Komplexität eine modell-inhärente Größe wiedergibt, ist die Kompliziertheit lediglich ein Produkt mangelnder Sorgfalt.

Die Komplexität kann lediglich durch Änderungen am (fachlichen) Modell selbst reduziert werden, z. B. durch Einschränkung der verfügbaren Daten. Die Kompliziertheit kann dagegen schon durch ein Redesign der Benutzungsschnittstelle ohne Verlust von Funktionalität oder Information geleistet werden.

Heutzutage stehen umfangreiche Bibliotheken von Oberflächenelementen zur Verfügung. Dank grafischer Benutzungsschnittstellen sowie moderner Monitore und Grafikkarten kann der Entwickler über viele Schriften und eine für das menschliche Auge nicht mehr unterscheidbare Auswahl von Farben (TrueColor) verfügen.

Die Forderung nach Simplizität und Klarheit schließt hier allerdings einen unbedachten Einsatz von zusätzlichen (neuen) Elementen, Schriften und Farben von vorne herein aus. Eine Verwendung aufgrund der technischen Machbarkeit oder bloßen Existenz sollte erschwert und – soweit möglich – verhindert werden.

Regeln:

- Warnung bei komplexen Dialogen mit Hilfe von Metriken (Anzahl Elemente im Dialog / pro Gruppe, Verhältnis überdeckter zur freier Fläche etc.).
- Standardisierung von Schriften und Bildern / Icons.
- Konzentration auf Aufgaben, Ziele und Abläufe mit Hilfe der IML, statt auf Elemente und Farben.
- Automatische Aufteilung in Sub-Dialoge bei zu hoher Komplexität.
- Verwendung von wenigen Standard-Elementen der MFC.

5.4.8 Labels

Zur Anordnung von Labels vor Eingabeelementen gibt es unterschiedliche Ansätze. Methoden, die Eingabeelemente variierend anordnen, scheiden wegen des Kriteriums „virtuelle Linien“ von vorneherein aus.

Die rechtsbündige Ausrichtung am Beginn des Eingabeelements ist aus dem gleichen Grund zumindest problematisch. Sie führt zu einer verstärkten Suchaktivität mangels starker virtueller Linien, die es erleichtern würden, den Textbeginn zu finden. Prinzipiell ist die linksbündige Ausrichtung am Beginn des größten Labels am sinnvollsten.

In früheren IBM-Richtlinien ([IBM 1989a], [IBM 1989b]) wird zur einfacheren Zuordnung bei Linksausrichtung und kürzeren Texten der Leerraum durch nachfolgende Punkte bis zum Eingabeelement gefüllt. Problematisch sind dabei die vielen zusätzlichen Elemente (Punkte/Linien), die das Prinzip der Klarheit und Simplizität verletzen.

Die in [Balzert 1996] vorgeschlagene wahlweise Ausrichtung, abhängig von der Längendifferenz, verstößt ebenfalls gegen das Konsistenzprinzip und Regeln der ISO 9241 [ISO 1998]. Wird eine Rechtsausrichtung gewählt, muss diese beibehalten werden.

Es empfiehlt sich also eine prinzipielle Linksausrichtung. Eine Aufhellung des grauen Hintergrunds hinter dem Label würde die Zuordnung des Labels zum Eingabeelement erleichtern, allerdings auch den Dialog wieder unregelmäßiger erscheinen lassen.

Abbildung 45: Aufhellung des Hintergrunds von Labels zur verbesserten Zuordnung

Grundsätzlich muss jedes Interaktionselement ein Label besitzen, um den Anwender über die Bedeutung des Elements zu informieren. Ausgenommen sind Elemente, die diesen Text schon enthalten. Dabei handelt es sich zumeist um Steuerelemente, also beispielsweise Buttons.

Der Text sollte dabei aussagekräftig, eindeutig, präzise und kurz sein [Kruschinski 1999]. Es sollten deshalb weder ganze Sätze noch Abkürzungen verwendet werden, es sei denn die Abkürzungen sind mit Sicherheit dem ganzen Benutzerkreis bekannt, z. B. „PC“.

Eine Kontrolle auf Sinnhaltigkeit der Texte ist nicht möglich. Allerdings könnte eine Rechtschreibprüfung für Führungs- und Hilfetexte durch das Ansteuern einer externen Rechtschreibprüfung eingesetzt werden.

5.4.8.1 Positionierung

Sinnvolle Positionen für Labels finden sich links und oberhalb, bei „kompakten“ IML-Gruppen (compact_group) auch unterhalb des zu beschreibenden Interaktionselements.

[Kruschinski 1999] sieht für einzeilige Elemente ein Label links vom Interaktionselement, für mehrzeilige ein darüber angeordnetes Label vor. Gleichzeitig kritisiert er allerdings die dadurch ausgelöste Unruhe in Fenstern mit gemischten Elementtypen. Eine gemischte Anordnung im selben Dialog ist also nicht sinnvoll.

Die Anordnung sollte daher für den konkreten Anwendungsfall grundsätzlich nochmals überdacht werden. Dialoge mit obenstehenden Labels wirken kompakter bzw. voller. Linksstehende Labels wirken geordneter, führen aber bei sehr hohen Elementen zu großen Leerräumen.

Zu den mehrzeiligen Interaktionselementen zählen beispielsweise Auswahllisten, Kombinationsfelder (ausgenommen Drop-Down-Combo-Box) und mehrzeilige Eingabefelder.

Steht das Label links von einem Eingabe- oder Auswahlelement, so ist darauf zu achten, dass sich das Label auf gleicher Höhe mit dem Text im Element befindet. Dazu ist es sinnvoll, einen Abstand („Offset“) zu definieren, der die Position relativ zur Position des oberen Elementrandes regelt.

Ausnahmen von der Positionierung vor dem Interaktionselement bestehen nur in folgenden Fällen:

- Das Interaktionselement ist allein in einer Spalte / einem Fenster.
- Das Interaktionselement wurde spaltenübergreifend angeordnet.
- Mehrere Elemente stehen nebeneinander in derselben Spalte.

Da in modernen Systemen derzeit zumeist Proportionalschriften zum Einsatz kommen, ist es eher sinnvoll, die reale Breite zu berechnen und eine maximale Differenz zwischen dem breitesten und

kürzesten Element der Spalte festzulegen. Wird dieser Wert überschritten, kann eine Warnung ausgegeben oder der Text gekürzt werden.

5.4.8.2 Aufhellung

Unter MS Windows 95, aber auch in anderen Systemen, sind die Dialoge durch einen grauen Hintergrund geprägt. Da auf diesem meist schwarze Schrift als Vordergrund verwendet wird, ist der Kontrast nur mäßig.

Einen Beitrag zur einfacheren Zugehörigkeitsbestimmung und besseren Lesbarkeit kann ein Transformator deshalb auch leisten, indem er den Hintergrund des Labels aufhellt. Die Farbgestaltung kann damit verbessert werden, so dass die Augen durch einen höheren Kontrast eine Entlastung erfahren.

5.4.8.3 Doppelpunkte

Für die Verwendung des Doppelpunkts am Ende von Labels gibt es sowohl Fürsprecher ([Galitz 1993], [Microsoft 1996]) als auch Gegner ([Balzert 1996]). Die modernere Variante verzichtet auf den Doppelpunkt, da eine zusätzliche optische Abtrennung von Label und Eingabeelement bei grafischen Benutzungsoberflächen nicht mehr nötig ist.

Bei alphanumerischen Displays war der Nutzen durch diese Trennung noch größer als die negative Wirkung durch informationsloses Rauschen. Der Einsatz von Doppelpunkten empfiehlt sich daher bei modernen GUI nicht mehr. Wird der Doppelpunkt trotzdem noch eingesetzt, muss dies konsistent geschehen.

Regeln:

- Positionierung der Labels linksseitig.
- Stehen mehrzeilige Elemente alleine in einer Spalte oder enthalten Dialoge wenige, große Elemente, können Labels zur Kompaktifizierung oberhalb stehen, dann aber im ganzen Dialog konsistent.
- Stehen mehrzeilige Elemente mit einzeiligen in einer Spalte, stehen Labels links. Ausnahme: Generelle Oben-Ausrichtung.
- Erzeugen von Labels für alle Elemente, ausgenommen Elemente, die den Text des Labels bereits erhalten, z. B. Dialogknöpfe.
- Textkontrolle auf zu lang, Sonderzeichen, Abkürzungen etc.
- Ausrichtung der Labels linksbündig am Beginn des größten Labels. Dieser wird mit minimalem Abstand der Labels zum zugeordneten Eingabeelement berechnet.
- Bei einer Differenz von mehr als ca. 9 Zeichen zwischen längstem und kürzestem Label, sollte eine Anpassung vorgenommen oder eine Warnung ausgegeben werden.
- Berechnung der realen Breite mit Proportionalschrift.
- Der vertikale Abstand („Offset“) vom oberen Rand des Interaktionselements wird definiert als die Randbreite, um die das Label nach unten versetzt wird, so dass der Text des Interaktionselements auf gleicher Höhe liegt.
- Die Hintergrundfarbe des Labels wird aufgehellt (z. B. hellgrau statt grau), um den Kontrast zu erhöhen.
- Großschreibung am Wortbeginn; gemischte Schreibweise.
- Ein Zeichen minimaler horizontaler Abstand zwischen Label und Eingabeelement.
- Vertikale Zentrierung bei einzeiligen Eingabeelementen; entsprechende Position relativ zur ersten Zeile bei mehrzeiligen Eingabeelementen.
- Es werden keine Doppelpunkte am Ende der Labels eingeblendet.

- Syntaktische Überprüfung: Klammern im Text müssen geschlossen sein.
- Die Länge wird kontrolliert und bei Überschreiten einer maximalen Länge (z. B. 40% der maximalen Spaltenbreite) eine Warnung ausgegeben.

5.5 Gruppen

Gruppen dienen der Übersichtlichkeit und semantischen Zuordnung. Das Gestaltungsmittel Gruppierung erleichtert somit den Ablauf kognitiver Prozesse beim Benutzer.

5.5.1 Layout-Regeln für Gruppen

Wichtig ist, dass trotz weitgehender Separierung der Gruppen über das Mittel der Nähe von Elementen und der Entfernung von anderen Gruppen eine optisch sichtbare Verbindung bestehen bleibt. Sind die Abstände zwischen den einzelnen Gruppen zu groß, zerfällt das Fenster in Einzelteile, die als zusammenhangslos wahrgenommen werden.

Zusammengehörige Informationen müssen also möglichst immer mit demselben, geringen Abstand dargestellt werden. Bei nicht zusammenhängenden Informationen ist im allgemeinen die Darstellung im selben Fenster nicht wünschenswert, so dass die oben beschriebene Zersplitterung von Dialogfenstern generell nicht möglich sein sollte.

Innerhalb von Gruppen gelten wiederum zusätzliche Layout-Regeln. Ein passender Abstand zwischen Elementen und Gruppenrahmen muss gefunden werden, der allerdings den Abstand gruppenloser Elemente nicht überschreiten sollte.

Der optische Eindruck ist dabei abhängig von der Höhe einer Textzeile, die in zeichen-/textorientierten Dialogen das Grundmaß darstellt. Für andere Dialogformen, wie z. B. grafische Spielprogramme, müssen andere Elemente als Maß verwendet werden.

[Kruschinski 1999] gibt für die vertikalen Randabstände die Höhe einer Zeile, für die horizontalen Abstände vier Zeichenbreiten (mittlere Zeichenbreite) vor. Eine Ausnahme bilden reine Checkbox- oder Radiobutton-Gruppen. Bei diesen genügt ein Abstand von zwei Zeichen.

Allerdings ist diese Ausnahme kaum umzusetzen, da sich bei einer Anordnung von mehreren Gruppen innerhalb einer Dialog-Spalte die Anzahl der virtuellen Linien erhöhen würde. Bei variierenden Abständen wäre zudem die Erwartungskonformität verletzt. Somit ist es sinnvoller, horizontal den gleichen Abstand wie vertikal einzusetzen. Das entspricht einer Zeichenhöhe unabhängig vom Elementtyp.

Bei einzeiligen Checkbox- oder Radiobutton-Gruppen (horizontale Folge) ist zudem darauf zu achten, dass der Abstand zwischen den Elementen gleich bleibt, d. h., dass die Ausrichtung nicht anhand der unterschiedlichen, sondern anhand der maximalen Beschriftungstextlängen erfolgt.

Regeln:

- Abstand zwischen Einzelelementen (atomare Gruppen) ½ Höhe einer Eingabezeile.
- Abstand zwischen Gruppen vertikal ebenfalls ½ Höhe einer Eingabezeile.
- Abstand zwischen Gruppen horizontal: Indirekte Festlegung mit Hilfe des breitesten Elements bzw. der breitesten Gruppe plus der einfachen Höhe einer Eingabezeile als Abstand.
- Abstand des ersten Elements einer Gruppe von deren Rahmen: 1 Textzeile.
- Vertikaler Abstand des letzten Elements einer Gruppe von deren Rahmen: 1 Textzeile.
- Horizontaler Abstand der Elemente einer Gruppe von deren Rahmen: 2 Zeichenbreiten.

5.5.2 Bildung und Typisierung von Gruppen

Die Bildung von Gruppen kann auf unterschiedlichen Ebenen geschehen.

Syntaktisch: Syntaktische Gruppen sind meist unvermeidbar. So stellt beispielsweise eine Radiobutton-Gruppe eine syntaktische Gruppe da. Allein aufgrund der Existenz der Auswahlgruppe existieren auch deren Elemente (Aggregat). Eine einzelne Darstellung ist damit sinnlos.

Semantisch: Semantische Gruppen ergeben sich oft aus dem Anwendungskontext. Sollen Optionen zum Speicherformat dargestellt werden, so haben die einzelnen Punkte einen inhaltlichen bzw. kontextuellen Zusammenhang. Auch hier ist eine Darstellung als Gruppe notwendig.

Ergonomisch: Die Bedienung erfordert oft die zusammenhängende Darstellung von Punkten, ohne dass ein direkter inhaltlicher Zusammenhang besteht. Oft handelt es sich dabei um eine Informationsabhängigkeit, die den Benutzer zum Nachschauen von Werten oder zur vorhergehenden Eingabe eines bestimmten Wertes zwingt. Es kann auch nur eine bestimmte Eingabereihenfolge wünschenswert sein oder ein bestimmter Elementtyp einfacher in der Gruppe einzugeben sein.

Lose Gruppen: Zusätzlich zu den oben beschriebenen Gruppen gibt es noch „lose Gruppen“, deren einzige Auswirkung die gleichzeitige Darstellung ist. Auswirkungen hat dies vor allem, wenn aus Platzgründen Register eingesetzt werden, da über diese Gruppen bestimmt werden kann, welche Elemente auf derselben Seite dargestellt werden müssen.

5.5.3 Registertypen

Mit Hilfe des Gruppierungselements Register können mehrere Gruppen oder InteractionCases in einem Dialogfenster zusammengefasst werden, auch dann, wenn der sichtbare Platz im Fenster nur für jeweils einen Block ausreichen würde.

Ein häufiges Einsatzgebiet für dieses Gruppierungselement sind Einstellungs-/Optionsdialoge: Hier kann eine Vielzahl von Einstellungen vorgenommen werden, die den Rahmen eines rein zweidimensionalen Dialogs sprengen würden. Hinzu kommt, dass oft nur Einstellungen zu einem bestimmten Themenbereich gemacht werden, so dass die Einteilung in einzelne Interaktionsblöcke und deren Verteilung auf Registerkarten durchaus Sinn machen.

Register können in drei unterschiedlichen Ausprägungen verwendet werden (vgl. [Kruschinski 1999]):

- ganzseitige Register mit integrierten Navigationselementen
- ganzseitige Register ohne integrierte Navigationselemente
- integrierte Register
- Wizards

Bei ganzseitigen Registern nimmt das Register im Fenster eine Vorrangstellung ein: Es dominiert und strukturiert das gesamte Dialogfenster. Die verwendeten Elemente werden mit wenigen bis keinen Ausnahmen auf den Registerkarten angeordnet.

Die Navigationselemente (im allgemeinen Buttons) sind bei **ganzseitigen Registern mit integrierten Steuerelementen** ebenso wie die restlichen Interaktionselemente auf den Registerkarten angeordnet. Das Register-Element füllt also den gesamten Raum innerhalb des Fensters aus.

Diese Form ist vor allem dann zu empfehlen, wenn sich auch die Navigationselemente für jede Registerkarte unterscheiden.

Ist dies nicht der Fall, d. h. können alle Registerkarten mit denselben Steuerelementen genutzt werden (z. B. OK, Abbrechen, Hilfe), so empfiehlt sich die Verwendung von **ganzseitigen Registern ohne integrierte Steuerelemente**. Diese Variante ist die derzeit am häufigsten verwendete.

Um eine lineare Durchwanderungsstrategie ergänzt, die nur die automatische und strikt aufeinanderfolgende Anzeige der Seiten ermöglicht, kann das integrierte Register leicht zu einem **Wizard** umfunktioniert werden.

Eine weniger dominante Form der Register sind **integrierte Register**. Hier tritt das Register nicht als Quasi-Fenster auf, sondern spielt die Rolle eines Elements bzw. einer Gruppe. So können beispielsweise unterschiedliche Datensätze oder Tabellen in den einzelnen Registerkarten aufgelistet werden.

Zusätzlich zu den o.g. drei Formen existieren noch **Mischformen**. Beispielsweise sind auch halbhohe Register möglich, die Interaktionselemente enthalten. Allerdings wird damit auch automatisch die Dialogstruktur komplizierter, so dass deren Einsatz, obgleich technisch möglich, wenig sinnvoll erscheint.

Eine Ausnahme ist beispielsweise das Anzeigen von Hilfe- bzw. Hinweistexten oder Suchergebnissen innerhalb eines gesonderten Rahmens.

5.5.4 Layout von Registern

Werden ganzseitige Register verwendet, sollte darauf geachtet werden, dass keine Scroll-Leisten im Fenster dargestellt werden oder Teile des Registers nicht sichtbar sind. Generativ wird das durch einen Zuschnitt auf eine Mindest-Bildschirmauflösung des Zielsystems erreicht.

Der Abstand zum Fensterrand ist weitgehend frei wählbar. Es ist aber sinnvoll etwa ½ bis 1 Zeilenhöhe in allen Richtungen einzuplanen.

Einzelne Registerkarten sind zwangsläufig immer gleich groß. Deshalb ist es erstrebenswert, einen möglichst ähnlichen Informationsumfang auf allen Registerkarten zu platzieren. Dazu ist es manchmal nötig, die Informationsmenge in komplexen Registerkarten einzuschränken. Möglich ist beispielsweise eine Auslagerung von Kann-Attributen in einen zusätzlichen Dialog oder eine andere Registerkarte.

Das Erscheinungsbild sollte zwischen den einzelnen Registerkarten nicht zu stark variieren. Gemeint sind hier die Gestaltungsprinzipien bzw. der Stil. Natürlich kann dabei bei Bedarf die Spaltenzahl oder die Art der Elemente variieren. Wichtig ist, dass der Anwender erkennen kann, dass er sich noch im selben Fenster, d. h. im selben Kontext, befindet.

5.5.4.1 Auslagerung von größeren Elementen

Mehrzeilige Textfelder, Tabellen, Grafiken und andere große Elemente können bei Bedarf auf Registerkarten ausgelagert werden. Speziell bei Tabellen und großen mehrzeiligen Textfeldern ist eine Auslagerung in eine nur für dieses Element reservierte Seite (Registerkarte) sinnvoll.

Als Beschriftung des Reiters an der Registerkarte kann dann einfach der Text des Elementlabels gewählt werden.

Speziell große, kombinierte Elemente wie Doppellisten zur M-aus-N-Auswahl eignen sich zur Auslagerung. Für die Grenz-Größe ab der die Auslagerung erfolgt, muss ein Schwellwert bestimmt werden.

5.5.4.2 Auslagerung von größeren Gruppen

Gruppen können ebenfalls sehr leicht ausgelagert werden. Allerdings sollten nur große Gruppen eine eigene Registerkarte bekommen. Bei zwei Gruppen, die knapp die Hälfte des verfügbaren Platzes einer Registerkarte beanspruchen, kann ein Reiter mit einer Doppelbeschriftung aus den beiden Gruppenbeschriftungen gebildet werden.

Wird eine einzelne Gruppe ausgelagert, so ist darauf zu achten, dass das Label nicht zusätzlich zur Beschriftung des Reiters eingeblendet wird. Auch der Gruppenrahmen muss ausgeblendet werden. Beides gilt nicht, wenn mehrere Gruppen zusammen auf einer Registerkarte platziert werden.

Gruppen eignen sich bei passender Größe sehr gut für die Auslagerung in eine eigene Seite. Ein Schwellwert sollte auch hier die Höhe regeln, ab der eine Auslagerung erfolgen kann. Nach [Kruschinski 1999] hat sich ein Wert ab der Höhe von fünf einzeiligen Elementen inklusive Abständen bewährt.

5.5.5 Auswahl der passenden Gruppendarstellung

Gruppen können auf verschiedene Arten dargestellt werden (vgl. [Hofmann 1998]):

- Nicht optisch sichtbar, d. h. beispielsweise unter Beibehaltung der Reihenfolge oder als Permutation der enthaltenen Elemente
- Gesetz der Nähe. Reine Gruppierung über die Nähe ohne weitere Hilfsmittel
- Als Gruppierungselement mit Gruppenrahmen mit oder ohne Überschrift
- Auslagerung in einen Sekundärdialog. Der Druckknopf, der den Sekundärdialog öffnet erhält als Aufschrift die Bezeichnung der Gruppe mit angehängtem „...“, das Sekundärfenster erhält den gleichlautenden Titel ohne Punkte.
- Positionierung auf eine eigene Registerkarte. Reicht der Platz nicht aus, werden Kann-Attribute auf die folgende Registerkarte ausgelagert.

Wird die Darstellung durch das Modell eindeutig vorgegeben, so muss diese Entscheidung auch umgesetzt werden.

Eine Gruppe kann bei HERBS in mehreren Formen vorkommen, die im Folgenden näher beschrieben werden:

- Unsichtbare Gruppierung
- Gerahmte Gruppe
- Aggregatlose Gruppe
- Sekundärdialog
- Registerkarte
- Wizard-Seite

Die Entscheidung, welche Form vorzuziehen ist, muss bei fehlender Vorgabe vom Transformator selbst getroffen werden. Generell sollte bei Überschreitung einer bestimmten Gruppengröße die Registerkarte vorgezogen werden. Ist die Gruppe relativ klein, so ist die gerahmte Gruppe besser: bei fehlender Beschreibung durch einen Namen die unsichtbare Gruppierung.

Der Einsatz einer Wizard-Seite ist nur bei streng sequentieller Bearbeitung sinnvoll. Sonst wirkt ein Wizard einschränkend und stört durch den erhöhten Navigationsaufwand.

5.5.5.1 Unsichtbare Gruppierung

Hier dient die Gruppe lediglich als Container (vgl. [Kruschinski 1999]). Die Elemente werden so zusammengefasst, als wären sie Teil einer gerahmten Gruppe.

Mit Hilfe dieser Gruppierungsmethode kann eine Trennung der enthaltenen Elemente verhindert werden. Beispielsweise wird so sichergestellt, dass sich die Elemente nicht über mehrere Spalten verteilen.

5.5.5.2 Gerahmte Gruppe

Die gerahmte Gruppe hat die gleichen Funktionen wie eine unsichtbare Gruppe, allerdings wird die Gruppierung mit Hilfe eines Rahmens visualisiert.

Für den Benutzer ist die Gruppe somit direkt sichtbar. Links oben im Rahmen wird zudem ein Beschreibungstext bzw. Label eingebettet, der bzw. das die Gruppe in Form eines Aggregat-Namens näher beschreibt.

5.5.5.3 Aggregatlose Gruppe

Die aggregatlose Gruppe hat die gleichen Funktionen und die gleiche Darstellung wie eine gerahmte Gruppe. Prinzipiell handelt es sich um eine Gruppe ohne Namen, da die aggregatlose Gruppe Elemente zusammenfasst, die nicht (alle) einer gemeinsamen Bezeichnung für die Gruppe zugeordnet werden können.

Aggregatlose Gruppen sollten nur selten und dann nur zu Strukturierungszwecken genutzt werden. Eine Aufteilung in gerahmte Gruppen ist der bezeichnungslosen „aggregatlosen Gruppe“ immer vorzuziehen.

5.5.5.4 Sekundärdialog

Eine große oder spezielle Gruppe kann in einen Unterdialog oder einen gleichberechtigten Sekundärdialog ausgelagert werden.

Die Gruppe kann dann in fast allen Belangen wie ein abhängiger Dialog (SCREEN) gehandhabt werden.

5.5.5.5 Registerkarte

Das Register eignet sich vor allem dann, wenn mehrere große Gruppen oder Abschnitte vorkommen. Es ermöglicht die Verteilung umfangreicher Interaktionen auf mehrere Seiten desselben Dialogs. Durch eine einfache Navigation können diese trotzdem übersichtlich und gut erreichbar präsentiert werden.

Allerdings sollten kleine Gruppen nicht auf einer Registerkarte allein stehen. Es würde so ein zu großer Leerraum entstehen, da die Größe aller Seiten von der größten bestimmt wird. Entweder prüft man dann ob eine Aufteilung in Registerkarten vom Umfang her überhaupt Sinn macht oder man fasst zwei Gruppen zusammen und ändert die Beschriftung des Reiters in der Form „[Gruppenname1]/[Gruppenname2]“.

5.5.5.6 Wizard-Seite

Eine Wizard-Seite ist eine spezielle Registerkarte. Es wird eine Navigationsbeschränkung eingebaut, die nur eine schrittweise Navigation um eine Seite vor oder zurück ermöglicht. Hierbei handelt es sich meist um die Buttons „Abbrechen“, „< zurück“, „weiter >“ und „Fertig stellen“.

Allgemeines Register und Wizard-Seite schließen sich im selben Dialogfenster gegenseitig aus. Die sequentielle Vorgehensweise muss also für das gesamte Dialogfenster gelten, da bei einer Mischung beider Konzepte die Dialogstruktur unklar wird.

Wizards sollten nur dann eingesetzt werden, wenn ein progressiver, vervollständigender Arbeitsablauf vorgesehen ist, bei dem eine lineare Vorgehensweise die Interaktion vereinfacht und beschleunigt. Die kognitive Belastung wird durch das Ausblenden noch nicht oder nicht mehr benötigter Informationen verringert.

Soll der Benutzer dagegen nur in der Reihenfolge nicht festgelegte Einstellungen tätigen, behindert ihn ein Wizard.

5.5.5.7 Informationen in Wizards

Sind die Abläufe in einem Wizard tatsächlich, wie gefordert, sequentiell, so kann es doch vorkommen, dass Informationen aus vorangegangenen Registerkarten nochmals benötigt werden.

Hierbei gibt es vier unterschiedliche Fälle:

	Selten benötigt	Häufig benötigt
Informationsmenge klein	Einblenden wo benötigt	Dauernd anzeigen: Zusatzfenster oder feststehende Information oberhalb des Navigationsbereichs
Informationsmenge groß	Zugriffsmöglichkeit wo benötigt (Button, Sub-Dialog)	Zusatzfenster, direkter Zugang Möglicherweise ist eine sequentielle Eingabe (Wizard) ungeeignet

Tabelle 2: Darstellung von Zusatzinformationen in Wizards.

Generell gilt dabei: Wo zusammengehörende Informationen auch zusammen eingegeben werden, wird redundante Informationsdarstellung unnötig. Dieser Zustand ist zwar erstrebenswert, jedoch meist nur bis zu einem bestimmten Prozentsatz sinnvoll erreichbar.

5.5.5.8 Regeln für Register, Gruppen und Wizards

Zusammenfassend können für Gruppen, Register und Wizards folgende Regeln aufgestellt werden:

- Ganzseitige Register mit integrierten Steuerelementen Randabstand ½ Dialogzeile in alle Richtungen.
- Ganzseitige Register ohne integrierte Steuerelemente Randabstand ½ Dialogzeile nach oben, links und rechts. Abstand zu den Steuerelementen unten ebenfalls ½ Zeile.
- Layout für ganzseitige Register wie für Fenster.
- Layout integrierte Register genauso wie für Gruppen.
- Logische / semantische Reihenfolge von links nach rechts (1. Schritt links), Wichtigkeit möglichst abnehmend.
- Die Größe des Registers richtet sich nach dem Platzbedarf der größten Registerkarte, d. h. nach dem Platzbedarf der Elemente auf dieser Registerkarte.
- Sind in Bezug auf den Platzbedarfs die Unterschiede zwischen den einzelnen Registerkarten gering, kann ein Ausgleich über eine Vergrößerung von skalierbaren Elementen erfolgen.
- Sind die Unterschiede groß, stehen folgende Möglichkeiten zur Verfügung:
    - der Entwickler bessert nach,
    - die größte Registerkarte wird in zwei separate Registerkarten aufspaltet,
    - zwei der kleinsten Registerkarten werden zusammengefasst,
    - Elemente (z. B. Kann-Attribute) werden in andere / zusätzliche Bereiche oder andere Registerkarten ausgelagert.
- Wird ein Element bei der Bearbeitung von einer folgenden Registerkarte benötigt, dann kann in dieser Form kein Wizard angelegt werden, da eine Einblendung nicht möglich ist.
- Wizard: Zuvor eingegebene und später benötigte Elemente werden eingeblendet.

5.6 Der IML-Transformator

Im Folgenden Abschnitt wird ein Überblick über das Konzept des Transformators gegeben. Die einzelnen Schritte sind in den zugehörigen Kapiteln beschrieben.

5.6.1 Genereller Ablauf

Das vom Entwickler erstellte Modell wird ohne interaktive Zwischenschritte mit Hilfe mehrerer Transformationsschritte in die finalen Artefakte umgewandelt.

Mit Ausnahme der Oberflächengenerierung werden die finalen Artefakte dabei direkt aus dem vervollständigten IML-Modell erzeugt. Die Resultate sind im Kapitel 7 näher beschrieben, der Referenz-IML-Transformator in Kapitel 8.1. Die spezielle Ablaufstruktur für die Oberflächengenerierung wird im folgenden Abschnitt erläutert.

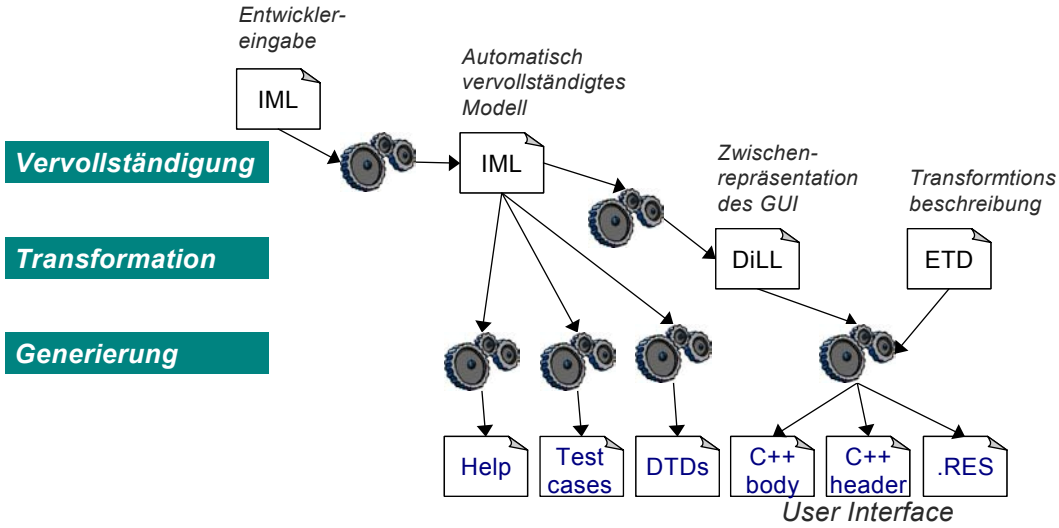

Abbildung 46: Genereller Ablauf des Transformationsprozesses

5.6.2 Oberflächengenerierung mit einer Pipelinestruktur

Während die meisten anderen Resultate ohne Zwischenstufen aus dem IML-Modell bzw. dem vervollständigten IML-Modell generiert werden, durchläuft die Oberflächengenerierung eine Pipeline-Struktur, vergleichbar der Visualisierungspipeline in der wissenschaftlichen Visualisierung [Ertl 2001].

Dies ist notwendig, um die sehr unterschiedlichen Transformationsergebnisse voneinander zu trennen, eine Bearbeitung dieser Ergebnisse zu ermöglichen und zudem jeden abgeschlossenen Transformationsblock unabhängig zu gestalten.

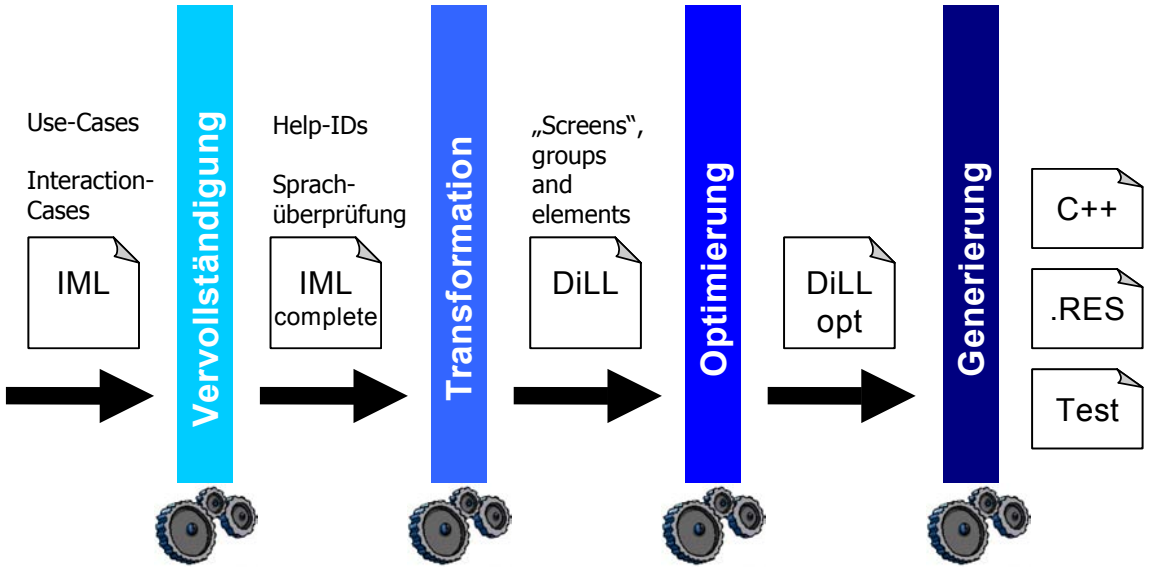

Abbildung 47: Transformations-Pipelinestruktur der Oberflächengenerierung

Vervollständigung: Zunächst wird das vom Entwickler erstellte IML-Modell eingelesen. Die Vervollständigungsstufe enthält eine Konsistenzprüfung, die sowohl auf syntaktischer Ebene durch das XML-Parsing als auch auf semantischer Ebene mit Hilfe von Prüfungen auf Vollständigkeit und Relationen erfolgt. Die eigentliche Vervollständigung ergänzt nach einem bestimmten Schema verschiedene IDs und kann abhängig von den Projekteinstellungen weitere Teile hinzufügen.

- Eingabeformat: IML (XML)
- Ausgabeformat: IML (XML)

Transformation: In der Transformationsstufe wird das IML-Modell über eine Abbildung auf Elemente, Kommandos und Ablaufsteuerung in ein DiLL-Modell transformiert. Aus der ablauforientierten Darstellung in IML wird so eine weitgehend plattformunabhängige Darstellung der zu generierenden Dialogstruktur. Zugleich findet schon bei der Festlegung der Elemente mit Hilfe des Mengengerüsts und anderer Information sowie der Positionierung von Labels ein bedeutender Teil der Optimierung statt.

- Eingabeformat: IML (XML)
- Ausgabeformat: DiLL (XML)

Optimierung: Wie schon erwähnt, enthält die Transformation die semantische Optimierung anhand von IML-modellabhängigen Entscheidungen. Die Optimierungsstufe enthält dagegen die Layoutoptimierung. Hier finden Spaltenumbrüche, Positionsverschiebungen, Vergrößerungen von Elementen und Abständen sowie Auslagerungen in separate Dialoge statt.

- Eingabeformat: DiLL (XML)
- Ausgabeformat: DiLL (XML)

Generierung: Anhand der Beschreibungen innerhalb des DiLL-Modells werden zugehörige Code- und Ressourcenfragmente in den Element-Transformationsbeschreibungen gesucht, ausgelesen und parametrisiert in eine weitgehend lineare Darstellung gebracht. Dann werden zu Zielbetriebssystem und -anwendungstyp passende Templates eingelesen und mit der linearen Darstellung befüllt. Das Ergebnis ist ein unkompilierter Oberflächenprototyp, der nach der Kompilierung ablauffähig ist.

- Eingabeformat: DiLL (XML)
- Ausgabeformat: Code (C++), Ressourcenbeschreibungen (.RC-Dateien)

5.7 Vorgehensweise zur Layoutgenerierung

Die sukzessive Vervollständigung des Layouts geschieht im vom XML-Parser generierten Baum bzw. in dem daraus generierten Objektmodell.

Die Elemente enthalten Attribute, die die relative Position und die Ausmaße angeben. Initial sind dies die minimalen Ausmaße, die sich im Laufe des Transformationsvorgangs noch vergrößern können.

In dem beispielhaft im Rahmen dieser Diplomarbeit implementierten Transformator verfügt jedes DiLL-Objekt[41] über eine Methode zur Berechnung des Platzbedarfs.

Diese Methode („CalculateDimensions“) berechnet für Elemente die Höhe und Breite und gibt diese zurück.

Objekte, die andere Objekte enthalten (Gruppen und SCREENs) berechnen ihre Dimensionen aus dem Maximum der untergeordneten Elemente und der Addition eigener zusätzlicher Werte wie z. B. Titelhöhe.

Das Maximum wird dabei bezüglich der Summe aus Position und Größe bestimmt, so dass für jedes Element eine maximale X- und eine maximale Y-Koordinate errechnet wird, die das Element-Ende markieren.

Auf der Grundlage dieser initialen Größenberechnung können die weiteren Entscheidungen wie

- Verteilung auf mehrere Registerkarten
- Sub-Dialoge
- Spaltenzahl
- Spaltenumbruch
- etc.

getroffen werden.

5.8 Anwendungsmodellabhängige Grundfunktionalität

5.8.1 Vordefinierte Anwendungsmodelle und Funktionen

Mit Hilfe von bestimmten Schaltern ist es möglich, den Anwendungstyp und generische Funktionen anzugeben, die direkt generiert werden sollen, ohne dass zusätzliche Informationen eingegeben werden müssen. Bei dem im Zuge dieser Arbeit erstellten Transformator geschieht dies beispielsweise über die Auswahl von Templates und die Generierung von zusätzlichen DiLL-Objekten.

Ist der Typ Mehrbenutzer gewählt, kann ein Login und eine Benutzerverwaltung generiert werden. Als weitere Stufe kann dazu ein Administrator-Menü angelegt werden.

Geeignet sind diese Schalter vor allem zur Bestimmung zu generierender Standard-Dialoge oder Standard-Funktionen wie der Fensterverwaltung, und zur Bestimmung der zu verwendenden Anwendungs-Templates.

Sollen bestimmte Dinge vom Anwender anpassbar sein, kann für die zugehörigen, generischen Befehle die Zugehörigkeit zu einem Menü über ein Tag definiert werden. Beispielsweise kann das Anpassen der Werkzeugleiste oder die Einstellung von Farben auf diese Weise dem Menü „Einstellungen“ zugeordnet werden.

[41] D.h jedes DiLL-Element, jede DiLL-Gruppe und auch jeder DiLL-SCREEN

Beispiele für generische Funktionen aus [Hofmann 1998] sind:

- Zuordnung von Elementen zur Toolbar
- Anwendungsverhalten (z. B. Sicherheitsabfragen vor Löschen)
- Sprache auswählen (Englisch, Deutsch, ...)
- Anpassung / Änderung von Daten für Auswahlelemente
- Fenster anordnen, kaskadieren, minimieren („ikonisieren“), wiederherstellen, schließen etc.

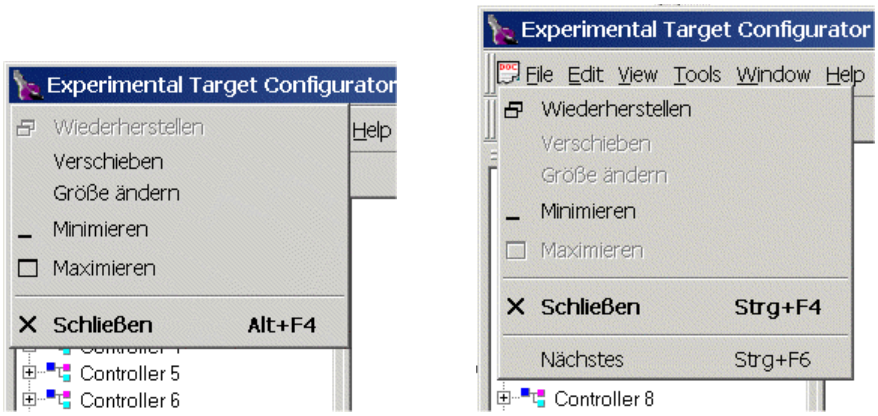

Abbildung 48: Generische Fenster-Funktionen des Hauptfensters und der Unterfenster

Zudem existieren Standardfunktionen, die für alle Anwendungen eines bestimmten Typs ohne explizite Angabe generiert werden können, beispielsweise „Öffnen“, „Speichern“, etc.

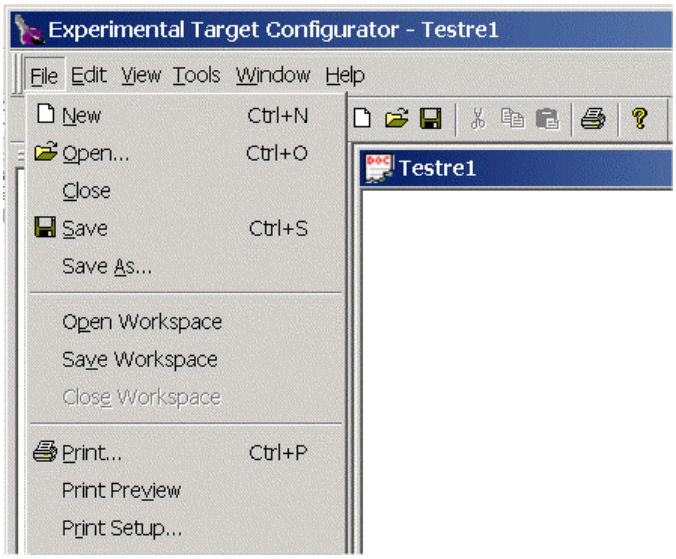

Abbildung 49: Standardfunktionen aus dem File-Menü bzw. der Toolbar ; generische Fenster-Funktionen

Standardfunktionen und generische Funktionen können sowohl über eine Generierung durch den Transformator erzeugt werden als auch durch die in Kapitel 7.1 näher beschriebenen Templates direkt im Code eingebettet sein.

5.8.2 Automatisierte Erstellung von Konfigurationsdialogen

Änderbare Listen und andere Elemente mit Vorgaben benötigen, je nachdem welche Änderungsberechtigung welchen Rollen zugeteilt wurde, unterschiedliche Konfigurationsdialoge. Bei Microsoft-Anwendungen werden diese Einstellungsdialoge meist unter „Extras-Optionen“ aufgeführt, wobei die Zuordnung von Optionen zu Extras zumindest fragwürdig erscheint [Wessel 2002].

Schon mit der Definition einer entsprechenden Interaktion im IML-Modell stehen die zur Generierung von Optionsdialogen nötigen Informationen zur Verfügung. Aus den Rollen, Änderungstypen und der Zuordnung der Listen zu InteractionCases und Use-Cases kann der Transformator direkt Optionsdialoge generieren und diese entsprechend strukturieren.

Die generierten Einstellungs-Dialoge sind, wie alle direkt aus dem IML-Modell abgeleiteten Dialoge, Teil des DiLL-Modells, wo sie in einen oder mehrere Options-SCREENs eingebettet werden.

Sind bestimmte Veränderungen nur von einem Administrator-Aktor vorzunehmen oder betreffen die Änderungen nur die jeweils aktuelle Benutzerrolle, müssen die entsprechenden Dialoge über speziell für diese Fälle gedachte Menüpunkte zugänglich gemacht und auf unterschiedliche Dialoge verteilt werden.

Die generierte Anwendung wird durch entsprechende Optionsdialoge besser individualisierbar, da beispielsweise jeder Benutzer bestimmte Vorgabewerte für sich einstellen und je nach Aufgabe auch wieder ändern kann.

Eine Erstellung von Konfigurationsdialogen durch den Transformator wird durch entsprechende Vorgaben für die Erweiterbarkeit im IML-Modell ermöglicht, so dass der Benutzer eine generierte und damit homogen aufgebaute Konfigurationsmöglichkeit vorfindet.

6 Ergonomische Dialogoptimierung

„Die Anordnung von Dialogelementen in einem Dialog ist zwar für die Benutzerfreundlichkeit der Anwendung nicht unwesentlich, aber dies rechtfertigt keineswegs den Aufwand, den mancher Programmierer auch heute noch in die Layouterstellung steckt. Schließlich kommt es nicht darauf an, ob ein Texteingabefeld ein Pixel weiter links oder weiter rechts steht, sondern darauf, dass beispielsweise links vom Texteingabefeld eine Beschriftung steht, die angibt, was denn in dieses Texteingabefeld einzugeben sei.“ [Nilson 1997]

Viele Teile der Dialog-Optimierung in ergonomischer Hinsicht lassen sich nicht mit Hilfe von Optimierungsverfahren im mathematischen Sinne erreichen. Wegen der Ausrichtung auf den Benutzer können viele Informationen nicht im herkömmlichen Sinne berechnet werden.

Daher besteht ein großer Teil der Optimierungen in der Herleitung und besseren Interpretation von Informationen aus der Spezifikation bzw. dem Interaktionsmodell. Diese entstehen in den frühen Phasen der Entwicklung und stellen damit das Kernthema dieser Diplomarbeit dar.

Das ursprüngliche Konzept zur Dialogoptimierung sah einen von allen anderen Transformations-Schritten unabhängigen Teil vor, der DiLL-Bäume einliest und sie in optimierter Weise wieder ausgibt.

Es hat sich allerdings gezeigt, dass für einen großen Teil der Optimierungen Informationen aus dem IML-Modell benötigt werden, die im DiLL-Modell nicht enthalten sind. Beispielsweise ist nicht bekannt, welche Einstellungen das übergeordnete InteractionCase eines Interaktionsschritts hat. Es ist im DiLL-Modell somit nicht klar, ob die Position des Labels und andere Einstellungen zwingend sind.

Zwar existiert über eine ID-Rückkopplung ein Zugriff auf das IML-Modell, jedoch ist diese Vorgehensweise weder vom Aufwand her vertretbar noch bringt sie eine Verbesserung der Ergebnisse mit sich. Aus diesem Grund wurde die Optimierung in zwei Teile aufgeteilt.

Teil 1 führt das Mapping auf die endgültigen Elementtypen durch und parametrisiert diese schon weitgehend. Auch Längeneinstellungen und Gruppenbildung, die Markierung von fakultativen und zwingend notwendigen Elementen[42] sowie die Generierung und Anordnung von Labels finden in diesem Teil statt.

Da Teil 1 vollständig in die IML-DiLL-Transformation integriert ist, wurden die hierzu notwendigen Regeln, Mengengerüste und Vorgehensweisen schon in Kapitel 5 beschrieben.

Teil 2 führt eine Optimierung des Layouts, d. h. der Anordnung der Elemente durch und hat zudem die Kompetenz, Elemente bei Bedarf und soweit zulässig zu vergrößern und in Unter- oder Folge-Dialoge auszulagern.

Für den zweiten Teil ist die Erarbeitung von Layoutgrundsätzen von großer Bedeutung. Daher werden zunächst die wichtigsten Grundsätze erläutert und die daraus resultierenden Schlüsse und Regeln vorgestellt.

Ein Überblick über Positionierungsverfahren wird ergänzt durch Überlegungen zum automatischen Layout und die hierzu notwendige Gewichtung von Elementen.

6.1 Layoutgrundsätze für die Optimierung

Die im Folgenden beschriebenen Grundsätze stellen die Basis für automatisches Layout und Layoutoptimierungen dar. Die exakte Quantifizierung ist teilweise nur abhängig vom Zielsystem möglich. Soweit möglich werden aber Werte angegeben und Algorithmenentwürfe skizziert.

[42] Sogenannte Kann- und Muss-Elemente

6.1.1 Informationsdichte und anzuzeigende Informationen

Die Informationsdichte in einem Fenster sollte zwischen 25% und 40% liegen [Tullis 1988]. Zu viele Informationen führen zu einer Überladung der Dialoge, zu wenige Informationen dagegen zu überproportional großen Leerräumen und kontraintuitiver Bedienung. Lokal, d. h. innerhalb einer Gruppe, kann dieser Wert überschritten werden und bis zu 60% betragen.

Im Zuge der Aufgabenangemessenheit sollten genau die zur Erledigung der Aufgabe notwendigen Informationen dargestellt werden.

Es sollten nur die Information angezeigt werden, die zur Erledigung der aktuellen Aufgabe notwendig sind. Unnötige Informationen stellen eine vermeidbare kognitive Belastung für den Anwender dar. Beispielsweise ist das aktuelle Datum für die Eingabe des Namens nicht von Belang.

Im Gegenzug sollte auch keine benötigte Information fehlen. Der Aufwand des Benutzers ist hier noch größer: Er muss nicht vorgesehene Arbeitsschritte für die Informationssuche abarbeiten, ohne einen Mehrwert zu erzielen.

Häufig werden Informationen auch als Referenz (Link) eingebunden. Werden diese Informationen immer benötigt, ist diese Vorgehensweise unpassend. Bei geringer Nutzungsfrequenz kann die Referenzierung als ein adäquates Mittel zur Reduktion der primär sichtbaren Informationen genutzt werden. Allerdings muss die Referenz so automatisiert sein, dass ihr der Benutzer mit geringem Aufwand folgen und auch wieder zurückkehren kann (z. B. Button, Hyperlink).

6.1.2 Proportion

[Galitz 1993] beschreibt verschiedene ästhetisch wirkende Seitenverhältnisse:

- Quadrat 1:1
- Verhältnis Wurzel aus zwei 1:1,414 (gleichzeitig DIN A[n] Format, quer)
- Goldener Schnitt mit 1:1,618
- Verhältnis Wurzel aus drei 1:1,732
- Doppeltes Quadrat 1:2

Charakteristisch ist hierbei, dass alle Werte im Bereich von 1:1 bis 1:2 liegen, also immer breiter, oder ebenso breit, wie hoch sind. Es bietet sich daher an, bei Dialogfenstern ein Seitenverhältnis in diesem Bereich zu wählen. Zudem macht auch das Seitenverhältnis der heute gängigen Bildschirme eine eher breite Darstellung sinnvoll.

Die Anatomie des menschlichen Auges, speziell die Augenstellung im Nahsichtbereich, sorgt dafür, dass eine breitere Darstellung angenehmer empfunden wird [Kruschinski 1999].

Eine künstliche, extreme Verbreiterung der Interaktionselemente sollte vermieden werden. Deshalb bietet sich bei umfangreicheren Dialogfenstern eine mehrspaltige (i.a. zweispaltige) Darstellung an.

Regeln:

- Spaltentrennung: Die Fläche der einspaltigen Darstellung aufgeteilt auf Spalten ergibt die theoretische Dialog-Geometrie – noch ohne die Sicherheit einer Umbruchmöglichkeit an genau dieser Stelle:

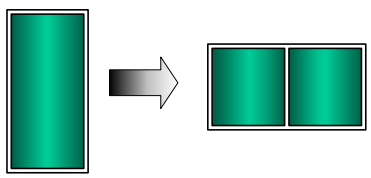

Abbildung 50: Theoretische Dialoggeometrie durch Spaltenumbruch nach Höhe.

- Überprüfung der Fenstergeometrie auf ein Höhe-Breite-Verhältnis zwischen 1:1 und 1:2.
- Abstandsvergrößerung horizontal und vertikal sowie Vergrößerung von Elementen bei Bedarf, d. h. zur Zeilenbildung, Symmetrieunterstützung etc.

6.1.3 Balance

Zieht man eine senkrechte Line in der Mitte des Fensters, so sollte beiderseits dieser Linie die Informationsdichte etwa gleich hoch sein, d. h. die Anzahl vergleichbarer Elemente sollte in weitgehend übereinstimmen. „Vergleichbar" bedeutet in diesem Zusammenhang, dass größere Elemente wie etwa Listboxen eine höhere Gewichtung erhalten als einfache Elemente wie Checkboxen oder einzeilige Textfelder.

Regeln:

- Ermittlung einer flächenorientierten Balance-Metrik, die aus Elementfläche und Sättigung einen Füllgrad berechnet sowie aus den Teilwerten für die einzelnen Spalten eine Balance-Wert ermittelt.
- Ist die Balance nicht innerhalb der Toleranzgrenzen, können Elemente auf andere Spalten verteilt oder schrittweise vergrößert werden. Eine Verkleinerung ist nicht möglich.

Ein Flächenberechnungsmechanismus ist beispielsweise über das für den Transformator entworfene Klassenmodell sehr einfach zu realisieren. Eine schnelle Bewertung von Layouts ist somit über diese Metrik möglich.

6.1.4 Symmetrie

Balancierte Fenster allein sind allerdings nur der erste Schritt in Richtung eines harmonischen Dialogs. Eine Verstärkung ist durch das hier beschriebene Symmetrie-Prinzip möglich.

Dieses Prinzip fordert eine gegenüberliegende Anordnung von gleichen bzw. ähnlichen Elementen. Sind beispielsweise zwei Adress-Blöcke vorhanden, müssen diese gegenüberliegend angeordnet werden. Die Bezugslinie ist dabei auch für die Symmetrie die Senkrechte durch die Fenstermitte [Kruschinski 1999].

„Ähnliche Elemente" können dabei eine Ähnlichkeit in Größe, Farbe, Form und ggf. auch Semantik aufweisen.

Bei Dialogen, die nicht zwei Spalten enthalten, ergeben sich allerdings Probleme mit dieser Definition der Symmetrie. Speziell bei ungeradzahligen Spaltenzahlen zerschneidet die Symmetrielinie eine Spalte.

Balance ebenso wie Symmetrie können auch direkt auf benachbarte Spalten angewandt werden. Trennlinie ist dabei die, evtl. für den Benutzer nicht unmittelbar sichtbare, Spaltentrennlinie.

Regel:

- Wo Elementreihenfolgen nach Maßgabe des IML-Modells variiert werden können, werden Elemente nach Größe und Typ (in dieser Reihenfolge) auf Ähnlichkeit überprüft und möglichst auf gleicher Höhe symmetrisch zugeordnet.

6.1.5 Linien, Harmonie und Ordnung

Der Benutzer bildet intuitiv Linien und daraus Blöcke auf dem Bildschirm. Eine virtuelle Linie setzt dabei existierende Kanten in ihrer gedachten Verlängerung fort.

Jedoch entsteht ein harmonischer bzw. geordneter Eindruck nur dann, wenn die Anzahl der virtuellen Linien gering ist. Das Gesetz der guten Gestalt liefert dabei die Basis für die Dialoggestaltung, wobei große Elemente mehr zum Eindruck beitragen als kleine und rechteckige Elemente gegenüber solchen ohne festen Umriss dominieren.

Eine Reduktion der virtuellen Linien ist vor allem dadurch möglich, dass viele Linien zusammenfallen. Zusätzlich verstärken die Einzellinien dabei die gemeinsame Linie und machen diese deutlicher. Eine versetzte, z. B. treppenförmige, Anordnung stellt also einen Regelverstoß dar. Ein solches Layout vermittelt den Eindruck von „springenden“ Linien.[43]

Außerdem ist darauf zu achten, dass sich bei einer mehrspaltigen Anordnung die in der ersten Spalte zugrundegelegte Linienordnung auch in den folgenden Spalten wiederfindet, da sonst ein Eindruck von Diskontinuität entsteht.

Diese Regel bezieht sich nicht nur auf Elemente, sondern auch auf Gruppen innerhalb eines Dialogfensters.

Generell ist es dabei in einer Layout-Spalte die Orientierung am breitesten Element am einfachsten, so dass z. B. Gruppenrahmen auf die gleiche Breite gebracht werden können.

Regeln:

- Ausrichtung der Interaktionselemente und Labels an „optischen“ Linien.
- Anpassung der Breiten im Rahmen der Vorgaben des Mengengerüsts.
- Anpassung der Gruppenbreite an das breiteste Objekt der Spalte.
- Anpassung der Labellängen an das längste Label.
- Jedes DiLL-Element und jede DiLL-Gruppe verfügt über eine optische Linie und eine optische Zeile als Grundlage für die Linienberechnung.

6.1.6 Ein Konzept zur Berechnung der optischen Linien

Aus dem vorhandenen Grundlayout können die optischen Linien errechnet werden. Ziel ist es dabei, die Zahl der optischen Linien zu minimieren.

Virtuelle Linien können durch verschiedene Dialogbestandteile hervorgerufen werden:

- Beginnendes Label
- Endendes Label
- Beginnendes Interaktionselement
- Endendes Interaktionselement
- Unterbrechung im Interaktionselement

Generell ist darauf zu achten, dass virtuelle Linien beginnender Elemente und virtuelle Linien resultierend aus hinteren Begrenzungen von Elementen nicht zusammenfallen sollten.

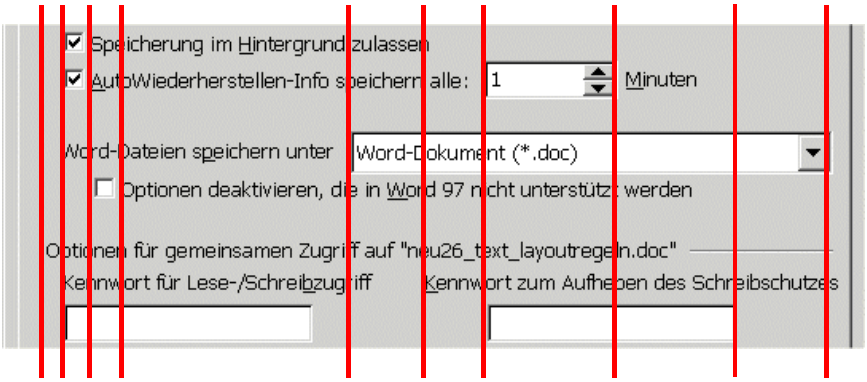

Abbildung 51: Virtuelle Linien in Dialogen (Quelle: MS WinWord 2000 Extras/Optionen/Speichern)

[43] Eine Ausnahme bildet hier eine schräge virtuelle Linie, wenn die Mehrzahl der Elemente auf dieser zu liegen kommt.

Vielmehr sollten Startlinien und ebenso Endlinien möglichst zusammenfallen, zwischen den beiden Linientypen jedoch ein Abstand vorhanden sein. Bei unterschiedlichen Textlängen ist das Ende natürlich variabel, da eine optische Repräsentation der einzugebenden Zeichenmenge für den Benutzer hier wichtiger ist.

6.1.6.1 Beginnendes und endendes Label

Labels erzeugen nur wenig ausgeprägte virtuelle Linien. Trotzdem stellen sie aufgrund ihrer hohen Gesamtzahl ein Problem dar. Die in dieser Diplomarbeit festgelegte generelle Linksausrichtung ermöglicht es, die Anzahl der durch beginnende Labels hervorgerufenen ausgeprägten Linien auf eine pro Spalte zu reduzieren.

Durch das ebenfalls in dieser Diplomarbeit vorgeschlagene Prinzip eines helleren Hintergrunds für Labels können auch die durch endende Labels entstehenden virtuellen Linien zusammengelegt werden. Zwar enden die Texte an unterschiedlichen Positionen, jedoch die gesamten Labels dann alle direkt vor den zugehörigen Interaktionselementen.

6.1.6.2 Beginnendes und endendes Interaktionselement

Beginnende einfache Interaktionselemente können durch Linksausrichtung auf eine vertikale Linie pro Spalte positioniert werden. Hinzu kommen virtuelle Linien, die durch nachfolgende Elemente in derselben Zeile und Spalte entstehen. Hierbei ist darauf zu achten, dass diese nach Möglichkeit auf derselben horizontalen optischen Linie ausgerichtet werden wie vorhergehende und folgende Elemente.

Das Ende der Interaktionselemente lässt sich nur begrenzt anpassen, da die Länge zugleich mit der Kapazität des Feldes korrelieren sollte.

6.1.6.3 Unterbrechung im Interaktionselement

Falls es sich um komplexe Interaktionselemente handelt, können auch innerhalb der Elemente virtuelle Linien entstehen. Beispielsweise kann eine erweiterbare Mehrfachauswahl zu zwei mit Buttons verbunden Listen führen. Anfang und Ende der beiden Listen sowie der Buttons markieren auch virtuelle Linien, die berücksichtigt werden müssen.

Abbildung 52: Erweiterbare Mehrfachauswahl als Beispiel für Elementunterbrechungen

6.2 Positionierungsverfahren für Elemente

Für die Positionierung von Elementen im Dialog gibt es zwei verschiedene Vorgehensweisen:

- manuelle Positionierung und
- schematische Positionierung

Die **manuelle Positionierung** erfordert einen Software-Ergonom. Zwar kann hier individuell und mit einem größeren semantischen Wissen entschieden werden. Jedoch fällt auf diese Weise auch die gesamte Automatisierung weg, und jede Änderung im Anforderungsmodell zieht einen großen Aufwand

nach sich – die Verknüpfung zwischen Anforderungsmodell und Oberflächenprototyp ist somit unterbrochen. Zudem ist oft kein Software-Ergonom über die ganze Projektdauer verfügbar.

Eine **schematische Positionierung** ist die Voraussetzung für eine automatisierte Generierung von Oberflächenprototypen. Mit Hilfe von Regeln können Dialoge generiert werden. Werden dabei Fehler im Anforderungsmodell festgestellt, kann nach erfolgter Verbesserung direkt ein weiterer Generierungsschritt durchlaufen werden.

Da dem Transformator nur begrenzte semantische Informationen zur Verfügung stehen, muss die endgültige Version der Benutzungsoberfläche von einem Software-Ergonomen nachbearbeitet werden. Es sei denn, das Anforderungsmodell ist mächtig genug, um alle erforderlichen Anpassungen des Software-Ergonomen schon hier vorwegzunehmen.

Dies ist die vermutlich nur in Einzelfällen erreichbare Vision der ergonomischen Oberflächengenerierung. Mit dem Einsatz einer sehr detaillierten und benutzernahen Sprache sind jedoch schon viele der benötigten Informationen verfügbar.

Gliederung: Im Folgenden soll nur auf schematische Positionierungsstrategien eingegangen werden, da eine manuelle Positionierung für den Transformator nicht in Frage kommt.

Die schematischen Verfahren können weiter unterteilt werden in:

- sequentielle, die nach der eingegebenen Reihenfolge arbeiten
- nicht-sequentielle, die die Eingabereihenfolge ignorieren

6.2.1 Sequentielle Positionierung

Die sequentielle Positionierung platziert Elemente genau in der Reihenfolge, in der sie übergeben wurden. Im Falle der IML würde die Positionierung also in der Eingabereihenfolge, im Falle des Transformators in dessen semantisch motivierter DiLL-Ausgabereihenfolge vonstatten gehen.

In der Praxis werde drei Positionierungsstrategien unterschieden [Kruschinski 1999]:

- Bei der **einspaltigen Positionierung** werden die Elemente einfach untereinander mit einer berechneten horizontalen Ausrichtung dargestellt. Daher ist eine einspaltige Positionierung von den drei Alternativen am einfachsten zu realisieren. Allerdings entstehen hier eher hohe als breite Dialoge, was diese Vorgehensweise nur bei wenigen, sehr breiten Elementen sinnvoll macht.
- Bei der **zweispaltigen Positionierung** muss zunächst das erste Element der zweiten Spalte bestimmt werden. Dann wird das Layout für die Elemente für jede der zwei Spalten nach dem Prinzip der einspaltigen Positionierung berechnet.
- Das **Right-Bottom-Verfahren** [Bodart 1994b] weicht von der streng linearen Vorgehensweise ab. Die Elemente werden nicht zwangsläufig untereinander angeordnet. Ist die Breite wesentlich geringer, so können Elemente oder Gruppen auch nebeneinander angeordnet werden. Ist die Breite ähnlich der anderer Elemente, so werden Elemente oder Gruppen untereinander angeordnet. Dabei wird die Auswahl getroffen, ob das folgende Element bezüglich des zuletzt eingefügten daneben (Right) oder darunter (Bottom) positioniert werden soll.[44]

6.2.2 Nicht-sequentielle Positionierung

Im Unterschied zur sequentiellen Positionierung wird bei Methoden zur nicht sequentiellen Positionierung die Reihenfolge nicht direkt übernommen. Vielmehr werden andere Systematiken zur Positionierung verwendet.

6.2.2.1 DON

Ein Kriterium, das dabei zum Einsatz kommen kann, ist die Größe, d. h. die Ausmaße, von Gruppen oder Elementen. Angewandt wurde diese Vorgehensweise von Kim und Foley beim DON-Projekt

[44] vgl. [Kruschinski 1999]: Algorithmus Seite 93ff

[Kim 1993]. Das breiteste Objekt wird dabei ganz oben platziert, während die kleinsten Objekte zum Schluss kommen. Durch die zweispaltige Positionierung ergibt sich meist eine größere, linke Spalte und eine kleinere, rechte Spalte. Beide Spalten können so gleichmäßig gefüllte werden.

Das Problem bei diesem Ansatz ist das Fehlen jeglicher Semantik. Ein praktischer Einsatz ist deshalb kaum sinnvoll.

6.2.2.2 Layout Appropriateness

Beim Prinzip der Layout Appropriateness [Sears 1993] bestimmen Benutzungshäufigkeit und -reihenfolge die Anordnung der Elemente. Häufig benutzte Elemente werden möglichst links oben angeordnet. Zur Minimierung der Wegstrecke, die der Benutzer mit Blicken über den Bildschirm zurücklegt, werden die Elemente dann möglichst in der Reihenfolge ihrer Benutzung abgelegt. Die Voraussetzung für dieses Verfahren ist die Kenntnis des Benutzerkreises und der Arbeitsabläufe:

„Die Strategie der Layout Appropriateness hat einen wesentlichen Nachteil: Die Benutzerwerte müssen bekannt sein. Aus diesem Grund ist es grundsätzlich schwer möglich, diese Verfahren für ein generierendes System einzusetzen. ***Eine Ausnahme würde bestehen, wenn dem System zusätzliche Daten über mögliche Arbeitsabläufe bereitstehen würden. Diese Daten könnten durch eine Aufgabenanalyse ermittelt werden.****“* [Kruschinski 1999]

Genau an dieser Stelle setzt die Generierung aus InteractionCases bzw. Use-Cases an. Dank der detaillierten Informationen aus den InteractionCases und der inhärenten Arbeitsabläufe von Use-Cases ist es möglich, Ablaufinformationen einzubringen und so eine rein auf syntaktischen, datenmodell-basierten Angaben beruhende Transformation mit Semantik zu erweitern.

Das Generator-Prinzip des IML-Transformators ist also vom Ansatz her verwandt mit dem Prinzip der Layout Appropriateness.

6.2.3 Vertikale Positionierung

Während rein aus einzeiligen Elementen bestehende Dialoge leicht auszurichten sind, müssen bei der Positionierung in Dialogen mit gemischten Elementtypen bestimmte Vorkehrungen getroffen werden.

Die schon beschriebene Vorgehensweise zur Vergrößerung der vertikalen Abstände hat das Ziel nebeneinander stehende Elemente auf gleicher Höhe auszurichten.

Werden kleinere Elemente neben größeren positioniert, wird zunächst nach deren vertikalem Beginn vorgegangen. Wird dabei kein Treffer erzielt, so werden in einem zweiten Schritt die unteren Kanten überprüft.

So können beispielsweise mehrere kleine Elemente neben einem großen Element angeordnet werden, während obere und untere Kante trotzdem abschließen. Sinnvoll ist dann noch eine Überprüfung auf gleiche Abstände, wenn mehr als zwei kleine Elemente neben dem großen stehen.

Eine wichtige Rolle spielt dabei wiederum das Konzept der optischen Linien – in diesem Falle der optischen Zeilen.

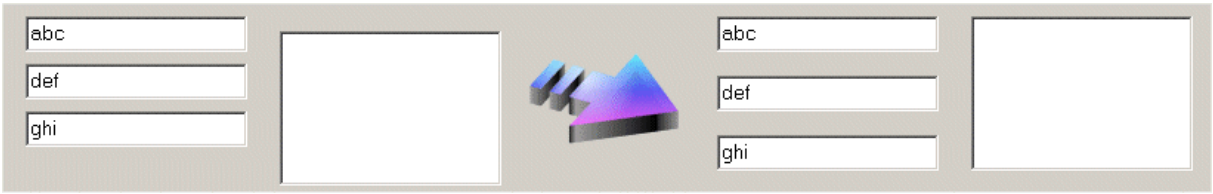

Abbildung 53: Zeilenweise Ausrichtung von Elementen unterschiedlicher Größe

6.2.4 Das Layoutprinzip von HERBS

Betrachtet man HERBS genauer, findet – vom IML-Modell aus betrachtet – *keine sequentielle* Anordnung statt. Es werden Elemente zum Teil umsortiert, da das IML-Modell nur einen möglichen Interaktionsablauf und die entsprechenden Spielräume enthält.

Nach der als Ausgangsbasis verwendeten DiLL-Reihenfolge, handelt es sich allerdings um ein weitgehend *sequentielles Verfahren*, auch wenn Teilweise Umgruppierungen vorgenommen werden.

Ausgehend von der IML-Modell kann der Vorgang dagegen eher als *Layout Appropriateness* betrachtet werden, da hier die Informationen aus dem Vorgehensmodell des Benutzers Verwendung finden.

Prinzipiell handelt es sich also um eine *Mischform* der beschriebenen Layoutansätze, aufgrund der Benutzerinformationen im IML-Modell.

Eine Erklärung zu schematischem und konkretem Layout [Nilson 1997] und dessen Umsetzung in HERBS befindet sich in Anhang C.

6.3 Gewichtung von Elementen

Betrachtet man Dialoge, in welchen mehrzeilige Textfelder und Tabellen sich mit einzeiligen Elementen abwechseln, entsteht oft ein Eindruck von Unordnung oder fehlender Harmonie.

Deshalb ist es ratsam, die Elemente so anzuordnen, dass Tabellen und mehrzeilige Textfelder sich auf alle Spalten weitgehend gleichmäßig verteilen. Eine Anordnung auf der gleichen Höhe und die Nutzung der gesamten Dialogbreite für ausgewählte Elemente tragen zu einem besseren Layout bei.

6.3.1 Gewichtung generell

Eine Möglichkeit, hier algorithmisch vorzubauen, ist ein Gewichtungssystem für Elemente. So müssen Tabellen und mehrzeilige Textelemente ein hohes Gewicht, abhängig von Ihrer Größe (Fläche) besitzen. Einzeilige Felder werden ebenfalls nach Größe (zumeist Länge) gewichtet. Checkboxen als sehr kleine und wenig flächenfüllende Elemente haben ein niedrigeres Gewicht.

Labels fallen aufgrund der geringen Fläche und optischen Auffälligkeit ebenfalls nur wenig ins Gewicht.

Das letztendliche Gewicht ist das Produkt aus Fläche und typindividuellem Gewichtungsfaktor. Die Höhe wird also mit der Breite multipliziert, was zunächst die Fläche ergibt. Ein Gewichtungsfaktor wird nun abhängig vom Typ bestimmt und mit dem Flächenwert multipliziert.

Für die meisten Elemente ist der Gewichtungsfaktor 1, da Eingabezeilen und andere Elemente vollständig ausgefüllt sind. Labels könnten dagegen beispielsweise mit ca. 0,4 gewichtet werden.

Zusammen mit der Maxime der Gleichverteilung und der Positionsangleichung großer Elemente können so harmonischere Dialoge erreicht werden.

6.3.2 Gewichte von Spalten

Wichtiger noch als bei Fenstern ist die Gewichtung bei Spalten. Diese dient zur Vermeidung sehr unterschiedlicher Füllgrade von Spalten, die ein unharmonisches Layout verursachen.

Mit Hilfe der Gewichtung kann festgestellt werden, ob einzelne Elemente die Spalte wechseln müssen. Hat eine Spalte ein zu hohes Gewicht, kann dies durch Abgabe eines Elements verringert werden.

Da die Reihenfolge oft weitgehend beibehalten werden muss, ist ein Ausgleich nicht immer möglich. Zudem darf immer nur das letzte Element einer Spalte in die nächste oder das erste Element einer Spalte in die vorhergehende Spalte verschoben werden.

6.3.3 Füllgrad von Fenstern

Der Füllgrad von Fenstern lässt sich über die Fläche ohne einen Gewichtungsfaktor berechnen. Dazu werden alle nicht überlappenden, mit Elementen gefüllten Rechtecke addiert. Die Summe ergibt den Füllwert F.

Das Produkt aus Höhe und Breite des Fensters abzüglich der Titelzeile ergibt den Gesamtinhalt G des Fensters.

Der Quotient B = F/G aus Füllwert F und Gesamtinhalt G ergibt den prozentualen Füllgrad des Fensters, der wiederum nach den in Kapitel 6.1.1 angegebenen Grenzen bewertet werden kann.

Bei zu hohem Füllgrad können die Abstände vergrößert werden. Führt dies zu einem zu großen Dialog, ist es auch möglich Teile auszulagern werden.

6.4 Initiale Positionierung der Elemente

Für die Optimierung der Positionen im Layout muss eine Ausgangssituation geschaffen werden, die den Rahmenbedingungen weitgehend genügt:

1. Schon bei der Erzeugung der DiLL-Objekte wird deren Typ festgelegt.
2. Die Größe eines Elements wird aus dem Typ und dem Mengengerüst bestimmt.
3. Handelt es sich um einen SCREEN, so werden zunächst die Gruppen und Elemente ausgerechnet. Handelt es sich um eine Gruppe, werden zunächst die enthaltenen Untergruppen und Elemente berechnet. Aus diesen untergeordneten Objekten wird die Gruppe bzw. der SCREEN dimensioniert. Diese haben beide initial die Größe 0 x 0.
4. Jeder SCREEN und jede Gruppe verfügen über eine „nächste freie Y-Position“.
5. Untergeordnete Elemente werden einer Gruppe bzw. einem SCREEN hinzugefügt. Elemente und Gruppen werden an der jeweils „nächsten freien Y-Position“ des übergeordneten Objekts eingefügt.
6. Nach dem Einfügen wird die „nächste freie Y-Position“ um die Höhe des Elements und den initialen Abstand zwischen Elementen nach unten verschoben. Alle Objekte werden also untereinander eingefügt.

Auf diese Weise entsteht ein einspaltiges Modell der in einem SCREEN enthaltenen Gruppen und Elemente.

Das Verfahren zu Positionierung erfolgt bottom-up, da zuerst die in der Hierarchie ganz unten gelegenen Elemente geometrisch erfasst und relativ positioniert werden müssen, bis die Geometrie der übergeordneten Elemente feststeht.

Elemente und Gruppen sind schon korrekt dimensioniert, wenngleich in manchen Fällen eine weitere Vergrößerung möglich ist und auch die Anordnung innerhalb der Gruppe und des SCREENs noch nicht endgültig feststeht.

Ist diese Positionierung abgeschlossen, kann es vorkommen, dass der Dialog wesentlich zu hoch und auch vom Seitenverhältnis her zu schmal ist.

Abhilfe kann durch Verbreiterung von Elementen und durch Spaltenbildung geschaffen werden. Die Spaltenbildung wird deshalb im nächsten Kapitel ausführlicher beschrieben.

6.5 Spaltenbildung

Wird ein bestimmter Grenzwert der Höhe bzw. des Dialog-Seitenverhältnisses überschritten, so müssen die Elemente eines Dialogs mehrspaltig angeordnet werden. Hierzu ist die Berechnung von Spaltenumbrüchen erforderlich.

Als Kriterien können dafür sowohl das Seitenverhältnis des Dialogfensters, d. h. des SCREENs, als auch die Anzahl der Interaktionselemente in einer Spalte dienen.

Die Grundlage für diese Entscheidung könnte beispielsweise auch wieder die „magische Zahl“ sieben plus/minus zwei bilden: Steigt die Anzahl der Elemente oder Gruppen pro Spalte über diese Zahl, so muss ein Spaltenumbruch stattfinden. Aufgrund der Größe des Bildschirms und der Aufnahmekapazität des Benutzers ist allerdings die Anzahl Spalten und somit auch die Anzahl gleichzeitig darstellbarer Elemente begrenzt.

Der entscheidende Wert ist die Fenstergeometrie. Wie schon im Kapitel 6.1.2 beschrieben, sollte das Verhältnis von Höhe zu Breite zwischen 1:2 und 1:1 liegen. Wurde ein zu hoher Dialog berechnet, sollte ein Umbruch erfolgen. Bei geringfügiger Überschreitung kann auch eine leichte Verbreiterung der Elemente und Gruppen das gewünschte Verhältnis herbeiführen.

Spaltenumbruch-Kriterien: Ein Spaltenumbruch erfolgt immer, wenn

- eine festgelegte Anzahl Elemente oder Gruppen überschritten wird (basierend auf der 7±2-Regel)
- das Höhe-Breite-Verhältnis des Fensters außerhalb des Intervalls [1/2, 1] liegt

Diese Regel kann jedoch von einer höherwertigen Regel wieder überschrieben werden, wenn bestimmte weitere Bedingungen gelten.

6.5.1 Unterschiedliche Spaltenhöhen

Das Problem bei Umbrüchen stellen Gruppen und große Elemente dar. Besteht ein Dialog nur aus einzeiligen Elementen, geht der Umbruch problemlos nach den oben genannten Regeln vonstatten. Gruppen und große Elemente können jedoch meist nicht getrennt werden, und so kommt es zu sehr unharmonischen Layouts.

Wird eine bestimmte Gewichtsdifferenz zwischen den Spalten überschritten, d. h. ist eine Spalte wesentlich stärker gefüllt als eine andere, so müssen andere Umbrüche, meist sogar andere Layouts gewählt werden. Möglichkeiten sind dabei:

- Reduktion der Spaltenzahl bei Inkaufnahme einer Seitenverhältnisüberschreitung
- Verteilung der Gruppe über mehrere Spalten, falls dies durch die Position innerhalb des Ablaufs möglich ist
- Belassen der Spaltenzahl, aber Beachtung von Harmonie-Regeln
- Vergrößern von skalierbaren Elementen
- Umsortieren entsprechend gekennzeichneter Elemente
- Gruppen oder sehr große Elemente spaltenübergreifend darstellen
- In Ausnahmefällen: Sortieren nach Abhängigkeiten

6.5.1.1 Reduktion der Spaltenzahl bei Inkaufnahme einer Seitenverhältnisüberschreitung

Die Reduktion der Spaltenzahl lässt sich nur dann durchführen, wenn die Dialoghöhe die voraussichtliche vertikale Größe des sichtbaren Bereichs nicht überschreitet.

Findet keine Überschreitung statt, wird die Spaltenzahl um eins verringert und der Layout-Algorithmus nochmals für eine Spalte weniger durchgeführt.

6.5.1.2 Verteilung der Gruppe über mehrere Spalten

Steht die problematische Gruppe bzw. das mehrzeilige Element,

- am Anfang,
- an einem Spaltenumbruch,
- am Ende
- oder an einer anderen „günstigen Position“[45],

so kann die Gruppe auch über mehrere Spalten ausgedehnt werden, falls dies durch die Position innerhalb des Ablaufs möglich ist. Am sinnvollsten ist dabei die Ausdehnung auf die volle Dialogbreite. Diese Variante ist allerdings nur für Gruppen geeignet, die mindestens soviele Elemente haben wie Dialogspalten vorhanden sind oder entsprechend vergrößerbare Elemente enthalten. Eine Ausdehnung über mehrere Spalten erzeugt bei einzeiliger Gruppenhöhe leicht sehr ungewöhnliche Layouts.

Den größten Effekt erzielt diese Vorgehensweise jedoch bei Verwendung der mit Abstand größten Gruppe in einem komplexen Fenster. Hier ist eine optische Aufwertung des gesamten Fensters die Folge.

6.5.1.3 Belassen der Spaltenzahl, aber Beachtung von Harmonie-Regeln

Eine weitere Möglichkeit, wie sie beispielsweise in [Kruschinski 1999] beschrieben wird, entsteht durch das Beibehalten des bisherigen Layouts von Gruppen bzw. Elementen bei ebenfalls gleichbleibender Spaltenzahl.

Hierbei ist dann allerdings zu beachten, dass die auf dem Spaltenumbruch liegende Gruppe nur dann in die nächste (rechte) Spalte verschoben werden darf, wenn die linke Spalte größer bleibt als die rechte. Dies gilt entsprechend auch für drei- und mehrspaltige Layouts. Für die Spaltengewichte (Kapitel 6.3.2) muss immer gelten:

$$\text{Spalte}(n).\text{Gewicht} \geq \text{Spalte}(n+1).\text{Gewicht}$$

Gleichung 6: Relation der Spaltengewichte nebeneinanderliegender Spalten

Ein entsprechender Algorithmus mit eingeschränkter Beachtung der Reihenfolge ist in [Kruschinski 1999] ab Seite 195 angegeben.

6.5.1.4 Vergrößern von skalierbaren Elementen

Ist eine bessere Aufteilung mit dem Ziel einer harmonischen Darstellung nicht möglich, so können skalierbare Elemente vergrößert werden. Skalierbare Elemente sind beispielsweise

- mehrzeilige Textfelder
- Tabellen
- Angezeigte Dokumente wie OLE-Links

Befinden sich skalierbare Elemente in der weiter rechts gelegenen Spalte, so kann diese vergrößert werden, indem diese Elemente in vertikaler Richtung vergrößert werden.

Ist die weiter rechts gelegene Spalte zu groß und enthält sie skalierbare Elemente, die noch sinnvoll verkleinert werden können, so kann ein harmonisches Layout auch über eine Verkleinerung dieser Elemente erzielt werden.

Wegen der weitgehenden Initialisierung mit minimalen Werten ist eine Verkleinerung allerdings meist nicht möglich. Besser ist dann meist eine Vergrößerung von Feldern der linken Spalte.

[45] wie schon erwähnt können bei ausreichendem Platz über oder unter einem spaltenübergreifenden Objekt auch innerhalb des Dialogs unterbrechende Elemente eingeführt werden

6.5.1.5 Umsortieren gekennzeichneter Elemente

Wurden Elemente oder Gruppen mit Hilfe der IML als in der Reihenfolge austauschbar markiert, muss überprüft werden, ob unterschiedlich große Objekte (Elemente/Gruppen) enthalten sind. Ist dies der Fall, so kann über eine Änderung der Reihenfolge die nachfolgende Spalte vergrößert werden. Die zu große Spalte wird dadurch verkleinert, die zu kleine Spalte im Gegenzug vergrößert.

Dabei ist darauf zu achten, dass nur entsprechend markierte Elemente umsortiert werden. Im Falle einer freien Umsortierung, würde der Dialogablauf verschlechtert und damit die Benutzbarkeit drastisch reduziert werden, was in schlechtem Verhältnis zur optischen Verbesserung steht.

In [Kruschinski 1999] wird ein Verfahren vorgeschlagen, das beliebige, allerdings möglichst wenige Elemente komplett neu einordnet, da bei dem dort beschriebenen Verfahren aus JANUS keine Kennzeichnung zur Verfügung steht.

Bei der Kennzeichnung können verschiedene Stufen angegeben werden:

1. None: Keine Umsortierung möglich
2. Block: Innerhalb des Blocks von block-verschiebbaren Elementen kann die Position verändert werden
3. Independent: Element/Gruppe kann beliebig verschoben werden, da es/sie unabhängig ist

Bei IML können sortierbare Blöcke in Form von Gruppen sowohl einzeln als auch geschachtelt auftreten, so dass sogar mehrere Ebenen der Vertauschung möglich sind, sofern diese vom Entwickler spezifiziert wurden.

Für größere Gruppen ist es sinnvoll, mehrere Permutationen zu testen. Dabei ist ein flächenminimaler Dialog das Ziel. Da der Aufwand für N Elemente dabei N! ist, können bei vielen Elementen längere Berechnungszeiten entstehen. Mit Verfahren, die diesen Optimierungsschritt nicht nach der Brute-Force-Methode angehen, können viele Fälle allerdings von vorneherein ausgeschlossen werden.

6.5.1.6 Sortieren nach Abhängigkeiten

Sind Abhängigkeiten zwischen Elementen definiert, so kann mit Hilfe dieser Relationen eine die Ablaufsemantik in Grenzen verletzende Layoutgestaltung eingesetzt werden. Kann das Layout nur mit Methoden, die den Ablauf antasten harmonisiert werden, so kann im Zuge dessen unter Beachtung der Abhängigkeiten eine Umsortierung stattfinden.

Die Beachtung von Abhängigkeiten bei der Reihenfolge garantiert dabei, dass nicht Elemente eingegeben werden müssen, deren Ergebnis von Einstellungen beeinflusst wird, die erst weiter hinten im Dialogablauf getroffen werden.

Allerdings ist die Bedienbarkeit einer solchen Struktur fragwürdig, da auch hier eine vollständige Umsortierung stattfinden kann, die die in das IML-Modell eingegebene Reihenfolge und Bedeutung der einzelnen Felder vollkommen vernachlässigt.

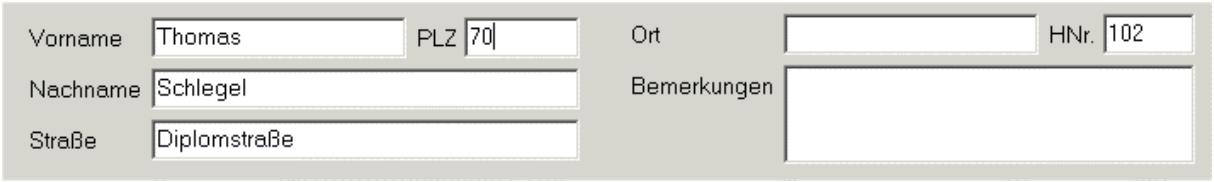

Abbildung 54: Beispiel für ergonomisch ungünstige Anordnung durch reine Layoutorientierung

6.5.1.7 Gruppen oder sehr große Elemente spaltenübergreifend darstellen

Die Spaltenübergreifende Darstellung wurde schon besprochen. Hier wird sie allerdings nicht wegen einer entsprechenden Festlegung im Modell, sondern rein nach Optimierungsentscheidungen des Transformators eingesetzt.

Eine häufige spaltenübergreifende Darstellung hat negative Auswirkungen auf das Layout, wenn sie nicht am Anfang oder am unteren Ende des Dialogs erfolgt.

Aus diesem Grund sollte eine derartige Lösung nur eingesetzt werden, wenn bessere Methoden zu keinem Ergebnis führen.

6.5.2 Ausrichtung und Zeilenbildung

Generell ist auf eine passende horizontale Ausrichtung zu achten, d. h. die Elementanfänge einer Spalte liegen auf möglichst wenigen senkrechten virtuellen Linien. Wird allerdings mehr als eine Spalte dargestellt, so muss auch auf die Zeilenbildung, d. h. vertikale Ausrichtung, geachtet werden.

Kommen nur einzeilige Elemente zum Einsatz, regelt sich das Layoutverfahren automatisch richtig, d. h. alle Elemente liegen auf einem regelmäßigen Gitter.

Kommen Elemente mit abweichender Höhe zum Einsatz, beispielsweise mehrzeilige Textfelder, so entsteht ein Gitter nicht mehr ohne Optimierung.

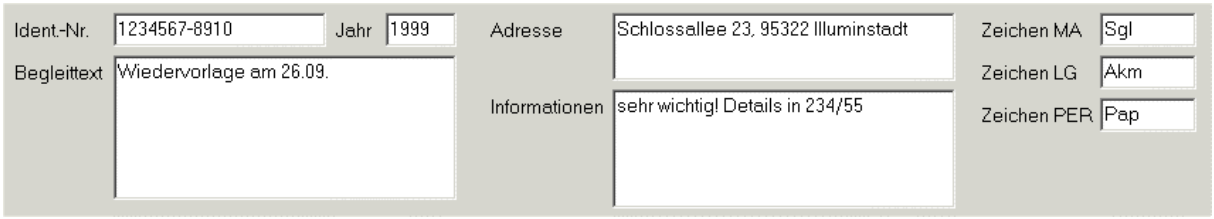

Abbildung 55: Gitterverlust durch unterschiedliche Elementhöhen

Mit Hilfe eines passenden Algorithmus muss nun eine Lösung gefunden werden, indem bei Bedarf Abstände in benachbarten Spalten passend vergrößert werden.

6.5.2.1 Zeilenbildung

Elemente sollten möglichst auf einer horizontal verlaufenden Linie positioniert werden, um die Anzahl der horizontalen virtuellen Linien zu verringern. Eine gute vertikale Positionierung kann nur durch Anpassung von Abständen erreicht werden.

Im Ausgangslayout sind schon minimale Abstände enthalten, so dass eine Neupositionierung nur über Umsortierung oder Vergrößerung der Abstände erfolgen kann.

Für diese zeilengetriebene Umsortierung kommen dieselben Kriterien wie im Falle der gewichtungsgetriebenen Umsortierung von Spalten zur Geltung. Eine Verletzung der Semantik kann nicht in Kauf genommen werden.

6.5.2.2 Entwurf eines Algorithmus

Mit Hilfe eines Algorithmus können Spalten verglichen und angepasst werden. Dazu werden für jede im Dialog vorkommende Y-Koordinate in jeder Spalte Elemente mit derselben Y-Koordinate gesucht. Werden diese gefunden, ist die Zeile ausgerichtet. Folgen in einer Spalte keine Elemente mehr, ist die Zeilenbildung für diese Spalte vollständig und damit auch korrekt.

Können zum ersten Element keine passenden Elemente gefunden werden, so wird der Bereich nach unten schrittweise oder in einem Schritt erweitert. Können auch im erweiterten Bereich nicht in jeder Spalte Elemente gefunden werden, werden nur die gefunden entsprechend ausgerichtet. Der Algorithmus läuft dann in gleicher Weise weiter.

Wurden Elemente im gewählten Bereich gefunden, die unterhalb liegen, werden die obersten Elemente solange nach unten verschoben, bis es auf gleicher Höhe mit dem untersten im Verschiebe-Bereich vorhandenen Element zu liegen kommt.

```
Solange Y-Koordinate kleiner N, Startwert 0
    Ist in keiner oder jeder Spalte noch nicht beendeten Spalte ein Element mit dieser Y-Koordinate? Wenn nicht:
        Suchbereich festlegen
        Alle Elemente im Suchbereich auf Höhe des untersten gefundenen bringen
        Neu Y-Koordinate ist die nächste unterhalb des Suchbereichs
```

Abbildung 56: Pseudocode-Algorithmus zur vertikalen Elementausrichtung

Oberkanten werden so auf Oberkanten abgebildet. Zusätzlich sollten Unterkanten mit einem erweiterten Algorithmus auf Unterkanten abgebildet werden.

6.5.2.3 Maximalverschiebung

Die Verschiebung darf einen Maximalwert von etwa einer Elementhöhe (einzeiliges Interaktionselement) nicht überschreiten, da sich sonst größere Leerräume bilden, die unerwünscht sind und zu Missinterpretationen führen. Für den oben beschriebenen Algorithmus werden Interaktionselemente, die in Gruppen organisiert sind, ebenfalls als Einzelelemente betrachtet.

6.5.2.4 Gruppenrahmen

Gruppenrahmen müssen gesondert betrachtet werden. Der Algorithmus für Interaktionselemente kann in gleicher Form auch auf gerahmte Gruppen angewandt werden. Allerdings befinden sich in den meisten Fällen keine Gruppen auf gleicher Höhe, so dass dieser Algorithmus selten Änderungen liefert und zuerst gestartet werden sollte.

Da auch die Gruppenrahmen optische Linien erzeugen, ist es sinnvoll, diese in den oben genannten Algorithmus zu integrieren.

Werden Elemente nach unten verschoben, müssen alle unterhalb liegenden Elemente ebenfalls verschoben werden.

6.6 Implizite Styleguides

Für die manuelle Entwicklung von Oberflächen spielen Styleguides heute eine enorm wichtige Rolle. Die Konformität zu einem Plattform-Styleguide wie dem von Apple [Apple 1992] oder Microsoft für Windows 95 [Microsoft 1996] kann über die Akzeptanz der Software durch die Benutzer entscheiden, da ein einheitliches „Look&Feel“ eine schnellere Einarbeitung und besseres Verständnis für die Applikation bewirkt. Gleiches gilt auch für firmen- oder branchenspezifische Styleguides.

Durch den flächendeckenden Einsatz von graphischen Benutzungsoberflächen ist auf dem Markt eine unüberschaubare Menge von Bibliotheken und damit Darstellungsmöglichkeiten verfügbar. Soll diesen Rechnung getragen werden und trotzdem eine Vereinheitlichung vorgesehen werden, entstehen oft Styleguides mit mehreren hundert Seiten[46].

Schon für einen Software-Ergonom sind derartige Richtlinien nicht überschaubar. Noch schwieriger gestaltet sich der Einsatz derartiger Styleguides für Entwickler, was meist direkt zur Nicht-Beachtung führt.

[46] z. B. [Sparkasse 1997]

Deshalb ist es wichtig, kompakte Darstellungen für Fälle zu entwickeln, die den Großteil der Entwicklungen abdecken können. In einem zweiten Schritt können diese dann zumindest, wie in dieser Diplomarbeit gezeigt, zu einem Transformator konsolidiert werden, der für die meisten Fälle eine direkte Gestaltung der Oberfläche basierend auf einem Modell ermöglicht.

Bestehen in einem weiteren Schritt schon Modelle für bestimmte Abläufe (beispielsweise IML-Patterns), so können viele Anwendungen basierend auf diesen mit ergonomischen Regeln überprüften Abläufen realisiert werden.

Zusammenfassend entsteht damit folgender Entwicklungspfad:

1. Styleguide
2. Kompaktifizierter Styleguide
3. Styleguide-basierter Transformator
4. Patterns für Abläufe, pattern- und frameworkbasierte Entwicklung
5. Entwicklung von unternehmensübergreifenden Standard-Patterns und deren Implementierungen (vgl. z. B. File-Open-Dialoge)

6.7 Größenanpassung

Die Größenanpassung wurde schon für das Layout verwendet. Weiterführende Konzept werden im Folgenden vorgestellt.

6.7.1 Registerkarten: Vergrößerung von Elementen bei zuviel Platz

Die größte Registerkarte bestimmt die Größe aller anderen Seiten. So kann es passieren, dass bei einer groß gewählten Seite andere Seiten kaum belegt sind.

Weisen einzelne Seiten einen zu geringen Füllgrad auf, ist es manchmal möglich, Registerkarten mit wenig Information etwas zu vergrößern, beispielsweise durch die Vergrößerung von Listen mit vielen Einträgen. Abhilfe kann hier nur mit dem Zusammenfassen von mehreren Seiten und mit der Vergrößerung von Elementen geschaffen werden.

Unterschiede bei Registerkarten ergeben sich aus der festgelegten Größe der Seite. Somit können Spalten und Elemente nach den unten beschriebenen Regeln vergrößert werden, bis die Seite weitgehend gefüllt ist. Auch Spalten können bis zu einem gewissen Grad vergrößert werden, um mehr Platz zu belegen.

Führt das Zusammenlegen mehrerer Seiten ebenso wie die sinnvolle Elementvergrößerung nicht zum Ziel, muss der Rest der Seite leer gelassen werden. Der Entwickler sollte dann eine entsprechende Warnung erhalten und das IML- oder DiLL-Modell, falls möglich, ändern.

6.7.2 Größenanpassung bei Eingabe- und Auswahlfeldern

Die grundlegende Forderung bei Eingabefeldern ist, dass der sichtbare Platz groß genug für alle geplanten Eingaben sein sollte. Insbesondere gilt dies für Zahlenwerte und Datumsangaben. Ist die Eingabelänge nicht exakt festgelegt, genügt eine Dimensionierung, die für die Mehrheit der Eingaben ausreicht. Dieser Wert kann aus durchschnittlicher und maximaler Länge berechnet werden.

Diese Regelung ist allerdings nur dann sinnvoll, wenn ein Höchstwert von ca. 40 Zeichen nicht überschritten wird. Sind die geplanten Eingaben länger, sollten mehrzeilige Eingabefelder verwendet werden.

Da sich diese ebenfalls nach den Gestaltungsregeln zu richten haben, ist eine breitere Darstellung meist nur dann sinnvoll, wenn es sich um ein oder maximal zwei Felder handelt. Diese müssen zudem eher am Ende des Interaktionszyklus stehen sein. Ist dies der Fall, kann sich das Feld über mehr als eine Spalte erstrecken bzw. im unteren Bereich des Dialogs in voller Breite dargestellt werden.

Bei mehrzeiligen Eingabefeldern ist es meist nicht sinnvoll, einen horizontalen Scrollbar einzusetzen. Die Darstellung über einen Zeilenumbruch zu bewerkstelligen, ist hier die bessere Alternative.

Bei Auswahlfeldern wie Listen oder vor allem Drop-Down-Combo-Boxen ist aus technischen Gründen oft keine mehrzeilige Darstellung einzelner Einträge möglich. Hier muss dann auf einen horizontalen Scrollbar zurückgegriffen werden, wenn nicht Teile unsichtbar bleiben sollen. Ein Scrollbar ist hier ungewöhnlich, sollte aber durch ausreichende Dimensionierung weitgehend zu vermeiden sein.

Ab einer Länge von vier Zeilen können Scrollbar verwendet werden. Allerdings sollte eine genauere Abwägung je nach minimaler, durchschnittlicher und maximaler Zeilenzahl vorgenommen werden. Die Grenze kann nicht exakt vorgeschrieben werden. Wichtig ist es also, einen möglichst guten Wert festzulegen.

Die Maximalzahl direkt erfassbarer Elemente liegt bei 7 ± 2. Ein Vorschlag für eine Festlegung ist folgende fallabhängige Vorgehensweise:

- Listenelement
  - ganze Spalte steht zur Verfügung (Label oben)

    ⇒ Maximalzahl, höchstens jedoch Spaltenhöhe
  - Nur ein Teil einer Spalte steht zur Verfügung

    ⇒ Durchschnittliche Zahl + 1, höchstens 8 Einträge
- Drop-Down-Element
  - Maximale Anzahl kleiner als verfügbare Höhe (Strecke bis zum unteren Rand)

    ⇒ zeige die maximale Anzahl an
  - Wenn nicht, aber durchschnittliche Anzahl kleiner als verfügbare Höhe

    ⇒ zeige die durchschnittliche Höhe + 1 an
  - Wenn nicht

    ⇒ zeige so viele wie möglich an; bei weniger als minimale Anzahl Warnung ausgeben
  - Maximal aber 9 Einträge, außer Ausnahme-Tag regelt dies
  - Minimal 3 Einträge (Zeilen)

Regeln:

- Bei Zahlen und Datumsangaben muss das Feld alle möglichen Eingaben in voller Länge darstellen können.
- Bei Texteingaben kleiner als 40 Zeichen sollte das Feld alle möglichen Eingaben in voller Länge darstellen können oder alternativ den Text verschieben, so dass der aktuelle Teil immer sichtbar ist.
- Bei Texteingaben, die keine sinnvolle Darstellung dieser Art mehr erlauben (größer als ca. 40-80 Zeichen) wird ein mehrzeiliges Eingabefeld verwendet.
- Mehrzeilige Eingabefelder werden horizontal im Dialogkontext dimensioniert (keine Abweichung größer als 25%).
- Sehr große Eingabefelder, die weiter hinten im Interaktionszyklus liegen, können sich über mehr als eine Spalte bzw. über die volle Dialogbreite erstrecken. Die Maximalzahl dieser Felder sollte bei 1-2 pro Dialog bzw. Registerkarte liegen.
- Horizontale Scrollbars sind bei mehrzeiligen Eingabefeldern nicht sinnvoll, stattdessen findet ein Zeilenumbruch statt.

- Bei mehrzeiligen Feldern (Liste, Combo-Box) ist ein horizontaler Scrollbar bei Überschreitung der sichtbaren Breite notwendig, es sei denn, das Feld kann einen Umbruch vornehmen und die Elemente mehrzeilig darstellen.
- Ab vier Zeilen Länge können Scrollbars verwendet werden.
- Zeige andere mehrzeilige Eingabeelemente ebenfalls wie die oben beschriebenen mehrzeiligen Eingabefelder an.

6.7.2.1 Gruppenrahmen anpassen

Gruppenrahmen sollten jeweils an das größte Element der Spalte angepasst werden, wiederum um die virtuellen Linien zu reduzieren. Ist der rechte Rand einer Gruppe weiter links positioniert als der rechte Rand des größten Elements oder der größten Gruppe, so muss der rechte Gruppenrand auf dieselbe virtuelle Line verschoben werden, auf der auch der rechte Rand des größten Elements oder der größten Gruppe liegt, d. h. die Gruppe wird nach rechts vergrößert.

Sind mehrere Gruppen in einer Spalte vorhanden, so müssen diese in jedem Fall durch Vergrößerung auf die gleiche Breite gebracht werden. Da bei der Vergrößerung von Gruppenrahmen nicht zwangsläufig eine Änderung der Eingabelängen von enthaltenen Elementen notwendig wird, ergeben sich aus der Rahmenvergrößerung nur größere Leerräume innerhalb der Gruppe, jedoch nicht insgesamt.

Ein verbesserter Algorithmus kann dieses Problem berücksichtigen und eine Breitenminimierung der größten Gruppe veranlassen oder in der kleinsten Gruppe Elemente horizontal hintereinander anordnen bzw. vergrößern. Auf diese Weise wird die ganze Gruppe verbreitert, bei Horizontalanordnung jedoch gleichzeitig in der Höhe gestaucht.

Gruppenrahmen müssen top-down angepasst werden, da sich die maximale Größe einer eingeschachtelten Gruppe (Gruppe in Gruppe) erst in Kenntnis der übergeordneten Gruppe bzw. der übergeordneten Gruppen berechnen lässt.

Zuerst werden daher die Gruppen, die direkt dem Fenster untergeordnet sind, vergrößert; im nächsten Schritt die in diesen enthaltenen Gruppen etc.

Es ergibt sich also ein Bottom-Up-Durchlauf zur Größenbestimmung der einzelnen Gruppen und ein nachfolgender Top-Down-Durchlauf zur Vereinheitlichung der Gruppengrößen.

6.7.2.2 Elemente einer Gruppe anpassen

Wurde eine Gruppe vergrößert, so müssen alle enthaltenen Elemente überprüft werden. Mehrzeilige Textfelder sollten nach Möglichkeit immer mit vergrößert werden, da so ungewollte Zeilenumbrüche in der Darstellung vermieden werden können. Diese Felder werden also um den gleichen Betrag vergrößert, wie die Gruppe, zu der sie gehören.

Haben Listen schon 110% der maximalen Breite der voraussichtlichen Einträge erreicht, sollten sie nicht weiter vergrößert werden, es sei denn es handelt sich um gekoppelte Listen, die noch unterschiedliche Größe haben. Gleiches gilt für Combo-Boxen und deren Drop-Down-Pendant.

Die maximale Eingabelänge darf bei Eingabefeldern nicht überschritten werden. Werden einzelne Felder vergrößert, stimmt die Relation zu anderen nicht mehr. Eine einzelne Vergrößerung ist daher normalerweise nicht sinnvoll.

Generell muss bei nebeneinander stehenden Elementen darauf geachtet werden, dass diese nur zusammen um den berechneten Betrag vergrößert werden. Ein Algorithmus könnte anhand einer Tabelle bestimmen, welches Element um wie viele Einheiten vergrößert wird. Beispielsweise würde eine Radiobutton-Gruppe neben einem mehrzeiligen Textfeld ihre Größe behalten, während das Textfeld maximal vergrößert werden würde.

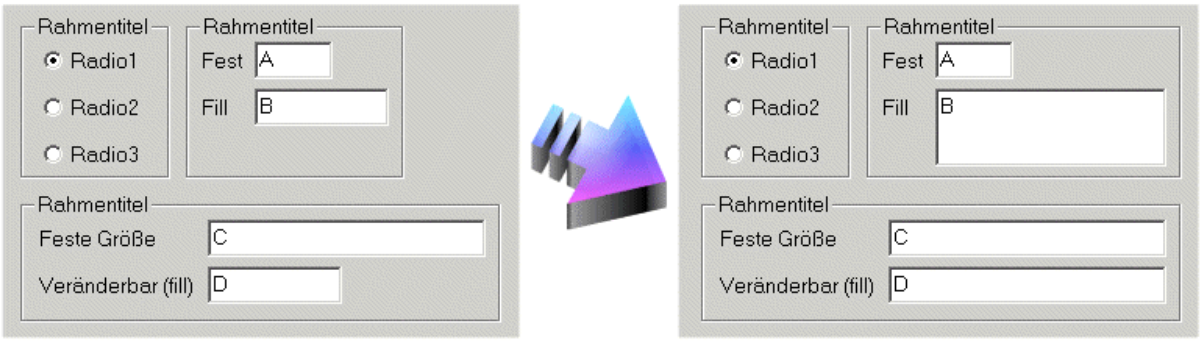

Abbildung 57: Anpassung der Gruppen- und Elementgröße

6.7.2.3 *Freie Elemente anpassen*

Sind Elemente keiner Gruppe zugeordnet, d. h. es handelt sich um freie Elemente, gelten dieselben Regeln wie für Elemente in Gruppen. Unterschiede ergeben sich nur durch den unterschiedlichen Rand: Während bei Elementen einer Gruppe der Abstand zum Gruppenrand maßgeblich ist, gelten für freie Elemente die Grenzen der Spalte. Im Sonderfall des einspaltigen Dialoges entsprechen die Spaltengrenzen den Fenstergrenzen.

6.7.2.4 *Generelle Größenanpassung*

Die Anpassung der horizontalen Ausmaße wurde schon eingehend beschrieben. Eine vertikale Anpassung kommt nur für mehrzeilige Elemente wie Listen, mehrzeilige Textfelder oder Tabellen in Frage.

Das Vorgehen zur Größenbestimmung wurde schon beschrieben. Ist die durchschnittliche oder sogar die minimale Größe noch nicht erreicht, so sollten die Elemente soweit möglich vergrößert werden.

Einen speziellen Fall, in dem die Vergrößerung besonders wünschenswert ist, stellt die ungleiche Spaltenhöhe dar. Hier sollten vertikal skalierbare Elemente in kleineren Spalten vergrößert werden, bis sie den Ausmaßen der größten Spalten angenähert sind. Bei zwei nebeneinanderliegenden Spalten muss die Höhe der linken Spalte jedoch immer größer oder gleich der Höhe der rechten Spalte sein.

6.8 Weitere Optimierungsmöglichkeiten

Schon bei der Umsetzung der IML-Beschreibungen in Elemente stehen viele weitere Optimierungsmöglichkeiten zur Verfügung, die ebenso große Verbesserungen bewirken können, wie dies mit der Anordnung und Dimensionierung von Elementen möglich ist.

So können Sonderfälle für sehr kleine InteractionCases berechnet werden. Oftmals werden beispielsweise mehrere Vorgehensweisen zur Auswahl gestellt. Werden diese mit Hilfe eines SELECT-Schritts und eines OK-Cancel-Navigationsmusters modelliert, würden – was für größere Dialog auch die beste Lösung wäre – zwei Radiobuttons und die Buttons „OK“ und „Abbrechen“ entstehen.

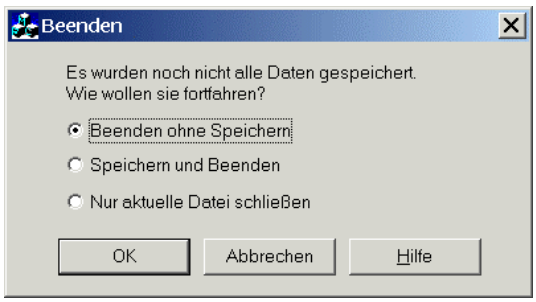

Abbildung 58: Beispiel für Dialogablaufoptimierung in speziellen Dialogen – Ausgangssituation

Möglich wäre aber auch die in diesem Fall bessere Darstellung mit nur einem Interaktionsschritt. Die beiden Optionen können so direkt ausgewählt werden. Die Alternative ist ein Abbruch. Der OK-Knopf ist damit unnötig geworden. [Raskin 2001]

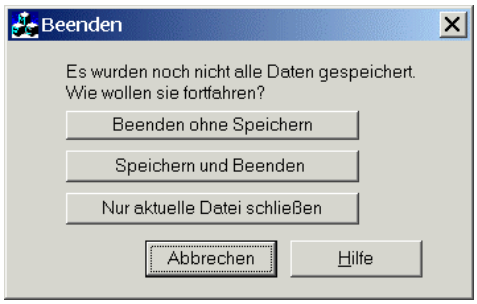

Abbildung 59: Beispiel für Dialogablaufoptimierung in speziellen Dialogen – lokale Optimierung

Die möglichen Sonderfälle beschränken sich größtenteils auf Auswahlmöglichkeiten, da Eingaben nicht anders abzudecken sind, es sei denn, die Länge der Eingabe ist beispielsweise auf ein Zeichen oder wenige Kombinationen begrenzt.

6.9 Metriken

Eine weitere Möglichkeit, Optimierungen vorzunehmen, oder zumindest vorzuschlagen und zu begründen, sind Metriken.

Das IML-Modell kennt alle Abläufe, Reihenfolgen, Interaktionsvorgaben und Aufrufzeiten für Fachkonzeptfunktionen. Aus dem DiLL-Modell können die Positionen und Dialoge hergeleitet werden.

Ausgestattet mit diesen Modellen können für die zu generierende Benutzungsoberfläche Interaktionsmetriken erhoben werden.

GOMS-Berechnungen: Mit Hilfe des GOMS-Modells lassen sich Zeiten für die Abarbeitung des Dialogs nach einem Standardszenario aus einzelnen Interaktionsschritten berechnen. Hierzu stehen nach [Raskin 2001] folgende Gesten zur Verfügung:

Abk.	Zeit	Gestenname	Beschreibung
K	0,2 s	Keying	Tastendruck auf Tastatur
P	1,1 s	Pointing	Zeigen auf eine Position am Bildschirm
H	0,4 s	Homing	Bewegen der Hand von der Tastatur zum GID[47] oder umgekehrt
M	1,35 s	Mentally Preparing	Mentale Vorbereitung auf den nächsten Schritt
R	?	Responding	Antwortzeit des Rechners

Tabelle 3: Zeitdauer für bestimmte Gesten.

Für welche Aktionen welche Gesten vermerkt werden müssen kann beispielsweise anhand einer Tabelle in [Raskin 2001] ermittelt werden. Mit einer Umsetzungstabelle für die Interaktionsschritte eines IML-Modell oder der Elementtypen des DiLL-Modells können direkt aus diesen Modellen entsprechende Werte erzeugt werden.

Aufgrund starker Schwankungen sind die Werte je nach Benutzer individuell sehr unterschiedlich. Ein Vergleich von Dialogen ist allerdings auch bei relativer Betrachtung möglich.

[47] Graphical Input Device, z. B. Maus

Hier kann zudem zwischen mauslastiger Bedienung und Bedienung mit Hilfe von Hotkeys und Tab-Taste unterschieden werden, so dass die Eignung für bestimmte Bedienungsarten ebenfalls abgelesen werden kann.

Auch weitere GOMS-Modelle wie CPM-GOMS und NGOMSL können umgesetzt werden. So ist es möglich, selbst Voraussagen über Lernphasen etc. miteinzubeziehen.

Der Vorteil ist, dass somit quantitative Bewertungskriterien zur Verfügung stehen, die beispielsweise für eine Entscheidung über die Aufteilung eines InteractionCase herangezogen werden können.

Informationseffizienz: Die Informationseffizienz wird bestimmt aus dem Quotienten des Mindestinformationsumfangs, der zur Durchführung einer Aufgabe notwendig ist und dem Informationsumfang, der tatsächlich vom Benutzer eingebracht wird. Bei einer Auswahl mit n Alternativen ist der Informationsumfang:

$$I = \frac{1}{n} \cdot {}_2\log n$$

Gleichung 7: Informationsumfang einer Alternative bei n gleich wahrscheinlichen Alternativen. [Raskin 2001]

Genauere Formeln und Erklärungen sind [Raskin 2001] zu entnehmen. Aussagen über die Zeicheneffizienz geben z. B. gute Hinweise auf Verbesserungsmöglichkeiten durch eine bessere Automatisierung.

Fitts Gesetz: Zur Bestimmung von Positionierzeiten bei Mausbedienung oder ähnlichen Positionierverfahren eignet sich das Gesetz von Fitt. Abhängig von Größe und Abstand des Ziels wird hier die Zeit errechnet, die ein Benutzer zur korrekten Positionierung benötigt.

Dabei ist D der Abstand bzw. Weg und S die Größe des Ziels in Positionierrichtung. Die Konstanten a und b werden experimentell oder durch vorhandene Testergebnisse ermittelt.

$$t = a + b \cdot {}_2\log(\frac{D}{S} + 1)$$

Gleichung 8: Berechnung der Positionierzeit t nach Fitts Gesetz (t in ms)

Sind die Wahrscheinlichkeiten jeder Auswahl nicht wie angenommen gleich, muss eine andere Formel angewandt werden [Raskin 2001].

Stärker am eigentlichen Dialog- und Anwendungsaufbau orientiert sind Metriken wie

- Anzahl Elemente eines bestimmten Typs von Elementen
- Anzahl aller Elemente
- Verhältnis von Eingabe- zu Auswahl- und zu Bearbeitungsfeldern
- Anzahl nicht näher spezifizierter, abstrakter Objekte
- Dialogdimensionen und durchschnittliche Bildschirmüberdeckung
- Anzahl Dialoge
- Anzahl Elemente pro Dialog
- Verhältnis direkt aufrufbarer und mittelbarer Dialoge, die nur als Folge aufgerufen werden

An diese Metriken können zudem Ratschläge für eine Umgestaltung von Modell und Oberfläche geknüpft und dem Entwickler zur Verfügung gestellt werden. Weitere Metriken können aus der Dokumentation zu entsprechenden Projekten wie UIDE (Kapitel 2.6.1.1) entnommen werden.

7 Finale Artefakte

Nachdem das IML-Modell vervollständigt und in ein per Zwischenschritt optimiertes DiLL-Modell umgesetzt wurde, können aus den vorhandenen Modellen Dateien generiert werden.

Diese gehören zu den verschiedenen Bestandteilen einer Software und entstehen alle am Ende des Generierprozesses als feststehende und zu archivierende Resultate – quasi als Blätter des Transformationsbaumes. Das heißt sie stellen „finale“ Artefakte dar.

Beispiele sind Ressourcen, Hilfen, Entwicklungsdokumente, Testbeschreibungen, Code-Fragmente oder ganze Applikationen in Native-Code.

Dabei sind zwei Gruppen von Resultaten zu unterscheiden:

- Aus dem DiLL-Modell generierte Artefakte, d. h. das User-Interface.
- Direkt aus dem IML-Modell generierte Artefakte.

Im Folgenden werden zunächst die aus dem DiLL-Modell mit Hilfe von Templates hergeleiteten Artefakte beschrieben. Der hierzu verwendete Transformationsprozess enthält eine Reihe von Konzepten, die im Folgenden zunächst erläutert und dann in den Gesamtzusammenhang gestellt werden.

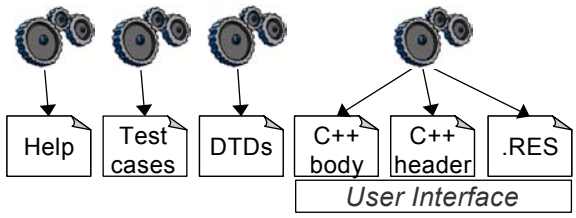

Abbildung 60: Finale Artefakte sind die Blätter des Transformationsprozesses

7.1 Das Template-Konzept

Das Anwendungsgerüst lässt sich nicht aus dem Modell herleiten, da abhängig von Programmiersprache und Plattform völlig unterschiedliche Konzepte zum Einsatz kommen, die in einer weitgehend plattformunabhängigen Sprache wie der IML nicht sinnvoll beschrieben werden können.

Möglichkeiten, das Anwendungsgerüst zu generieren sind folgende:

- Speicherung als Array von Strings im Generatorcode
- Einbettung als Liste in das IML-Modell
- Vorgabe mit Hilfe von Code-Dateien, die den Ergebnis-Dateien entsprechen

Eine unveränderbare Einbettung in den Generator ist kontraproduktiv, da keinerlei Veränderungen oder Verbesserungen vorgenommen werden können und alle Änderungen am Zielsystem (Compiler, Betriebssystem etc.) zu einem neuen Transformator, zumindest aber zu einer Neukompilierung, führen.

Die Einbettung in das IML-Modell ist möglich. Dabei stellt sich allerdings die Frage, woher die Daten kommen sollen. Eine Einbettung in das IML-Modell ist nur durch den Entwickler oder eine Standardvorgabe innerhalb der DTD möglich, wobei eine Einbettung durch den Entwickler zu fehleranfällig und aufwändig wäre.

Eine Einbettung als Standardvorgabe in die DTD führt zur Abhängigkeit von einer Konfiguration, so dass entweder diese Konfiguration genutzt werden muss oder wiederum eine Einbettung durch den Entwickler stattfinden muss.

Die beste Alternative ist also das Anlegen von Template-Dateien. Auf diese Weise ist es möglich, für jede Zielumgebung oder sogar für bestimmte Anwendungstypen jeder Zielumgebung einen Satz von Template-Dateien zur Verfügung zu stellen, die nach Ergänzung durch den Transformator eine vollständige, compilierfähige Anwendung ergeben.

Dabei können die Template-Dateien sogar vom Entwickler ergänzt werden, sowie neue Konfigurationen beispielsweise durch Kopieren und Verändern angelegt werden. Die Template-Dateien müssen bestimmte Marken enthalten, die Einfügestellen durch eine eindeutige Markierung und einen eindeutigen Index innerhalb dieser identifizieren.

Im Transformator wird der Token ***#_<ID>_#*** genutzt, d. h. die erste Einfügestelle wird mit ***#_1_#*** bezeichnet. Dieser Token wird beim Einfügen durch das zugehörige generierte Codefragment ersetzt.

Jede Einfügestelle kann damit durch die Kombination von Template-Dateiname und Indexwert eindeutig beschrieben werden. Ein Konzept, das diese Referenzierung nutzt, wird im Folgenden vorgestellt.

7.2 Code- und Ressourcen-Datei-Generierung

Sind geeignete Templates für das gewählte Zielbetriebssystem und den gewählten Anwendungstyp vorhanden, müssen noch die Ressourcenbeschreibungen und der Code für die vorgesehenen Einfügestellen generiert werden.

Viele Generatoren wählen für die Generierung in der letzten Stufe eine weitgehend hart codierte Umsetzungsfunktion, die für einmal festgelegte Elementtypen die endgültige Repräsentation ausgibt.

Probleme ergeben sich dann bei der Versorgung mehrerer Plattformen und Programmiersprachen: Die Ausgabe ist nur schwer veränderbar. Eine Änderung des Ausgabeformats zieht teilweise eine Änderung am Transformator nach sich. Auch ist der Transformator dann vollkommen plattformabhängig.

Die bessere Vorgehensweise ist also die Bereitstellung von Beschreibungen für jeden Elementtyp. Diese Beschreibungen müssen dann vom Transformator nur noch anhand der vorhandenen Informationen parametrisiert und in die Zieldateien geschrieben werden.

7.2.1 Eine Element-Transformationsbeschreibung

Für jeden Elementtyp muss dabei eine parametrisierbare Beschreibung existieren, die ihrerseits alle Text- und Codefragmente enthält und zudem mit Hilfe von Ziel-Dateiname und Ziel-Index eine Zuordnung zu den Einfügestellen der Template-Dateien enthält.

Die **Element Transformation Description (ETD)** ist eine solche Beschreibungsform. Sie existiert in Form einer XML-Datei, deren Syntax mit der ETD-DTD für alle Dateien dieser Art festgelegt wurde. ETD wurde ebenfalls im Zuge dieser Arbeit entwickelt. Die Transformationsbeschreibung ETD enthält Zähler, Spezifikationen für einzelne Elemente und generische Fragmente.

Zähler (DEFINE_COUNTER) können zur Festlegung eindeutiger Bezeichner oder zur Vergabe von eindeutigen Konstanten genutzt werden. Startwert und Schrittweite ermöglichen eine Vorgabe linearer Reihen.

Generische Fragmente (GENERIC_FRAGMENT) gleichen Bruchstücken von Elementspezifikationen. Die in generischen Fragmenten enthaltenen Beschreibungen können per Referenz in Elementspezifikationen eingebunden werden. Auf diese Weise werden redundante Beschreibungen für mehrere Elementtypen vermieden.

Elementspezifikationen (ELEMENT_SPEC) sind einem bestimmten Elementtyp (ElementType) zugeordnet. Sie beschreiben alle für das Hinzufügen eines derartigen Elements notwendigen Ergänzungen in Code und Ressourcen. Zu diesem Zweck enthalten sie eine beliebige Anzahl von Fragmenten (FRAMENT). Werte zur Dimensionierung des beschriebenen Elementtyps im Dialog ergänzen die Beschreibung.

ETD-Fragmente: Jedes Fragment ist einem Template und einem Indexwert innerhalb dieses Templates zugeordnet. Diese Zuordnung ermöglicht das Einfügen von Code und Ressourcen-Texten an exakt bezeichneten Stellen innerhalb der Templates.

Einfache Texte, Zähler, Zeilenvorschübe und applikationsweit eindeutige Bezeichner können so für jeden Elementtyp passend eingefügt werden.

Die Mehrfachnutzung desselben Zählerwerts ist für den Einsatz dieses Wertes an mehreren Stellen von großer Bedeutung. Zu beachten ist dabei die Reihenfolge der Templatebefüllung: Das inkrementieren eines Zählers muss immer bei der ersten Verwendung erfolgen. Die Reihenfolge der Mehrfachnutzungen danach ist beliebig, muss allerdings nach der Inkrementierung erfolgen.

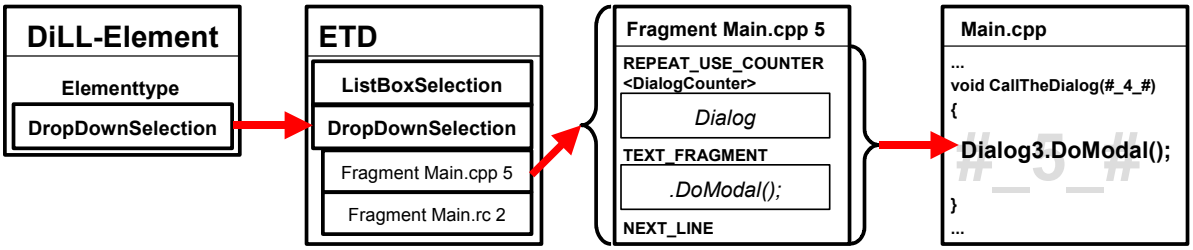

Abbildung 61: Beispielhafter Ablauf der Codegenerierung

Anker: Das ursprünglich geplante rekursive Befüllen der Templates mit Code erwies sich als unzureichend, da beispielsweise Dialogklassen nicht nur an einer Stelle, sondern in Event-Handlern und Konstruktoren befüllt werden müssen.

Aus diesem Grund wurden sogenannte Anker (ANCHOR) eingeführt, die eine durch untergeordnete Elemente zu befüllende Stelle innerhalb des generischen Codes anzeigen. Auf diese Weise können Codefragmente eines Eingabefelds an mehreren Stellen des übergeordneten Dialogs generiert werden.

Zudem können per FOR_EACH Code-Fragmente für jedes untergeordnete Element generiert und mit dessen Attributen parametrisiert werden.

Zugriff auf Parameter (VARIABLE): Abhängig von den Werten im DiLL-Modell müssen für jedes Element andere Werte verwendet werden. Über den VARIABLE-Zugriffsmechanismus können die Attribute des aktuellen Elements ausgelesen werden. Dabei kann aus folgenden Quellen gelesen werden:

- Attribute des Elements
- Attribute seiner Geometrie (GEOMETRY)
- Untergeordnete CONTENT-Knoten[48], die beliebige Name-Value-Paare enthalten können

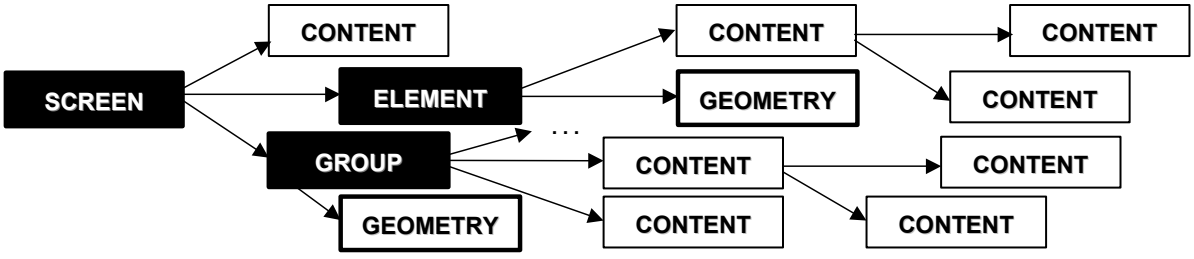

Abbildung 62: Datenquellen für VARIABLE-Zugriffe im DiLL-Modell

[48] eine Beschreibung des CONTENT-Konzepts befindet sich in Kapitel 5.1.2.5

Beliebige Ergebnisse: Durch die weitgehende Generizität ist es möglich, mehrere Oberflächenelemente für einen DiLL-Elementtyp zu generieren, beispielsweise einen Menüeintrag und ein Toolbar-Element. Über die per ETD generierte Funktionalität hat der Generator allerdings ebenso wie über die Template-Dateien keine weiteren Informationen, so dass keine Korrektheitsprüfung stattfinden kann.

7.2.2 Codebasis

Kernstück einer generierten Anwendung ist der Code. Während zumindest für einen Prototypen andere Teile noch geringere Bedeutung haben, ist für Entwickler und partizipierende Benutzer die Anwendung an sich zunächst das zentrale Ergebnis des Generierschritts.

Während der Arbeiten an den Metamodellen und am Generator stellte sich schon früh heraus, dass trotz einer Verallgemeinerung des Ansatzes immer Code-Teile mit generiert werden mussten. Die Hoffnung, dass ein allgemeiner Ansatz mit Ressource-Files auf Code-Generierung ganz verzichten könne, scheiterte schon am Dialogablauf.

Eine rein statische Generierung reduziert die Akzeptanz durch die Entwickler bedeutend, da die Anbindung der Basisapplikation und des Fachkonzepts an das generierte statische GUI einen wesentlich höheren Aufwand erfordert und keine lauffähigen Prototypen erzeugt werden können.

Die Vorteile der Codegenerierung liegen in der Berücksichtigung der Dynamik und der Anbindung der restlichen Architektur an die Oberfläche, so dass ein compilier- und ablauffähiger Oberflächenprototyp entsteht.

7.2.3 Manuelle Änderungen und Code-Aktualisierung

Solange nur ein Oberflächen-Wegwerfprototyp generiert werden soll, bietet die beschriebene Vorgehensweise alle notwendigen Möglichkeiten.

Probleme ergeben sich vor allem dann, wenn eine zweite Generierung unter Berücksichtigung der schon vorhandenen, manuell geschriebenen Anwendungsteile durchgeführt werden soll. Der hier nötige Aufwand ist vergleichbar mit der Code-Generierung bei integrierten CASE-Werkzeugen wie Innovator [MID 1997] oder Rational Rose [Rational 1999].

Änderungen am Template sind jederzeit möglich und – sofern ausreichend – auch die einfachste Vorgehensweise.

Änderungen außerhalb der generierten Blöcke könnten durch die Markierung dieser Blöcke im Template verfolgt werden, da die entsprechenden Teile bei inhaltlicher Abweichung anstatt des Codes im Template kopiert werden können.

Probleme treten auf, wenn Änderungen innerhalb eines generierten Code-Blocks vorgenommen werden. Änderungen sind hier nur durch die Generierung entsprechender IDs als Kommentare möglich, so dass jede generierte Zeile einem IML- bzw. DiLL-Objekt zugeordnet werden kann.

Versionsvergleiche und ähnliche Vorgehensweisen sind auch möglich, allerdings sehr aufwändig. Das Problem ist mit einfachen Mitteln also nicht zu lösen. Eine Vorgehensweise, ähnlich der bei der Code-Generierung von integrierten CASE-Tools üblichen, bietet sich also an.

Die durch die Diplomarbeit geschaffene Version beschränkt sich auf das Oberflächenprototyping mit Überschreiben und stellt keinen über die Anpassung von Templates hinausgehenden Aktualisierungs-Mechanismus zur Verfügung.

7.2.4 Windows-Ressource-File-Format : Generierung von RC-Dateien

Unter Windows können weitgehend compilerunabhängige Ressourcen-Dateien verwendet werden. Somit ist eine Trennung von Quellcode und großen Teilen der statischen Beschreibung der Benutzungsoberfläche (Ressourcen-Beschreibung) möglich. Neben Interaktionselementen können hier auch Schriften und Grafiken eingebunden werden. Ein Ressource-Compiler übersetzt die RC-Dateien bevor sie per Linker zum restlichen Programm gebunden werden.

Die Position der Elemente wird in Dialogeinheiten beschrieben, die abhängig von der Größe der Systemschrift berechnet werden. Positionen werden relativ zum Fenster angegeben, auch dann, wenn es sich um Elemente innerhalb von Gruppen handelt.

Stilparameter, sichtbarer Name und ein Index ergänzen die Parametrisierung.

```
CONTROL <Name>, <Identifier>, <Elementtyp>, <Styleparameter>, <X-Position>,
<Y-Position>, <Breite>, <Höhe>

CONTROL „Label", ID_Txt_Name, „STATIC", SS_LEFT | WS_CHILD | WS_VISIBLE |
WS_GROUP, 20, 20, 30, 10
```

Abbildung 63: Generelle und beispielhafte Struktur einer Elementdefinition im MS-RC-Format.

Der Resource-Compiler überträgt die .RC-Dateien ins .RES-Format. Die .RES-Datei kann dann direkt gelinkt werden. Enthaltene Ressourcen, Controls oder Statements können dann von der Applikation bei Bedarf geladen werden.

Das Problem der Microsoft Resource-Files, bei manueller Erstellung durch einen Entwickler, ist die Positionierung relativ zum Fenster, und dies auch bei Elementen in untergeordneten Gruppen. Die automatisierte Positionsberechnung des Transformators ist hier ein Vorteil für den Entwickler.

Die Positionierung anhand von Dialogeinheiten hat den Vorteil, dass bei geänderter Systemschriftart die Dialoggröße entsprechend angepasst wird, da Dialogeinheiten unter Windows relativ zu Systemschriftgröße definiert sind.

Somit ist das RC-Format, auch aufgrund seiner Verbreitung, der beste Kandidat für das Layout-Ausgabeformat. Andere Formate sind entweder sprachabhängig oder nutzen Containerkonzepte, die einer Positionsgenauen Generierung entgegenstehen.

Das einzige Problem ist die Bestimmung der Positionen im Fenster, die hier schon vor der RC-Generierung stattfinden muss. Die entsprechenden Werte müssen also vor der Generierung im DiLL-Modell berechnet worden sein, und von dort als Attribut oder CONTENT einzulesen sein. Vom Template- und ETD-Mechanismus her sind die Ressourcen-Dateien dem Code gleichgestellt, und werden parallel mit derselben ETD erzeugt.

7.3 Generierung von Hilfe und Dokumentation

Während Handbücher oftmals kaum gelesen und nur in seltenen Fällen zur Referenz herangezogen werden, wird eine gute kontextsensitive Hilfe von vielen Benutzern gern angenommen.

Die meisten Online-Hilfen entstehen allerdings erst gegen Ende des Softwareentwicklungsprojekts, so dass der anfangs meist relativ gute Kontakt zu Kunden und Benutzern oftmals nur noch marginal ist und zudem Zeitdruck das Projekt beherrscht.

Grund genug also, die Hilfetexte möglichst früh, und möglichst passend zum Modell des Benutzers von der zu entwickelnden Anwendung B(A), zu verfassen. Das IML-Modell sieht für jedes Element eine Beschreibung und teilweise auch eine Kurzbeschreibung vor.

Tooltip: Die Kurzbeschreibung kann unter Windows direkt in einen „Tooltip", unter anderen Betriebssystemen in eine entsprechende Kurzhilfe oder einen eingeblendeten Hinweis umgewandelt werden. Unter Windows werden die Tooltips innerhalb der Ressourcen-Datei eingetragen und zugeordnet.

CHM- und HTML-Dateien: Die vorhandenen Beschreibungstexte können direkt in ein HTML-Format übertragen werden. Anhand der ebenfalls im IML-Modell vorhandenen Hilfe-IDs kann zudem der Zugriff auf die entsprechenden Seiten direkt eingebettet werden.

Während eine normale HTML-Hilfe einen Zugriff über den Web-Browser benötigt, können die ähnlich zu erzeugenden CHM-Dateien unter Windows mit dem entsprechenden Hilfeprogramm von Microsoft angezeigt werden. Der Vorteil hierbei sind erweiterte Funktionen zur Navigation zwischen den Hilfe-Kontexten.

Standardtexte: Für jeden Dialog kann eine Standard-Hilfe schon ohne Zusatzinformationen generiert werden. Beispielsweise ist das Durchlaufen der Elemente mit Hilfe der Tabulatortaste vorwärts und rückwärts möglich – allerdings dem Anwender oft nicht bekannt.

Erzeugt wird also eine Standardhilfe für die Oberfläche des gewählten Zielsystems und eine Beschreibung der vom Transformator verwendeten Elementtypen.

Generische Hilfetexte für einzelne Elemente sind ebenfalls möglich. Während die beschriebenen Standardtexte für Fenster oder Bereiche immer generiert werden können, kann in Abhängigkeit von Elementen und Elementkombinationen teilweise auch eine erweiterte Hilfe generiert werden – beispielsweise eine Übersicht über die Bedienung der enthaltenen Elemente.

Hierzu wird eine Bibliothek von Texten benutzt, die jedem Interaktionselementtyp einen entsprechenden Hilfetext zuordnet. Allerdings wird hierzu das DiLL-Modell zusätzlich zum IML-Modell benötigt, da die Elementtypen erst innerhalb des DiLL-Modells festgelegt werden. Die Text kann mit der Elementbezeichnung und weiteren Modellinformationen parametrisiert werden.

Übersichten über die Hilfeinhalte zu einem Dialog oder Use-Case können ebenfalls automatisch generiert werden. Im IML-Modell sind bereits die Elementzusammensetzung und die Elementreihenfolge vorhanden. Das Resultat kann beispielsweise eine Überblicksseite sein.

Ablaufbeschreibungen: Zudem stehen die vielfältigen Beschreibungstexte aus den Use-Cases zur Verfügung, die ebenfalls in die Beschreibung integriert werden können. Da aus dem IML-Modell direkt die Benutzungsschnittstelle hergeleitet wird, stimmen die Ablaufbeschreibungen mit der erzeugten Oberfläche überein und können somit verwendet werden.

Kontextmenü: Das für jedes InteractionCase, für jedes Element und teilweise auch für jeden Listeneintrag definierbare Kontextmenü steht schon im IML-Modell, spätestens aber im DiLL-Modell, zur Verfügung. Aus dem Kommandonamen und den mit ihm verbundenen Funktions- und Dialogaufrufen kann wiederum eine Beschreibung generiert und bei Bedarf auch per Link in den Beschreibungstext für das jeweilige Objekt integriert werden.

Entwicklerdokumentation: Aus den im Modell vorhandenen Informationen, speziell den Entwicklungs-Beschreibungstexten, können Handbuchtexte, Übersichten und Checklisten für Entwicklerhandbücher erzeugt werden.

Use-Case-Diagramme und weitere interaktionsorientierte Diagramme können ebenfalls aus den vorhandenen Informationen erzeugt werden, so auch Relationsdiagramme, die die vielfältigen Relationstypen zeigen, beispielsweise ein erweitertes Use-Case-Diagramm.

Zudem ist es möglich aus den Relationen Checklisten oder Empfehlungen für den Umgang mit bestimmten Use-Cases abzuleiten.

Eine Darstellung der implizit enthaltenen Aufrufrelationen von InteractionCases kann zudem einen guten Überblick über die feineren Dialogabläufe liefern, wie es durch Use-Cases-Diagramme nicht möglich ist.

Ablaufdiagramme, beispielsweise in Form von endlichen Automaten, könnten aus der Interaktionsbeschreibung im IML-Modell ebenfalls erzeugt werden. Auch Task Object Charts (TOC, [Ziegler 1996]) wären ein mögliches Zielformat.

Anwendungs-Datenmodell: Aus den Datenbeschreibungen und den Definitionen im Use-Case-Bereich des IML-Modells können Schaubilder der Datenzusammenhänge generiert werden. Mögliche Diagramme sind beispielsweise Entity-Relationship-Diagramme (ERD) oder OOA-Diagramme wie sie bei JANUS [Hofmann 1998] Verwendung finden. Dabei stellt das IML-Modell sowohl Datentypen als auch Relationen zur Verfügung.

Die Implementierung des Fachkonzepts muss allerdings nicht zwangsläufig dieselbe Struktur haben, so dass die aus dem IML-Modell generierten Datenmodellansichten für die Entwicklung nur Vorschlagscharakter haben und lediglich die aus Benutzersicht logisch erscheinende Modellierung wiedergeben.

Projektdokumentation: Einträge über Tests, Reviews und andere für Projektablauf und -status relevante Daten können ebenfalls aus dem IML-Modell gelesen und in einen Statusbericht oder eine Projektablauf-Dokumentation umgewandelt werden.

Aufwandsabschätzungen für Teile des Projekts, einzelne Systeme und das ganze Projekt können zu einem Bericht verarbeitet und ausgegeben werden, da eine Schätzung und ein Status für die einzelnen Teile im IML-Modell integriert ist.

Generations-Dokumentation: Während der Generierung werden viele Entscheidungen getroffen und einige Abbildungen durchlaufen. Zur Unterstützung der Entwickler wäre deshalb eine Dokumentation dieses Prozesses hilfreich.

So könnte der gesamte generierte Code nochmals in Form einer HTML-Datei erzeugt werden, die zu den vorhergehenden DiLL-Objekten und von diesen aus zum IML-Modell Links enthielte. Der Entwickler könnte so in der HTML-Datei die entsprechende Stelle ansteuern und würde per Link zu einer Dokumentation der entsprechenden DiLL-Objekte, IML-Objekte und Abbildungsentscheidungen geleitet.

Begriffslexikon: Durch eine schon in der Spezifikation klar geregelte Begrifflichkeit können in der Analyse vorhandene Missverständnisse beseitigt und neue vermieden werden, was sich vor allem positiv auf die Kommunikation mit Benutzern und Anwendern auswirkt.

Die XML-basierte IML sorgt durch einen Begriffsdefinitionsblock für jedes Projekt für die zentrale Ablage von projektrelevanten Begriffen. Aus diesen Begriffsdefinitionen könnte durch Sprachwahl und Sortierung ein Begriffslexikon erstellt werden.

Außerdem wäre die Generierung einer Liste von Begriffsdefinitionen innerhalb der Hilfe und des Benutzerhandbuchs möglich – dank der zentralen Definition im IML-Modell.

7.4 Testfälle

Die Generierung von Testfällen und Testhilfsmitteln trägt wesentlich zur Erwartungskonformität, Fehlerrobustheit und Qualität der Software bei, da hier überprüft wird, ob die vom Benutzer erwartete Funktionalität tatsächlich vorhanden ist und ob fehlerhafte Eingaben abgefangen werden.

7.4.1 Übernahme von Testfällen

Die einfachste Möglichkeit zur Testfall-Generierung ist die Verwendung der in TEST_CASE-Blöcken direkt verfügbaren Texte. Dabei ist es möglich, eine Report-Software entsprechend zu parametrisieren oder Texte für einen Browser oder Texteditor zu generieren, die ein entsprechendes Ausfüllen der Formulare ermöglichen.

Die IML sieht für die Ansteuerung von einer Report-Software aus eine direkte Beschreibung von Status und Ausführungsdatum vor. Zudem ermöglicht sie die Angabe einer Referenz auf eine externe Protokolldatei oder deren textuelle Einbindung.

7.4.2 Automatische Generierung von Testfällen aus dem IML-Modell

Für den Entwickler interessanter ist die Möglichkeit einer automatischen Testfallgenerierung aus den InteractionCases heraus.

Use-Cases, die im Hauptmenü ansprechbar sind, können als Ausgangspunkt benutzt werden. Dabei können Abläufe für den Normalfall und alternative Abläufe berücksichtigt werden. Unter anderem sind folgenden Punkte zu berücksichtigen:

- Für jeden Ablauf muss ein separater Testfall generiert werden. Eine Wiederholung kann eine Zuordnung der Ergebnisse erschweren.
- Für jedes BRANCHING_IF müssen zwei Durchläufe generiert werden: Einmal bei erfüllter Bedingung und einmal bei nicht erfüllter Bedingung.

- Das Verhalten für Eingaben an den Grenzen der Definitionsbereiche des Mengengerüsts sowie außerhalb und innerhalb muss getestet werden (Minimum, Maximum).
- Abhängige Elemente müssen auf Deaktivierung durch die steuernden Elemente überprüft werden.
- Die Bedienung von Navigationsknöpfen muss getestet werden.
- Selektionen können nur mit Einträgen getestet werden, die schon zum Generierzeitpunkt im IML-Modell oder in externen aber referenzierten Dateien existieren.

Die so ermittelten Testfälle können sowohl in der Art einer Durchführungsanweisung als auch in Form eines Berichts über den Testverlauf ausgegeben werden, den der Testverantwortliche während des Tests ausfüllt.

Um diese Texte zu generieren, müssen für die in IML verwendeten Interaktionen jeweils Template-Sätze formuliert werden, die mit Hilfe der im Modell enthaltenen Informationen parametrisiert werden können.[49]

7.4.3 Automatische Generierung von Testprogrammen aus dem IML-Modell

Wie die Applikation selbst, können auch Testtreiber-Templates gefüllt werden. Beispielsweise kann eine Testklasse zusätzlich zum restlichen Code angelegt werden, die für jeden Testfall eine Methode enthält.

Eine Switch-Routine steuert diese dann abhängig von der Durchlaufnummer an, die beispielsweise in einer Datei gespeichert sein und dann je Durchlauf inkrementiert werden kann.

Eine Funktionalität zur Generierung einer Log-Datei kann dabei schon im Testtreiber-Template vorhanden sein. Das Ergebnis kann wiederum zu einem Report oder zu einem Eintrag in einer globalen Test-Log-Datei konsolidiert werden.

Soll die Oberfläche wie von einem Anwender angesteuert werden, kann keine direkt im Code wirkende Testroutine genutzt werden. Vielmehr muss dann entweder eine Ansteuerung für eine externes Test-Tool generiert werden oder die Mauspositionierung und Tastatureingaben müssen direkt an das Betriebssystem übergeben werden, so dass die Ansteuerung aus Applikationssicht nicht mehr von einer wirklichen Ansteuerung durch den Benutzer zu unterschieden ist.

7.4.4 Checklisten für Reviews

BUSINESS_RULES, d. h. fachliche Rahmenbedingungen der Anwendungsdomäne, und IMPLEMENTATION_NOTES, d. h. in der Implementierung zu berücksichtigende Hinweise, sind zwei Beispiele für IML-Einträge, die als Checklisten Fehler vermeiden helfen.

Prinzipiell lassen sich alle aufzählungsartig im IML-Modell aufgeführten Informationen zu Grundlagen für Reviews und Überprüfungen verarbeiten. Die einfache Generierung von HTML aus XML mit Hilfe von XSLT[50] liefert hier Anknüpfungspunkte für HTML-Ausgaben aller im Modell vorhandenen Informationen.

7.5 Daten-/Domainbeschreibung

Soll das Fachkonzept bzw. die Datenhaltung per Generierung an die Oberfläche angebunden werden, müssen auch Teile der Datenhaltung generiert werden. Ein Ansatz hierzu wird im Folgenden beschrieben.

[49] in der Art von: „Der Benutzer <ActorName> wählt bei <DISPLAYED_NAME> den Eintrag <VALUE_ITEM_ENTRY> aus."
[50] XSL-Transformationen arbeiten auf XML und können trotz ihrer Mächtigkeit leicht implementiert werden.

7.5.1 Generierung von Businessobjekten aus DTDs

Bei ETAS steht ein DTD-basierter Generator zur Verfügung, der es ermöglicht, aus DTDs direkt Businessobjekte für die Anwendungen zu generieren. Die Businessobjekte kapseln dabei den Datenzugriff auf das XML-basierte Datenbanksystem Object-Server.

Die Fassaden-Businessobjekte (FBO) kapseln dagegen die direkt mit der Oberfläche zusammenhängenden Funktionen wie das Überprüfungen von Eingabewerten. Sie sorgen zudem für Methodenaufrufe und Datenaustausch zwischen Businesslogik und GUI.

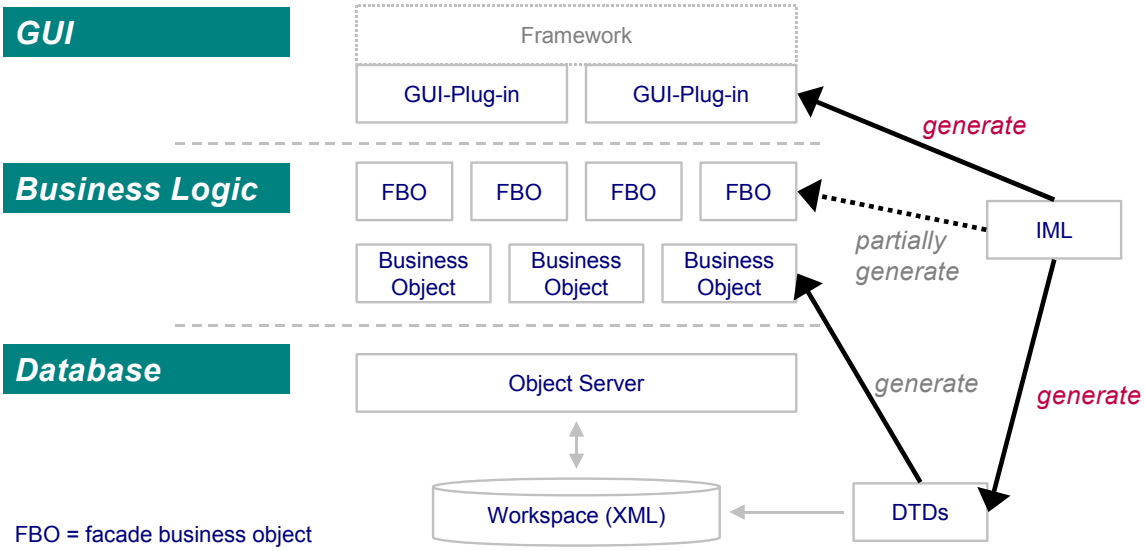

Abbildung 64: YUKON-Applikationsstruktur mit Funktion der DTD und der IML

Können DTDs aus dem IML-Modell generiert werden, ist es also in vielen Fällen möglich die wiederum aus den DTDs generierten Businessobjekte mit der Oberfläche zu verbinden. Zumindest aber existiert für jedes Businessobjekt auch eine entsprechende Repräsentation auf Seiten der Benutzungsschnittstelle.

Die IML-Transformation ermöglicht eine generative Herangehensweise in der Art einer „Zangenbewegung“ durch die Generierung der Oberflächenfunktionalität und der entsprechenden Datenstruktur-Repräsentation, so dass nur der funktionale Teil des Fachkonzepts ohne die Datenhaltung ergänzt werden muss.

7.5.2 DTD-Generierung

Die DTDs für ETAS-Systeme sind hierarchisch aufgebaut. Jeder Eintrag kann über Listen untergeordneter Einträge verfügen, wobei die Listen als Tag modelliert sind, die wiederum über eine beliebige Anzahl der untergeordneten Einträge verfügen können:

```
<!ELEMENT TASK (PROCESSES)>
<!ELEMENT PROCESSES (PROCESS*)>
<!ELEMENT PROCESS EMPTY>
```

Abbildung 65: Beispiel einer DTD für Tasks im Experimental Target Configurator

Da eine Darstellung aller auszuwählenden Hierarchiestufen erfolgen soll, kann diese als SELECT angegeben werden. Jedes SELECT verfügt über eine Werteliste (VALUE_LIST). Hier sind die Wurzelknoten für die zu generierenden DTDs als Wert-Einträge (VALUE_ITEM) enthalten.

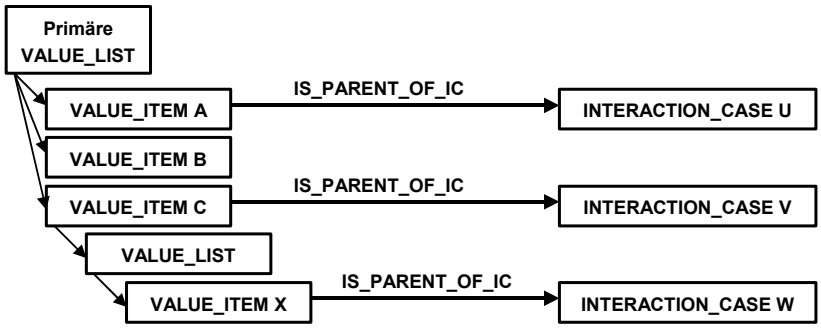

Abbildung 66: Beispielhafte Struktur der hierarchischen Wertedefinitionen für die DTD-Generierung

Ist ein dem Knoten zugeordnetes InteractionCase vorhanden, kann dieses über IS_PARENT_OF_IC angegeben werden. Die im referenzierten InteractionCase enthaltenen Felder werden dann zu Attributen des, aus dem Wert-Eintrag erzeugten, Tags in der DTD.

Aus der Liste in der Abbildung oben würde folgender DTD-Code generiert:

```
<!ELEMENT ROOT_NODE (A_List, B_List, C_List)>
<!ELEMENT A_List (A*)>
<!ELEMENT B_List (B*)>
<!ELEMENT C_List (C*)>
<!ELEMENT C (X_List)>
<!ELEMENT X_List (X*)>
<!ELEMENT X EMPTY>
```

Abbildung 67: Beispiel für generierten DTD-Code

Die in den InteractionCases U, V und W enthaltenen Elemente würden zusätzlich als Attribute zu den Elementen A, C und X definiert.

Angenommen InteractionCase V enthält das Eingabefeld Priority mit Vorgabewert 0 und das Auswahlfeld AutoStart mit den Listeneinträgen „True“ und „False“, wobei „True“ Standardwert ist, dann würde der folgende Eintrag für C generiert:

```
<!ATTLIST C
    Priority CDATA "0"
    AutoStart (True | False) "True"
>
```

Abbildung 68: Generierte Attribute in der DTD

Die DTD-Generierung kann also direkt aus dem Interaktionsmodell in IML vorgenommen werden. Nötig ist dazu nur die Definition eines Hauptauswahlelements das beispielsweise über seinen Namen, oder die Angabe eines speziellen Elementtyps, identifiziert wird. Alle weiteren Informationen können mit Hilfe dieser Referenz aus dem IML-Modell herausgelesen werden.

Die Definitionen können dabei wahlweise in eine DTD pro Ebene aufgeteilt werden. Sinnvoller und einfacher zu generieren ist jedoch eine DTD pro vollständigem Baum.

Die später auf dieser Grundlage entworfene Architektur ist für die IML nicht verfügbar, auch wenn für jeden Typ der Oberfläche bzw. des Baumes ein entsprechender Eintrag in der Datenbank und im Businessobjekt existiert.

8 Realisierung, Anwendung und Evaluation des Verfahrens

Neben den in der Aufgabenstellung zur Diplomarbeit angegebenen Anforderungen, erfüllen die IML und die weiteren Bestandteile von *HERBS* noch eine Reihe weiterer Funktionen. Im Folgenden sollen deshalb Möglichkeiten und Grenzen des Verfahrens näher beleuchtet werden.

Wie in der Aufgabenstellung vorgeschlagen erfolgte die prototypische Anwendung der erarbeiteten Methodik in Form eines Transformators, der mit Hilfe einer automatisierten Umsetzung der IML in einen Oberflächenprototypen die Einhaltung software-ergonomischer Grundsätze sicherstellt.

8.1 Implementierung eines Transformators

Um den Nachweis der Realisierbarkeit der Konzepte zu führen, wurde ein Transformator entwickelt, der die geschilderten Konzepte beispielhaft umsetzt. Dabei war vor allem die Durchgängigkeit des Transformationsprozesses wichtig, d. h. die Abbildung aller genannten Stufen.

Der vorliegende Transformator wurde mit Microsoft Visual C++ realisiert, da dies auch die Zielsprache für den Codegenerator sein sollte. Um einen einfachen Zugriff auf die XML-Dateien zu erhalten, wurden die Microsoft XML Core Services 4.0 in Verbindung mit C++-Wrapperklassen aus [Wiegand 2002] genutzt.

Der Transformator arbeitet als Kommandozeilenprogramm, könnte aber, falls erforderlich, auch auf eine grafische Oberfläche umgestellt werden.

Einlesen und Vervollständigen: Das Einlesen und Parsen erfolgt weitgehend durch die oben genannten Werkzeuge. Für die Vervollständigung von IDs wurde eine generische Routine in C++ implementiert, die durch unterschiedliche Parametrisierung alle IDs wie gewünscht ergänzen kann. Eine Prüfung auf Vollständigkeit der Texte in allen Ziel-Sprachen wird ebenfalls durchgeführt. Das Ergebnis wird in eine neue Datei geschrieben.

Die grundlegenden Werte werden von einem ProjectData-Objekt eingelesen und während des ganzen Transformatorablaufs bereitgehalten. Sie stehen damit allen Stufen zur Verfügung.

IML-DiLL-Transformation: Diese Stufe durchläuft folgende Schleifen:

```
Für alle Systeme
    Erzeuge neues Hauptfenster; Suche Start-Use-Case; Suche dessen Start-InteractionCase;
    Für alle Use-Cases im System
        Erzeuge Menüeintrag mit Aufruf des jeweiligen Start-InteractionCase etc.;
    Für alle InteractionCases im System
        Prüfung auf Typ, Unterordnung etc.;
        Für alle Gruppen und Elemente in diesem InteractionCase
            {Behandle Gruppe; Behandle enthaltene Untergruppen und Elemente | Behandle Element}
```

Für IML-Gruppen und IML-Elemente werden ein oder mehrere DiLL-Objekte generiert. Platzberechnung, Label-Erzeugung, Typenauswahl und alle anderen Schritte werden auf diesem Modell ausgeführt. Dabei werden zunächst alle Elemente weitgehend linear untereinander erzeugt.

Auch die Layoutoptimierung kann auf diesem Modell stattfinden und dabei Spaltenumbrüche und Geometrien berechnen.

Das Hauptfenster (DiLLMasterScreen) enthält eine Liste aller Fenster (DiLLScreen). Da die Fenster Elemente und Gruppen, sowie die Gruppen wiederum Elemente und Gruppen, enthalten, kann das ganze DiLL-Modell durchwandert werden. Das Ergebnis wird mit Hilfe dieser Hierarchie gespeichert, indem jedes DiLL-Objekt vom Hauptfenster ausgehend sich und seine Child-Elemente in ein XML-Fragment schreibt. Alle Fragmente zusammen bilden die DiLL-Datei.

Die verwendete DiLL-Klassenhierarchie wurde bereits in Kapitel 5.1.3 beschrieben. Die Implementierung ist in den entsprechenden Dateien zu finden (DiLLElement.cpp, ButtonGroup.cpp, DiLLObject.cpp, DiLLGroupClass.cpp, DiLLScreen.cpp, DiLLMasterScreen.cpp).

Weitere Module mit ihren Funktionen sind:

- ProjectData.cpp: Stellt eingelesene und erzeugte Daten wie Projektname und Transformationsparameter für jede Transformatorstufe zur Verfügung.
- GlobalFunctions.cpp: Enthält Funktionen wie XML-Attributerzeugung und XPath-Suchstringerzeugung, die von allen Modulen benötigt werden.
- DOMXMLDemo.cpp: Hauptprogramm, das die Dokumente lädt und die einzelnen Stufen ansteuert.
- DiLLHandler.cpp: Steuert die Umwandlung von IML in DiLL und spricht die DiLL-Objekte von DiLLMaasterScreen bis DiLLElement an.
- ButtonGroup: Repräsentiert eine Navigations- oder Funktionsknopfgruppe und ist eine durch Ableitung erzeugte Spezialisierung von DiLLGroup.
- ValueListHandler.cpp: Behandelt Wertelisten in IML-SELECT-Tags und erzeugt daraus Content-Tags und Elementtypisierungsempfehlungen für das DiLL-Modell.
- Synthesis.cpp: Liest die ETD und erzeugt mit Hilfe dieser und des DiLL-Modells die lineare XML-Repräsentation der Code-Fragmente.
- FinalArtifact.cpp: Liest die Templates ein, füllt diese mit den Informationen aus den in Synthesis.cpp erzeugten Code-Fragmenten und schreibt die Dateien in ein ebenfalls erzeugtes Projektverzeichnis.

Die in Synthesis und FinalArtifact enthaltenen Teile setzen die gesamte in Kapitel 7.2 beschriebene Funktionalität für die Code-Generierung aus dem DiLL-Modell um. Allerdings muss dazu die ETD alle im DiLL-Modell erzeugten Typen (ElementType) enthalten und korrekt umsetzen können.

Die Templates für FinalArtifact erzeugen ein vollständiges MS-Visual C++ Projekt mit Elementen aus der Stingray-Bibliothek.

Der Transformator kann aus einem einfachen Modell eine ablauffähige Anwendung generieren, die Dialoge mit allen beschriebenen Navigationsmustern und -knöpfen sowie Gruppen und Auswahlelemente enthält. Dabei ist die Umsetzung auf einen Auswahltyp beschränkt, kann jedoch leicht erweitert werden.

Alle Dialoge werden über das Hauptmenü angebunden und sind von diesem aus aufrufbar. Entsprechende Klassen für Dialoge, und Attribute für Elemente, werden ebenfalls erzeugt, so dass der Code direkt erweitert werden kann.

Mit Hilfe der Templates werden Projekt- und Workspacedatei miterzeugt, so dass die Anwendung direkt kompiliert und ohne Änderungen lauffähig ist.

Alle konzeptuellen Schritte sind soweit ausgeführt, dass darauf aufbauend eine Erweiterung zum vollständigen IML-Transformator durchgeführt werden kann.

Die verwendeten DTDs der IML, der DiLL, und der ETD, sowie die zugehörigen Modelle, sind im Anhang enthalten.

8.2 Anwendung auf ein aktuelles Projekt

Um die Praxistauglichkeit zu untersuchen, wurde die Methodik mit Hilfe des Transformators beispielhaft auf eine Anforderungsbeschreibung aus einem aktuellen ETAS-Entwicklungsprojekt angewandt – die Oberfläche des Experimental Target Configurators.

Der Experimental Target Configurator (ETC) wurde als Teil einer Tool-Suite konzipiert. Eine kurze Beschreibung befindet sich in Kapitel 3.1.2.

Eine – auch beim ETC zu berücksichtigende – Besonderheit ist die schon in Kapitel 7.5 beschriebene hierarchische Modellierung eines SELECT-Elements für die Erstellung des Stingray-TreeView-Elements.

Zunächst wird eine ES-1000-Basiskomponente als Wurzelknoten erstellt. Wahlweise kann diese Stufe auch entfallen, da bisher nur jeweils eine ES 1000 enthalten sein kann. Sollen mehrere verwendet werden können, müssen diese als Wurzelelement aufgelistet werden.

Die weitere Modellierung orientiert sich an den enthaltenen Komponenten und gibt diese in Form eines 0..N-Mengengerüsts wieder, wobei N die maximale Anzahl der Bausteine ist.

Interrupts der Controller etc. können als „pool“ oder „applicationwide_pool“ deklariert werden, um Doppelbelegungen zu vermeiden.[51] Die Attribute der Einträge werden als InteractionCases modelliert und können so vom Transformator berücksichtigt werden.

Die einzelnen Karten, die darauf vorhandenen Bauteile und weitere Detaillierungsstufen wurden mit einem SELECT modelliert. Dabei konnten auch Kontextmenüs etc. angegeben werden. Für jeden Eintragstyp wurde ein InteractionCase modelliert und per ON_FOCUS oder CALL_IC bzw. IS_PARENT_OF_IC mit dem Baumauswahlelement verbunden.

Die Modellierbarkeit ist also nachgewiesen, d. h. das IML-Metamodell ist mächtig genug, die notwendige Oberflächenfunktionalität zu modellieren.

Die Umsetzung durch den Transformator liefert eine lauffähige Anwendung, die eine entsprechende Navigation enthält. Von der modellierten Hierarchie unabhängige Dialoge werden direkt in das Menü eingefügt. Die zu Einträgen im Baum gehörenden Dialoge sind ebenfalls als Ressourcen vorhanden.

Generier-Ergebnis: Das Anwendungsgerüst ist in dieser Form lauffähig. Über Menü erreichbare Dialoge werden richtig eingebunden und aufgerufen. Der Aufruf direkt über den Baum muss manuell eingegeben werden, da eine Eventbehandlung für den Treeview in den Templates nicht vorgesehen ist.

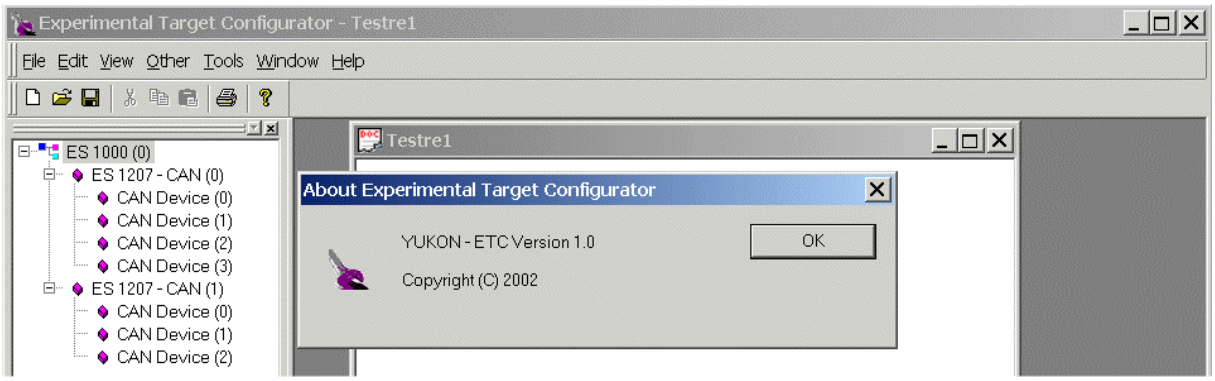

Abbildung 69: Beispielansicht für den generierten und verbesserten ETC-Prototyp

Die Navigation der Dialoge wird korrekt abgebildet. Alle Navigations-Patterns und -knöpfe werden richtig übernommen und angeordnet.

Die Dialoge werden zwar weitgehend vollständig erzeugt, enthalten aber nur wenige unterschiedliche Elementtypen, da diese – wie schon in den vorangegangenen Kapiteln beschrieben – noch in den Transformator integriert werden müssen.

Die Erzeugung eines Prototyps ist also möglich. Für eine vollständige Umsetzung fehlen jedoch noch bestimmte Elemente, wie z. B. Radiobuttons. Zudem wird die Dynamik nur in Bezug auf Menüaufrufe umgesetzt. Eine weitergehende Dialogsteuerung wird nicht generiert.

[51] Das Pool-Konzept wurde in Kapitel 4.5.2.3 erläutert.

Deshalb müssen die entsprechenden Elemente manuell durch andere ersetzt und die weitere Dialogsteuerung manuell realisiert werden. Da Dialogklassen und -attribute schon generiert werden, kann das Fachkonzept direkt an die Oberfläche angebunden werden.

Einen genaueren Überblick über die bisher realisierten Teile von HERBS bietet der folgende Abschnitt.

8.3 Ausbaustufen von Methode und Transformator

Der Ausbau der einzelnen Stufen und Konzepte folgt weitgehend einem Trichter-Prinzip: Eine konzeptuell breite Basis wird exemplarisch bis zur letzten Stufe und Spezialisierung durchgeführt, wobei spätere Ergänzungen in der Breite vorbereitet, aber nicht durchgeführt werden.

Wie im Folgenden beschrieben, wird durch den Transformator der vollständige Prozess zwar seiner Länge nach abgedeckt, jedoch nur für wenige Elemente abgeschlossen, die exemplarisch für andere zu betrachten sind. Folgende Schritte werden abgedeckt:

Überblick über GUI-Generatoransätze: Ein umfassender Überblick über die existierenden Ansätze zur ergonomischen Oberflächengenerierung ist im Grundlagenkapitel und teilweise im Anhang enthalten und deckt alle dem Autor bekannten software-ergonomischen Generatoren ab.

Modellierungskonzept: Die Interaktionsmodellierungssprache IML wurde soweit wie möglich entwickelt, vervollständigt und auf die Verwendungsfähigkeit zur Generierung der angesprochenen finalen Artefakte überprüft. Eine DTD mit der vollständigen Modellierungssprache steht zur Verfügung.

IML-Vervollständigung: Alle zur Oberflächengenerierung benötigten Vervollständigungen[52] wurden in die Implementierung integriert. Das Prinzip der Vervollständigung steht für Erweiterungen zur Verfügung und ist dokumentiert. Durch die XML-Basierung können beliebige weitere, externe Vervollständigungsschritte ergänzt werden.

Plattformunabhängige Dialogrepräsentation: Mit der Dialog Layer Language (DiLL) steht eine weitgehend plattformunabhängige Repräsentation der Dialogstruktur für Erweiterungen und Optimierungen sowie als Basis für die Oberflächengenerierung. Es existiert eine komplette DTD und ein Generator für die DiLL. Erweiterungen der DiLL sind zumindest im dynamischen Teil nötig, durch die XML-Basierung aber auch mit einfachen Mitteln möglich.

IML-DiLL-Transformation: Die Umsetzung in ein korrektes und weiterverarbeitungsfähiges DiLL-Modell mit Navigation, Menüstruktur und Elementen wird von der Transformationsstufe erledigt. Allerdings ist die Anzahl der Elemente sehr stark eingeschränkt. Auch die Überprüfungen auf Mengengerüst etc. sind rudimentär, so dass hier Erweiterungsbedarf besteht. Eine Auswertung für einen Großteil des IML-Modells ist jedoch strukturell vorhanden.

DiLL-Layout-Optimierung: Für die Layout-Optimierung wurden bereits Objektstrukturen angelegt, die über entsprechende Funktionalität und Erweiterungsmöglichkeiten zur Positionierung und Berechnung von Metriken sowie optischen Zeilen und Spalten verfügen. Die grundlegenden Regeln wurden in dieser Arbeit beschrieben, jedoch nur soweit implementiert, wie für die Transformation in eine initiale Oberfläche notwendig. Algorithmen anderer Ansätze können jedoch auf die DiLL-Objektstruktur angewandt werden.

ETD: Die Transformationsbeschreibung für Elemente in Code und Ressourcen wird auf eine geringe Anzahl Dialoge, Gruppen und Elemente begrenzt, jedoch sind alle Konzepte für komplexe Strukturen vorhanden. Eine leichte Erweiterbarkeit wird durch die XML-Basierung und die Tatsache gesichert, dass neue Elemente rein durch Erweiterung der ETD hinzugefügt werden können. Der Transformator arbeitet hier vollkommen generisch.

Template-Mechanismus: Der Template-Mechanismus zur Bereitstellung eines Applikationsgerüsts ist vollständig implementiert. Anhand eines für den Transformator erstellten MFC-Stingray-Anwendungstemplates wurde mit Hilfe eines einfachen Modells eine lauffähige Applikation erzeugt.

[52] Dabei handelt es sich um Generierung von Hilfe- und Item-IDs, Erstellung aller nicht spezifizierten Attribute sowie Vollständigkeitsüberprüfung für alle mehrsprachigen Einträge nach vorgegebenem Muster.

Allerdings müssen für andere Umgebungen entsprechende Templates erzeugt werden, wenn keine MFC-Stingray-Anwendung mit hierarchischem Navigationselement entstehen soll.

DiLL-Code-Transformation: Die Transformation des DiLL-Modells mit Hilfe der ETD in eine weitgehend lineare XML-Darstellung und deren Einfügen in die vorgesehenen Templates sowie die Erzeugung eines vollständigen, ablauffähigen Projekts für Microsoft Visual C++ 6 ist bereits möglich. Da dieser Teil komplett generisch gehalten ist, konnte er vollständig und mit allen Konzepten implementiert werden.

Testfälle: Beschriebene Testfälle können extrahiert und zu einer textuellen Darstellung weiterverarbeitet werden. Für zu generierende Testfälle sind die entsprechenden Informationen wie Grenzen, Mengengerüst etc. vorhanden. Jedoch müsste für eine Erzeugung von vollständigen Testfällen oder für eine Generierung von Testtreibern ein weiterer Generator geschrieben bzw. der vorhandene um Testfunktionen erweitert werden. Entsprechende Hinweise befinden sich im Kapitel 7.4.

Hilfe: Für die Hilfegenerierung stehen die entsprechenden Einträge aus dem IML-Modell, versehen mit einer ID, zur Verfügung. Aus diesen Angaben kann eine Hilfe-Datei erzeugt werden, die mit den bereits in das Generat integrierten Konstanten angesteuert werden kann. Die Erzeugung der Hilfedateien würde ebenfalls einen neuen Generatorschritt erfordern.

Gesamtdurchlauf: Der Transformator ist so konzipiert, dass er Zwischenergebnisse (IML complete, DiLL und lineare Darstellung) speichert, jedoch die komplette Pipeline einmal durchläuft. Um zwischengeschaltete Änderungen vorzunehmen, müssen die einzelnen Stufen getrennt werden, was jedoch sehr einfach zu realisieren ist: Die einzelnen Stufen werden direkt im Hauptprogramm aufgerufen. Ein erfolgreicher Gesamtdurchlauf wurde mit zwei IML-Modellen durchgeführt. Durch die oben genannten Einschränkungen wird jedoch nur ein Teil der Elemente erzeugt.

8.4 Vergleich mit anderen Methoden und Aufwandsabschätzung

Im Folgenden soll *HERBS* mit anderen, bisher eingesetzten, Methoden zur Oberflächenentwicklung verglichen werden, vor allem hinsichtlich der Verteilung des Aufwands über die einzelnen Phasen eines Entwicklungsprojekts hinweg.

8.4.1 Vergleich nach Methoden

8.4.1.1 Ablauforientierung

Die grundlegende Orientierung an Interaktionen und Abläufen ist, wie schon mehrfach erwähnt, ein Konzept, das *HERBS* von den meisten anderen modellbasierten Generatoransätzen unterscheidet.

Mit diesem Modellierungsunterschied verändert sich gleichzeitig auch die Herangehensweise. Während bei datenzentrierten Ansätzen Zustände, Entitäten und deren Relationen das Denken in der Spezifikationsphase bestimmen, steht bei *HERBS* die Interaktion von Benutzer und Anwendung im Vordergrund.

8.4.1.2 Prozessunterstützung

Aus Prozesssicht stellt der „Human-Centered Design Process“ für Vergleiche im Bereich der softwareergonomischen Oberflächenentwicklung einen guten Bezugspunkt dar. Von den notwendigen strategischen Überlegungen über den Entwurf bis zum Betrieb des Systems werden bei diesem alle Bereiche in insgesamt sieben Kategorien aufgeteilt. Jede Kategorie verfügt wiederum über Einträge, die einer Check-Liste vergleichbar sind.

Auf diese Weise können auch Modelle auf ihre Vollständigkeit bezüglich „Human-Centered Design Process“[53] überprüft werden. Der in dieser Arbeit vorgestellte *HERBS*-Ansatz deckt mit Hilfe der IML alle sieben Kategorien und, soweit möglich, nahezu alle Unterpunkte ab. Die Prozessunterstützung ist also bereits im IML-Metamodell vorhanden.

[53] Vgl. Human-Centered Design Process Kapitel 2.1.3

8.4.1.3 Single Source

Gerade bei datenzentrierten Ansätzen werden oft mehrere Modelle erstellt, die unterschiedliche Aspekte abdecken. Zudem sind Projektinformationen selbst in CASE-Tools kaum vorhanden. Bei einem IML-Modell befinden sich dagegen alle Informationen im selben Format in derselben Datei.

8.4.1.4 Genereller Aufwandsvergleich

Die grundlegenden Eigenschaften des Konzepts geben schon Anhaltspunkte für die Änderung der Aufwandsverteilung im Vergleich zur textuellen Spezifikation und Entwicklung ohne Generator, d. h. der bisherigen Standard-Vorgehensweise.

So erhöht sich der Aufwand im allgemeinen bei einer Formalisierung der Spezifikation. In den folgenden Phasen sinkt er jedoch durch eine zentrale Datenhaltung und die Generierung von Anwendungsteilen, Projektdokumenten und Testfällen weit unter das Niveau der Standard-Vorgehensweise.

Treten mit Modellierung und Anbindung der nicht generierten Teile keine durch den *HERBS*-Ansatz bedingten Probleme auf, kann Aufwand eingespart werden. Die Neueinführung ist jedoch – abhängig vom bisherigen Einsatz ähnlicher Modellierungsmethoden wie Use-Cases und entsprechender Prozesse – mit einer zusätzlichen Investition verbunden.

Allerdings ist das primäre Ziel der in dieser Arbeit vorgestellten Methode nicht die Optimierung des Entwicklungsaufwands, sondern die Optimierung der Entwicklungsergebnisse aus Benutzersicht. Eine Reduzierung von Fehlern und Aufwand ist hier ein erwünschter Nebeneffekt.

8.4.1.5 Vergleich mit UIMS-Codegenerierung

Die Code-Generierung für UIMS ist deutlich einfacher zu implementieren. Zudem ist eine Änderung der generierten Oberfläche einfacher, da hierzu reiner UIMS-Code angepasst wird. Die Anbindung an die Implementierung des Fachkonzepts gestaltet sich jedoch schwieriger, da die entsprechenden Teile in einer anderen Programmiersprache vorliegen.

Die Bearbeitung einer in Native-Code generierten Benutzungsschnittstelle vereinfacht sich also erheblich, wenn das Fachkonzept und die vom GUI-Generator erzeugten Dateien dieselbe Programmiersprache nutzen.

8.4.2 Vergleich nach Projektphasen

Im Folgenden soll der Aufwand für die einzelnen Phasen eines Softwareentwicklungsprojekts anhand eines Vergleichs mit anderen Ansätzen verdeutlicht werden.

8.4.2.1 Einführungsphase

Die Einführungsphase hängt entscheidend von der Managementunterstützung und der Einstellung der Entwickler ab. Wie jede neue Methode erfordert auch die IML einen Einarbeitungsaufwand, der abhängig von entsprechenden Vorkenntnissen ist.

Stehen schon Kenntnisse in XML oder sogar ein IML-Editor zur Verfügung und werden Use-Cases auch bisher schon eingesetzt, liegt der Einführungsaufwand deutlich unter dem Aufwand für Methoden mit proprietären Modellierungsformaten.

Da anfangs noch keine Templates vorhanden sind – falls diese nicht direkt hinzugekauft werden können oder aus freien Quellen verfügbar sind – nimmt der Aufwand mit zunehmender Standardisierung erst bei Folgeprojekten ab.

Werden benutzerzentrierte Prozesse eingesetzt, bieten das benutzernahe IML-Modell und das frühe, iterative und inkrementelle Prototyping viele Vorteile. Das Roll-Out für *HERBS* im Zuge der Einführung eines benutzerzentrierten Softwareentwicklungsprozessmodells oder Reifegradmodells wie CMM ist sinnvoll. Speziell die noch in der Entwicklung befindliche, auf Usability abzielende, CMM-Variante würde sich dazu eignen.

Der höhere Aufwand für Einführung und Erlernen muss, wie bei CASE-Verfahren üblich, immer durch Vorteile bei der Bildung neuer Versionen oder bei der Wartung ausgeglichen werden.

8.4.2.2 Analyse und Spezifikation

Da ein IML-Modell sehr umfangreich ist, um hochwertige Ergebnisse erzeugen zu können, ist es Methoden zur direkten Modellierung wie bei [Herczeg 1995] während der Analysephase in punkto Aufwand weit unterlegen. Speziell im Vergleich zu einer rein textuellen Beschreibung, beispielsweise in einer User Requirements Specification (URD), benötigt die Formalisierung zusätzlichen Aufwand.

Steht schon ein Use-Case-Modell zur Verfügung, verringert sich der Aufwand erheblich, da die Use-Cases direkt in IML umgesetzt und verfeinert werden können. Andere – meist datenzentrierte – Methoden können Use-Cases kaum verwenden. Zudem muss dann ein Entitätenmodell erstellt werden, das aus den Anforderungen und Abläufen aus Sicht des Benutzers nicht direkt ableitbar ist.

Der Abgleich der Analyse mit Kunden und Benutzern erzeugt bei solchen Modellen einen wesentlich höheren Aufwand und eine größere Kommunikationsproblematik als bei Use-Case- und IML-basierten Ansätzen.

Stehen schon fertige Datenmodelle zur Verfügung und sollen nur entsprechende Datensätze editiert werden, sind Methoden wie bei JANUS oder GENIUS wesentlich schneller und einfacher einzusetzen, da der Entwickler das vorhandene Datenmodell zügig umsetzen kann. Dem Software-Engineering entspricht allerdings ein Daten-Entwurf vor der Analyse.

Prototyping: Für die oberflächenprototyping-gestützte Analyse und Spezifikation stellt die UI-Generierung eine große Hilfe dar. Müssen iterativ mehrere Prototypen gebaut werden, steigt der Aufwand bei manueller Erstellung schnell über denjenigen der eigentlichen Spezifikation.

Prototyping wird oft parallel zur Spezifikation, aber unabhängig von dieser, betrieben und erzeugt damit einen höherem Aufwand. Mit *HERBS* besteht die Möglichkeit, den Anwender mit Hilfe einer generierten Oberfläche direkt mit einzubeziehen. Am Prototyp kann der spätere Benutzer die Bedienung testen und mit seiner Intention vergleichen.

Stehen schon Templates für Standard-InteractionCase oder sogar Standard-Use-Cases zur Verfügung kann die Spezifikation der einfacheren Dialoge wesentlich schneller durchgeführt werden, als dies bei Netzdarstellungen oder ähnlichen Repräsentationen der Fall ist.

Funktionale Anforderungen können zwar im Modell integriert werden, müssen aber trotzdem formuliert werden.

8.4.2.3 Entwurf

Bereits in der Entwurfsphase kann sich der Entwickler bei IML auf einen Oberflächenprototypen stützen, so dass die Oberflächenelemente schon zur Verfügung stehen. Der Entwickler muss also noch das Fachkonzept, beispielsweise in Form von Klassendiagrammen, erstellen und die Anbindung an die Oberfläche konzeptionieren.

Bei einigen anderen generativen Konzepten steht ebenfalls ein Oberflächenprototyp zur Verfügung. Wird allerdings nur eine auf dem Modell basierende Entwurfsunterstützung (IDE) gegeben oder steht allgemein keine Generator-Funktionalität zur Verfügung, muss die Oberfläche manuell modelliert werden. Der Aufwand hierfür ist wesentlich größer.

Erzeugt die generierte Oberfläche für den Entwurf keine Schnittstellenprobleme, dann sorgt sie für einen geringeren Aufwand durch den Wegfall einer sonst notwendigen Modellierung, auch der Oberfläche. Wird der Entwurf stark eingeschränkt, erhöht sich der Aufwand sowohl für den Entwurf als auch für die folgende Implementierung.

8.4.2.4 Implementierung

Bei nicht-automatisierten Ansätzen muss immer alles implementiert werden, selbst wenn dafür von entsprechenden UI-Design-Tools bestimmte Unterstützungen gegeben werden können. Generatoren können stattdessen teilweise finalen Code für die Oberfläche erzeugen.

Die Anbindung des Fachkonzepts ist bei Generatoransätzen abhängig von den entsprechenden Konstrukten der Modellierungssprache. Können Methoden- bzw. Funktionsaufrufe ins Modell integriert werden, ist die Anbindung wie bei IML über das Modell möglich.

Erfolgt die Anbindung manuell, ist eine Generierung bei Oberflächenveränderungen nicht mehr möglich. Bei vielen Ansätzen ist deshalb nur die Generierung eines Wegwerf-Prototypen möglich, was den Aufwand für die (Neu-)Implementierung stark erhöht.

Der IML-Transformator kann soweit ausgebaut werden, dass nicht nur Wegwerf-Oberflächenprototypen generiert werden können. Solange allerdings kein Konzept zur Berücksichtigung manueller Änderungen im Generierergebnis existiert, steigt der Aufwand mit jeder Änderung des Modells oder der Oberfläche. Dem Autor ist allerdings kein Generator-Ansatz bekannt, der manuelle Änderungen innerhalb des Generierergebnisses berücksichtigt.

Sind Generatoren wie JANUS in CASE-Werkzeuge eingebunden, können zum Teil deren Funktionen mitgenutzt und so der Aufwand reduziert werden.

Die Anbindung des Fachkonzepts an die generierte Oberfläche ist bei IML über bestimmte Einträge möglich. Bei Generatoren ohne dieses Konzept liegt der Aufwand für die Implementierung wesentlich höher.

Im Vergleich zur Vorgehensweise ohne Generator ist der Aufwand bei native Code erzeugenden Ansätzen wie IML oder JANUS wesentlich geringer, da nur bei diesen eine vollständige Anwendung in der Fachkonzept-Implementierungssprache erzeugt wird und die Anbindung und Verwendbarkeit sichergestellt ist.

8.4.2.5 Test

Können, wie bei IML, Testfälle oder sogar halb- oder vollautomatisch arbeitende Testtreiber generiert werden, beträgt in anderen Ansätzen der Aufwand durch die konventionelle, d. h. manuelle, Erstellung und Durchführung von Testfällen ein Vielfaches.

Auch ohne eine Automatisierung ist allein die frühe Angabe von Testfällen und die Existenz einer benutzernahen Modellierung mit entsprechendem Mengengerüst für die Testfallerstellung sehr hilfreich.

Da für den Blackbox-Test die Ablauf-Pfade aus dem Modell abgelesen werden, geht die Testfallentwicklung wesentlich schneller, fehlerärmer und mit höherer Abdeckung vonstatten. Eine Aufwandsersparnis durch schnellere Testfallerstellung und früh entdeckte Fehler ist die Folge.

8.4.3 Vergleich nach Entwicklungsaufgaben

8.4.3.1 Compilerwechsel und Zielplattformwechsel

Compilerwechsel führen bei generatorlosen Ansätzen zu einer Portierung oder Neuimplementierung, bei schlechten Spezifikationen und Entwürfen oft sogar zu einer Neuentwicklung.

Wird die Zielplattform gewechselt, reicht meist eine Portierung aus, die bei unklarer Implementierung und Dokumentation nur durch Re-Engineering, Neuimplementierung oder Neuentwicklung gelöst werden kann.

Sowohl bei Compiler- als auch bei Zielplattformwechsel sind Generatoransätze dann weit überlegen, wenn eine entsprechende native-code-generierende Endstufe für die Plattform bzw. den Ziel-Compiler zur Verfügung steht. Muss diese selbst entwickelt oder durch Veränderung aus einer existierenden Endstufe erzeugt werden, steigt der Aufwand sprunghaft an.

Kann keine entsprechende Endstufe zur Verfügung gestellt werden, muss anhand des vorhandenen Modells neu entwickelt oder die reine Implementierung portiert werden. Die Probleme sind dann – mit Ausnahme der besseren Spezifikation – ähnlich denen generatorloser Ansätze.

Die formale Modellierung und vor allem das Generatorprinzip verringern den Aufwand aber in jedem Fall.

8.4.3.2 Variantenbildung / Sprachen

Oft muss eine einsprachige Software an mehrere Sprachen oder Länder angepasst werden, oder es sind unterschiedliche Varianten für unterschiedliche Zielgruppen oder Märkte notwendig.

Ein generativer Ansatz ist dabei bei nicht-marginalen Änderungen anderen Ansätzen überlegen. Die IML leistet bei Sprachwechseln durch die direkt unterstützte Mehrsprachigkeit und Hilfegenerierung hier mehr als vergleichbare Ansätze ohne direkt unterstützte Mehrsprachigkeit.[54]

Da vom IML-Transformator generierte Oberflächen strukturell und von der Fachkonzeptanbindung her sprachunabhängig weitgehend stabil bleiben, muss das Fachkonzept nicht angepasst werden. Im Idealfall lassen sich so Varianten in anderen Sprachen direkt ohne manuelle Änderungen generieren.

Die Übersetzungen stehen von Anfang an im IML-Modell oder können zu einem beliebigen Zeitpunkt ergänzt werden – dann immer direkt im Kontext des Modells, so dass auch die Satz-Semantik leichter zu erfassen ist und die Übersetzung beschleunigt und auch korrekturärmer wird. Der Transformator überprüft zudem die Texte für jede Sprache auf Vollständigkeit.

Der Aufwand für anderssprachige Oberflächen verringert sich also mit IML bedeutend – vor allem gegenüber herkömmlichen Entwicklungen, aber auch in Bezug auf andere Generatoren.

8.4.3.3 Re-Engineering

Beim Re-Engineering von IML-Projekten oder allgemein modellbasierten, generatorgestützten Projekten ist der Aufwand wesentlich geringer, da nur manuell ergänzte Teile rücktransformiert werden müssen, die Entwicklungsstufen der Oberfläche jedoch zur Verfügung stehen.

Wurde die gesamte Entwicklung nicht modellbasiert durchgeführt oder stehen die Modelle nicht mehr zur Verfügung, ist ein Re-Engineering in Richtung eines IML-Modells wesentlich aufwändiger und deshalb nicht zu empfehlen. Nur bei Releasewechseln oder neuen Oberflächenkonzepten wäre dies sinnvoll.

Alternativen sind dann konventionelles Re-Engineering oder eine Neuentwicklung, die dann IML-gestützt erfolgen kann. Bei einem Einsatz von HERBS ohne bereits existierendes IML-Modell ist eine Neuentwicklung – zumindest der Oberfläche – meist besser.

[54] bei JANUS werden Führungstexte beispielsweise nur einsprachig angegeben.

8.5 Grenzen

EDV-Systeme verarbeiten, womit sie gefüttert werden.
Kommt Mist rein, kommt Mist raus.

– André Kostolany, ungarischer Börsenspekulant (1908 - 1999) –

Neben den vielen Vorteilen eines IML-Modells und der daraus möglichen Generierung finaler Artefakte, bestehen für *HERBS* – speziell für die Implementierung des Transformators – bestimmte Grenzen. Diese sollen im Folgenden aufgezeigt werden.

Modellinhärente Beschränkungen: Mit der Wahl von IML als Modellierungssprache entstehen automatisch Beschränkungen hinsichtlich der generellen Modellierbarkeit. Während in einer rein natürlichsprachlichen Spezifikation jeder Anwendungstyp in ähnlich sinnvoller Weise beschrieben werden kann, ist dies mit IML nicht möglich.

IML wurde für stark interaktive Systeme konzipiert, die zudem Dialogstrukturen aufweisen. Deshalb ist die Spezifikation einer CAD-Software oder einer graphischen Audiobearbeitungssoftware zwar möglich, jedoch können die Konstruktionsbereiche nur abstrakt spezifiziert werden, so dass die entsprechende Funktionalität manuell implementiert werden muss. Eine Erweiterung der IML in dieser Richtung ist schwierig, wäre aber denkbar.

DiLL: Noch stärker als die IML findet in der Dialog Layer Language (DiLL) eine Dialogorientierung statt. Durch die Konzentration auf SCREENs als primäres Strukturierungskriterium eignet es sich kaum für komplexe Anwendungen, die immer auf demselben Bereich im selben Fenster arbeiten, wenn keine Dialoge verwendet werden.

Ein **Generierschritt** ist in seiner „Intelligenz“ immer beschränkt, d. h. im allgemeinen werden Sonderfälle nicht komplett abgedeckt, so dass bestimmte Fälle immer manuell eingegeben werden müssen.

Speziell im Falle des Referenz-IML-Oberflächengenerators liegt nur eine rudimentäre Implementierung vor, die nicht alle Tags berücksichtigt und somit keine Abbildung für den ganzen IML-Definitionsbereich ermöglicht.

Eine Neugenerierung der Hilfe, des Benutzerhandbuchs und der Dialoge ist direkt möglich. Für einen iterativen Einsatz muss allerdings manuell hinzugefügter Code oder Text erhalten bleiben, so dass hierfür Mechanismen entwickelt und implementiert werden müssen.

Die generelle Anbindung an den Code, der für das Fachkonzept erstellt oder ebenfalls generiert wird, ist nur zu einem geringen Prozentsatz erprobt, so dass hier für eine zuverlässige Anbindung weiterer Entwicklungsaufwand für Generator oder generierte Software nötig wird.

Die **Element Transformation Description (ETD)** ist in der Lage, beliebig Textblöcke an spezifizierten Stellen von Templates einzufügen. Die Beschränkung auf Tokens kann durch eine Redefinition auf Zeilen oder Zeichen ersetzt werden.

Dabei besteht jedoch die Beschränkung auf weitgehend lineares Einfügen. Zwar bestehen sogenannte Anker, die auch ein Einfügen in generierte Teile ermöglichen, jedoch ist dieses Konzept begrenzt und erschwert bei höherer Schachtelungstiefe die Modellierung. Hierarchische Konstrukte sind sehr schwer anzubinden.

HERBS ist in seiner Konzeption nur für bestimmte Anwendungstypen geeignet: Technische Anwendungen wie Konfiguratoren und andere weitgehend dialogorientierte Systeme. Betriebliche Anwendungssysteme können bei einer direkten Abbildung der Datenbank auf die Oberfläche einfacher mit datenzentrierten Ansätzen generiert werden. Ist die Ausrichtung eher prozessorientiert, eignet sich *HERBS* besser.

8.6 Einsatzmöglichkeiten

Die interaktions- und ablauforientierte Vorgehensweise von *HERBS* zielt auf weitgehend dialogorientierte Systeme. Nicht vorgangsgesteuerte Anwendungen wie beispielsweise reine Malprogramme eignen sich für die Entwicklung mit *HERBS* kaum.

Kommen jedoch technische, dialogorientierte Systeme wie Konfiguratoren oder strukturierte Definitionswerkzeuge zum Einsatz, kann fast die ganze Benutzungsschnittstellenfunktionalität auf ein IML-Modell abgebildet werden und daraus ein DiLL-Modell und letztlich ein Benutzungsoberflächenprototyp erzeugt werden.

Auch für betriebliche Anwendungssysteme, vor allem für prozessorientierte Anwendungen, eignet sich *HERBS* sehr gut.

Die Modellbildung fällt vor allem in Bereichen leichter, die auch eine Use-Case-Modellierung nahe legen, da hier zunächst die Anwendungsfälle erstellt und diese mit vorhandenen Informationen vorbelegt werden können.

Mit Hilfe dieser Grundlage können Details diskutiert und verändert werden, die alle direkt in das Modell einfließen. Stehen für einen Dialog ausreichende Informationen zur Verfügung, kann der Transformationsprozess durchlaufen und ein Prototyp zur weiteren Diskussion generiert werden.

Besonders gut eignet sich *HERBS* für den Einsatz in Projekten

- mit benutzerorientierter Vorgehensweise
- zur Entwicklung interaktionslastiger Anwendungen
- mit Oberflächenprototyping
- zur Neuentwicklung eines existierenden Systems, das portiert und/oder ergonomischer gestaltet werden soll
- mit getrennter Oberflächenentwicklung
- mit dem Fokus auf einfache Wartbarkeit.

9 Zusammenfassung und Ausblick

Jedes Instrument trägt den Geist seines Entwicklers in sich.
– Werner Karl Heisenberg –

Ziel dieser Arbeit war es, ein software-gestütztes Verfahren zu konzipieren und zu erproben, das es ermöglichen soll, software-ergonomische Methoden schon in den ersten Phasen einer Applikationsentwicklung einzusetzen.

Mit *HERBS*[55] steht nun ein Konzept zur Verfügung, das eine software-ergonomische Vorgehensweise, ausgehend von den frühen Phasen der Anwendungsentwicklung, unterstützt, zugleich aber auch eine Bearbeitung und Nutzung der Ergebnisse über alle Entwicklungsphasen ermöglicht.

Möglich wird dies durch ein umfangreiches Metamodell, das eine an Interaktionen und Abläufen orientierte Modellierung bietet. Die hierzu entworfene Interaction Modelling Language (IML) ist XML-basiert und liegt in Form einer DTD vor.

Diese formalisierte Darstellung ermöglicht eine Umsetzung von IML-Modellen mit Hilfe eines Transformators in eine Anwendungsoberfläche. Hierbei kommt eine pipelineartige, mehrstufige Struktur zum Einsatz.

Zunächst werden Ergänzungen und Überprüfungen am IML-Modell vorgenommen. Nachfolgend wird eine weitgehend zielsystemunabhängige Dialogrepräsentation erstellt, die mit Hilfe der ebenfalls in dieser Arbeit entwickelten Dialog Layer Language (DiLL) beschrieben wird.

Diese Repräsentation kann in einem weiteren Schritt optimiert und in native Code bzw. Ressourcenbeschreibungen umgesetzt werden. Dabei finden Elementbeschreibungen (ETD) Verwendung, die für jedes erzeugte Oberflächenelement die vorgesehenen Code-Fragmente zur Verfügung stellen und mit diesen die Anwendungs-Templates befüllen.

Auf diese Weise entsteht eine compilier- und ablauffähige Anwendung. Diese enthält bereits die Oberflächenfunktionalität, benötigt aber noch eine Implementierung des Fachkonzepts.

Neben der so entstandenen Oberfläche können weitere finale Artefakte wie Testfälle, Hilfen, Checklisten etc. generiert werden, die informationstechnisch schon im IML-Modell repräsentiert sind bzw. aus diesem abgeleitet werden können.

HERBS ist bedingt durch die XML-Basiertheit und die weitgehend generische Implementierung des Transformators sehr offen im Hinblick auf mögliche Erweiterungen und Anpassungen gestaltet, so dass viele Möglichkeiten zur Weiterentwicklung des Ansatzes über diese Arbeit hinaus bestehen und auch notwendig sind.

Benutzertypisierung: Wünschenswert wäre eine abhängig vom Aktor unterschiedliche Beschreibung, so dass Menschen mit körperlichen Einschränkungen wie Sehbehinderungen in der Beschreibung berücksichtigt werden könnten. Bisher werden nur Transformationen für Benutzer ohne Typisierung ermöglicht.

Über Konfigurationen können nur andere Abläufe, jedoch nicht andere Interaktionskonzepte generiert werden. Die IML ist aber ohne weiteres in dieser Richtung erweiterbar. Neue Interaktionskonzepte könnten dann mit entsprechenden Transformatoren und ETDs verarbeitet werden.

Editor: Die Verwendung von XML ermöglicht zwar die Nutzung entsprechender Standard-Anwendungen als Editoren, für die sehr komplexen und vielstufigen IML-Modelle wäre jedoch die Entwicklung eines passenden Editors wichtig, der Hilfen, Wizards und eine Lernunterstützung bereitstellen könnte. Speziell InteractionCases und Relationen wären mit geeigneten Editoren besser zu modellieren. Zudem könnten bei Referenzen (ID / IDREF) immer die passenden bzw. möglichen Werte eingeblendet werden.

[55] Human-centered Ergonomic Requirements specification and Building of Software

Prozessunterstützung: Methoden und Metamodelle können Prozesse nur unterstützen und ergänzen. Aus diesem Grund eignet sich die IML vor allem für den Einsatz mit benutzerorientierten Entwicklungsprozessen, beispielsweise in der Art des Human-Centered Design Process.[56] Bestehende Entwicklungsprozesse könnten daher an die generative Vorgehensweise und frühe Modellierung angepasst, und die Benutzerpartizipation in existierenden Ansätzen verstärkt, sowie auf *HERBS* fokussiert, werden.

Generierprozess: Eine Einbindung oder Ergänzung moderner CASE-Werkzeuge würde eine bessere Integration in den Entwicklungsprozess – und speziell die Code-Generierung – ermöglichen und die Softwareentwicklung der vollständig generierten Implementierung ein gutes Stück näher bringen.

GUI-In-The-Loop: Wenn auch ein Re-Engineering vom Prototyp aus bisher noch nicht vorgesehen ist, wäre es denkbar, den Oberflächenprototyp mit dem Modell in einer Art „Live-Generierung" zu verbinden und durch entsprechenden Zusatzcode auch eine Rückkopplung zu ermöglichen.[57]

Änderungen am Modell wären dann direkt im Prototyp sichtbar. Änderungen am Prototyp würden sich direkt auf das Modell auswirken. Grafische und formale Spezifikation würden so interaktiv Hand in Hand gehen, da sowohl am Modell als auch am Prototyp Änderungen vorgenommen werden könnten, die dann in beiden gleichzeitig sichtbar wären.

Bei einer Ergänzung mit CASE-Tools für die funktionalen Bestandteile der Spezifikation und den Entwurf könnte Softwareentwicklung interaktiv betrieben werden, wie dies für die Benutzungsschnittstelle schon mit „Programmieren durch Zeigen" ansatzweise möglich ist.

Die Vision von *HERBS* ist die benutzerorientierte Entwicklung und Generierung von Oberflächen und ganzen Anwendungen mit Hilfe eines IML-Modells, das alle entwicklungsprojektrelevanten Informationen enthält und möglichst große Teile der Software generiert: Kontextsensitive Hilfe, UI-Testfälle, GUI, Dokumentation für Benutzer und Entwickler, Checklisten etc.

Die IML könnte dabei als Grundlage und Datenhaltung für alle Entwicklungsschritte dienen und mit Hilfe einer Integration von CASE- und Projektmanagementtools eine möglichst vollständige Abdeckung des ganzen benutzerzentrierten Softwareentwicklungsprozesses bieten.

[56] vergleiche Kapitel 2.1.3
[57] vergleichbar mit Hardware-In-The-Loop bei Steuergeräten