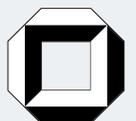
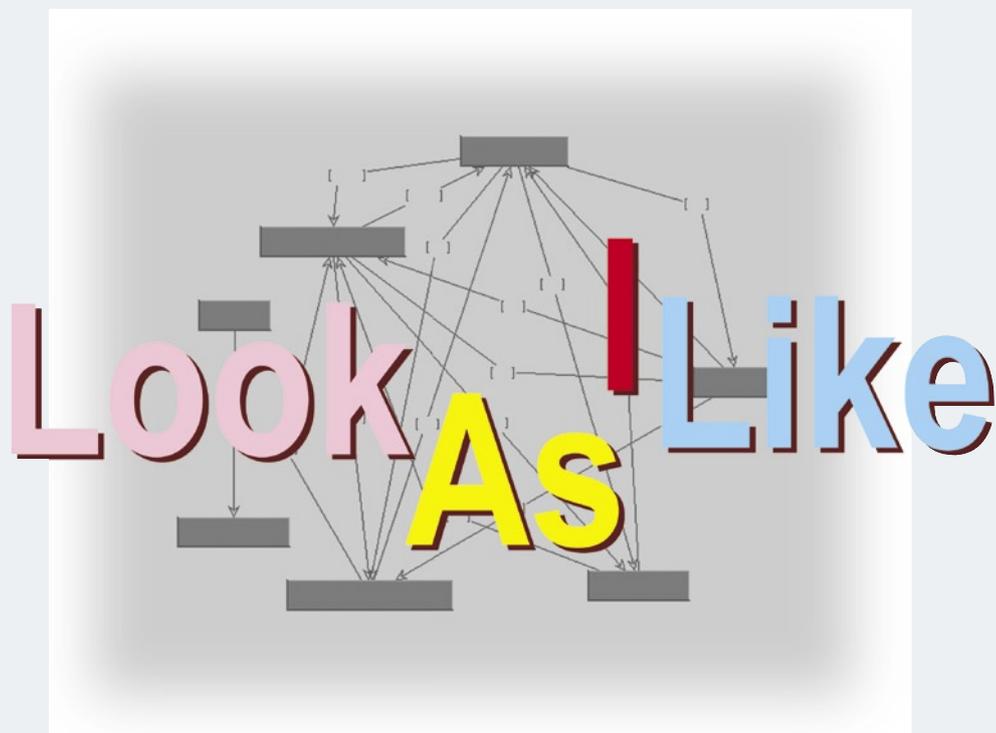


Andreas Judt

Konfigurierbare Benutzerschnittstellen zur Vereinfachung formularbasierter Datenerfassung

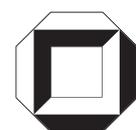


Andreas Judt

**Konfigurierbare Benutzerschnittstellen zur Vereinfachung
formularbasierter Datenerfassung**

Konfigurierbare Benutzerschnittstellen zur Vereinfachung formularbasierter Datenerfassung

von
Andreas Judt



universitätsverlag karlsruhe

Dissertation, Universität Karlsruhe (TH)

Fakultät für Informatik, 2004

Referenten: Prof. Dr. Walter F. Tichy, Prof. Dr. Alfred Schmitt

Impressum

Universitätsverlag Karlsruhe
c/o Universitätsbibliothek
Straße am Forum 2
D-76131 Karlsruhe
www.uvka.de



Dieses Werk ist unter folgender Creative Commons-Lizenz
lizenziert: <http://creativecommons.org/licenses/by-nc-nd/2.0/de/>

Universitätsverlag Karlsruhe 2005
Print on Demand

ISBN 3-937300-70-8

Konfigurierbare Benutzerschnittstellen zur Vereinfachung formularbasierter Datenerfassung

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

der Fakultät für Informatik

der Universität Fridericiana zu Karlsruhe (TH)

genehmigte

Dissertation

von

Andreas Judt

aus Siegen, NRW

Tag der mündlichen Prüfung:

25. Oktober 2004

Erster Gutachter:

Prof. Dr. Walter F. Tichy

Zweiter Gutachter:

Prof. Dr. Alfred Schmitt

Zusammenfassung

Bei der Implementierung formularbasierter Benutzerschnittstellen werden die Bedürfnisse einzelner Benutzer aus verschiedenen Gründen nicht berücksichtigt. Beispielsweise verursachen mehrere Varianten der Benutzerschnittstelle für ein einziges Programm einen hohen Entwicklungs- und Pflegeaufwand.

Selbstanpassende Benutzerschnittstellen sind entweder zu komplex oder verfehlen ihr Ziel aufgrund fehlender Informationen über die Bedürfnisse der Benutzer. Benutzerschnittstellen werden in der Praxis von Programmierern und wenigen ausgewählten Benutzern definiert. Eine so implementierte Benutzerschnittstelle stellt eine universelle Lösung für die von der Entwicklergruppe bedachten Anwendungssituationen dar. Benutzer können die resultierende Benutzerschnittstelle für ihre Anwendungsfälle zwar verwenden, jedoch wären in vielen Fällen Vereinfachungen der Formulare realisierbar und wünschenswert. Benutzer sind daher oft und berechtigt mit der zur Verfügung gestellten Benutzerschnittstelle unzufrieden.

Die Idee dieser Arbeit ist es, den Graben zwischen Benutzer und Entwicklergruppe dadurch zu schließen, daß der Benutzer die Eingabe von Daten in Formulare selbständig für beliebige Anwendungsfälle vereinfachen kann. Die Vereinfachung erfolgt durch Umstrukturierung der Formularfelder bei gleichzeitiger automatischer Anpassung der Berechnung der Auswahlmenge und Konsistenzprüfung der Eingabedaten. Die Benutzerschnittstelle unterstützt dabei durch Vorschläge zur Änderung der Formulare.

Das Ziel wurde durch eine erweiterte Strukturdefinition komponentenbasierter grafischer Benutzerschnittstellen erreicht, die eine unterliegende objektorientierte Programmiersprache und die Serialisierbarkeit der Interaktionselemente voraussetzt. Bestehende Implementierungen grafischer Interaktionselemente werden spezialisiert, um Abhängigkeiten zu definieren und Berechnungen der Auswahlmenge bzw. Konsistenzprüfungen mit einer Qualitätsaussage zu versehen. Aus dem Interaktionsverhalten wird ein Benutzermodell erzeugt, das gemeinsam mit der Struktur und Elementabhängigkeiten eine Analyse der Benutzerschnittstelle ermöglicht.

Der Hauptinhalt der Arbeit besteht in der Entwicklung von Verfahren zur Analyse und Beobachtung der Benutzerinteraktion, mit denen aus der Struktur und den Abhängigkeiten der Formularfelder Möglichkeiten zur Vereinfachung identifiziert werden. Hierfür werden komponentenbasierte Bibliotheken grafischer Benutzerschnittstellen erweitert. Die Machbarkeit des Konzepts wird durch eine Prototypenimplementierung mit Java und Swing nachgewiesen und die Verwendbarkeit alternativer Technologien dargestellt. Anhand von Fallbeispielen werden Vereinfachungen von Benutzerschnitt-

stellen durch Änderung der Interaktionsfolgen gezeigt.

Das Ergebnis der Arbeit ist ein Verfahren zur Erweiterung bestehender komponentenbasierter Technologien, die die Unterstützung des Benutzers bei der Vereinfachung der Dateneingabe für beliebige Anwendungssituationen erlaubt.

Danksagung

Um eine Dissertation erfolgreich abzuschließen, benötigt man die Unterstützung seines gesamten Umfelds. Die wichtigste ist dabei die nichtfachliche: die Unterstützung der Familie. Ich danke meiner Frau Alexandra für die unendliche Geduld, mit der sie diese Arbeit begleitet und mir Freiräume geschaffen hat. Meine Tochter Antonia hat dabei mit ihrer Frage „Papa, wieviele Seiten hast Du schon fertig?“ regelmäßig nach ihrer Gutenachtgeschichte für die richtigen Motivationsschübe gesorgt und damit ihren Beitrag zu dieser Arbeit geleistet.

Für die fachliche Unterstützung bedanke ich mich herzlich bei meinem Referenten Walter F. Tichy. Seine Anregungen und Tips bei der Problemlösung haben mir dabei sehr geholfen. Besonders dankbar bin ich für die skeptische Hinterfragung des entwickelten Verfahrens, was wesentlich zur Evolution dieser Arbeit beigetragen hat. Ebenso möchte ich meinem Korreferenten Alfred Schmitt danken, der mit seiner konstruktiven Kritik einen wichtigen Beitrag zur Qualität dieser Arbeit geleistet hat. Ein weiterer Dank geht an Michael Philippsen, der als Abteilungsleiter Softwaretechnik am Forschungszentrum Informatik an der Universität Karlsruhe meine ersten Ideen kritisch betrachtet hat und mir immer ein wertvoller Diskussionspartner war. Ebenso danke ich Friedbert Frass, Hans-Peter Meier und Oliver Meier für ihre Unterstützung bei der Evaluierung des Verfahrens.

„Eine Dissertation muß man komponieren.“ Mein „großer Bruder“ Wolfgang Bott hat nicht nur mit diesen Worten sondern insbesondere mit seiner unermüdlichen Energie bei der Korrektur zur Gestaltung dieses Buches beigetragen. Schön, wenn man solche Kollegen hat.

Besonders möchte ich mich bei Katja Abel bedanken, mit der ich meine Erfahrungen bei der Entwicklung dieser Arbeit austauschen konnte. Von ihr habe ich gelernt, daß sich Promotionen fachübergreifend erstaunlich ähneln.

Ein herzliches Dankeschön gebührt ebenso Elke Mainz, Kerstin Wimmer und Katrin Schmitt für die Fehlerkorrektur von Text und Layout dieses Buches. Besonders habe ich mich darüber gefreut, daß sie sich über alle meine Erwartungen hinweg engagiert haben. Abschließend gilt mein Dank allen, die mich bei dieser Arbeit unterstützt haben und hier nicht explizit genannt wurden.

Inhaltsverzeichnis

1	Einleitung	9
1.1	Motivation	9
1.2	Aufgabenstellung	10
1.3	Kategorisierung der Anwendergruppe	10
1.4	Situation in der Praxis	11
1.5	Beiträge dieser Arbeit	11
1.5.1	Verfahren zur Analyse der Benutzerinteraktion	12
1.5.2	Verfahren zur Unterstützung des Anwenders bei der Vereinfachung von Dateneingaben	13
1.5.3	Konzept zur Programmierung anpaßbarer Komponentenstrukturen	13
1.6	Gliederung	14
2	Stand der Technik: Anpassung der Schnittstelle an den Benutzer	15
2.1	Konzepte für den Entwurf benutzerorientierter Schnittstellen	15
2.1.1	Konsistenz von Benutzerschnittstellen	16
2.1.2	Beurteilung der Komplexität von Benutzerschnittstellen	16
2.1.3	Benutzermodellierung	16
2.1.4	Benutzerzentrierter Entwurf	18
2.1.4.1	Benutzerbeteiligung im Entwurf	18
2.1.4.2	Benutzerzentrierte Evaluierung	19
2.1.5	Automatisch anpassende Benutzerschnittstellen	21
2.1.6	Endbenutzer Programmierung	22
2.2	Plattformneutrale Beschreibungen	22
2.2.1	UIML	23
2.2.2	XIML	23
2.2.3	XForms	23
2.2.4	XAML	23
2.3	Verwandte Arbeiten	23
2.3.1	Dynamic Forms	24
2.3.2	Adaptive Forms	24
2.4	Zusammenfassung	25
3	Analyse der Benutzerinteraktion	27
3.1	Ereignisbasierte Evaluierung von Benutzerschnittstellen	27
3.2	Techniken zur Analyse von Ereignisströmen	30
3.2.1	Synchronisation und Suche	31
3.2.2	Transformation von Ereignisströmen	31

3.2.3	Quantitative Erfassung von Ereignissen	31
3.2.4	Sequenzerkennung und Sequenzvergleich	32
3.2.5	Charakterisierung von Sequenzen	32
3.2.6	Visualisierung	32
3.2.7	Unterstützung integrierter Evaluierung	32
3.3	Lösungsansatz zur ereignisbasierten Modellierung des Benutzers . . .	33
3.3.1	Auswahl relevanter Ereignisse	34
3.3.2	Berechnung der bevorzugten Eingabefolge	35
3.4	Zusammenfassung	35
4	Unterstützung des Benutzers bei der Vereinfachung von Dateneingaben	37
4.1	Anforderungen an die Vereinfachung formularbasierter Datenerfassung	37
4.2	Formulierung von Abhängigkeiten zwischen Dateneingaben	39
4.3	Ein Beispiel für formularbasierte Datenerfassung: Adresseingabe . . .	42
4.4	Strukturanalyse der Benutzerschnittstelle	45
4.5	Zusammenfassung	48
5	Benutzergesteuerte Modifikation von Komponentenstrukturen	51
5.1	Ansatz zur technologieutralen Modellierung	52
5.1.1	Modellierung der Komponentenstruktur	52
5.1.2	Die Modellierung des Abhängigkeitsgraphen	55
5.1.3	Die Modellierung des Kontrollflußgraphen	57
5.2	Vorgehensweise bei der Programmierung	58
5.2.1	Festlegen der Komponenten	59
5.2.2	Einfügen der Gestalten	60
5.2.3	Verknüpfung der Komponenten	60
5.2.4	Einfügen von Abhängigkeits-Beziehungen	61
5.2.5	Registrieren externer Hörer	61
5.2.6	Speichern der Benutzerschnittstelle	61
5.3	Bedienung der konfigurierbaren Benutzerschnittstelle	61
5.4	Zusammenfassung	62
6	Evaluierung	63
6.1	Vorgehensweise bei der Evaluierung von Dateneingaben	64
6.2	Vervollständigung von Adressdaten	65
6.2.1	Initiale Eingabe des Postleitzahl	65
6.2.2	Initiale Eingabe des Wohnorts	68
6.2.3	Ergebnis	69
6.3	Austausch von CAD/CAM Konstruktionsdaten	69
6.3.1	Austausch mit einem Konstruktionsbüro	73
6.3.2	Kürzel definiert den Kontakt	75
6.3.3	Geänderte Kontaktdaten	76
6.3.4	Projektbezogene Konfiguration	77
6.3.5	Ergebnis	77
6.4	Das Verwaltungssystem IHKdezent2	78
6.4.1	Spezialimplementierungen	78
6.4.2	Machbarkeit und Aufwand	79
6.4.3	Ergebnis	79
6.5	Zusammenfassung	81

INHALTSVERZEICHNIS

7 Zusammenfassung und Ausblick	85
7.1 Technische Weiterentwicklung	85
7.1.1 Integration erweiterter Komponenten in Entwicklungsplattformen	86
7.1.2 Begrenzung des Rechenaufwandes	87
7.2 Ausblick	88
7.2.1 Nutzung kognitiver Fähigkeiten des Anwenders	88
7.2.2 Post-WIMP: flexible Spracherkennung	89
7.2.3 Aspekte der Verwendbarkeit	89
A Details der Implementierung	91
A.1 Auswahlmenge und Konsistenzprüfung	91
A.2 Die Erzeugung des Klassenmodells	94
A.3 Das Klassenmodell der konfigurierbaren Benutzerschnittstelle	94
A.4 Konfigurierbare Elemente mit Swing	96
A.4.1 Die Klasse ExtLelement	98
A.4.2 Die Klasse ExtStyle	100
A.4.3 Die Klasse ExtCalculator	100
A.5 Die XML Serialisierung von Komponenten	100
A.6 Ein Programmbeispiel	103
A.7 Der Zugriff auf das Datenmodell	110
A.8 Die Strukturbeschreibung mit erweiterten Komponenten	111
A.9 Ein Beispiel der Anwenderunterstützung	115
A.10 Probleme mit Java Swing	119
A.11 Zusammenfassung	120
B Glossar	121

Kapitel 1

Einleitung

1.1 Motivation

Benutzerschnittstellen bilden das Bindeglied zwischen Mensch und Maschine. Die effiziente Bedienung von Computern erlaubt die einfache Steuerung komplexer Prozesse. Softwareergonomie spielt daher bei der Programmentwicklung eine wesentliche Rolle. Hierzu gehört auch die Entwicklung leicht bedienbarer Benutzerschnittstellen. Beim Entwurf komplexer Dialoge muß insbesondere die starke Unterschiedlichkeit der Anwender beachtet werden. Diese müssen mindestens in Anfänger und Experte unterschieden werden. Während Anfänger leicht verständliche Dialoge vorziehen, wünschen sich Experten ein knappes und effizient gestaltetes Erscheinungsbild.

Benutzerschnittstellen zur Datenerfassung orientieren sich üblicherweise an der Vorgehensweise beim Ausfüllen von Formularen. Anwender sehen die Anzeige korrespondierender Felder, bewegen den Mauszeiger auf deren Position und geben erfragte Daten dort ein. Bei dieser Form der Interaktion müssen Benutzer die Beschreibung der Felder, gültige Eingabewerte und die Eingabemethode kennen. Darüber hinaus müssen sie in der Lage sein, auf Fehlermeldungen zu reagieren. Die Formularform eignet sich für erfahrene Benutzer bei regelmäßiger Verwendung. Betrachtet man die Menge der verschiedenen Anwendungssituationen, so lassen sich bei Abhängigkeiten zwischen Feldern bereits in der Entwurfsphase Vereinfachungsmöglichkeiten identifizieren, mit denen eine Effizienzsteigerung durch Vermeidung redundanter Eingaben erzielt wird. Hierbei können allerdings nur allgemeingültige Aussagen umgesetzt werden.

Anwendungssituationen mit nicht allgemeingültigen Vereinfachungsmöglichkeiten müßten dann als Variante der Benutzerschnittstelle zur Verfügung gestellt werden. Diese Fälle basieren häufig auf dem Prozeßwissen des Anwenders und sind zeitlich variant. Wünsche zur Anpassung der Benutzerschnittstelle zur Vereinfachung von Dateneingaben können somit zu beliebigen Zeitpunkten bei der Verwendung des Programms auftreten. Die Programmierung und Verwaltung von Varianten wird häufig aufgrund strenger Budgetierung der Programmentwicklung nicht realisiert. Eine Anpassung sollte also vom Anwender selbst vorgenommen werden. Die vom Programmierer implementierten Berechnungen von Auswahlmengen der Felder könnten mit dem Prozeßwissen des Anwenders für die Vereinfachung der Dateneingabe der Benutzerschnittstelle genutzt werden.

1.2 Aufgabenstellung

In dieser Arbeit wird das Konzept der *konfigurierbaren Benutzerschnittstelle* entwickelt. Es erlaubt die Vereinfachung von formularbasierter Datenerfassung für spezifische Anwendungssituationen. Diese Modifikation wird vom Benutzer selbständig und ohne Programmieraufwand vorgenommen. Anwender können somit durch ihr Prozeßwissen eine Effizienzsteigerung allgemeingültiger, ergonomischer Benutzerschnittstellen erzielen, die durch Programmierung in der Praxis nicht umsetzbar ist. Diese Arbeit leistet einen Beitrag zur Entwicklung effizienter, verwendbarer Benutzerschnittstellen, die heute eine große wissenschaftliche Herausforderung darstellt. Erfolge in der Effizienzsteigerung sind jedoch allgemein nicht leicht quantifizierbar. Der Nachweis der Vereinfachung wird in dieser Arbeit durch die Messung benötigter Interaktionen erreicht.

Die Herausforderung bei der Umsetzung des Konzepts besteht darin, Benutzer in die Lage zu versetzen, ihr Prozeßwissen bei der Vereinfachung der Datenerfassung einzubringen. Hierbei darf der erzielte Vorteil auf Anwenderseite nicht durch signifikanten Mehraufwand bei der Entwicklung verloren gehen.

Die Aufgabenstellung wird mit folgenden Thesen formuliert:

These 1 Anwender können Dateneingaben mit ihrem Prozeßwissen selbständig und ohne Programmieraufwand vereinfachen. Damit kann eine Effizienzsteigerung der Benutzerschnittstelle erzielt werden.

These 2 Durch Beobachtung der Benutzerinteraktion kann der Anwender bei der Vereinfachung von Dateneingaben unterstützt werden.

These 3 Komponenten formularbasierter Benutzerschnittstellen können mit einem einheitlichen Konzept so erweitert werden, daß Dialogstrukturen vom Anwender modifiziert werden können. Die Konsistenz der Formularfelder bezüglich ihrer Dateneingaben bleibt dabei erhalten.

Die Verwendung von Komponenten in einer objektorientierten Programmiersprache sichert die Anwendbarkeit des im folgenden beschriebenen Klassenmodells. Konfigurierbare Benutzerschnittstellen können möglicherweise auch ohne diese Einschränkung modelliert werden. Es ist jedoch anzunehmen, daß dabei stark unterschiedliche Modelle entstehen. Die Serialisierbarkeit einer Komponente sichert ihre Analysierbarkeit und Wiederherstellbarkeit zu beliebigen Zeitpunkten des Programmablaufs.

1.3 Kategorisierung der Anwendergruppe

Der Entwurf einer Benutzerschnittstelle sollte mit dem Verstehen der angestrebten Benutzergruppe beginnen. Hierzu gehören auch gesellschaftliche Informationen wie Alter, Geschlecht, körperliche Fähigkeiten, Bildung, kultureller und ethnischer Hintergrund, Schulung, Motivation, Ziele und Persönlichkeit. Benutzerkategorien werden aufgrund der vielfältigen Daten, die für eine Profilbildung erhoben werden müssen, stark abstrahiert.

Shneiderman [50] schlägt eine generische Unterteilung der Benutzer in drei Kategorien vor:

1.4 Situation in der Praxis

- Anfänger oder Erstbenutzer
- Erfahrene, regelmäßige Anwender
- Experten mit häufiger Benutzung (*engl. „power user“*)

Die Vereinfachung von Dateneingaben setzt für den Benutzer das Verständnis des unterliegenden Prozesses und insbesondere der eigenen Anwendungssituation voraus. Anfänger und Erstbenutzer eignen sich dafür nicht. Die Zielgruppe gehört somit den beiden letzten Kategorien an: erfahrene, regelmäßige Anwender und Experten mit häufiger Benutzung.

1.4 Situation in der Praxis

Beim Entwurf von Benutzerschnittstellen existieren verschiedene Strategien zur Erreichung der Verwendbarkeit und Nützlichkeit für den Anwender. Anpassende Systeme ermöglichen die Vereinfachung von Dateneingaben, jedoch nach dem Prozeßverständnis der Entwickler [12, 5]. Änderungen der Benutzerschnittstelle – automatisch oder interaktiv – sind vom Benutzer oft nicht nachvollziehbar und führen dann zum Verlust der Akzeptanz.

Die Adaption verschiedener Anwendungssituationen kann durch Endbenutzer-Programmierung [13] erreicht werden. Diese erfordert neben Programmierkenntnissen des Benutzers einen hohen Entwicklungsaufwand. Darüberhinaus kann die Korrektheit und Konsistenz der Benutzer-Lösung nicht ohne zusätzliche Mechanismen gewährleistet werden. Eisenberg [13] stellt fest, daß Systeme, die keine Programmierung durch den Anwender erlauben, kaum Möglichkeiten der Anpassung an Benutzerbedürfnisse bieten. Insbesondere weist er auf die Notwendigkeit der Modifikation durch den Benutzer hin, um die Effizienz eines Systems zu steigern. Die Vereinfachung der Eingabe abhängiger Daten ließe sich mit Endbenutzer-Programmierung lösen, jedoch können Fähigkeiten zur Programmierung bei Anwendern nur in den seltensten Fällen vorausgesetzt werden.

Benutzerbeteiligung im Entwurf [37] findet heute als einfache benutzerzentrierte Technik Anwendung in der Programmentwicklung. Auf diese Weise implementierte Benutzerschnittstellen verkörpern einen Kompromiß vieler Anwendungsszenarien. Benutzerzentrierte Evaluierung [26] integriert Anwender während des Entwurfs und der Implementierung. Verbesserungen der Bedienbarkeit und Nützlichkeit finden in mehreren Iterationen statt, bieten aber ebenfalls keinen Raum für die Vereinfachung von Dateneingaben für spezifische Anwendungssituationen. Die Interpretation von Daten aus der benutzerzentrierten Evaluierung ist Gegenstand der aktuellen Forschung. Gewonnene Ergebnisse können heute noch nicht mit allgemeingültigen Methoden zur Effizienzsteigerung des Programmentwurfs genutzt werden.

1.5 Beiträge dieser Arbeit

Zur Umsetzung des Konzepts der konfigurierbaren Benutzerschnittstellen liefert diese Arbeit folgende Beiträge:

Beitrag 1 ein Verfahren zur Analyse der Benutzerinteraktion

Beitrag 2 ein Verfahren zur Unterstützung des Anwenders bei der Vereinfachung von Dateneingaben

Beitrag 3 ein Konzept zur Programmierung anpaßbarer Komponentenstrukturen

Die Beiträge korrespondieren dabei mit den Thesen, die in der Aufgabenstellung formuliert wurden.

1.5.1 Verfahren zur Analyse der Benutzerinteraktion

Aus dem Interaktionsverhalten des Anwenders werden Rückschlüsse auf wichtige Dateneingaben gezogen. Für die Erfassung wird ein Benutzermodell generiert. Die Analyse basiert auf der Sammlung verschiedenster Ereignisse, die bei der Bedienung der Benutzerschnittstelle auftreten. Abbildung 1.1 veranschaulicht die Analyse der Benutzerinteraktion.

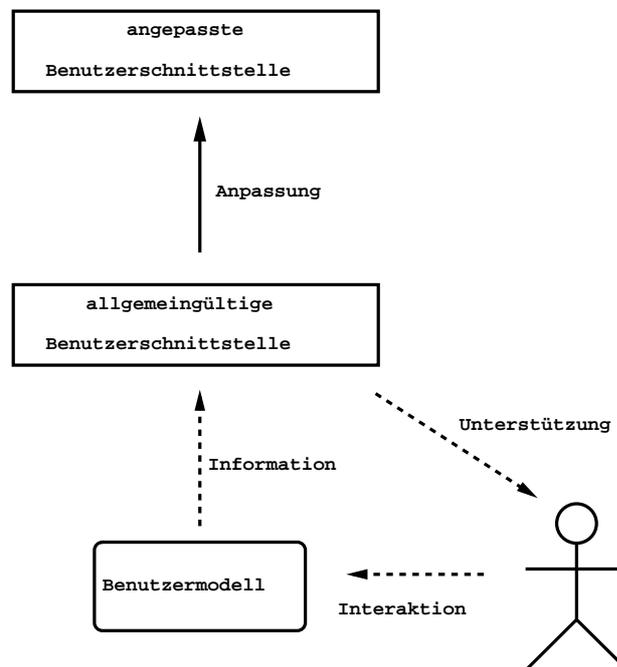


Abbildung 1.1: Konzept der Vereinfachung formularbasierter Datenerfassung

1.5.2 Verfahren zur Unterstützung des Anwenders bei der Vereinfachung von Dateneingaben

Anwender sollen von der konfigurierbaren Benutzerschnittstelle bei der Vereinfachung der Dateneingaben unterstützt werden. Hierfür wird ein Benutzermodell entwickelt, das die bevorzugte Interaktionsfolge der Dateneingabe berechnet. Basierend auf dieser Folge, der Strukturdefinition und den Abhängigkeiten zwischen den Dateneingaben wird ein Verfahren zur Beurteilung alternativer Interaktionsfolgen entwickelt. Basierend auf der Menge bestimmlicher Folgen werden Strukturänderungen vorgeschlagen. Die Konfiguration der Benutzerschnittstelle kann beispielsweise durch ein Gestaltungssystem oder durch Manipulation des Anwenders erfolgen. Abbildung 1.1 veranschaulicht die Unterstützung des Benutzers.

1.5.3 Konzept zur Programmierung anpaßbarer Komponentenstrukturen

Im Verlauf dieser Arbeit wird argumentiert, daß das Konzept zur Vereinfachung von Dateneingaben für beliebige komponentenbasierte Programmbibliotheken anwendbar ist. Betrachtet man Komponenten als Bausteine, die von außen vollständig parametrierbar sind, dann kann der Zustand einer beliebigen Komponente gespeichert und zu einem beliebigen Zeitpunkt wiederhergestellt werden. Für die Vereinfachung von

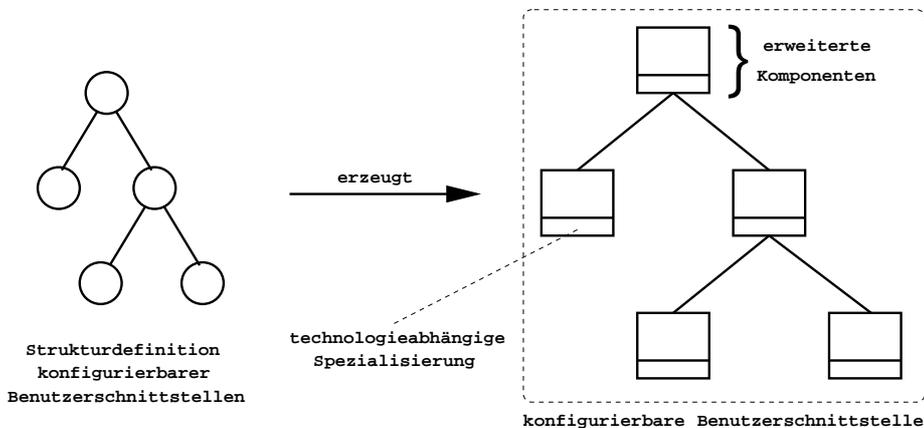


Abbildung 1.2: Strukturdefinition konfigurierbarer Benutzerschnittstellen

Dateneingaben müssen Formularfelder, die von einander abhängende Daten speichern, auf Eingaben mit der Berechnung von Auswahlmengen reagieren. Hierfür wird die Komponentenstruktur der Programmbibliothek durch Bildung von Unterklassen um die nachfolgend beschriebene Funktionalität erweitert. Diese Spezialisierung ist teilweise technologieabhängig, z.B. bei der Zuweisung von Daten an Formularfelder. Abbildung 1.2 skizziert die Erzeugung einer konkreten Benutzerschnittstelle aus einer technologieutralen Strukturdefinition.

1.6 Gliederung

Kapitel 2 beschreibt den Stand der Forschung der Mensch-Maschine Interaktion und ihre Anwendung in der Praxis. Im Verlauf des Kapitels werden benutzerzentrierte Techniken in Entwurf und Evaluierung vorgestellt. Anschließend werden Konzepte zur Vereinfachung von Dateneingaben dargestellt und mit dieser Arbeit verglichen. In Kapitel 3 wird ein Verfahren zur Modellierung des Benutzers dem Interaktionsverhalten entwickelt. Kapitel 4 beschreibt ein Verfahren zur Unterstützung des Anwenders bei der Vereinfachung von Dateneingaben. Kapitel 5 stellt ein Konzept zur Programmierung anpaßbarer Komponentenstrukturen vor. Anwender können eine Benutzerschnittstelle damit selbständig modifizieren. Kapitel 6 evaluiert die Machbarkeit des Konzepts mit Hilfe von Anwendungssituationen, in denen Vereinfachung von Dateneingaben durch Modifikation der Benutzerschnittstelle erzielbar sind. Diese werden anhand verschiedener prototypischer Implementierungen dargestellt und erläutert. Das abschließende Kapitel 7 faßt die Ergebnisse dieser Arbeit zusammen und formuliert technische Schwierigkeiten bei Implementierung und Evaluierung. Abschließend werden Anwendungen des Elementkonzepts jenseits grafischer Benutzerschnittstellen skizziert. Anhang A beschreibt die Implementierung des Prototyps Look_{AS}^ILike.

Kapitel 2

Stand der Technik: Anpassung der Schnittstelle an den Benutzer

Die Forschung der Mensch-Maschine Interaktion (*Human-Computer Interaction, HCI*) beschäftigt sich neben technischen Aspekten der Maschine auch mit psychosozialen Fähigkeiten des Menschen. Ein hervorragendes technisches Konzept genügt nicht, um die Zufriedenheit des Benutzers zu erreichen. Vielmehr müssen Benutzerschnittstellen für die Fähigkeiten, das Umfeld und die Ziele des Anwenders optimiert werden.

Abschnitt 2.1 stellt die Konzepte für den Entwurf benutzerorientierter Schnittstellen vor. Abschnitt 2.2 beschreibt Techniken zur Erzeugung von Benutzerschnittstellen mit deklarativen Ansätzen. Sie dienen zur Speicherung der modifizierten Benutzerschnittstelle des Anwenders. Die hier vorgestellten Techniken eignen sich neben komponentenbasierten Programmbibliotheken für die Umsetzung des Konzepts dieser Arbeit. Abschnitt 2.3 stellt dieser Arbeit vergleichbare Verfahren gegenüber.

2.1 Konzepte für den Entwurf benutzerorientierter Schnittstellen

Verwendbarkeit (*engl. usability*) und Nützlichkeit (*engl. usefulness*) von Benutzerschnittstellen verkörpern die Kernziele der Mensch-Maschine Interaktion und sind nur schwer voneinander zu trennen. Die Herausforderung beim Entwurf einer Benutzerschnittstelle besteht nicht in der Etablierung neuer Technologien, sondern in der Nützlichkeit für seine Anwender. In einem verwendbaren System dient die Mensch-Maschine Interaktion dem Menschen und nicht der Maschine [40]. Diese größtenteils unscharf formulierten Aussagen umfassen ein breites Forschungsgebiet, das in den letzten drei Dekaden verschiedenste Ansätze zur Unterstützung von Benutzern bei der Erreichung ihrer Ziele hervorgebracht hat. Dieser Abschnitt beschreibt die wesentlichen Forschungsrichtungen und ordnet konfigurierbare Benutzerschnittstellen darin ein.

2.1.1 Konsistenz von Benutzerschnittstellen

Nielsen [41] bewertet die Konsistenz einer Benutzerschnittstelle als einen der wesentlichen Faktoren der Verwendbarkeit. Grudin [18] benutzt den Begriff der Konsistenz einer Benutzerschnittstelle in gegenseitig überlappenden Aspekten des Entwurfs:

Interne Konsistenz beschreibt die Konsistenz der Benutzerschnittstelle zu sich selbst. Der Konsistenzbegriff bezieht sich hier auf physikalische und grafische Struktur, Namensgebung, Befehlsumfang, Interaktionsmöglichkeiten und Dialogformen.

Externe Konsistenz beschreibt die Konsistenz einer intern konsistenten Benutzerschnittstelle zu weiteren Schnittstellen im Arbeitsumfeld des Benutzers. Externe Inkonsistenz kann beispielsweise durch gleiche Benennung von Objekten in verschiedenen Benutzerschnittstellen erfolgen, die unterschiedliche Funktionalitäten besitzen. Externe Konsistenz wird häufig auf Kosten der internen Konsistenz hergestellt.

Korrespondenz zur realen Welt beschreibt die Konsistenz von Objekten einer Benutzerschnittstelle zu Objekten der realen Welt. Konsistenz zur realen Welt bedeutet, daß Symbole oder Bezeichnungen in der Benutzerschnittstelle und ihre Gegenstücke der realen Welt die gleiche Bedeutung besitzen.

Der Begriff *Konsistenz* wird in den Arbeiten von Nielsen, Grudin, Kellog et.al. [41] für die Struktur einer Benutzerschnittstelle verwendet. Der Konsistenzbegriff in dieser Arbeit bezieht sich auf die Konsistenz der Eingabedaten. Die konsistente Gestaltung bildet einen Stützpfiler bei der Unterstützung des Anwenders. Die Forschungsergebnisse von Nielsen et.al. [41] können in das Konzept dieser Arbeit bei der Implementierung mit einer konkreten Technologie integriert werden.

2.1.2 Beurteilung der Komplexität von Benutzerschnittstellen

Die Messung der Komplexität einer Benutzerschnittstelle beschäftigt die Forschung seit mehr als zwei Dekaden [50, 57]. Verschiedene Metriken und Systeme wurden entwickelt, um möglichst einfach bedienbare und leicht zu erlernende Benutzerschnittstellen zu entwickeln und ihre Effizienz zu messen. Die meisten Verfahren wurden für konkrete Anwendungen definiert. Allgemeingültige Erkenntnisse, die zur Entwicklung eines Entwurfswerkzeugs führen, konnten bisher nicht konstatiert werden. Tullis [57] stellt insbesondere fest, daß ein solches Werkzeug niemals das Urteilsvermögen des Entwicklers ersetzen könne, jedoch effektive Unterstützung leisten würde.

2.1.3 Benutzermodellierung

Das erwartete Verhalten einer Benutzerschnittstelle resultiert aus dem *mental*en Modell des Benutzers, das sein Verständnis widerspiegelt. Die Erwartungen, die ein System von seinem Benutzer hat, resultieren aus dem *Benutzermodell* [3].

Modelle sind Abstraktionen eines Objekts oder Prozesses, die Aspekte desselben beschreiben. Menschen konstruieren Prozesse üblicherweise in ihrer Vorstellungskraft. Diese sogenannten mentalen Modelle definieren Prozesse als Menge einzelner Schritte. Sie können unvollständig, inkonsistent und fehlerhaft sein. Allen [3] definiert folgende Charakteristiken, um Schlußfolgerungen über das mentale Modell eines Menschen zu ziehen:

2.1 Konzepte für den Entwurf benutzerorientierter Schnittstellen

Voraussagen Benutzer können den nächsten Schritt in einem Prozeß voraussagen und wissen, welche Dateneingaben sich auf welche Teile des Prozesses auswirken.

Erklärung und Diagnose Benutzer können die Ursachen eines Ereignisses und die Diagnose seines Ursprunges erklären.

Training Menschen, die geübt in der Bearbeitung eines Prozesses sind, erfüllen diese Aufgabe besser als Ungeübte. Es wird erwartet, daß Ausbildung den Umgang mit der Anwendung erleichtert.

Andere Rückschlüsse lassen sich ebenfalls aus Reaktionszeiten, aus Augenbewegungen und Rückfragen zum Prozeß ermitteln.

Nachdem mentale Modelle nur im Kopf des Benutzers existieren, wird aus gezogenen Rückschlüssen ein *konzeptuelles Modell* erzeugt. Konzeptuelle Modelle dienen der Analyse mentaler Modelle.

Benutzermodelle besitzen Parameter, um Benutzer voneinander zu unterscheiden. Parameter können explizit vom Anwender angegeben werden oder vom Programm aus den vorangegangenen Eingaben oder dem Benutzerverhalten erzeugt werden:

Explizite Profile Anwender müssen für diese Modellierungstechnik ein Interessenprofil erzeugen. Ein wesentliches Problem hierbei ist, daß Benutzer durchaus keine präzise Vorstellung über ihre Vorlieben besitzen oder keine ehrliche Antwort geben wollen – beispielsweise bei Angabe der Expertise. Die Pflege des Profils bedeutet weiterhin für den Anwender einen Pflegeaufwand, den er üblicherweise nicht unmittelbar mit der Nützlichkeit der Benutzerschnittstelle assoziiert.

Rückschlüsse aus dem Benutzerverhalten Die Sammlung von Informationen über den Benutzer beschränkt sich auf die vom Modell benötigten Daten. Hierbei können vom Modell falsche Schlüsse gezogen werden. Rückschlußbasierte Modelle können jedoch präzisere Informationen über den Benutzer besitzen als explizite Profile.

Ein wichtiger Bestandteil anpassbarer, interaktiver Systeme ist die Fähigkeit, Benutzer zu modellieren. Kass et.al. [27] definieren ein allgemeines Benutzermodell und beschreiben einen Modellierungsansatz, der in vielen Anwendungen verwendbar sein könnte. Dieser Ansatz konzentriert sich auf die Darstellung, Verwaltung und Sammlung von Aspekten, die langzeitliche Aussagen über den Anwender treffen können. Kass konzentriert sich darauf, das Systemverhalten auf Basis des Benutzermodells dynamisch anzupassen. Dafür definiert er Benutzermodelle als *Modelle, die eine wohldefinierte Menge an Fähigkeiten besitzen, die in verschiedenen Situationen und Systemen verwendet werden können*. Benutzermodelle sind für Anwendungen mit mindestens einer der folgenden Eigenschaften vorteilhaft:

1. Die Anwendung versucht, ihr Verhalten an einzelne Benutzer anzupassen.
2. Die Anwendung übernimmt die Verantwortung für eine erfolgreiche Mensch-Maschine Kommunikation.
3. Die Klassen potentieller Benutzer sind sehr unterschiedlich.

In allen Fällen muß das Programm den Benutzer und seine Aktionen verstehen, um sein eigenes Verhalten zu kontrollieren. Die Effektivität eines Benutzermodells hängt unmittelbar vom darunter liegenden Prozeß ab. Ein allgemeingültiges Vorgehen bei der Modellierung des Benutzers läßt sich nicht angeben. Konfigurierbare Benutzerschnittstellen unterstützen Anwender bei der Vereinfachung der Dateneingaben durch Beobachten des Interaktionsverhaltens. Hierfür wird eine Benutzermodellierung angegeben, die bevorzugte Eingabesequenzen des Benutzers approximiert. Die Funktionsweise des Benutzermodells wird in Abschnitt 3.3 und Anhang A.9 beschrieben.

2.1.4 Benutzerzentrierter Entwurf

Karat [25] definiert den benutzerzentrierten Entwurf (*engl. user-centered design*) folgendermaßen: *A user centered design process is one that sets users or data generated by users as the criteria by which a design is evaluated or as the generative source of design ideas.*¹ Benutzerzentrierter Entwurf stellt einen multidisziplinären Ansatz dar, der Anwenderwissen für ein scharfes Verständnis der Benutzer- und Prozeßanforderungen in Entwurf und Evaluierung der Benutzerschnittstelle nutzt. Vredenburg, Mao et.al. [31, 58] konstatieren, daß benutzerzentrierter Entwurf die Verwendbarkeit und Nützlichkeit eines Produkts verbessert. Der hohe Ressourcenaufwand ist dabei die Ursache für die stark unterschiedliche Verbreitung dieses Ansatzes und stellt sein Kosten-Nutzen Verhältnis in Frage. Rosenbaum et. al. [47] stellen eine geringe Durchdringung benutzerzentrierten Entwurfs in der Industrie fest. Nach ihren Schlußfolgerungen wird die Anwendung durch eine Ablehnungshaltung der Entwickler oder durch die Unkenntnis über nutzbare Techniken verhindert.

Konfigurierbare Benutzerschnittstellen versuchen die Akzeptanz der Entwickler dadurch zu erreichen, daß bei geringfügig höherem Entwicklungsaufwand ein gutes Kosten-Nutzen Verhältnis entsteht. Ein weiterer Vorteil besteht darin, daß nicht alle möglichen Spezialfälle im Entwurf bekannt sein müssen, sondern der Benutzer später selbständig die Schnittstelle an die Spezialfälle anpassen kann.

In der industriellen Entwicklung haben sich benutzerbeteiligender Entwurf und benutzerzentrierte Evaluierung als praxistaugliche Einschränkung erwiesen.

2.1.4.1 Benutzerbeteiligung im Entwurf

Die Beteiligung des Benutzers in allen Phasen der Entwicklung verspricht die Verbesserung von Nützlichkeit und Verwendbarkeit. Muller, Haslwanter und Dayton [37] stellen jedoch fest, daß diese Disziplin (*engl. participatory design*) keine anerkannte Definition besitzt. Generell bedeutet Benutzerbeteiligung in der Entwicklung, daß Benutzer qualifizierte Aussagen über ihre Bedürfnisse und Ziele in den Entwicklungsprozeß einbringen. In der Praxis wird eine Gruppe ausgesuchter Anwender in den Entwicklungsprozeß integriert.

Shneiderman [50] betont, daß die Qualität des Entwurfs nicht mit der Zahl beteiligter Anwender steigt. Insbesondere bleibt fraglich, wie Bedürfnisse von Benutzern, deren Teilnahme im Entwurf abgelehnt wurde, in das System integriert werden. Hierbei spielen auch nichttechnische Aspekte eine wichtige Rolle im Programmmentwurf:

¹aus [25], Seite 161

2.1 Konzepte für den Entwurf benutzerorientierter Schnittstellen

The experienced user interface architect knows that organizational politics and the preferences of individuals may be more important than the technical issues in governing the success of an interactive system.²

Benutzerbeteiligung im Entwurf versucht, eine verwendbare und nützliche Benutzerschnittstelle zu schaffen. Nicht allgemeingültige Bedürfnisse können dabei nicht umgesetzt werden. Weiterhin bleibt zu beantworten, ob eine ausgewählte Benutzergruppe die Bedürfnisse aller Anwender reflektieren kann. Anforderungen der Benutzer sind zeitlich variant und heterogen. Ein benutzerbeteiligter Entwurf ist somit lediglich ein Kompromiß aller in der Entwicklungsphase bekannten Anforderungen. Konfigurierbare Benutzerschnittstellen realisieren einen alternativen Ansatz: Anwender können konfigurierbare Benutzerschnittstellen bedürfnisgerecht für beliebige Situationen anpassen. Die Ursprungskonfiguration wird dabei benutzerbeteiligt entworfen.

2.1.4.2 Benutzerzentrierte Evaluierung

Mit benutzerzentrierter Evaluierung wird die Verwendbarkeit einer Benutzerschnittstelle aus Informationen der Anwender überprüft. Karat [26] definiert die Verwendbarkeit eines Produkts folgendermaßen: *Usability of a product is the extent to which the product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use.*³ Die Evaluierung durch Benutzer kann überraschende Mißverständnisse bei der Konzeption eines Systems hervorbringen. Leider liefert die Identifikation von Problemen der Verwendbarkeit noch nicht das Ziel der benutzerzentrierten Evaluierung. Dieses besteht vielmehr darin, ein System so anzupassen, daß die Bedürfnisse der Anwender erreicht werden und so Zufriedenheit mit dem Produkt erzielt wird. Karat [26] beschreibt folgende Informationsquellen benutzerzentrierter Evaluierung für verschiedene Stadien des Entwicklungsprozesses:

Verbale Berichte dienen der Erfassung kognitiver Prozesse des Benutzers bei der Bedienung des Systems. Sie können jederzeit eingesetzt werden, wenn allgemeine Informationen benötigt werden und eignen sich bestens zur Identifikation von Problemen.

Fragebögen ergeben eine subjektive Antwort und sind bestens für spezifische Fragen geeignet. Fragebögen können zu jeder Zeit in der Entwicklung eingesetzt werden, sofern eine scharfe Fragestellung besteht.

Anwendungsdaten liefern spezifische Maßstäbe der aktuellen oder simulierten Nutzung des Systems. Anwendungsdaten werden für spezifische Tests von alternativen Strukturen oder für vollständige Verwendung der Benutzerschnittstelle genutzt. Sie sind bestens geeignet für konkurrierende Evaluierung und die Erhebung von Performanzdaten.

Entwurfsdurchgänge mit Benutzern dienen Entwicklern zur Akzeptanzsteigerung des Entwurfs. Sie werden in frühen Entwicklungsstadien angewendet und sollten mehrfach wiederholt werden.

²aus [50], Seite 474

³aus [26], Seite 691

2 Stand der Technik: Anpassung der Schnittstelle an den Benutzer

Heuristische Prüfungen liefern ein Expertenurteil der Entwickler zur allgemeinen Akzeptanz der Benutzerschnittstelle. Diese Durchgänge filtern Verwendbarkeitsprobleme in frühen Entwicklungsstadien und können Informationen des Benutzers ergänzen.

Theoriebasierte Analysen sagen das Verhalten eines erfahrenen Benutzers voraus und können Anwendertests zur Beurteilung der Benutzerfreundlichkeit ersetzen. Dieses Verfahren findet Anwendung, wenn Testfälle aufgrund erwarteter großer Unschärfen der Meßdaten kaum auswertbar sind.

Karat [26] betont, daß bisher nicht alle evaluationsspezifischen Faktoren bekannt sind und die Forschung heute keine Methoden kennt, deren Effektivität zu maximieren. Verbesserungen in der benutzerzentrierten Evaluierung bedeuten bessere Datenerhebung und Analyse, um ihre Ergebnisse als Entwurfsstrategien in den Entwicklungsprozeß zu integrieren.

Manber [30] evaluierte das Portal MyYahoo!, das verschiedene Ansätze zur Personalisierung seines Erscheinens realisiert. Er formuliert die Verwendung von Personalisierung durch den Benutzer folgendermaßen: *In particular, if people are not sure how things work, they are less prone to experimentation, because they are afraid of breaking something, or getting into a state that cannot be undone. Any personalization feature should encourage experimentation.*⁴ Manber betont weiter, daß Personalisierung einer Benutzerschnittstelle Anwender zum Experimentieren ermutigen sollte. Häufig besteht bei Benutzern die Angst, etwas zu zerstören oder in nicht lauffähigem Zustand unwiderruflich zu hinterlassen. Die Personalisierung des Yahoo! Portals ermöglicht die Komposition der Benutzerschnittstelle aus einzelnen, unabhängigen Applikationen. Aus der Benutzermodellierung werden semantische Inferenzen als Approximation des Informationsbedarfs berechnet. Daraus werden dann potentiell nützliche Zusatzinformationen präsentiert. Manber stellt fest, daß *der Großteil der Anwender das Portal ohne Anpassung benutzt*. Hierfür nennt er drei mögliche Gründe:

1. Die allgemeine Seite ist so gut, daß keine Aufwände erforderlich sind.
2. Die Werkzeuge zur Anpassung sind so kompliziert, daß sich der Benutzer die Mühe nicht macht.
3. Viele Benutzer benötigen keine komplexe Anpassung.

Manber glaubt, daß der Grund der geringen Benutzung aus einer Kombination dieser drei Gründe besteht. Als weitere Erkenntnis stellte sich heraus, daß Anwender Personalisierung häufig nicht verstehen. Für eine benutzerzentrierte Evaluierung konfigurierbarer Benutzerschnittstellen ist zu erwarten, daß Fragebögen und direkte Rückmeldung aufgrund der Unberechenbarkeit des Benutzerverhaltens ein unscharfes Ergebnis liefern. Karat [26] weist auf die Diskrepanz von Testdaten aus der Verwendung von Prototypen gegenüber einer produktreifen Implementierung hin. Die theoriebasierte Analyse [26] ermöglicht alternativ eine scharfe Darstellung der Verwendbarkeit und Nützlichkeit durch Simulation eines Experten-Benutzers (*power user*).

⁴aus [30], Seite 37

2.1 Konzepte für den Entwurf benutzerorientierter Schnittstellen

Benutzerzentrierte Evaluierung verkörpert einen ressourcenintensiven Weg, um eine verwendbare Benutzerschnittstelle aus Informationen der Anwender zu entwickeln. Obwohl dieser Ansatz größten Wert auf die Zufriedenheit der Benutzer legt, bietet er kein Konzept für die Erfüllung spezifischer Bedürfnisse, da als Ergebnis der Entwicklung eine verbesserte, allgemeingültige Benutzerschnittstelle resultiert. Nachdem Anforderungen aller Benutzer häufig nicht zur Entwicklungszeit bekannt sind, eignet sich das Verfahren im Kontrast zu konfigurierbaren Benutzerschnittstellen für das ständig variierende Umfeld der Anwender nicht.

2.1.5 Automatisch anpassende Benutzerschnittstellen

Anpassende Systeme (*adaptive systems*) versuchen, die Benutzerschnittstelle für die Prozesse, die Fähigkeiten und die Expertise der Anwender anzupassen. Edmonds [12] unterscheidet Anpassung in drei Kategorien:

1. durch den Benutzer gefordert,
2. durch das System erfragt oder
3. automatisch.

Benyon und Murray [5] bezeichnen die ersten beiden Kategorien als benutzerangepaßt (*customized*). Die automatische Anpassung der Benutzerschnittstelle erfordert ein detailliertes und präzises Modell des Benutzers (genauer: der Bedürfnisse des Benutzers). Die Modellierung muß also die Eigenschaften des Benutzers erfassen, die zur Anpassung der Benutzerschnittstelle benötigt werden.

Anpassende Benutzerschnittstellen modellieren das Interaktionsverhalten des Anwenders für eine Menge konkreter Eigenschaften. Das System trifft die Entscheidung zur Anpassung aus den erfaßten Eigenschaften. Aufgrund der Komplexität können Benutzer häufig nur aus wenigen Eigenschaften modelliert werden. Die automatische Anpassung des Systems kann dadurch auf das Unverständnis des Benutzers treffen, der die Entscheidung nicht nachvollziehen kann. Automatische Anpassung kann weiterhin das Neuerlernen der Bedienung erfordern, wodurch die Produktivität des Benutzers durch zusätzliche Lernphasen geschwächt wird. Die Akzeptanz automatisch anpassender Benutzerschnittstellen ist sowohl bei Entwicklern als auch bei Anwendern fragwürdig. Entwickler müssen neben der Modellierung des Benutzers Schlüsse auf unbefriedigte Bedürfnisse ziehen und Ansätze zur Verbesserung mit Techniken zur automatischen Anpassung bereitstellen. Weiterhin benötigt das Verfahren eine intensive benutzerzentrierte Evaluierung, um aus der Beurteilung korrespondierende Adaptionen zu entwerfen. Durch die Vielfältigkeit der Anwendungssituationen konnten bisher keine allgemeingültigen Entwurfsverfahren entwickelt werden. Insbesondere bleibt fraglich, wie erkannte Defizite klassifiziert und daraus eine Menge möglicher Adaptionen zur Verbesserung der Benutzerschnittstelle entwickelt werden sollen.

Automatisch anpassende Systeme werden aufgrund ihrer Komplexität, mangelnder Nachvollziehbarkeit ihrer Adaptionen und der Budgetierung der Programmentwicklung kaum in die Praxis umgesetzt. Konfigurierbare Benutzerschnittstellen ermöglichen die Anpassung der Benutzerschnittstelle nach Edmonds' Kategorien 1 und 2: durch den Anwender oder das Benutzermodell erkannte Verbesserungsmöglichkeiten [12].

2.1.6 Endbenutzer Programmierung

Eisenberg [13] definiert Endbenutzer Programmierung (*end-user programming*) als *Benutzung einer Programmiersprache von nicht-professionellen Programmentwicklern*. Diese Programmieretechnik tendierte in der letzten Dekade zunehmend zu Werkzeugen mit grafischen Benutzerschnittstellen und wird als direkte Manipulation (*direct manipulation*) [16] kategorisiert. Endbenutzer Programmierung findet in der Praxis selten Anwendung. Folgende Gründe sprechen gegen die Integration einer Programmiersprache für Endbenutzer:

Erlernbarkeit Programmierung ist kompliziert. Aus ökonomischer Sicht sind einfacher erlernbare Systeme erfolgreicher platzierbar, da die Programmierung durch den Anwender einen unkalkulierbaren Zeitaufwand produziert.

Sinn der Programmierung Programmierung bedeutet unnötigen Aufwand. Die Dauer von Implementierung und Test sind nur sehr schwer schätzbar.

Für die Programmierung durch den Benutzer finden sich folgende positive Argumente:

Anpassbarkeit Nichtprogrammierbare Systeme ermöglichen kaum benutzererzeugte Erweiterungen. Programmierung erlaubt dagegen die Umsetzung von Effizienzsteigerungen, die ohne linguistische Ansätze verborgen blieben.

Ausdrucksmöglichkeit Programmiersprachen bieten ein Medium, mit dem Anwender Systemkonzepte modifizieren können. Benutzer können so ihre Ideen der Systemoptimierung umsetzen.

Kommunikationsmedium Die Programmierung durch Endbenutzer kann über mehrere Anwender verteilt werden, um gemeinsam Erweiterungen zu entwickeln. Potenziale und Erfahrungen lassen sich auf diese Weise erfolgreich kombinieren.

Erweiterte Fähigkeiten Die Endbenutzer Programmierung von Systemen erweitert die kognitiven Fähigkeiten ihrer Anwender. Die so erreichte Erfahrung bei programmtechnischen Problemlösungen kann bei zukünftigen Systemanwendungen eingesetzt werden.

Als Systeme zur Endbenutzer Programmierung etablierten sich beispielsweise AutoCAD [23] oder Emacs [1]. Der Einsatz der Endbenutzer Programmierung erfordert die Programmierfähigkeit der Anwender. Selbst die Verwendung grafischer Werkzeuge verlangt algorithmisches Denken.

Für Dateneingaben kann angenommen werden, daß Benutzer diese Voraussetzung nicht besitzen. Konfigurierbare Benutzerschnittstellen verfolgen daher einen orthogonalen Ansatz. Anwender benötigen keinerlei Programmierkenntnisse bzw. algorithmisches Verständnis zur Vereinfachung einer Benutzerschnittstelle. Optimierungen können ohne ein zusätzliches Programmierwerkzeug vorgenommen werden.

2.2 Plattformneutrale Beschreibungen

In der letzten Dekade etablierten sich zunehmend XML-basierte Darstellungen [9] grafischer Benutzerschnittstellen. Neben einfacher Beschreibungen der Benutzerschnittstelle in Entwicklungswerkzeugen wie SunONE [36], NetBeans [34] oder JBuilder

2.3 Verwandte Arbeiten

[6] wurden zunehmend plattform- und geräteunabhängige Repräsentationen entwickelt [56, 43, 2, 55, 14, 63, 66, 32]. Neutrale XML-Beschreibungen eignen sich für die Erzeugung von Gestalten in konfigurierbaren Benutzerschnittstellen.

2.2.1 User Interface Markup Language (UIML)

UIML [2] wurde als Metasprache zur Beschreibung grafischer Benutzerschnittstellen entwickelt. Hierbei dient UIML als neutrales Austauschformat beliebiger Schnittstellen – unabhängig von Interaktion, Sprache, Betriebssystem und Plattform. Für die Beschreibung einer konkreten Benutzerschnittstelle muß UIML um ein Vokabular von Schnittstellenteilen, Eigenschaften und Ereignissen erweitert werden. Dieser Ansatz trennt Daten von ihrer Repräsentation und ermöglicht damit ähnlich zu Swing [51, 53] den Austausch der grafischen Repräsentation von Interaktionselementen.

2.2.2 Extensible Interface Markup Language (XIML)

XIML [14] verkörpert eine Sprache zur Repräsentation und Manipulation von Interaktionsdaten auf abstraktem oder konkretem Niveau. Die Schnittstellendefinition erlaubt sowohl Strukturbeschreibungen als auch die Definition von Laufzeitoperationen, wobei die XIML-Definition die Darstellung der Interaktionsobjekte in einer konkreten Benutzerschnittstelle nicht formuliert. Das erforderliche Wissen über die Zielplattform und ihre grafischen Interaktionselemente, das während der Entwicklung durchaus Lücken aufweisen kann, bildet den wesentlichen Schwachpunkt dieses Ansatzes [56].

2.2.3 XForms

Die XForms 1.0 Spezifikation des World Wide Web Consortium (W3C) [63] versucht, die nächste Generation von Formularen für Internetseiten zu etablieren. Diese Formulare werden in umgebende Sprachen wie XHTML [62] oder VoiceXML [64] eingebettet und erlauben die Trennung von Datenmodell, Darstellung und Prozeßmodell.

2.2.4 Extensible Application Markup Language (XAML)

XAML [32] definiert grafische Benutzerschnittstellen in Microsofts kommendem Betriebssystem mit dem Codenamen Longhorn. XAML ermöglicht die Strukturbeschreibung als Hierarchie von Objekten, Eigenschaften und Logik. Diese adressiert die Formulierung einer grafischen Benutzerschnittstelle und deren Abbildung auf Objekte der Common Language Runtime [11] ohne Programmieraufwand.

2.3 Verwandte Arbeiten

Die im folgenden beschriebenen Konzepte beschreiben Ansätze zur Optimierung der Benutzerschnittstelle durch Vereinfachung von Dateneingaben. Ihre Zielsetzung ist mit dem Ziel konfigurierbarer Benutzerschnittstellen vergleichbar. Die folgenden Abschnitte beschreiben *Dynamic Forms* und *Adaptive Forms* und vergleichen sie mit dem Konzept dieser Arbeit.

2.3.1 Dynamic Forms

Girgensohn et. al. [17] stellen mit Dynamic Forms ein Konzept für grafische Benutzerschnittstellen vor, das mit dem Konzept dieser Arbeit vergleichbar ist. Dynamic Forms werden mit einer deskriptiven Sprache (Field Definition Language, FDL) definiert und als C++-Implementierung erzeugt. Als Entwicklungswerkzeug wird ein sog. *Form Editor* zur Verfügung gestellt, der Format, Beschreibung und Feldabhängigkeiten der Werte formuliert. Komplexere Konfigurationen müssen in der textuellen Konfigurationsdatei vom Programmierer manuell vorgenommen werden. Dynamic Forms bestehen aus Texteingaben und passen eine dem Eingabefeld zugeordnete Gestalt durch Änderung der Schriftart oder der Feldhöhe an. Im *Form Editor* kann eine Gestalt zur Darstellung ausgewählt werden. Dynamic Forms setzen auf automatische Gestaltung der Benutzerschnittstelle und eliminieren Eingabefelder, die aus dem Wert anderer Eingabedaten eindeutig bestimmt werden können. Hierbei werden eingebettete Formulare (*nested forms*) zum Ein- und Ausblenden von Teilformularen verwendet. Für die Sicherstellung der vollständigen Dateneingabe wird im Programm eine *checklist* hinterlegt, die durch Einfärbung der Interaktionselemente auf fehlende Dateneingaben hinweist. Unnötige Eingaben eindeutig bestimmbarer Elementwerte können umgangen werden. Abhängige Elemente (*active fields*) können mit einer oder mehreren Prozeduren ihren Wert aus den Elementwerten vorausgesetzter Elemente berechnen.

Das Konzept dieser Arbeit entwickelt kein neues Programmierparadigma für grafische Benutzerschnittstellen, sondern verkörpert eine intelligente Erweiterung grafischer Interaktionselemente zur Vereinfachung formularbasierter Datenerfassung. Die Programmierung konfigurierbarer Benutzerschnittstellen erfordert nur geringen Mehraufwand für den Entwickler. Dynamic Forms werden deklarativ als C++-basierte grafische Benutzerschnittstellen erzeugt. Konfigurierbare Benutzerschnittstellen erzeugen automatisch aus der Implementierung eine XML-basierte Konfigurationsdatei. Das Konzept dieser Arbeit geht nicht davon aus, daß abhängige Daten überwiegend eindeutig aus den Werten der übrigen Elemente berechnet werden. Vielmehr zeigt die Erfahrung, daß Auswahlmengen sehr häufig berechenbar sind, jedoch Abhängigkeiten, die eindeutige Werte ergeben, kaum auftreten. Somit sind unnötige Felder, die in der Benutzerschnittstelle unterdrückt werden können, nur selten enthalten. Abhängige Dateneingaben können ohne neues Programmierparadigma durch Benutzerbeteiligung im Entwurf zur Vereinfachung genutzt werden. Dynamic Forms können Dateneingaben für spezifische Anwendungsfälle zur Laufzeit nicht vereinfachen.

2.3.2 Adaptive Forms

Frank et.al. [15] stellen mit *Adaptive Forms* ein Verfahren vor, das Dateneingaben in Benutzerschnittstellen durch gezieltes Ein- bzw. Ausblenden von Dialogteilen vereinfacht. Hierbei wird die Abhängigkeitsbeziehung so gestaltet, daß einer Dateneingabe in Element *X* weitere Daten folgen müssen. Erfolgt keine Eingabe in *X*, so werden die abhängigen Interaktionselemente unterdrückt. Die Platzierung abhängiger Elemente erfolgt in Leserichtung (*forward-look-ahead*) in der Nähe des übergeordneten Elements. Typische Anwendungen des Konzepts sind Formulareingaben, bei denen Unterpunkte detailliertere Angaben erfragen, ähnlich zu einem Formular in Papierform. Hierfür wurde eine kontextfreie Grammatik zur Spezifikation des Formulars entwickelt, die keine Rekursionen erlaubt. Damit sind zyklische Abhängigkeitsgraphen

2.4 Zusammenfassung

zwischen Unterformularen nicht definierbar. Die grafische Benutzerschnittstelle wird dann aus der Formularspezifikation erzeugt.

Adaptive Forms beschreiben Abhängigkeiten zur Anzeige von Interaktionselementen für benötigte Details innerhalb von Formularen. Die Vereinfachung von Dateneingaben wird durch das Ausblenden irrelevanter Formulareile erreicht. Das Verfahren bietet kein Konzept zur Berechnung von Auswahlmengen und Konsistenzprüfungen. Die in Aussicht gestellte Programmbibliothek würde eine intuitive Entwicklung von grafischen Benutzerschnittstellen ermöglichen. Insbesondere bieten *Adaptive Forms* aber keinen Einfluß auf die Gestaltung der Benutzerschnittstelle. Konfigurierbare Benutzerschnittstellen setzen Vereinfachungen von Dateneingaben wesentlich flexibler ohne Beschränkung auf Detailinformationen um.

2.4 Zusammenfassung

Der Stand der benutzerzentrierten Techniken zeigt, daß formularbasierte Benutzerschnittstellen kaum für Spezialfälle des Anwenders durch einen Programmierer vereinfacht werden können. Das Konzept dieser Arbeit nutzt – anders als die vorgestellten Verfahren der statischen Analyse wie beispielsweise *Dynamic Forms* – das Wissen des Benutzers über Vereinfachungsmöglichkeiten für Spezialfälle. Die im mentalen Modell des Anwenders existierende Vorstellung über Vereinfachungsmöglichkeiten kann vom Programmierer nicht zur Optimierung der allgemeingültigen Benutzerschnittstelle verwendet werden. Insbesondere können Anwender ihre Vorstellungen häufig nicht präzise formulieren. Das Verfahren greift hier die Idee von Manber [30] auf, Benutzer zum Experimentieren zu ermutigen. Anwender sollen ihre Vorstellung selbst durch Experimentieren und mit Unterstützung der Benutzerschnittstelle vereinfachen.

Benutzerzentrierte Konzepte versuchen, Benutzerschnittstellen für die Bedürfnisse der Anwender zu optimieren. Allgemeingültige Verfahren dieser Techniken konnten sich für den Programmentwurf und die Evaluierung nicht etablieren. Die aktuelle Forschung beschäftigt sich mit der Effizienzsteigerung von Benutzerschnittstellen aus Ergebnissen der Evaluierung. Benutzerzentrierte Ansätze leiden heute unter verschiedenen Schwächen. Häufig sind Verfahren nur für konkrete Einsatzgebiete entwickelt und wenig flexibel für den Wechsel des Anwendungsumfelds. Die hier vorgestellten Verfahren sind häufig schwierig zu implementieren und wegen des hohen Aufwands und aufgrund der Budgetierung der Programmentwicklung nicht in der Praxis anwendbar.

Anwender erreichen die Vereinfachung konfigurierbarer Benutzerschnittstellen durch selbständige Anpassung von Formularen. Die Anpassung vereinfacht die Datenerfassung, indem die Zahl der notwendigen Eingaben reduziert wird oder die Auswahlmengen eingeschränkt werden. Hierbei unterstützt das System mit Vorschlägen zur Modifikation der Interaktionsfolge. Optimierungen erfolgen ausschließlich mit dem Einverständnis des Benutzers oder durch seine Initiative und sichern auf diese Weise seine Akzeptanz.

2 Stand der Technik: Anpassung der Schnittstelle an den Benutzer

Kapitel 3

Analyse der Benutzerinteraktion

Als Grundlage zur Unterstützung des Anwenders muß sein Interaktionsverhalten analysiert werden. Aus diesem Verhalten kann auf Basis des Benutzermodells die bevorzugte Eingabefolge berechnet werden. Abschnitt 3.1 stellt Techniken der ereignisbasierten Evaluierung vor. Abschnitt 3.2 erläutert Verfahren zur Analyse von Ereignissequenzen. Die Sequenzanalyse von Ereignisströmen kann für die Berechnung des Benutzermodells genutzt werden. Abschnitt 3.3 beschreibt die Adaption der Verfahren zur ereignisbasierten Modellierung des Benutzers.

3.1 Ereignisbasierte Evaluierung von Benutzerschnittstellen

Ereignisbasierte Evaluierung beinhaltet die Analyse des Interaktionsverhaltens. Diese Techniken können zur ereignisbasierten Modellierung des Benutzers im Konzept dieser Arbeit verwendet werden.

Ereignisse (*events*) werden von grafischen Benutzerschnittstellen als Information der Benutzerinteraktion bereitgestellt [4, 65, 51]. Das Verhalten des Benutzers kann damit auf verschiedenen Abstraktionsebenen erfaßt werden. Die Kombination von Ereignissen und dem Wissen über das Anwendungsumfeld ermöglicht die Extraktion von Daten zur Analyse der Nützlichkeit und Verwendbarkeit einer Benutzerschnittstelle.

Hilbert und Redmiles [21] kategorisieren Klassen technischer Ansätze:

- Synchronisation und Suche
- Transformation von Ereignisströmen
- Quantität und Erfassung von statistischen Summationen
- Sequenzerkennung und Sequenzvergleich
- Charakterisierung von Sequenzen

- Visualisierung
- Unterstützung integrierter Evaluierung

Die aufgelisteten Techniken dienen der Analyse der Verwendbarkeit einer grafischen Benutzerschnittstelle. Nielsen [42] definiert Verwendbarkeit (*usability*) mit fünf Attributen:

- Lernbarkeit
- Effizienz
- Erinnerung
- Fehler
- Zufriedenheit

Hilbert und Redmiles [21] definieren die Evaluierung der Verwendbarkeit (*usability evaluation*) als Identifikation potentieller Aspekte, die Attribute der Verwendbarkeit eines Systems oder Geräts auf partielle Benutzergruppen, partielle Prozesse oder partielle Kontexte beeinflussen. Grudin [19] betrachtet die Eigenschaften *Verwendbarkeit* (*usability*) und *Zweck* (*utility*) als Unterkategorien der Verallgemeinerung *Nützlichkeit* (*usefulness*). Die Evaluierung der Verwendbarkeit wird weiter unterschieden in *formativ* und *summativ*. Während formative Evaluierung eine Rückkopplung zum Entwickler beabsichtigt, signalisiert summativ Evaluierung die Produktauglichkeit eines Systems. Die Erfassung der Verwendbarkeitsdaten (*usability data*) wird von Sweeny et.al. [54] mit folgenden Indikatoren versehen:

- Online Verhalten/Performanz
- Offline (nichtverbales) Verhalten
- Auffassung/Verständnis
- Einstellung/Urteil
- Streß/Nervosität

Tabelle 3.1 zeigt eine Übersicht über die Sammlung von Verwendbarkeitsdaten. Die Performanz bzw. das Online Verhalten eines Benutzers kann durch die Analyse von Ereignissen der Benutzerschnittstelle erfolgen. Sanderson und Fisher [48] kategorisieren Ereignisse nach ihrer Dauer. Die resultierenden Frequenzbänder besitzen Schwingungsdauern von 10 Millisekunden bis zu einem Jahr. Kürzer dauernde Ereignisse werden als hochfrequent angenommen, länger dauernde als niederfrequent. Frequenzbänder. Die Evaluierung der Verwendbarkeit basiert auf hochfrequenten Ereignissen, die Sanderson und Fisher wie folgt charakterisieren:

- Synchron/asynchrone Ereignisse
- Komposition mehrerer Ereignisse aus höheren Frequenzbändern
- Inferenz über Frequenzbänder

3.1 Ereignisbasierte Evaluierung von Benutzerschnittstellen

	Techniken der Datenerfassung		
Indikatoren der Verwendbarkeit	Aufzeichnung von Ereignissen	Audiovisuelle Aufnahmen	spätere Kommentare
Online	•	•	
Offline		•	
Auffassung	•	•	•
Verständnis			•
Streß			

	Techniken der Datenerfassung		
Indikatoren der Verwendbarkeit	verbale Befragung	Fragebögen	psychophysikalische Aufzeichnungen
Online			
Offline			
Auffassung	•	•	
Verständnis	•	•	
Streß			•

Tabelle 3.1: Indikatoren zur Sammlung von Verwendbarkeitsdaten (aus [21])

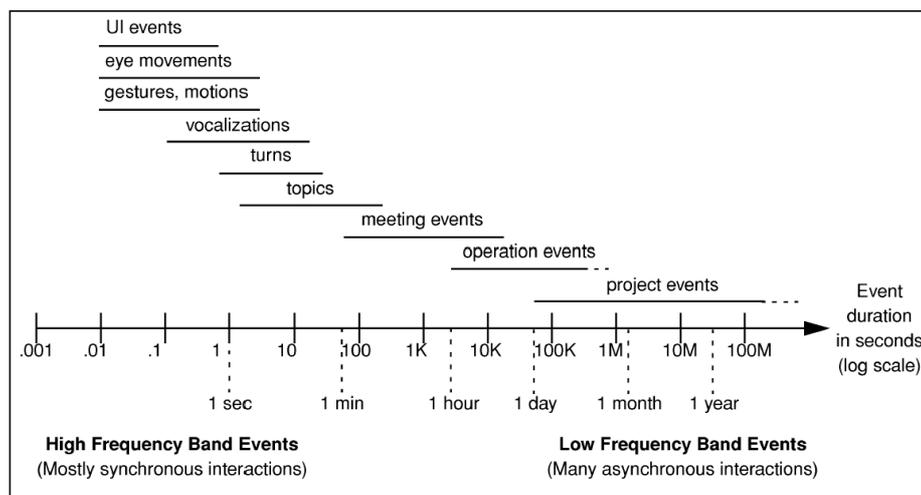


Abbildung 3.1: Frequenzspektrum von Ereignissen (aus [48])

Ereignisse können häufig als grammatikalische Strukturen formuliert werden. Grammatiken werden in verschiedenen Disziplinen zur Charakterisierung der Struktur sequentieller Daten verwendet. Die Fähigkeit zur Identifikation von Interaktionsmustern wird hierbei zur Suche von Äquivalenzklassen und zyklischen Interaktionen genutzt. Ein großes Problem stellen hierbei kontextuelle Hintergründe der Ereignisse dar: viele Ereignisse können nur durch die Betrachtung assoziierter Ereignisse interpretiert werden. In zahlreichen Fällen können Ereignisse mit kontextbezogenen

Assoziationen nur unmittelbar nach ihrem Auftreten interpretiert werden, eine asynchrone Auswertung der Ereignissequenz wäre erfolglos.

Hilbert et.al. definieren Abstraktionsebenen für hochfrequente Ereignisse. Tabelle 3.2 zeigt die Ebenen der Abstraktion. Physikalische Ereignisse werden als untere

ziel-/problembezogene Ereignisse (z.B. Aufnahme einer Bestellung)
umgebung-/prozeßbezogene Ereignisse (z.B. Eingabe einer Adresse)
Ereignisse der abstrakten Interaktionsebene (z.B. Werteingabe in Interaktionselementen)
Ereignisse der Benutzerschnittstelle (z.B. Fokuswechsel, Tastaturereignisse)
gerätespezifische Ereignisse (z.B. hardwareerzeugte Interrupts)
physikalische Ereignisse (z.B. Handbewegung)

Tabelle 3.2: Abstraktionsebenen hochfrequenter Ereignisse

Ebene definiert. Typischerweise erzeugen Interaktionen des Benutzers Ereignisse verschiedener Ebenen.

Die Betrachtung der transformierten Ereignisströme aus der Benutzerinteraktion bildet die Basis für die Evaluierung der Benutzerschnittstelle. Hilbert und Redmiles [21] unterscheiden zwischen *formativer* und *summativer Evaluierung*. Formative Evaluierung wird verwendet, um eine Rückkopplung in den Entwurfsprozeß zu schaffen. Entwickler erhalten Informationen darüber, ob Entwurfsentscheidungen zu einer verwendbaren Benutzerschnittstelle geführt haben. Summative Evaluierung beurteilt den Grad der Verwendbarkeit einer Benutzerschnittstelle. Dieser Grad wird genutzt, um Fortschritte in der Verwendbarkeit überarbeiteter Versionen der Benutzerschnittstelle nachzuweisen. Summative Evaluierung eignet sich dadurch auch für den Vergleich zweier Benutzerschnittstellen anhand von Verwendbarkeitsmetriken.

In dieser Arbeit werden alternative Formularstrukturen anhand der erwarteten, notwendigen Interaktionen des Anwenders miteinander verglichen. Durch die Betrachtung aller Alternativen werden diejenigen Formularstrukturen gefunden, die möglichst wenige bzw. durch Auswahlmengen vereinfachte Dateneingaben erfordern. Der folgende Abschnitt beschreibt Verfahren zur Analyse von Ereignisströmen von Benutzerschnittstellen.

3.2 Techniken zur Analyse von Ereignisströmen

Ereignisströme können mit verschiedenen Techniken und Zielsetzungen analysiert werden. Im folgenden werden die Ansätze der ereignisbasierten Analyse klassifiziert. Techniken zur Analyse von Sequenzen in Ereignisströmen können zur Identifikation bevorzugter Eingabefolgen des Anwenders genutzt werden.

3.2 Techniken zur Analyse von Ereignisströmen

3.2.1 Synchronisation und Suche

Ereignisse von Benutzerschnittstellen liefern Informationen zum Benutzerverhalten, das automatisiert erfaßt, durchsucht, gezählt und analysiert werden kann. Die Schwierigkeit besteht darin, Ereignisse höheren Interesses aus dem Ereignisstrom zu filtern. Zusätzlich können kontextuelle Informationen im Ereignisstrom fehlen, was zu falscher oder unmöglicher Interpretation führt. Synchronisation und Suche können Ereignisdaten mit semantisch wertvollen Beobachtungen des Benutzers verknüpfen. Diese Verknüpfung – beispielsweise synchronisiert mit Videoaufzeichnungen – kann zur Gewinnung höherwertiger Ereignisse als ergänzende Information aus dem Ereignisstrom genutzt werden. Diese Form der Extraktion von Ereignissen erzeugt etwa den zehnfachen Aufwand der eigentlich erfaßten Videosequenz [48, 54].

3.2.2 Transformation von Ereignisströmen

Ereignisströme werden mit verschiedenen Zielsetzungen transformiert, beispielsweise zur Erkennung menschlicher Verhaltensmuster oder zur Bereitstellung von Daten für Analysewerkzeuge. Die *Selektion* von Ereignissen extrahiert interessante Sequenzen vom sogenannten *Rauschen*. Hierzu müssen Bedingungen zur Extraktion relevanter Sequenzen spezifiziert werden. Die *Abstraktion* synthetisiert neue Ereignisse aus einem Ereignisstrom und ergänzt dabei möglicherweise semantische Informationen. *Aufzeichnung* bindet die Erzeugung neuer Ereignisströme in die Resultate von Selektion und Abstraktion ein. Damit können semantisch identische, äquivalente partielle Sequenzen erkannt und mit Analysetechniken kombiniert werden. Die Schwierigkeit dieser Technik besteht darin, daß bei Selektion, Abstraktion und Aufzeichnung Daten verworfen werden, die eine wichtige Rolle in der Analyse spielen. Insbesondere die Auswahl relevanter Ereignisse durch den Anwender verursacht u. U. den Verlust nützlicher Informationen.

3.2.3 Quantitative Erfassung von Ereignissen

Die Erfassung quantitativer Daten erlaubt die Analyse des Onlineverhaltens der Benutzer mit automatisierten Werkzeugen. Diese Art der Analyse ermöglicht den Einsatz von Metriken zur Analyse eines Ereignisstroms. Folgende Metriken können beispielsweise auf einen Ereignisstrom angewendet werden:

Performanz-Zeit Wieviel Zeit wird zur Ausführung eines Prozesses benötigt?

Weg des Mauszeigers Ist die Summe der Distanzen zwischen Mausklicks unnötig hoch?

Häufigkeit von Kommandos Welche Kommandos werden am häufigsten benutzt, welche gar nicht?

Häufigkeit von Kommandopaaren Welche Kommandopaare werden am häufigsten gemeinsam genutzt? Könnten sie näher zueinander platziert werden?

Abbruch und Umkehrung Welche Dialoge werden häufig abgebrochen, welche Kommandos werden häufig zurückgerollt?

Wechsel physikalischer Geräte Wechselt der Benutzer unnötigerweise zwischen Tastatur und Maus? Welche Funktionen können mit dem Wechsel assoziiert werden?

Die Schwierigkeit dieser Technik liegt in der Verknüpfung von Ergebnissen der Metrik mit Funktionen der Benutzerschnittstelle. Identifizierte Probleme, z.B. häufiges Umkehren von Aktionen, deutet auf eine mißverständliche Struktur der Benutzerschnittstelle hin. Aus den erkannten Fehlern muß anschließend eine Strategie zur Verbesserung entwickelt werden. Aufgrund der Vielfältigkeit von Anwendern und Fehlerursachen werden hier Modelle betrachtet. Das kann dazu führen, daß tatsächliche Ursachen aufgrund der Reduktion von Informationen nicht mehr erkannt werden können.

3.2.4 Sequenzerkennung und Sequenzvergleich

Die Analyse von Sequenzen erlaubt die Erkennung von Mustern in Ereignisströmen. Mit dieser Technik können Analytiker anstatt einzelner Ereignisse interessante Sequenzen betrachten. Die Spezifikation von Sequenzen erfolgt beispielsweise grammatikalisch oder mittels regulärer Ausdrücke [48]. Die große Menge der erzeugten Daten bereitet hier Probleme: sie ist schwierig zu interpretieren und führt häufig nicht zur Identifikation von Schwächen der Verwendbarkeit.

Der Sequenzvergleich ermöglicht die Identifikation partiell übereinstimmender Ereignismuster. Diese Technik erlaubt den Vergleich aufgezeichneter Ereignisfolgen mit erwarteten Ereignismustern. Hiermit können Divergenzen in der Interaktion mit dem System erkannt und Defizite in der Verwendbarkeit identifiziert werden. Die wesentliche Schwäche dieser Technik besteht darin, daß sowohl der Ereignisstrom als auch die zu erkennenden Muster segmentierbar sein müssen. Zusätzlich wird für die Analyse der Verwendbarkeit auf partielle Übereinstimmungen Expertenwissen vorausgesetzt.

3.2.5 Charakterisierung von Sequenzen

Bei der Charakterisierung von Sequenzen wird versucht, ein Modell zu generieren, das interessante Eigenschaften des Ereignisstromes identifiziert. Hieraus können Prozeßmodelle oder grammatikalische Strukturen des Ereignisstroms extrahiert werden. Der Einsatz dieser Technologie gilt als zeitaufwendig und erfordert die Einbeziehung von Experten. Weiterhin ist unklar, ob die Charakterisierung tatsächlich verwendbare Analysedaten produziert [21].

3.2.6 Visualisierung

Die Visualisierung stellt Analyseergebnisse in einer Form dar, mit der Experten visuelle Muster zur Interpretation der Daten nutzen können. Abbildung 3.2 zeigt die Visualisierung von Lokalität und Dichte von Mausclicks [20]. Ein wesentlicher Nachteil der Visualisierung besteht darin, daß die Implementierung der Darstellung von der tatsächlichen Beschaffenheit der Analysedaten abhängt und kaum mit Standardwerkzeugen umsetzbar ist.

3.2.7 Unterstützung integrierter Evaluierung

Umgebungen, die eine flexible Komposition von Transformation, Analyse und Visualisierung ermöglichen, leisten eine integrierte Unterstützung in der Entwicklung. Einige Umgebungen erlauben die Verwaltung umgebungsspezifischer Artefakte, Prozesse

3.3 Lösungsansatz zur ereignisbasierten Modellierung des Benutzers

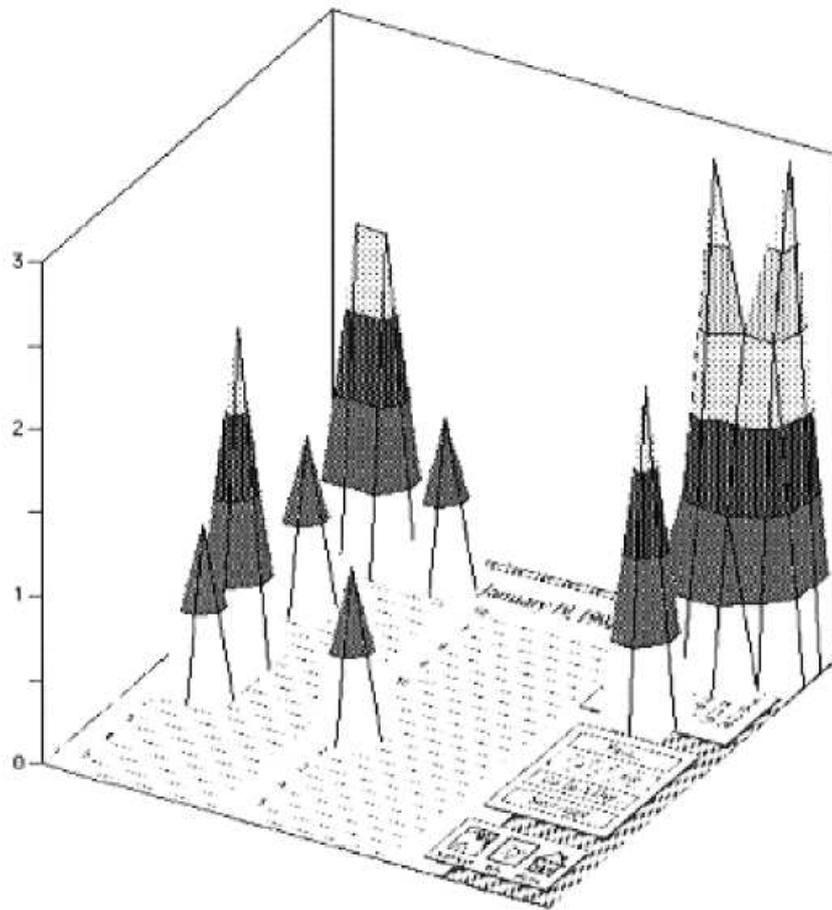


Abbildung 3.2: 3D-Darstellung der Mausclick-Dichte in einer grafischen Benutzerschnittstelle (aus [20])

und Daten bzw. Ergebnisse der Analyse. Die Verwaltung multipler Datenquellen und -formate, die aus Ereignissen erzeugt wurden, und die Komposition mehrerer Techniken reduziert damit signifikant die Verwaltung und Integration der Analysedaten.

3.3 Lösungsansatz zur ereignisbasierten Modellierung des Benutzers

Programmbibliotheken grafischer Benutzerschnittstellen bieten verschiedene Ansätze zur Reaktion auf die Interaktion des Anwenders. Komponentenbasierte Implementierungen bieten ereignisbasierte Reaktionsmöglichkeiten als Rückrufmechanismen: Über vereinbarte Schnittstellen werden Operationen bereitgestellt, die von den Interaktionselementen ausgelöst werden. Die relevanten Ereignisse der Benutzerschnittstelle werden dabei zu Ereignissen der abstrakten Interaktionsebene zusammengefaßt. Für die Modellierung des Benutzers werden alle relevanten Ereignisse gesammelt und daraus eine bevorzugte Interaktionsfolge berechnet. Abbildung 3.3 verdeutlicht das Prin-

zip der ereignisbasierten Benutzermodellierung. Das Interaktionsverhalten des Anwenders wird durch die in der Benutzerschnittstelle auftretenden hochfrequenten Ereignisse erfaßt. In die Modellierung des Benutzers können neben hochfrequenten auch niederfrequente Ereignisse einfließen. Aus der Modellierung des Benutzers wird anschließend die bevorzugte Eingabefolge berechnet.

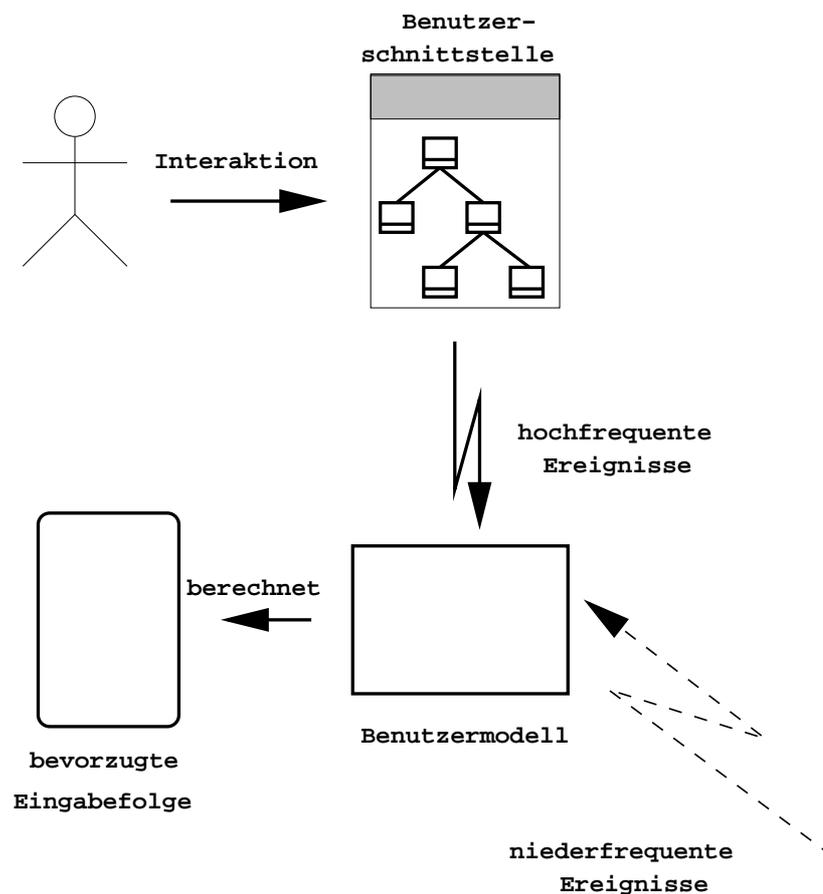


Abbildung 3.3: Prinzip der ereignisbasierten Benutzermodellierung

3.3.1 Auswahl relevanter Ereignisse

Die Klassifizierung relevanter Ereignisse hängt grundsätzlich vom Einsatzgebiet der Benutzerschnittstelle und dem technischen Umfeld ab. Audiovisuelle Beobachtungen des Anwenders finden in der Praxis nur selten Verwendung. Interaktionsbasierte Ereignisse können mit verschiedenen Methoden erfaßt und zur Modellierung des Benutzers verwendet werden. Das in Abbildung 3.4 markierte Frequenzband vereinigt relevante, interaktionsbasierte Ereignisse. Die Kategorisierung semantischer Ereignisse erfolgt spezifisch für eine konkrete Implementierung. Üblicherweise werden hierbei komponentenspezifische Ereignisse auf semantische Ereignisse der konfigurierbaren Benut-

3.4 Zusammenfassung

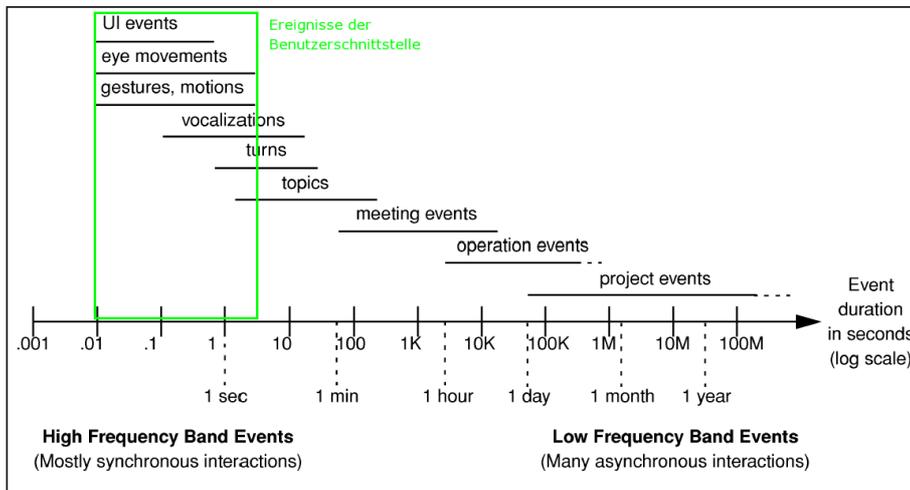


Abbildung 3.4: Frequenzbereich zur Berechnung des Benutzermodells (aus [20])

zerschnittstelle abgebildet¹. Kapitel 5 und Anhang A demonstrieren dieses Vorgehen.

3.3.2 Berechnung der bevorzugten Eingabefolge

Die Folge der erfaßten Ereignisse bildet die Grundlage zur Berechnung der vom Anwender bevorzugten Eingabefolge. Auch hier kann erst für eine konkrete Implementierung bestimmt werden, welche semantische Aussage diese Folge verkörpert. Wiederkehrende Interaktionssequenzen können beispielsweise auf fehlerhafte Eingaben hinweisen. Ein einfacher Ansatz besteht darin, die Wichtigkeit der Interaktionselemente nach der Reihenfolge der Dateneingaben zu bewerten.

Für die Berechnung der Eingabefolge sind die hier vorgestellten Techniken zur Analyse von Ereignisströmen anwendbar. Hierfür können hochfrequente Ereignisse der Benutzerschnittstelle und niederfrequente Ereignisse aus dem Anwendungsumfeld des Benutzers verwendet werden. Generell eignen sich Verfahren zur ereignisbasierten Evaluierung zur Erzeugung eines Benutzermodells für konfigurierbare Benutzerschnittstellen.

3.4 Zusammenfassung

Die Analyse der Anwenderinteraktion stellt eine Vorstufe zur Betrachtung der Nützlichkeit einer Benutzerschnittstelle dar. Techniken zur Evaluierung der Verwendbarkeit können für die Berechnung der bevorzugten Interaktionsfolge des Anwenders angepaßt werden und finden bei der Erzeugung des Benutzermodells Anwendung. Für den Prototypen Look_{AS}^ILike wurde die Modellierung des Benutzers aus hochfrequenten Ereignissen implementiert. Das Benutzermodell in Anhang A implementiert eine Transformation von Ereignissen der Benutzerinteraktion und liefert eine Sequenz von

¹ Im Prototypen wurden die Hörer *ActionListener*, *ItemSelectedListener* und *ItemInputListener* als semantisch angesehen.

3 Analyse der Benutzerinteraktion

Formularfeldern als bevorzugte Interaktionsfolge zur Beurteilung und Vereinfachung der konfigurierbaren Benutzerschnittstelle.

Kapitel 4

Unterstützung des Benutzers bei der Vereinfachung von Dateneingaben

Zur Beurteilung der Ablauffähigkeit von Berechnungen der Auswahlmengen und Konsistenzprüfungen müssen zunächst Abhängigkeiten zwischen Formularfeldern analysiert werden. Für die Unterstützung des Anwenders wird daraus ein Verfahren entwickelt, das die bevorzugte Eingabefolge mit der Struktur der Benutzerschnittstelle vergleicht. Aus der Struktur kann ein Kontrollfluß angenommen werden, der einen wesentlichen Bestandteil der Analyse darstellt. Aus Kontrollfluß und den Abhängigkeiten kann dann die Komplexität der Benutzerschnittstelle für die Anwenderunterstützung gemessen werden.

Abschnitt 4.1 analysiert die Anforderungen, die an die Vereinfachung einer formularbasierten Benutzerschnittstelle gestellt werden. Abschnitt 4.2 führt die benötigten Definitionen und Begriffe zur Beschreibung des Konzepts ein. Im anschließenden Abschnitt 4.3 erläutert eine konfigurierbare Benutzerschnittstelle anhand eines Formulars zur Adresseingabe. Abschnitt 4.4 beschreibt die Berechnung des Kontrollflusses, aus dem die angenommene Eingabefolge der Formularfelder berechnet wird.

4.1 Anforderungen an die Vereinfachung formularbasierter Datenerfassung

In diesem Abschnitt werden die Anforderungen für die Vereinfachung formularbasierter Benutzerschnittstellen formuliert. Es wird beschrieben, wie konfigurierbare Benutzerschnittstellen die Anforderungen erfüllen. Neben den technischen Grundlagen spielt die einfache Bedienung durch den Anwender eine wichtige Rolle beim Entwurf des Konzepts der konfigurierbaren Benutzerschnittstelle.

Die Vereinfachung der Datenerfassung bei formularbasierten Benutzerschnittstellen erfolgt durch Einsparen von Eingaben in Formularfelder, deren Werte bereits implizit durch die Werte anderer Formularfelder bestimmt sind. Diese Beziehungen zwischen Formularfeldern müssen vom Entwickler im Programm definierbar sein.

4 Unterstützung des Benutzers bei der Vereinfachung von Dateneingaben

Die Benutzerschnittstelle muß diese Beziehungen zur Laufzeit des Programms zur Einsparung der Dateneingaben verwenden und dabei die bereits erfaßten Daten nutzen können. Der Anwender möchte die Benutzerschnittstelle für seine Spezialfälle vereinfachen. Die Vereinfachung wird dadurch erreicht, daß der Benutzer nur die Felder eingibt, deren Daten unbedingt notwendig sind. Für seinen Spezialfall sind alle übrigen Daten bereits implizit angegeben. Die Benutzerschnittstelle sollte also die implizit gegebenen Werte mit Hilfe der Beziehungen zu bereits eingegebenen Daten automatisch ergänzen. Die Benutzerschnittstelle sollte für den Spezialfall nur die notwendigen Eingaben in der erforderlichen Reihenfolge darstellen.

Die Beziehungen zwischen Formularfeldern werden als Abhängigkeiten definiert. Abhängigkeiten werden als Regeln in konfigurierbare Elemente integriert. Zum Start der Benutzerschnittstelle werden alle Abhängigkeiten zu einem Abhängigkeitsgraph zusammengefügt. Aus dem Abhängigkeitsgraph ermittelt die Benutzerschnittstelle die Beziehungen zwischen den Formularfeldern. Nach jeder Dateneingabe in ein beliebiges Formularfeld prüft das System die Erfüllung neuer Abhängigkeiten und prüft die Konsistenz bei bereits vorliegenden Werten berechneter Auswahlmengen abhängiger Elemente. Die Ausnutzung der Transitivität eines Baumes von erfüllten Abhängigkeiten ermöglicht die Berechnung der Auswahlmengen aller in Beziehung stehenden Formularfelder. Im Falle der Eindeutigkeit kann einem Formularfeld ein konsistenter Wert zugeordnet und damit die Dateneingabe gespart werden.

Um die Felder zu identifizieren, die im Spezialfall des Anwenders die erforderlichen Informationen enthalten, beobachtet die Benutzerschnittstelle die Eingabefolge durch Aufzeichnen und Analysieren der im Programm auftretenden Ereignisse der Interaktionselemente. Die angenommene bevorzugte Eingabefolge wird mit allen Eingabefolgen verglichen. Folgen, die eine bessere Nutzung der Abhängigkeiten zwischen den Feldern ermöglichen, werden als bessere Eingabefolgen angesehen. Folgen, die Abhängigkeiten besser nutzen, speziell Folgen, die mit der Eingabefolge des Anwenders vergleichbar sind, sind Kandidaten für die Vereinfachung der Datenerfassung durch Strukturänderung der Benutzerschnittstelle. Ein Werkzeug, das hier prototypisch entwickelt wurde, schlägt dem Benutzer die Änderung der Struktur zur Vereinfachung der Datenerfassung vor. Die Erweiterung der Komponentenbibliothek grafischer Benutzerschnittstellen erlaubt die Modifikation der Struktur durch den Benutzer. Beim Prototypen Look_{As}^ILike kann der Anwender in einen Editiermodus umschalten und die Struktur des Formulars durch Verschieben der Felder verändern und anschließend speichern. Damit kann der Anwender selbständig Änderungen der Benutzerschnittstelle für beliebige Spezialfälle mit Unterstützung des Systems zur Laufzeit vornehmen, ohne daß Programmierung erforderlich ist.

Die folgenden Abschnitte beschreiben die Modellierung von Abhängigkeiten zwischen Formularfeldern und die Analyse und Beurteilung beliebiger Eingabefolgen des Formulars. Das Beispiel eines Adressformulars zeigt, daß Beziehungen bereits bei wenigen Feldern sehr komplex sein können. In diesem Beispiel wird gezeigt, wie Zusammenhangskomponenten des Abhängigkeitsgraphen die Komplexität der Beurteilung deutlich reduzieren können.

4.2 Formulierung von Abhängigkeiten zwischen Dateneingaben

Modifizierbare Komponentenstrukturen werden im Konzept dieser Arbeit technologie-neutral definiert. Hierfür wird zunächst die Gestalt definiert, die das Aussehen bzw. Erscheinungsbild einer Formulareingabe festlegt. Anschließend wird die Berechnung der Auswahlmenge und Konsistenzprüfung für eine Abhängigkeit angegeben. Auf diese Weise wird der Teil des Systems, der die Reaktion auf abhängige Dateneingaben implementiert, aus der Programmlogik entkoppelt und mit einem Formularfeld verknüpft. Diese Verbindung von Gestalten und Berechnungen führt zur Definition des Elements. Es bettet Gestalten und Algorithmen zur Berechnung der Auswahlmenge und Konsistenzprüfung der Abhängigkeitsbeziehungen ein.

Definition Gestalt

Die Gestalt eines Elements stellt die visuelle Darstellung eines Formularfelds zur Dateneingabe dar.

Im Falle einer grafischen Benutzerschnittstelle wird die Gestalt mit grafischen Primitiven zur Anzeige implementiert und nutzt Eingabegeräte wie Tastatur oder Maus zur Interaktion.

Definition Berechnung

Eine Berechnung kalkuliert die Auswahlmenge und die Konsistenzprüfung eines Elements aus den Werten von mindestens einem konfigurierbaren Element der Benutzerschnittstelle. Sie wird mit Referenzen auf die vorausgesetzten Elemente parametrisiert.

Definition konfigurierbares Element

Ein konfigurierbares Interaktionselement, kurz konfigurierbares Element, vereinigt mindestens eine Gestalt und eine beliebige Zahl von Berechnungen.

Definition Wert eines Elements

Der Wert eines konfigurierbaren Elements wird vom Anwender eingegeben oder ist eine beliebige Teilmenge der Auswahlmenge.

Der Wert eines konfigurierbaren Elements wird von der Gestalt als Selektion in der Auswahlmenge dargestellt. Für konfigurierbare Elemente wird angenommen, daß Mengenelemente der Auswahlmenge jeweils durch einen einzigen Parameter darstellbar sind. Mengenelemente sind Texte (String), Zahlen (Double, Float, Integer). Alle übrigen Datentypen lassen sich durch ihre Serialisierung auf einen Text (String) abbilden. Im Fall der nichteindeutigen Selektion wird der Wert eines konfigurierbaren Elements durch einen mengenwertigen Datentyp (z.B. eine Liste) beschrieben. Im folgenden werden Begriffe zur Formulierung von Abhängigkeiten zwischen konfigurierbaren Elementen definiert. Aus diesen wird der Abhängigkeitsgraph erzeugt.

Definition abhängiges Element

Ein abhängiges Element enthält Berechnungen, die Werte weiterer Elemente zur Berechnung der Auswahlmenge und Konsistenzprüfung benötigen.

Definition vorausgesetztes Element

Ein vorausgesetztes Element ist ein konfigurierbares Element, das in mindestens einer Berechnung eines anderen Elements benötigt wird.

Definition Abhängigkeitsgraph

Der Abhängigkeitsgraph wird als Tupel (E, V) der Knoten- und Kantenmenge definiert. Die Knotenmenge E besteht aus den konfigurierbaren Elementen. Die Kantenmenge V beschreibt die Abhängigkeitsbeziehungen zwischen den konfigurierbaren Elementen der Benutzerschnittstelle. Eine Kante $X \rightarrow Y$ zwischen zwei Knoten X und Y des Abhängigkeitsgraphen bedeutet, daß eine Wertänderung in X eine Änderung der Auswahlmenge in Y bewirken kann. Eine Wertänderung in Y erfordert die Konsistenzprüfung des Wertes von X . Man sagt: X ist vorausgesetztes Element von Y bzw. Y ist ein von X abhängiges Element. Der Abhängigkeitsgraph ist gerichtet, potenziell zyklisch und unzusammenhängend. Eine Kante $X \xrightarrow{q} Y$ bedeutet, daß die Auswahlmenge von Y mit der Güte q aus dem Wert von X berechnet werden kann.

Ein Abhängigkeitsgraph ist ein gerichteter, zyklischer Graph, bei dem jede Kante mit mehreren Güten markiert sein kann. Im optimalen Fall enthält die Auswahlmenge von Y eine eindeutige Selektion, so daß der Wert von Y eindeutig bestimmt ist. Damit können die Auswahlmengen der abhängigen Elemente von Y automatisch neu berechnet werden. In diesem Fall folgt die Änderung der Auswahlmengen der von Y abhängigen Elemente. Ist die Auswahlmenge von Y nach Werteingabe in X leer, so deutet dies auf einen inkonsistenten Wert von X hin. Daraus folgt, daß entweder in X oder in einem Element W mit $W \xrightarrow{+} X$ ein inkonsistenter Wert vorliegt: aus den Daten, die in den Abhängigkeiten zur Berechnung der Auswahlmenge verwendet werden, konnte kein gültiger Wert ermittelt werden. Wurde in X ein Wert eingegeben, wird die Konsistenzprüfung aller vorausgesetzten Elemente ausgelöst.

Der Entwickler formuliert für jede Abhängigkeit zwei Attribute zur Klassifizierung der Berechnung:

1. Die Abhängigkeit besitzt entweder genau ein oder mehrere vorausgesetzte Elemente.
2. Die Vorbelegung ergibt grundsätzlich eine Selektion oder nicht.

Abbildung 4.1 skizziert eine Abhängigkeit. Die Elemente V_1 , V_2 und V_3 werden vom abhängigen Element A zur Berechnung mit Güte q seiner Auswahlmenge vorausgesetzt. Die Güte einer Abhängigkeit wird als Attribut seiner Kanten im Abhängigkeitsgraphen notiert. Gehört eine Kante zu mehreren Abhängigkeiten, werden alle Güten notiert.

Definition Güte einer Abhängigkeit

Die Güte einer Abhängigkeit ist ein Maß für die Auswahlmenge, die aus einer Abhängigkeit heraus berechnet werden kann.

Definition Güte eines abhängigen Elements

Die Güte eines abhängigen Elements ist ein Maß für die Auswahlmengen, die aus allen seinen Abhängigkeiten berechnet werden können.

Die Güte einer Abhängigkeit wird aus den Attributen zur Vereinfachung der Analyse wie folgt festgelegt.

Güte 1 (*,*) Die Auswahlmenge des Elements läßt sich aus dem Wert von mindestens zwei weiteren Elementen bestimmen.

Güte 2 (1,*) Die Auswahlmenge des Elements läßt sich aus dem Wert eines einzigen weiteren Elements bestimmen.

4.2 Formulierung von Abhängigkeiten zwischen Dateneingaben

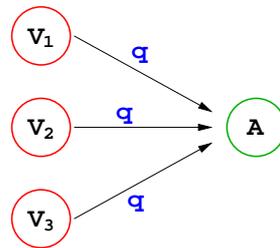


Abbildung 4.1: Beispiel einer Abhängigkeit

Güte 3 (*,1) Der Wert des Elements läßt sich aus dem Wert von mindestens zwei weiteren Elementen eindeutig bestimmen.

Güte 4 (1,1) Der Wert des Elements läßt sich aus dem Wert eines einzigen weiteren Elements eindeutig bestimmen.

Eine Metrik für die Güte eines abhängigen Elements wird in Abschnitt 4.4 entwickelt. Der Abhängigkeitsgraph bietet für jeden Knoten X folgende Mengen:

- Menge aller abhängigen Elemente (direkte Nachfolger): $Succ(X)$
- Menge aller vorausgesetzten Elemente (direkte Vorgänger): $Pred(X)$

Eine Elementmenge, die keinerlei Abhängigkeiten zu den übrigen Elementen der konfigurierbaren Benutzerschnittstelle besitzt, kann unabhängig von den verbleibenden Elementen die Konsistenz ihrer Werte prüfen oder Auswahlmengen berechnen. Das führt zur Definition der Elementgruppe.

Definition Elementgruppe, kurz Gruppe

Elementgruppen sind die Zusammenhangskomponenten des Abhängigkeitsgraphen.

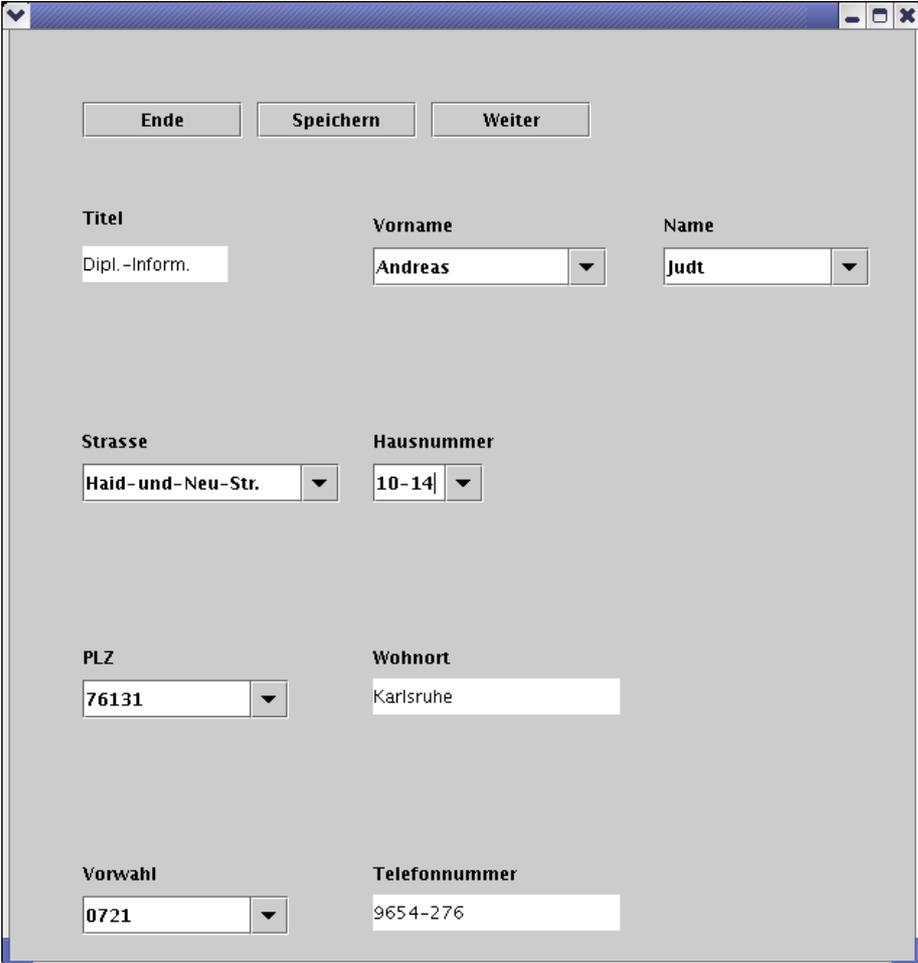
Gruppen vereinfachen den Rechenaufwand bei der Beurteilung alternativer Eingabefolgen für das Formular. Abschnitt 4.4 zeigt ein Beispiel für die Wichtigkeit der Vereinfachung. Einelementige Gruppen stellen einen Trivialfall dar und führen zu folgender Definition.

Definition triviale Gruppe

Sei E die nichtleere Menge der Elemente einer konfigurierbaren Benutzerschnittstelle. Eine einelementige Gruppe $G \subseteq E$, $G = \{1\}$ heißt triviale Gruppe.

Die Berechnung der Auswahlmenge kann abhängig von der Eingabefolge der Formularfelder über verschiedene Abhängigkeiten erfolgen. Für den Fall, daß in einem Element bereits eine Auswahlmenge existiert, können beide Mengen verschieden sein. Hat der Benutzer einen Wert eingegeben, so sollte dieser mit der neu berechneten Auswahlmenge vereinigt werden. Der Schnitt beider Mengen würde die Auswahlmöglichkeiten beschränken, wobei die Vereinigung möglicherweise zu viele Auswahlmöglichkeiten bietet. In beiden Fällen muß aber davon ausgegangen werden, daß eine Wertänderung die Neuberechnung der Auswahlmenge abhängiger Elemente oder die Änderung des Wertes verursacht.

- Schnitt der Mengen



The screenshot shows a window with a grey background. At the top, there are three buttons: "Ende", "Speichern", and "Weiter". Below these are several form fields arranged in a grid-like fashion. Each field has a label above it and a text input area with a small downward arrow on the right side, indicating a dropdown menu. The fields and their values are: "Titel" with "Dipl.-Inform.", "Vorname" with "Andreas", "Name" with "Judt", "Strasse" with "Haid-und-Neu-Str.", "Hausnummer" with "10-14", "PLZ" with "76131", "Wohnort" with "Karlsruhe", "Vorwahl" with "0721", and "Telefonnummer" with "9654-276".

Abbildung 4.2: Benutzerschnittstelle für Adresseingabe

- Vereinigung der Mengen
- Änderung der Auswahlmenge, aber Beibehaltung der Selektion.

Im Prototypen Look_{AS}^ILike wurde der Austausch der Mengen unter Beibehaltung der Selektion implementiert.

Der folgende Abschnitt zeigt die Erfassung von Adressdaten mit einer konfigurierbaren Benutzerschnittstelle.

4.3 Ein Beispiel für formularbasierte Datenerfassung: Adresseingabe

Grafische Benutzerschnittstellen zur Adresseingabe werden üblicherweise so implementiert, daß Felder der Adressdaten in der Reihenfolge eines Briefkopfs eingegeben

4.3 Ein Beispiel für formularbasierte Datenerfassung: Adresseingabe

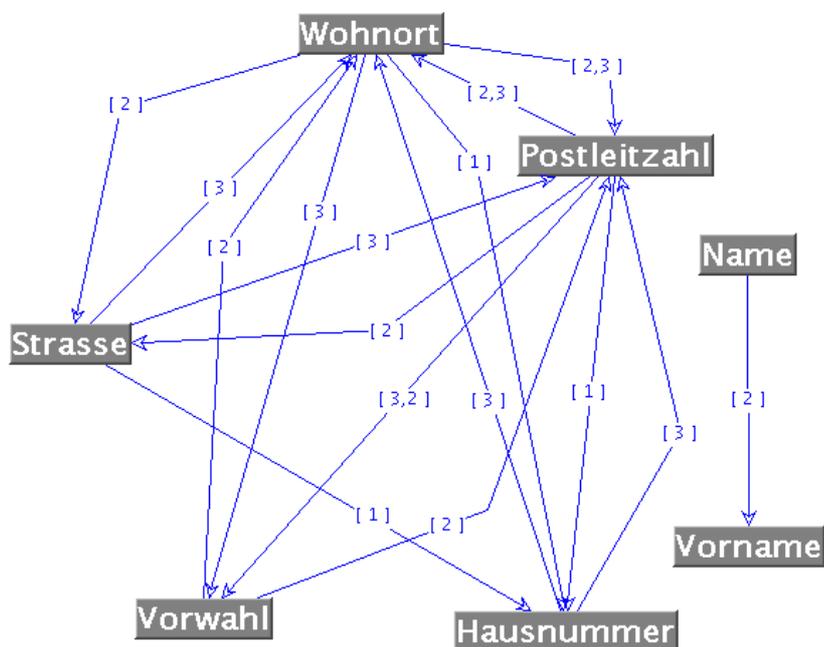


Abbildung 4.3: Abhängigkeitsgraph für Adresseingabe

werden. Die Telefonnummer wird im Anschluß eingegeben. Die Felder einer Adresse besitzen viele Abhängigkeiten, die man zur Vereinfachung der Dateneingabe verwenden kann. Aus bereits vorhandenen Daten und den Abhängigkeiten können Auswahlmengen für noch nicht eingegebene Elementwerte berechnet und die Konsistenz bereits verfügbarer Daten geprüft werden. Ein Adressverzeichnis in einer relationalen Datenbank dient hierfür als Informationsquelle. Abbildung 4.2 zeigt eine herkömmliche grafische Benutzerschnittstelle zur Adresseingabe als Formularform. Abbildung 4.3 zeigt den verwendeten Abhängigkeitsgraphen. Die Benutzerschnittstelle enthält folgende Interaktionselemente: Titel, Vorname, Name, Straße, Hausnummer, Postleitzahl, Wohnort, Vorwahl und Telefon. Die Abhängigkeiten werden wie folgt definiert.

Vorname_A Zu jedem auswählbaren Vornamen muß ein Name existieren. Aus der Eingabe des Namens läßt sich eine Liste möglicher Vornamen erzeugen.

Straße_A Aus einer Postleitzahl kann die Liste aller zugehörigen Straßennamen erzeugt werden.

Straße_B Aus dem Wohnort kann die Liste aller zugehörigen Straßennamen erzeugt werden. Hierbei ist zu beachten, daß Wohnorte mehrfach vorkommen können und somit die Straßen aller Wohnorte als Auswahl erzeugt werden müssen.

Hausnummer_A Aus Postleitzahl, Wohnort und Straße läßt sich die Liste der auswählbaren Hausnummern erzeugen. Die Postleitzahl schränkt die Liste der Hausnummern weiter ein, da Straßen auch mehrere Postleitzahlgebiete überspannen können.

4 Unterstützung des Benutzers bei der Vereinfachung von Dateneingaben

Wohnort_A Ein Postleitzahlgebiet kann mehrere Wohnorte enthalten. Daher kann aus einer angegebenen Postleitzahl eine Liste von zugehörigen Wohnorten erzeugt werden.

Wohnort_B Eine Vorwahl kann wie eine Postleitzahl mehrere Wohnorte enthalten. Daher kann aus einer angegebenen Vorwahl eine Liste von zugehörigen Wohnorten erzeugt werden.

Wohnort_C Ein Wohnort läßt sich eindeutig aus Postleitzahl, Straße und Hausnummer bestimmen.

Postleitzahl_A Eine Postleitzahl läßt sich eindeutig aus Straße, Hausnummer und Wohnort bestimmen. Da eine Straße mehrere Postleitzahlgebiete überspannen kann, wird hier die Hausnummer zur Eindeutigkeit benötigt.

Postleitzahl_B Ein Wohnort kann mehrere Postleitzahlgebiete enthalten. Aus der Angabe eines Wohnorts kann eine Liste der Postleitzahlen aller Namensduplikate erzeugt werden.

Postleitzahl_C Aus einer Vorwahl kann die Liste aller im Vorwahlbereich enthaltenen Postleitzahlgebiete erzeugt werden.

Vorwahl_A Aus einer Postleitzahl läßt sich die Liste der Vorwahlbereiche ermitteln. Man beachte, daß sich Postleitzahlgebiete und Vorwahlbereiche nicht überdecken.

Vorwahl_B Die Vorwahl läßt sich eindeutig aus Wohnort und Postleitzahl bestimmen.

Die gewählten Bezeichnungen finden in den nachfolgenden Kapiteln weitere Verwendung. Formularfelder können mehrfache Abhängigkeiten besitzen. Diese werden durch ihre Namen unterschieden. Abhängigkeiten werden als Regeln implementiert. Beispielsweise sind *Wohnort_A* und *Wohnort_B* zwei unterschiedliche Abhängigkeiten des Formularfelds *Wohnort*.

Abbildung 4.3 zeigt den Abhängigkeitsgraphen dieses Beispiels. Die Güten der Abhängigkeiten könnten durch den Programmierer detaillierter angegeben werden. Beispielsweise könnte die Güte der Abhängigkeit die Zahl seiner vorausgesetzten Elemente beinhalten. Diese Klassifizierung gibt jedoch keine Auskunft darüber, welche Güte die daraus berechnete Auswahlmenge besitzt. Die hier festgelegten Güten geben vielmehr Auskunft darüber, wieviele weitere Formularfelder für die Berechnung benötigt werden und ob eine Selektion daraus zu erwarten ist.

Die Größe der Auswahlmenge eines Interaktionselements kann üblicherweise nicht vorab bestimmt werden. Daher werden mehrelementige Auswahlmengen danach klassifiziert, ob die Werte von ein oder mehreren weiteren Elementen zur Berechnung erforderlich sind. Die in Abbildung 4.3 visualisierten Abhängigkeiten werden in der herkömmlichen Programmentwicklung üblicherweise nicht an die Implementierung der Interaktionselemente geknüpft. Häufig werden diese Beziehungen zwischen den Interaktionselementen im Entwurf nicht explizit formuliert sondern „wachsen“ im Laufe des Entwicklungsprozesses. Die Komplexität des Abhängigkeitsgraphen verdeutlicht, daß eine explizite Formulierung der Abhängigkeitsbeziehungen einen unvermeidbaren Aufwand für den Entwickler bedeuten würde.

4.4 Strukturanalyse der Benutzerschnittstelle

Grafische Benutzerschnittstellen bieten Anwendern die Möglichkeit, an verschiedensten Stellen der Benutzerschnittstelle zu interagieren. Die Interaktionsfolge der Daten kann also von der visuellen Reihenfolge vollkommen abweichen. Für konfigurierbare Benutzerschnittstellen wird die visuelle Folge der Elemente als Interaktionsfolge angenommen. Mit dem Algorithmus *YXNummerierung* in Abbildung 4.4 wird aus dem Strukturgraphen die Nummerierung der Formularfelder und Behälter berechnet¹. Sie beschreibt die Reihenfolge der angenommenen Dateneingaben. Der Start des Algorithmus erfolgt mit dem Knoten *Eingang*. Der Eingangsknoten markiert die Wurzel des Komponentenbaums und hat alle zum Programmstart angezeigten eigenständigen Behälter als Nachfolger. *Eingang* besitzt dabei keine Darstellung in der Benutzerschnittstelle².

Die angenommene Eingabefolge der Formularfelder wird zur Beurteilung der Komplexität verwendet³. Die Berechnung eines Elements X ist genau dann ausführbar, wenn seine vorausgesetzten Elemente in einem Pfad $Eingang \xrightarrow{+} X$ enthalten sind. Um die Auswahlmengen aller Elemente einer konfigurierbaren Benutzerschnittstelle berechnen zu können, müssen für jedes Element X alle vorausgesetzten Elemente im Pfad $Eingang \xrightarrow{+} X$ liegen. Ein Element kann mehrere vorausgesetzte Elemente und mehrere Abhängigkeiten besitzen. Die Betrachtung des Abhängigkeitsgraphen ohne Sichtung mehrerer zusammenhängender Kanten einer Abhängigkeit würde somit keine Aussage über ihre Berechenbarkeit liefern⁴. Der Vergleich der Pfadmengen des Abhängigkeitsgraphen umfaßt also nur einen Teil der Analyse.

Aus der Aufzählung aller Pfade [49] werden Erreichbarkeiten und die Güte der Vorbelegung betrachtet. Die optimale Struktur einer konfigurierbaren Benutzerschnittstelle berechnet die Auswahlmenge aller Elemente mit der bestmöglichen Güte. Hierfür werden alle möglichen Permutationen der Datenelemente betrachtet. Um einen Rechenaufwand zu verhindern, der mit der Fakultät der Elementzahl steigt ($O(n!)$), werden hier die Gruppen der Elemente betrachtet. Der Abhängigkeitsgraph aus Abbildung 4.3 besitzt beispielsweise folgende Gruppierung.

Gruppe 1 Titel

Gruppe 2 Vorname, Name

Gruppe 3 Straße, Hausnummer, Postleitzahl, Wohnort, Vorwahl

Gruppe 4 Telefon

Ohne Gruppierung müßten $9!=362.880$ Permutationen betrachtet werden. Gruppe 1 und Gruppe 4 sind triviale Gruppen und besitzen keine Permutationen, Gruppe 2 besitzt 2 Permutationen und Gruppe 3 besitzt $5!=120$ Permutationen. Die Menge der Permutationen aller Elemente sei mit P_0 , die der Gruppe 2 mit P_2 und die der Gruppe 3 mit P_3 bezeichnet. Durch die Gruppierung müssen $|P_2| + |P_3| = 2 + 120 = 122$ Gütebetrachtungen durchgeführt werden. Für jede Permutation $p_0 \in P_0$ läßt sich je

¹Die Klassifikation der Elemente wird in Abschnitt A.3 definiert.

²vergleichbar mit dem Dokumentknoten der Programmbibliothek SWT der Eclipse Plattform [22]

³siehe Abschnitt 2.1.2

⁴Abbildung 4.1 in Abschnitt 5.2.4 zeigt die grafische Darstellung einer Abhängigkeit.

4 Unterstützung des Benutzers bei der Vereinfachung von Dateneingaben

```
/* Annahme: ein Knoten in der graphischen Oberflaeche
   ist entweder ein Formularfeld (Datenelement) oder
   ein Behaelter (Fenstererelement), der andere Behaelter
   und Formularfelder enthalten kann)*/

int YX-Nummerierung(Start: Knoten, Startnummer: int) {

  /* nummeriert die von Start erreichbaren
     Formularfelder und Behaelter in xy-Ordnung,
     beginnend mit der Startnummer und gibt
     die hoechste zugewiesene Nummer+1 zurueck. */

  if (StartTyp == Datenelement) {

    Start.setzeNummer(Startnummer);
    return Startnummer+1;

  } else { /* Behaelter */

    N = Menge der direkten Nachfolger von Start;

    sortiere N nach yx-Koordinaten (linke obere Ecke)
    der Knoten, egal ob Behaelter oder Formularfeld;

    Knotennummer = Startnummer;

    for (int i=0; i<N.length; i++) {
      Knotennummer =
        YX-Nummerierung(N.getElement(i), Knotennummer);
    }
    return Knotennummer;
  }
}

Aufruf: Knotennummer = YX-Nummerierung(Eingang,0);
```

Abbildung 4.4: Algoritimus zur Berechnung der Eingabefolge

eine Permutation aus $p_2 \in P_2$ und $p_3 \in P_3$ finden, deren Elementreihenfolge der Folge der Elemente in p_2 und p_3 entspricht. Die Gütebetrachtungen der Permutationen aller Elemente lassen sich damit aus den Güten der 122 Permutationen berechnen. In diesem Beispiel wurden also 362.758 Gütebetrachtungen eingespart, also 99,96%. Allgemein gilt, daß möglichst viele kleine Gruppen einen vertretbaren Rechenaufwand erzeugen. Kapitel 5 erläutert, daß die Güteanalyse erfahrungsgemäß sehr schnell und ohne spürbare Wartezeiten für den Benutzer ausgeführt werden kann.

Eine Abhängigkeit wird als berechenbar angesehen, wenn alle vorausgesetzten Elemente einen Wert besitzen. Für die Beurteilung der Gesamtqualität einer Elementreihenfolge werden folgende Meßgrößen für jedes Element definiert:

4.4 Strukturanalyse der Benutzerschnittstelle

D = Anzahl der berechenbaren Abhängigkeiten

G = Summe der Güten der Abhängigkeiten

P = Anzahl der berechenbaren abhängigen Elemente

J = Beurteilung des Elements

Für Tripel $V_e = (D, G, P)$ eines Elements e wird folgende Metrik [7] definiert: Seien $V_i, V_j \in \mathcal{N}^3$ mit

$$\begin{aligned}V_i &= (D_i, G_i, P_i) \\V_j &= (D_j, G_j, P_j)\end{aligned}$$

Die Metrik ρ sei wie folgt definiert:

$$\rho(V_i, V_j) = k_D(D_j - D_i)^2 + k_G(G_j - G_i)^2 + k_P(P_j - P_i)^2$$

Die Konstanten k_D , k_G und k_P definieren Faktoren zur Gewichtung der jeweiligen Meßgrößen. Für den Prototypen wurden folgende Konstanten gewählt:

$$\begin{aligned}k_D &= 1 \\k_G &= 1 \\k_P &= 2\end{aligned}$$

Die Idee bei der Wahl der Konstanten war, daß die Eingabefolge umso besser ist, je mehr Abhängigkeiten durch die Eingabe eines Formularfelds berechenbar werden. Daher wurde k_P größer gewählt. Die Beurteilung J_e des Elements e berechnet sich als Abstand des Meßgrößentripels vom Nullpunkt des Raumes:

$$\begin{aligned}J_e &= \rho(0, V_e) \\&= k_D \cdot D^2 + k_G \cdot G^2 + k_P \cdot P^2\end{aligned}$$

Sei $E = \{e_0, e_1, \dots, e_{n-1}\}$ eine Elementmenge. Die Beurteilung J_E der Elementmenge E berechnet sich als Summe der Elementbeurteilungen:

$$\begin{aligned}J_E &= \sum_{i=0}^{n-1} \rho(0, e_i) \\&= \sum_{i=0}^{n-1} J_{e_i}\end{aligned}$$

Sei $T = \{t\}$ eine triviale Gruppe. Für die Beurteilung J_T der einelementigen Menge T gilt:

$$\begin{aligned}J_T &= J_t \\&= k_D \cdot 0 + k_G \cdot 0 + k_P \cdot 0 \\&= 0\end{aligned}$$

Die Güte einer trivialen Gruppe ist somit Null. Das bedeutet, daß die Position dieses Elements im Strukturgraphen bei der Gütebetrachtung keine Rolle spielt, da keine abhängigen Dateneingaben existieren. Um die Güte einer Benutzerschnittstelle zu

4 Unterstützung des Benutzers bei der Vereinfachung von Dateneingaben

bestimmen, werden die Permutationen der Gruppen summiert, die im Kontrollflußgraphen enthalten sind. Ist dieser zyklisch, können für jede Gruppe mehrere Permutationen in die Summe einfließen.

Gelingt die Unterteilung in *kleine* Gruppen nicht, ist ein hoher Zeitaufwand zur Beurteilung der möglichen Permutationen zu erwarten. Bei einer Gruppe von 10 Elementen wären $10! = 3.628.800$ Tests erforderlich, wobei abhängig von der verwendeten Metrik durchaus Vereinfachungen möglich sind. Die Analyse der Permutationen kann der Benutzerschnittstelle als Information zur Verfügung gestellt werden, jedoch müssen weiterhin große Datenmengen zur Laufzeit des Programms durchsucht werden. Selbst bei Verwendung effizienter Suchalgorithmen [29] können große Rechenaufwände entstehen. Abschnitt 7.1.2 und Anhang A.9 erläutern einen Lösungsvorschlag. Die hier vorgestellte Metrik zur Berechnung der Güte orientiert sich an der Zahl der berechenbaren Abhängigkeiten. Eine konkrete Implementierung sollte in der Lage sein, alternative Metriken zu verwenden, die andere Schwerpunkte der Beurteilung setzen.

Die Beurteilung aller Permutationen kann nun dafür verwendet werden, einerseits die Struktur zu bewerten und andererseits als Basis einer interaktiven Hilfe bei der Vereinfachung von Dateneingaben dienen. Die Anpassung der Benutzerschnittstelle an die Anwendungssituation benötigt darüberhinaus die Analyse des Interaktionsverhaltens. Mit den gesammelten Daten wird der Anwender aus Sicht der konfigurierbaren Benutzerschnittstelle modelliert. Man spricht hier von einem Benutzermodell [3]. Das Benutzermodell liefert die Analyse des Eingabeverhaltens als Reihenfolge von Elementen – also einen Pfad des Kontrollflußgraphen. Graphen, die diesen Pfad enthalten, verkörpern Strukturen, die sich potenziell für das Interaktionsverhalten des Anwenders eignen.

Die konkrete Unterstützung des Anwenders sollte nur minimalen Aufwand bedeuten und einfach bedienbar sein. Üblicherweise lehnen Benutzer Aufwände ab, die nicht offensichtlich der Erreichung ihrer Ziele dienen. Ein Verfahren darf dabei aber nicht autoritär eigene Vorstellungen durchsetzen. Die Ziele des Benutzers müssen selbst dann umsetzbar sein, wenn die Beurteilung ergibt, daß andere Strukturen objektiv geeigneter wären [60].

Abschnitt A.9 des Anhangs A erläutert eine beispielhafte Vorgehensweise bei der Unterstützung des Anwenders.

4.5 Zusammenfassung

Die Unterstützung des Anwenders basiert auf einem Vergleich zwischen Strukturanalyse und Berechnungen aus dem Benutzermodell. Die Betrachtung der Eingabefolgen in Bezug auf die Erfüllung von Abhängigkeiten zwischen den Formularfeldern bietet die Grundlage für die Berechnung alternativer Formularstrukturen. Strukturen werden in Gruppen unterteilt und ihre Güte gemessen. Aus dem Benutzermodell und alternativen, einfacheren Dateneingaben schlägt die Benutzerschnittstelle Anpassungen vor, die der Anwender im laufenden Betrieb vornehmen kann.

Die Strukturanalyse erzeugt mit steigender Gruppengröße sehr hohe Rechenauf-

4.5 Zusammenfassung

wände. Die Praxis zeigt jedoch, daß Gruppen mit mehr als 7 Elementen in der Regel nicht sinnvoll vereinfacht werden können. Der Prototyp reduziert den Aufwand dadurch, daß die Strukturanalyse erst nach mehreren Interaktionen (Reduzierung von Freiheitsgraden der Gruppe) gestartet wird.

Die Abschnitte 5.1.2, 5.1.3 und Anhang A.1 erläutern den Entwurf des Abhängigkeitsgraphen, des Kontrollflußgraphen und der Berechnung.

4 Unterstützung des Benutzers bei der Vereinfachung von Dateneingaben

Kapitel 5

Benutzergesteuerte Modifikation von Komponentenstrukturen

Für die Konfigurierbarkeit der Benutzerschnittstelle müssen während der Programmierung verschiedene Vorarbeiten geleistet werden. Neben der Definition der Komponentenstruktur werden insbesondere alle bekannten Abhängigkeiten zwischen Elementen formuliert. Abbildung 5.1 veranschaulicht das Prinzip der Programmierung. Einzelne Beziehungen zwischen den Formularfeldern werden als Abhängigkeiten formuliert. Aus ihnen wird der resultierende Abhängigkeitsgraph erzeugt und seine

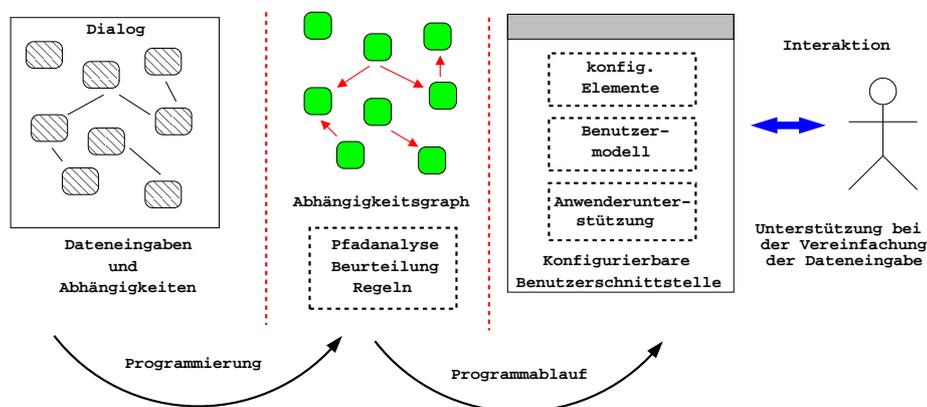


Abbildung 5.1: Prinzip der Entwicklung einer konfigurierbaren Benutzerschnittstelle

Pfade analysiert. Die Pfade beschreiben Reihenfolgen von Elementen, die Ketten von Berechnungen zur Auswahlmengen ermöglichen. Ein einziger Pfad, der alle Elemente enthält, wäre beispielsweise ein optimaler Abhängigkeitsgraph, wenn eine Abhängigkeit mit einem eindeutigen Wert zwischen jedem benachbarten Knotenpaar $A \rightarrow B$ existiert. Bis auf die erste Eingabe könnten so alle Formularfelder automatisch ausgefüllt werden.

Anhang A beschreibt Details der Implementierung und des Klassenmodells konfigurierbarer Benutzerschnittstellen. Der Prototyp Look_{AS}^ILike wurde aus dem Klassenmodell für Java und Swing [51] spezialisiert.

5.1 Ansatz zur technologieneutralen Modellierung

Die Implementierung der konfigurierbaren Benutzerschnittstelle umfaßt drei Datenstrukturen:

- die Menge der konfigurierbaren Elemente,
- den Abhängigkeitsgraphen und
- den Kontrollflußgraphen.

Diese werden aus einer technologieneutralen Document Type Definition (DTD) erzeugt. Das entstandene Klassenmodell wird dann für eine konkrete Technologie durch Bildung von Unterklassen spezialisiert.

5.1.1 Modellierung der Komponentenstruktur

Konfigurierbare Elemente erweitern die Interaktionselemente um ein Datenmodell, Methoden zur Berechnung der Auswahlmenge der Konsistenzprüfung und der Verwaltung von Abhängigkeitsbeziehungen zu weiteren Elementen der Benutzerschnittstelle. Die Abhängigkeitsbeziehung zwischen den Elementen werden im Programmwurf aus dem Prozeßwissen gewonnen und formuliert. Der Abhängigkeitsgraph wird dann aus der Parametrisierung der Berechnung automatisch erzeugt. Die Berechnung ist Bestandteil des Elements und wird vom Programmierer implementiert. Die Größe der Auswahlmenge kann in der Entwicklung nur selten vorausgesagt werden. Aus diesem Grund erlauben konfigurierbare Elemente die Einbettung mehrerer Gestalten, die für verschieden große Datenmengen angepaßt werden können.

Abbildung 5.2 veranschaulicht die Struktur eines konfigurierbaren Elements. Diese werden technologieneutral aus ihrer XML Darstellung [9] entwickelt. Abbildung 5.3 zeigt die korrespondierende Document Type Definition (DTD). Eine Elementliste (*elementlist*) besitzt mindestens ein Element (*element*). Dieses erhält eine Verhaltensbeschreibung (*behaviour*), eine Konfiguration (*configuration*), eine Berechnung (*calculator*) und eine Strukturdefinition (*structure*). Die Verhaltensbeschreibung und die Konfiguration sind optional, Berechnung und Strukturdefinition sind obligatorisch. Einem Element wird ein logischer Name (*name*) und ein Elementtyp (*type*) zugeordnet. Elementtypen werden als Eingang (*entry*), Ausgang (*exit*), Fenster (*panel*), Daten (*data*), Marke (*label*) und Aktion (*action*) kategorisiert. Eingang und Ausgang stellen Anfangs- und Endpunkt des Kontrollflusses dar. Der Eingang stellt das Wurzelement der Strukturbeschreibung dar und besitzt eine nichtleere Menge an Fensterelementen als Nachfolger. Fensterelemente besitzen Fensterelemente, Datenelemente, Markenelemente oder Aktionselemente als Nachfolger. Datenelemente und Markenelemente besitzen keine Nachfolger. Aktionselemente besitzen Fensterelemente oder den Ausgang als Nachfolger. Das Ausgangselement exitisiert wie das Eingangselement genau einmal und beendet die konfigurierbare Benutzerschnittstelle. Markenelemente enthalten beschreibende Texte oder andere grafische Primitive (z.B. Linien, Rahmen,

5.1 Ansatz zur technologieneutralen Modellierung

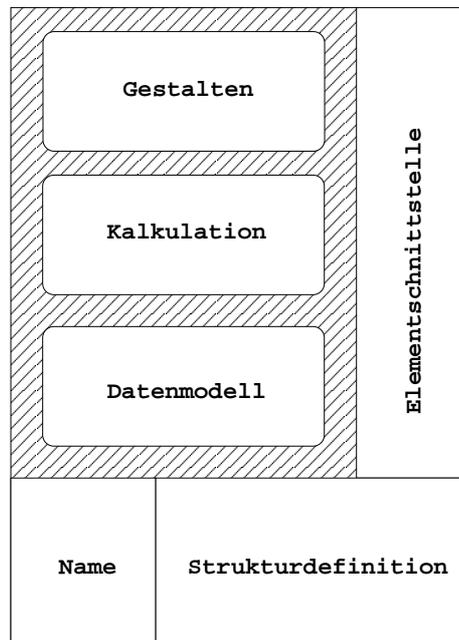


Abbildung 5.2: Struktur eines konfigurierbaren Elements

Bilder, etc.) und sind nicht interaktionsfähig. Datenelemente verkörpern die Interaktionselemente der konfigurierbaren Benutzerschnittstelle. Fensterelemente besitzen ein Verhalten (*behaviour*) im unterliegenden Fenstersystem. Dieses definiert, ob ein Fenster parallel, modal oder ersetzend geöffnet wird. Für jedes Fensterelement wird definiert, ob es direkter Nachfolger eines Fensterelements oder eigenständiges Fenster (*standalone*) ist. Fensterelemente, Datenelemente, Markenelemente und Aktionselemente besitzen eine Konfiguration, die eine initiale Auswahlmenge, die Position im verknüpften Fensterelement, alle Gestalten, einen Hinweistext für den Benutzer und einen Kommentar definiert. Die Konfiguration besitzt weiterhin Einstellungen des Datenmodells. Dieses definiert (*fixeditems*), ob Wert und Auswahlmenge geändert werden dürfen. Weiterhin werden das Datenformat (*format*), die Akzeptanz von Neueingaben (*inputable*) und Mehrfachselektierbarkeit (*multiselect*) festgelegt. Das Attribut *enabled* aktiviert bzw. deaktiviert die Interaktion des Elements. Das Aussehen eines deaktivierten Elements hängt von der jeweiligen Implementierung der Gestalten ab. Die initiale Auswahlmenge enthält eine Liste (*item*) möglicher Werte. Das Attribut *value* definiert den Wert, *default* definiert, ob der Wert initial selektiert wird. Die Position des Elements (*position*) wird als (x|y)-Koordinatenpaar (*xpos*, *ypos*) gespeichert.

Die Gestalten (*style*) werden technologiespezifisch als Rohdaten (*#PCDATA* [9]) gespeichert. Für konfigurierbare Benutzerschnittstellen wird angenommen, daß der Zustand einer beliebigen Gestalt der unterliegenden komponentenorientierten Programmbibliothek in dieser Form gespeichert werden kann. Komponentenzustände können vollständig serialisiert und zu einem beliebigen Zeitpunkt wiederhergestellt werden. Diese Voraussetzung wird von modernen komponentenorientierten Imple-

5 Benutzergesteuerte Modifikation von Komponentenstrukturen

```
<!-- DTD for configurable elements -->

<!ELEMENT elementlist (lelement+) >
<!ELEMENT lelement (behaviour?, configuration?,
calculator, structure) >
<!ATTLIST lelement name CDATA #REQUIRED>
<!ATTLIST lelement type (entry|exit|data|panel|
action|label) #REQUIRED>
<!ELEMENT behaviour EMPTY>
<!ATTLIST behaviour standalone (yes|no) #IMPLIED>
<!ATTLIST behaviour openingmode (parallel|modal|
replace) #IMPLIED>
<!ELEMENT configuration
(initial, position, style+, caption?, comment?)>
<!ATTLIST configuration format (text|number) #IMPLIED>
<!ATTLIST configuration multiselect (yes|no) #IMPLIED>
<!ATTLIST configuration inputable (yes|no) #IMPLIED>
<!ATTLIST configuration enabled (yes|no) #IMPLIED>
<!ATTLIST configuration fixeditems (yes|no) #IMPLIED>
<!ELEMENT initial (item*)>
<!ELEMENT item EMPTY>
<!ATTLIST item value CDATA #REQUIRED>
<!ATTLIST item default (yes|no) #IMPLIED>
<!ELEMENT position EMPTY>
<!ATTLIST position xpos CDATA #REQUIRED>
<!ATTLIST position ypos CDATA #REQUIRED>
<!ELEMENT style (#PCDATA)>
<!ATTLIST style width CDATA #REQUIRED>
<!ATTLIST style height CDATA #REQUIRED>
<!ATTLIST style type (large|medium|single) #REQUIRED>
<!ATTLIST style minimumitemnumber CDATA #IMPLIED>
<!ATTLIST style default (yes|no) #IMPLIED>
<!ELEMENT calculator (#PCDATA)>
<!ELEMENT caption (#PCDATA)>
<!ELEMENT comment (#PCDATA)>
<!ELEMENT structure (successor*)>
<!ELEMENT successor EMPTY>
<!ATTLIST successor name CDATA #REQUIRED>
```

Abbildung 5.3: Document Type Definition (DTD) der Komponentenstruktur

mentierungen erfüllt, z.B. Microsoft .NET [11], SWT [22] oder Swing [51]. Beim Erzeugen werden Gestalten typbedingt mit Verhaltensbeschreibung, Konfiguration und Position versehen. Für jede Gestalt wird die Größe (*width*, *height*) festgelegt. Ein Typ (*type*) definiert maximal¹ drei Gestalten, die einen einzelnen Wert (*single*), eine kleine (*medium*) oder eine große Auswahlmenge (*large*) darstellen. Für jede Gestalt kann eine Mindestgröße der Auswahlmenge definiert werden, der als Schwellwert

¹willkürlich o.B.d.A. festgelegt

5.1 Ansatz zur technologieneutralen Modellierung

für den Austausch der Darstellung fungiert, wobei eine Gestalt als Standard (*default*) definiert werden kann.

Die Einstellungen der Gestalt (*style*) werden auf die konkrete grafische Komponente abgebildet. Die Interaktion zwischen Gestalt, Datenmodell und Element wird durch die Erweiterung der Komponenten um die Schnittstelle *DataModelControl* erreicht.

Ein konfigurierbares Element besitzt eine Beschreibung (*caption*) und ein frei verwendbares Kommentarfeld (*comment*). Die Beschreibung kann beispielsweise als Tooltip des Elements benutzt werden. Die Eingliederung des Elements in die Struktur der konfigurierbaren Benutzerschnittstelle wird für jedes Element in seiner Strukturbeschreibung (*structure*) vermerkt. Diese enthält eine Liste von Nachfolgern (*successor*), welche ihrerseits aus logischen Elementnamen besteht. Aus den Strukturbeschreibungen der Elemente wird der Strukturgraph der konfigurierbaren Benutzerschnittstelle erzeugt. Er ist zusammenhängend, zyklisch und gerichtet.

Abbildung 5.4 zeigt das Modell der entwickelten Komponentenstruktur. Es besitzt einen technologieneutralen Teil und eine technologieabhängige Erweiterung für eine konkrete Programmbibliothek grafischer Benutzerschnittstellen. Die Bezeichnungen der Klassen entsprechen den XML Elementen und der oben beschriebenen Funktion. Das Klassenmodell wurde aus der Document Type Definition (DTD) (vgl. Abbildung 5.3) erzeugt. Hilfsklassen wurden im Modell vernachlässigt. Technologieabhängige Erweiterungen wurden mit dem Präfix „*Ext*“ im Klassennamen gekennzeichnet. Zusätzlich wird eine Klasse benötigt, die folgende Aufgaben übernimmt:

- Laden und Speichern der konfigurierbaren Benutzerschnittstelle,
- Verwalten einer zentralen Registratur der Elemente,
- Verwaltung des Abhängigkeitsgraphen,
- Steuern der Strukturanalyse und
- Weiterleiten von Ereignissen an das Benutzermodell.

Diese Klasse *LookAsLikeDoc* implementiert diese Funktionen und verkörpert den Ursprung der konfigurierbaren Benutzerschnittstelle.

Alternativ kann die konfigurierbare Benutzerschnittstelle als UML-Modell [44] entwickelt werden. Der Vorteil der Verwendung von XML [9] besteht darin, daß Elemente mit einer DTD oder einer Schema-Datei leicht definierbar sind. Die Speicherung der konfigurierbaren Benutzerschnittstelle erfolgt in XML, wobei die Konsistenz der Daten mittels DTD oder Schema leicht prüfbar sind. Die Gestalten der Elemente werden dabei als serialisierte Objekte in die Elementstruktur eingebettet.

5.1.2 Die Modellierung des Abhängigkeitsgraphen

Für die Modellierung der Abhängigkeiten kann eine Document Type Definition (DTD) [9] angegeben werden. Sie definiert die Knoten- und Kanteninformationen, aus denen der Abhängigkeitsgraph der Elemente erzeugt wird. Abbildung 5.5 zeigt die DTD des Abhängigkeitsgraphen. Aus der Document Type Definition kann für eine konkrete Im-

5 Benutzergesteuerte Modifikation von Komponentenstrukturen

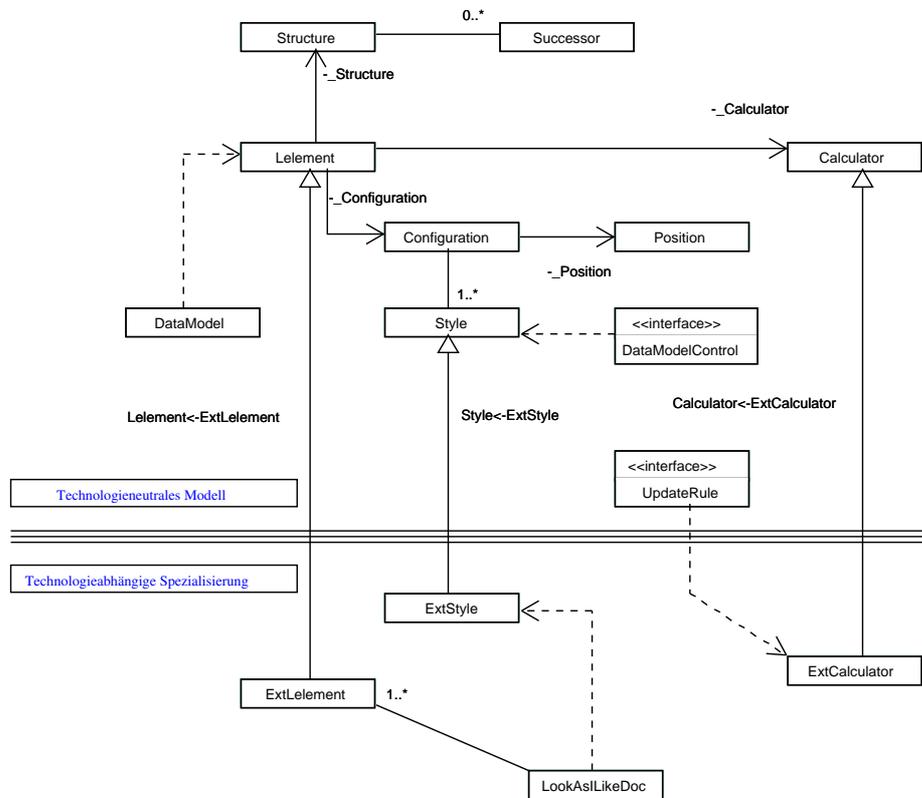


Abbildung 5.4: Klassenmodell der Komponentenstruktur

```

<!-- DTD for the dependencies graph -->
<!ELEMENT dependencieslist (dependency+) >
<!ELEMENT dependency (node, predecessor+)>
<!ATTLIST dependency priority (1|2|3|4) #REQUIRED>
<!ATTLIST dependency name CDATA #REQUIRED>
<!ELEMENT node (#PCDATA)>
<!ELEMENT predecessor (#PCDATA)>

```

Abbildung 5.5: Document Type Definition (DTD) des Abhängigkeitsgraphen

plementierung ein korrespondierendes Klassenmodell erzeugt werden². Abbildung 5.6 zeigt ein erzeugtes Klassenmodell. Abhängigkeiten werden in *dependencieslist* aufgelistet. Eine Abhängigkeitsbeziehung wird im XML Element *dependency* gespeichert. Sie wird einem Element *node* zugeordnet und besitzt eine Menge vorausgesetzter Elemente, die im Graphen als Vorgänger *predecessor* notiert werden. Die Güte der Vorbelegung wird als Kantengewicht *priority* notiert. Jede Kante eines vorausgesetzten Elements zu einem abhängigen Element wird gewichtet. Bei der Generierung wur-

²Für den Prototypen wurde Java XML Binding API [35] verwendet.

5.1 Ansatz zur technologieneutralen Modellierung

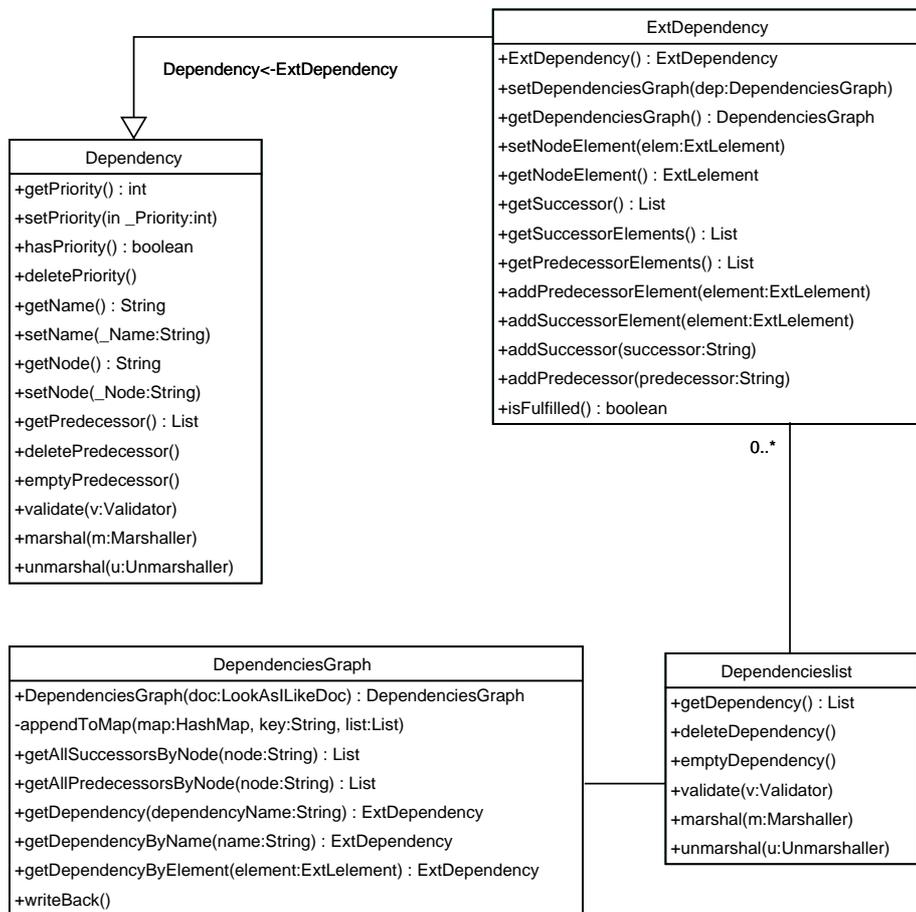


Abbildung 5.6: Klassenmodell des Abhängigkeitsgraphen

die *priority* als Datentyp *int* erzeugt. Die DTD schränkt die Güteangabe auf die Werte $\{1, 2, 3, 4\}$ ein. *ExtDependency* erweitert *Dependency* um die Nachfolgerbeziehung, die aus den Abhängigkeiten beim Aufbau des Graphen berechnet wird. Zusätzlich wird eine Referenz auf die konfigurierbaren Elemente (*ExtLelement*) des Knotens, seiner Vorgänger und seiner Nachfolger gespeichert. Sie wird zur Prüfung erfüllter Abhängigkeiten benötigt. Die Klasse *DependenciesGraph* repräsentiert den Abhängigkeitsgraphen. Sie verkörpert gleichzeitig die Registratur für die Abhängigkeiten. Für jedes Element kann somit geprüft werden, welche beste Güte der Abhängigkeiten zu einem beliebigen Zeitpunkt berechenbar ist.

5.1.3 Die Modellierung des Kontrollflußgraphen

Der Kontrollflußgraph der konfigurierbaren Benutzerschnittstelle beschreibt die Varianten der Navigation. Er wird aus der Strukturdefinition berechnet und dient dazu, alle im Abhängigkeitsgraphen relevanten Pfade zur Beurteilung der Benutzerschnittstelle zu extrahieren. Der Kontrollflußgraph beschreibt das angenommene Vorgehen beim Ausfüllen des Formulars und damit die Bedienung der Benutzerschnittstelle.

5 Benutzergesteuerte Modifikation von Komponentenstrukturen

Weiterhin begrenzt seine Pfadmenge die Komplexität bei der Beurteilung der Struktur der konfigurierbaren Benutzerschnittstelle.

Die Datenstruktur des Kontrollflußgraphen wird ähnlich zu der des Abhängigkeitsgraphen aufgebaut, um für die Beurteilung der Struktur mehrere Pfadmengen vergleichen zu können. Der wesentliche Unterschied besteht darin, daß Abhängigkeiten eine Vorgängerbeziehung und die Elementstruktur eine Nachfolgerbeziehung formuliert. Abbildung 5.7 zeigt die Document Type Defintion (DTD) [9] des Kontrollflußgraphen, Abbildung 5.8 das erzeugte Klassenmodell. Analog erweitert *ExtControlflow* die Klas-

```
<!-- DTD for the controlflow graph -->

<!ELEMENT controlflowlist (controlflow+)>
<!ELEMENT controlflow (node, successor+, predecessor+)>
<!ATTLIST controlflow priority (0|1|2|3|4|5|6|7|8|9) #REQUIRED>
<!ATTLIST controlflow preemption (0|1|2|3) #REQUIRED>
<!ELEMENT node (#PCDATA)>
<!ELEMENT successor (#PCDATA)>
<!ELEMENT predecessor (#PCDATA)>
```

Abbildung 5.7: Document Type Definition (DTD) des Kontrollflußgraphen

se *Controlflow* um die Berechnung der Vorgängerbeziehung. Ein Kontrollflußobjekt erhält zwei Gütesaussagen: eine Schätzung für den Interaktionsaufwand (*priority*) und die Güte der besten Vorbelegung unter Annahme eines optimistischen Kontrollflusses. Die Klasse *ControlFlowGraph* übernimmt nicht nur die Funktion einer Registratur, sondern speichert die Güte aller Kanten. *optimumPriorityByNode* liefert eine Aussage über die Erreichbarkeit innerhalb des Kontrollflusses.

5.2 Vorgehensweise bei der Programmierung

Die Programmierung einer konfigurierbaren Benutzerschnittstelle erfolgt in diesen nachfolgend beschriebenen Schritten:

1. Festlegen der Komponenten
2. Einfügen der Gestalten
3. Verknüpfung der Komponenten
4. Einfügen von Abhängigkeits-Beziehungen
5. Registrieren externer Hörer
6. Speichern der Benutzerschnittstelle in einer Konfigurationsdatei

Die vom Programmierer so erzeugte Benutzerschnittstelle kann vom Anwender aus der Konfigurationsdatei geladen, modifiziert und gespeichert werden.

5.2 Vorgehensweise bei der Programmierung

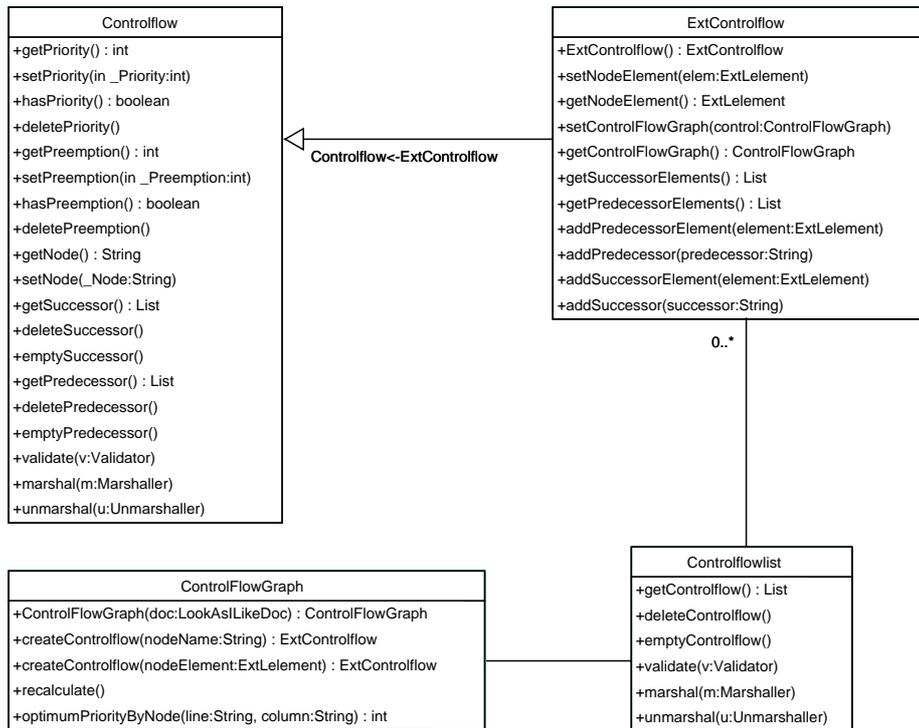


Abbildung 5.8: Klassenmodell des Kontrollflußgraphen

5.2.1 Festlegen der Komponenten

Für den Entwurf der konfigurierbaren Benutzerschnittstelle muß zunächst eine Menge konfigurierbarer Elemente implementiert werden. Diese enthalten die in Abbildung 5.9 skizzierten Bestandteile. Details der Modellierung können in Anhang A.4 nachgelesen werden. Die benötigten Komponenten der Benutzerschnittstelle werden analog zur Programmierung mit der unterliegenden Technologie ausgewählt. Das Konzept dieser Arbeit unterscheidet anzeigbare konfigurierbare Elemente in

- Fensterelement,
- Markenelement,
- Datenelement und
- Aktionselement.

Fensterelemente können selbständig, modal oder eingebettet sein. Markenelemente visualisieren passive Komponenten, z.B. Bilder, Linien oder Rahmen. Datenelemente dienen der Eingabe von Daten. Es enthält ein Datenmodell, das die Auswahlmenge und den Auswahlzustand des Elements speichert. Zwischen den Datenelementen werden später die aus dem Prozeßwissen bekannten Abhängigkeiten definiert. Aktionselemente stellen Komponenten dar, die eine Reaktion auslösen. Aktionselemente können beispielsweise Knöpfe oder Ikone darstellen.

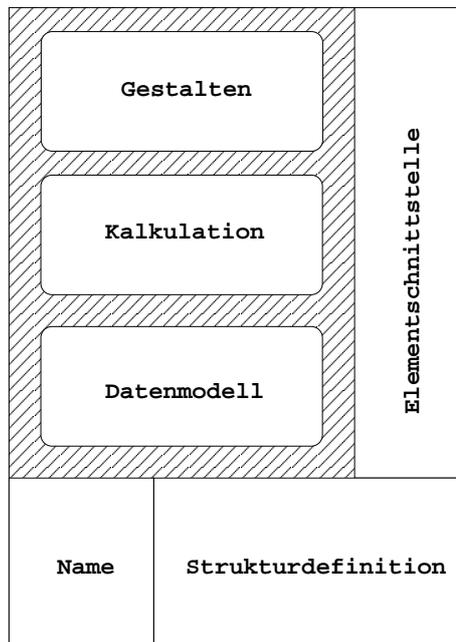


Abbildung 5.9: Struktur eines konfigurierbaren Elements

5.2.2 Einfügen der Gestalten

Gestalten definieren das visuelle Erscheinen der Elemente. Fensterelemente, Markenelemente und Aktionselemente erhalten eine einzige Gestalt, während Datenelemente mehrere Gestalten verwenden können. Da die Größe der Auswahlmenge eines Datenelements nicht vorab bestimmbar ist, kann die konfigurierbare Benutzerschnittstelle zwischen mehreren Gestalten – je nach Größe der Auswahlmenge – automatisch wechseln.

Komponenten der unterliegenden Programmbibliothek müssen um eine Schnittstelle für den Zugriff des Datenmodells erweitert werden. Dies erfolgt durch Implementierung einer Unterklasse. Diese Unterklassen werden als Kontrollkomponenten bezeichnet³. Das Datenmodell eines Elements speichert Wertänderungen nach Interaktion des Benutzers und löst die Benachrichtigung abhängiger Elemente aus.

5.2.3 Verknüpfung der Komponenten

Die Komponentenstruktur der konfigurierbaren Benutzerschnittstelle wird äquivalent zur Komponentenstruktur der unterliegenden Technologie definiert. Zusätzlich werden zwei Knoten *Eingang* und *Ausgang* eingefügt. Die Nachfolger des Knotens *Eingang* werden beim Start der Benutzerschnittstelle angezeigt. Der Knoten *Ausgang* definiert das Schließen der Benutzerschnittstelle. Alle Aktionen, die das Schließen der Benutzerschnittstelle nach sich ziehen, erhalten diesen Knoten als Kind.

³Für den Prototypen wurden etwa 20 Kontrollkomponenten implementiert.

5.3 Bedienung der konfigurierbaren Benutzerschnittstelle

5.2.4 Einfügen von Abhängigkeits-Beziehungen

Die Abhängigkeitsbeziehungen zwischen Datenelementen enthalten folgende Informationen:

1. Name des abhängigen Elements,
2. Liste der vorausgesetzten Elemente und
3. Güte der Vorbelegung.

Diese lassen sich aus dem Prozeßwissen leicht formulieren. Insbesondere trägt eine einfache Klassifizierung der Güte in 4 Fälle zur Anwendbarkeit des Konzepts bei (vgl. Abschnitt 4.3). Alle Abhängigkeiten werden automatisch durch Analyse der Berechnung erzeugt. Die Berechnung besteht aus mehreren Komponenten – den Regeln.

Eine Regel beschreibt eine Abhängigkeit und implementiert ein Methodenpaar zur Berechnung der Auswahlmenge und der Konsistenzprüfung. Die konfigurierbare Benutzerschnittstelle stellt hierbei die Rahmenimplementierung der Berechnung im Datenelement zur Verfügung. Der Entwickler implementiert die Regeln aus den ihm bekannten Abhängigkeiten. Jede Regel wird der Berechnung des korrespondierenden abhängigen Elements hinzugefügt – in diesem Beispiel Element A. Abbildung A.12 des Anhangs A.6 zeigt ein Beispielprogramm.

5.2.5 Registrieren externer Hörer

Hörer koppeln Interaktionselemente mit der Geschäftslogik. Der Entwickler registriert Hörer der abstrakten Interaktionsebene – z.B. „Aktion ausgelöst“, „Daten eingegeben“, oder „Wert geändert“ – bei Fenster-, Aktions- und Datenelementen. Die Elemente bilden diese auf technologiespezifische Hörer der Gestalten ab.

5.2.6 Speichern der Benutzerschnittstelle

Die Komponentenstruktur wird zum Abschluß in eine Konfigurationsdatei gespeichert. Die Abbildungen A.10, A.11, und A.12 des Anhangs A.6 zeigen eine vollständige Beispielimplementierung.

5.3 Bedienung der konfigurierbaren Benutzerschnittstelle

Anwender erzeugen ihre konfigurierbare Benutzerschnittstelle aus der von Entwickler gespeicherten Konfigurationsdatei. Beim Starten des Programms werden die Beobachtung des Benutzers, der Abhängigkeitsgraph, Kontrollflußgraph, Strukturanalyse und das Werkzeug zur Anwenderunterstützung automatisch gestartet. Der Anwender kann die Struktur aus eigener Initiative oder aufgrund von Änderungsvorschlägen der Unterstützung selbständig modifizieren und speichern⁴.

Anhang A.9 erläutert beispielhaft die Unterstützung des Anwenders durch den

⁴Der Prototyp bietet hierfür Pop-Up Menüs und Drag-and-Drop.

Prototypen. In der Evaluierung in Kapitel 6 werden verschiedene Situationen dargestellt, in denen eine konfigurierbare Benutzerschnittstelle Vereinfachungen vorschlägt.

5.4 Zusammenfassung

Die Programmierung konfigurierbarer Benutzerschnittstellen erfolgt ähnlich zur Programmierung mit der unterliegenden Programmbibliothek grafischer Benutzerschnittstellen. Der wesentliche Vorteil dieses Konzepts liegt in der Entkoppelung und Strukturierung der Reaktion auf abhängige Dateneingaben. Diese wird üblicherweise unstrukturiert in die Geschäftslogik integriert und ist später kaum mehr zu überblicken. Konfigurierbare Benutzerschnittstellen betten diese Implementierungen als Regeln in die betreffenden abhängigen Elemente ein. Ein weiterer Vorteil besteht darin, daß die Berechnungen die Transitivität der Abhängigkeiten nutzen, um möglichst viele Auswahlmengen nach der Interaktion des Anwenders zu berechnen. Die Erfahrung zeigt, daß auch bei Abhängigkeiten, die als nicht eindeutig definiert wurden, häufig eindeutige Auswahlmengen berechnet werden können. Diese Verkettung von Berechnungen trägt zusätzlich zur Vereinfachung der Dateneingaben bei und muß vom Entwickler nicht implementiert werden.

Kapitel 6

Evaluierung

Die Evaluierung von Benutzerschnittstellen verkörpert einen schwierigen Aspekt benutzerzentrierter Entwicklung. Konfigurierbare Benutzerschnittstellen erschweren die Evaluierung zusätzlich durch die Tatsache, daß Vereinfachungsmöglichkeiten von Dateneingaben bereits optimierter Strukturen im Prozeßwissen des Anwenders verankert sind. Für den Nachweis der aufgestellten Thesen dieser Arbeit müssen folgende Aussagen überprüft werden:

These 1 Anwender können Dateneingaben mit ihrem Prozeßwissen selbständig und ohne Programmieraufwand vereinfachen. Damit kann eine Effizienzsteigerung der Benutzerschnittstelle erzielt werden.

These 2 Durch Beobachtung der Benutzerinteraktion kann der Anwender bei der Vereinfachung von Dateneingaben unterstützt werden.

These 3 Komponenten formularbasierter Benutzerschnittstellen können mit einem einheitlichen Konzept so erweitert werden, daß Dialogstrukturen vom Anwender modifiziert werden können. Die Konsistenz der Formularfelder bezüglich ihrer Dateneingaben bleibt dabei erhalten.

Der Nachweis der These 1 stützt sich auf die Verwendung der Benutzermodellierung. Das Benutzermodell konfigurierbarer Benutzerschnittstellen beobachtet das Interaktionsverhalten des Anwenders auf Basis protokollierter Ereignisse. Dieses wurde von den Gestalten ausgelöst und zur Voraussage der bevorzugten Eingabefolge verwendet. Der Vergleich dieser Folge mit der Pfadmenge des Abhängigkeitsgraphen erlaubt die Beurteilung der aktuellen Struktur und das Vorschlagen einfacherer Alternativen. Die Abschnitte 2.1, 4.4 und Anhang A.9 plausibilisieren die Machbarkeit.

These 2 wird in den folgenden Abschnitten durch die Implementierung prototypischer Benutzerschnittstellen nachgewiesen. Diese simulieren Teile großer Systeme, deren Dateneingaben bereits für allgemeingültige Fälle optimiert wurden. Look_{As}^I-Like unterstützt die Modifikation der Benutzerschnittstelle durch den Anwender. Darauf basierend werden für die prototypischen Implementierungen Vereinfachungsmöglichkeiten der Dateneingabe für typische Anwendungsfälle gezeigt.

These 3 wurde durch den Prototypen Look_{As}^I-Like nachgewiesen. Das technologie neutrale Konzept (vgl. Kapitel 4) und die Randbedingungen der Spezialisierung

plausibilisieren, daß konfigurierbare Benutzerschnittstellen mit weiteren komponentenbasierten grafischen Benutzerschnittstellen realisierbar sind. Microsoft .NET Plattform [11] oder Java mit Simple Widget Toolkit (SWT) [22] eignen sich beispielsweise für weitere Spezialisierungen. Funktionsfähige prototypische Implementierungen mehrerer Benutzerschnittstellen zeigen die Machbarkeit des Konzepts. Insbesondere zeigt der Prototyp des IHKdezent2 Moduls Faktura, daß die Serialisierbarkeit der Komponenten der Grafikbibliothek zur Umsetzung des Konzepts in einer objektorientierten Programmiersprache genügt.

6.1 Vorgehensweise bei der Evaluierung von Dateneingaben

Die Evaluierung von Benutzerschnittstellen erfordert die Anwendung benutzerzentrierter Techniken. Bei einer Erstimplementierung können Nützlichkeit und Verwendbarkeit durch frühzeitige Evaluierung im Dialog mit Anwendern verbessert werden. Wird versucht, eine Verbesserung der Anwendungssituation durch eine neue Technologie basierend auf Urteilen der Benutzer nachzuweisen, können aufgrund psychologischer Aspekte und persönlicher Präferenzen der Benutzer unberechenbare Seiteneffekte bei der Beurteilung des Anwenders auftreten. Für die Evaluierung mit Benutzerumfragen muß somit eine geeignete Interpretation der erhaltenen Daten gefunden werden.

Benutzerschnittstellen, die den gleichen Prozeß implementieren, können nur dann sinnvoll analysiert werden, wenn der Anwender in beiden Systemen eine äquivalente Kategorisierung besitzt. Das Neuerlernen eines zweiten Systems kann hier bereits aufgrund des Aufwandes eine Ablehnungshaltung auslösen. Ein Vergleichssystem kann somit durch Benutzer nicht objektiv beurteilt werden. Die Fehlerfreiheit der Implementierung kann einen weiteren Grund für ein subjektives Urteil der Benutzer verkörpern. Sowohl die Existenz von Programmfehlern als auch die Performanz der Systeme beeinflussen den Eindruck der Verwendbarkeit eines Systems. Speziell produziert der Einsatz verschieden leistungsfähiger Rechner unerwünschte Seiteneffekte. Ein direkter Vergleich von Systemen erfordert insbesondere äquivalente Programmqualität. Bei den in den Abschnitten 6.3 und 6.4 vorgestellten Systemen wurden Teile prototypisch implementiert. Eine vollständige Implementierung beider Systeme würde einen Entwicklungsaufwand mehrerer Mannjahre erfordern. Aus diesem Grund kann keine objektive Beurteilung der Prototypimplementierung durch Benutzer der Originalsysteme angenommen werden.

Für die Evaluierung von Benutzerschnittstellen existieren Techniken, die nicht zwingend das Urteil ausgewählter Benutzer erfordern [26]. Generell beschäftigen sich theoriebasierte Techniken mit der Analyse der Frage, welche Aktionen ausgeführt werden müssen, um das System effizient bedienen zu können. Hierfür wird ein Benutzer, der das System bedient, theoretisch betrachtet. Unter der Annahme der Korrektheit aufgestellter Theorien kann diese Analyse anwendungsbezogene Evaluierungen substituieren. Damit kann die theoriebasierte Analyse eine Auskunft über das vorausgesagte Interaktionsverhalten liefern. Die Modellierung des angenommenen Benutzers und seiner Interaktion stellt einen wesentlichen Aspekt für die Güte der Analyse dar. Heute existieren jedoch keine allgemeingültigen Verfahren und Werkzeuge, um

6.2 Vervollständigung von Adressdaten

theoriebasierte Analysen direkt in den Entwurfsprozeß der Benutzerschnittstelle zu integrieren.

Die Evaluierung dieser Arbeit betrachtet die folgenden Aspekte der prototypischen Implementierungen:

- die Vereinfachung der Dateneingabe bei nicht allgemeingültigen Anwendungssituationen,
- die Vergleichbarkeit der Implementierungen der konfigurierbaren Benutzerschnittstelle und der ursprünglichen Benutzerschnittstelle und
- die Vollständigkeit der Implementierung.

Die Vereinfachung der Dateneingabe wird anhand ausgewählter Anwendungssituationen gezeigt. Die Dateneingabe einer Benutzerschnittstelle wird als einfacher angesehen, wenn

1. die vollständige Angabe aller benötigten Daten weniger Interaktionen erfordert als eine sequentielle Dateneingabe des ursprünglichen Systems,
2. eine vollständige Dateneingabe durch eine alternative Interaktionsfolge möglich ist oder
3. vormals frei eingebbare Interaktionselemente eine konsistente Wertselektion durch die Berechnung einer Auswahlmenge ermöglichen.

Die folgenden Abschnitte beschreiben beispielhafte Implementierungen konfigurierbarer Benutzerschnittstellen. Zunächst werden Vereinfachungsmöglichkeiten bei der Eingabe von Adressdaten argumentiert. Die zugehörige Implementierung wird in Anhang A.6 erläutert.

6.2 Vervollständigung von Adressdaten

Die Eingabe von Adressdaten wurde in den Kapiteln 4, 5 und Anhang A als Beispiel zur Veranschaulichung des Konzepts verwendet. Abbildung 4.2 zeigt die herkömmliche Struktur einer Adresseingabe, Abbildung 4.3 zeigt den Abhängigkeitsgraphen der Interaktionselemente. Die Gruppierung wurde in Abschnitt 4.4 vorgenommen. Die Analyse der Struktur mit der dort definierten Metrik ergibt die in Tabelle 6.1 dargestellten besten Interaktionsfolgen. Die Permutationen der Gruppe 3 verdeutlichen, daß die Elemente Wohnort und Postleitzahl die beste Güte der Auswahlmengen abhängiger Elemente liefern.

6.2.1 Initiale Eingabe des Postleitzahl

Die Eingabe der Postleitzahl „67663“ als erste Interaktion löst die nachfolgend beschriebenen Berechnungen aus. Das Dokument verwaltet die Berechnungen, um Zyklen und Redundanzen zu vermeiden. Zunächst wird versucht, die bestmögliche ablauffähige Regel zur Konsistenzprüfung des Elements *Postleitzahl* zu finden. Nachdem keine weiteren Elementwerte vorliegen, kann die Konsistenz der Postleitzahl nicht geprüft werden und erhält den Status „ungeprüft“. Im Falle einer positiven oder negativen Konsistenzaussage würden alle vorausgesetzten Elemente mit der Konsistenzprüfung

Gruppe	Eingabereihenfolge
2	Name, Vorname
3	Postleitzahl, Wohnort, Strasse, Hausnummer, Vorwahl
	Postleitzahl, Wohnort, Strasse, Vorwahl, Hausnummer
	Postleitzahl, Wohnort, Vorwahl, Strasse, Hausnummer
	Wohnort, Postleitzahl, Strasse, Hausnummer, Vorwahl
	Wohnort, Postleitzahl, Strasse, Vorwahl, Hausnummer
	Wohnort, Postleitzahl, Vorwahl, Strasse, Hausnummer

Tabelle 6.1: Auszug bester Interaktionsfolgen der Adresseingabe

beauftragt. Im nächsten Schritt werden die Berechnungen der Auswahlmengen aller abhängigen Elemente ausgelöst. Die ablauffähige Regel *Strasse_A* verkörpert die bestmögliche Regel, die den Wert des Elements *Postleitzahl* für die Berechnung nutzt. Nach der Vorbelegung des Elements *Strasse* wird erfolglos versucht, erneut die Konsistenzprüfung für das Element *Postleitzahl* auszulösen. Im folgenden Schritt wird versucht, das Element *Hausnummer* vorzubelegen. Die einzige Regel *Hausnummer_A* setzt die Elemente *Strasse*, *Postleitzahl* und *Wohnort* voraus. Darauf folgend wird versucht, das Element *Wohnort* vorzubelegen. Dies gelingt mit der Regel *Wohnort_A*, die als eindeutige Selektion den Wert „Kaiserslautern“ liefert. Die neue Selektion löst die Neuberechnung der Auswahlmenge von *Strasse* mit Regel *Strasse_B* aus. Das abhängige Element *Hausnummer* kann aufgrund des fehlenden Wertes von *Strasse* nicht vorbelegt werden. Die Selektion des Wohnorts löst die Konsistenzprüfung des Elements *Postleitzahl* mit Regel *Postleitzahl_B* aus und setzt den Status „konsistent“. Der neue Status aktiviert die Konsistenzprüfung des Elements *Wohnort* mit Regel *Wohnort_A*. Das Element *Wohnort* erhält den Status „konsistent“. Im letzten Schritt löst die Eingabe der Postleitzahl die Vorbelegung des Elements *Vorwahl* aus. Hierfür wird die Regel *Vorwahl_B* als bestmögliche gewählt, da sie der Vorwahl aus den Werten von *Postleitzahl* und *Wohnort* einen eindeutigen Wert zuordnen kann. Die Selektion der Vorwahl löst die Neuberechnung der Auswahlmengen von *Postleitzahl* mit Regel *Postleitzahl_C* und *Wohnort* mit Regel *Wohnort_B* aus, die jeweils die selektierte Vorwahl für die Berechnung verwendet. Da keine neue Selektion erfolgt, wird keine weitere Aktion ausgelöst. Abbildung 6.1 zeigt die Selektion und Konsistenz der Elemente nach Eingabe der Postleitzahl. Tabelle 6.2 erläutert die Bedeutung der Elementfärbung. Das Element *Strasse* besitzt aufgrund fehlender Selektion den Status „inkonsistent“. Da für das Element *Hausnummer* keine ablauffähige Regel existiert, gilt ihr Wert als „ungeprüft“. Die Eingabe einer Hausnummer verändert diesen Status nicht. Die hier dargestellte Interaktion selektiert drei der fünf Elemente eindeutig. Die Selektion aus der Auswahlmenge des Elements *Strasse* löst die Vorbelegung des Elements *Hausnummer* aus. Die Elemente der Gruppe 3 sind in diesem Fall mit drei

6.2 Vervollständigung von Adressdaten

Ende Speichern Weiter

Titel Vorname Name

Strasse Hausnummer

PLZ Wohnort

Vorwahl Telefonnummer

Abbildung 6.1: Selektion und Konsistenz nach Eingabe der Postleitzahl

Farbe	Konsistenz
grün	konsistent
rot	inkonsistent
gelb	ungeprüft
blau	unprüfbar

Tabelle 6.2: Legende der Konsistenzanzeige von Elementwerten

Abbildung 6.2: Selektion und Konsistenz nach Eingabe von Wohnort und Straße

Interaktionen konsistent eingegeben.

Die Neuberechnung der Auswahlmenge des Elements *Strasse* wird ausgelöst, da keine Selektion vorliegt. Bei gleicher Güte der Vorbelegung liefert die Regel *Strasse_A* 127 Werte, *Strasse_B* 899 Werte. Betrachtet man die Abhängigkeiten ohne Gesamtzustand, so liefert die Berechnung der Auswahlmenge ausschließlich konsistente Werte. Im Fall der Adresseingabe ist die Auswahlmenge von *Strasse_A* in der Auswahlmenge von *Strasse_B* enthalten.

6.2.2 Initiale Eingabe des Wohnorts

Die Eingabe des Wohnorts „Kaiserslautern“ löst die nachfolgend beschriebenen Berechnungen aus. Nach Setzen des Status „ungeprüft“ wird die Berechnung der Auswahlmenge des Elements *Strasse* mit der Regel *Strasse_B* ausgelöst. Das Element *Hausnummer* kann nicht vorbelegt werden, da die einzige Regel *Hausnummer_A* nicht ablauffähig ist. Im nächsten Schritt wird das Element *Postleitzahl* mit der Regel *Postleitzahl_B* eindeutig vorbelegt und selektiert. Die Selektion der Postleitzahl löst die Neuberechnung der Auswahlmenge des Elements *Strasse* aus.

Als folgende Interaktion sei die Straße „Kurt-Schumacher-Straße“ selektiert.

6.3 Austausch von CAD/CAM Konstruktionsdaten

Nach der Konsistenzprüfung von *Postleitzahl* und *Wohnort* wird die Berechnung der Auswahlmengen der Elemente *Hausnummer* und *Vorwahl* angestoßen. Die Vorwahl wird eindeutig vorbelegt und selektiert und löst die Neuberechnung der Auswahlmengen der Elemente *Wohnort* und *Postleitzahl* ohne Änderung der Selektion aus. Abbildung 6.2 zeigt die Selektion und Konsistenz der Elemente nach Eingabe des Wohnorts und Selektion der Straße. Analog zur initialen Eingabe der Postleitzahl kann die konsistente Dateneingabe der Gruppe mit drei Interaktionen erreicht werden.

Wählt man als zweite Interaktion die Selektion der Postleitzahl, läuft die Berechnung der Auswahlmengen und Konsistenzprüfungen analog zur initialen Eingabe der Postleitzahl ab.

6.2.3 Ergebnis

Die Eingabe von Adressdaten zeigt deutliche Vereinfachungsmöglichkeiten. Diese hängen einerseits wesentlich von der Reihenfolge der Interaktionen ab und andererseits ermöglichen geschickt gewählte Eingaben eindeutige Werte bei abhängigen Elementen. Diese lösen ihrerseits weitere Berechnungen von Auswahlmengen aus. Die Eingabe der Postleitzahl ermöglicht die eindeutige Vorbelegung von zwei Elementen und die Berechnung der Auswahlmenge eines dritten Elements.

Die Transitivität der Berechnungen aus eindeutigen Elementwerten führt bei statischer Analyse im Programmwurf nicht zum Ziel. Sie schafft jedoch in vielen Anwendungssituationen konfigurierbarer Benutzerschnittstellen starke Vereinfachungen der Dateneingabe.

Die Verbesserung der Bedienbarkeit wird durch Umstrukturierung der Interaktionselemente erreicht. Die Benutzerschnittstelle schlägt entsprechende Modifikationen vor. Mit der Fixierung einzelner Elementwerte können weitere fallbasierte Vereinfachungen vorgenommen werden. Der folgende Abschnitt erläutert diese Möglichkeiten.

6.3 Austausch von CAD/CAM Konstruktionsdaten

Beim Versenden von CAD/CAM Konstruktionsdaten innerhalb der Zuliefererkette in der Automobilindustrie [24] werden Projektmitarbeiter mit der nachfolgend beschriebenen Situation konfrontiert.

Heutzutage arbeiten Konstruktionsabteilungen mit den an einem Projekt (Auto, Systemkomponente, etc.) beteiligten Firmen eng zusammen. In der Konstruktionsphase werden die Einzelteile, die von den Projektpartnern konstruiert werden, im CAD System virtuell zusammengebaut. Neben der Kontrolle der Einbaumaße und -toleranzen werden kinetische Untersuchungen und Crashsimulationen im Rechner durchgeführt. Hierfür müssen die Konstruktionsdaten der Projektpartner zur Koordination innerhalb der Zuliefererkette transportiert werden. Durch kurze Laufzeiten der Daten wird die Zahl der Iterationen für Optimierungen erhöht, so daß die Güte der Entwicklung bei gleichzeitiger Reduzierung der Entwicklungszeit steigt und die Entwicklungskosten drastisch sinken.

Konstrukteure korrespondieren dabei häufig mit mehreren Projektpartnern. Da

die Welt der CAD/CAM-Systeme stark heterogen in Bezug auf Plattformen, Betriebssysteme und Datenformate ist, sind vor dem Transport der Daten Qualitätskontrollen und Datenkonvertierungen notwendig. Speziell unterscheiden sich häufig beim gleichen Partner unter den Fachabteilungen CAD Systeme, Datenformate oder die Qualitätsvorgabe der Daten auch bei abteilungsübergreifend verwendeten CAD Systemen. Im Extremfall muß der Konstrukteur jedem Kontakt auf Partnerseite unterschiedliche Formate und Qualitätskriterien liefern. Die Informationen über den Datenaustausch werden entweder zwischen den Konstrukteuren oder am Projekt beteiligten Abteilungen verhandelt, so daß häufig keine zentrale Anlaufstelle in der Projektgruppe existiert, die alle Datenaustauschvereinbarungen kennt.

Während das Versenden der Daten noch vor einigen Jahren manuell durch eine Zentralabteilung pro Firma bzw. Standort ausgeführt wurde, bieten Automatismen [24, 45, 46] heutzutage schnellere Datentransporte, Sicherheitsmechanismen, automatische Konvertierungen der Konstruktionsdaten, Qualitätskontrollen, automatisches Versenden/Empfangen von Daten und eine transparente, zentrale Statuskontrolle des Austauschvorgangs für den Konstrukteur. Die Rolle der Zentralabteilung reduziert sich auf die Überwachung der Systemressourcen, des Dateneingangs, die Beratung der Konstrukteure bei der Korrektur fehlerhafter Konstruktionsdaten und die Planung von Modernisierungen der Infrastruktur. Eine unterliegende Datenbank kennt alle Datenaustauschvereinbarungen, so daß die notwendigen Datenkonvertierungen, Aufbereitungen, Qualitätskontrollen und Versendetechniken daraus gewählt werden können. Diese Auswahl erfolgt automatisch, sobald der Konstrukteur vollständige Informationen über seinen Kontakt und die zu versendenden Daten besitzt und die Datenaustauschvereinbarung dieses Kontakts in der Datenbank gespeichert wurde. Eine vollständige Automatisierung für den Austausch von Konstruktionsdaten wurde in [24] entwickelt. Die Benutzerschnittstelle des Programms erfragt beim Konstrukteur die notwendigen Informationen, um die erforderliche Datenaustauschvereinbarung in der Datenbank zu identifizieren und diverse Sicherheitskriterien zu prüfen. Die Parameter, die der Konstrukteur dabei zur Identifikation von Kontakten beim Absender bzw. Empfänger angeben muß, werden nachfolgend beschrieben. Der Kontakt des Absenders ist häufig bereits durch die Kontoinformation des Benutzers vollständig definiert, jedoch werden Daten durchaus auch im Auftrag eines Projektmitarbeiters versendet, wodurch die Korrektur des Absenders notwendig werden kann. In diesem Fall müssen aus der Kontakt Datenbank zwei Kontakte ausgewählt werden. Ein Kontakt wird eindeutig durch die Felder *Firma*, *Abteilung*, *Name* und *Kürzel* bestimmt.

Firma Identifikation des Absenders. Die Unterscheidung des Standorts einer Firma liefert standortspezifische Informationen im elektronischen Datenaustausch (*electronic data interchange, EDI*). Firmen besitzen üblicherweise standortbezogene Übergabepunkte für Konstruktionsdaten. Die Identifikation des Benutzers definiert den Standort nicht eindeutig, da Konstrukteure projektbezogen über verschiedene Standorte hinweg eingesetzt werden.

Abteilung Abteilung der Firma des Kontakts. Die Datenbank enthält für jede Abteilung spezifische Informationen der Datenaustauschvereinbarung, z.B. Standortdaten, vorhandene CAD Systeme, akzeptierte Datenformate, Qualitätsvorgaben oder Ansprechpartner für den Datenaustausch.

Name Eindeutige Benutzererkennung des Konstrukteurs. Unter dieser Kennung sind in der Datenbank alle notwendigen Informationen über den Konstrukteur erfragbar:

6.3 Austausch von CAD/CAM Konstruktionsdaten

Abteilung, Adresse, Telefon, E-Mail, verwendete CAD Systeme, Position in der Abteilung, Datenaustauschberechtigung, Kürzel, etc.

Kürzel Kennung des Kontakts. Die Kennung hängt häufig vom Zielformat, Quellformat oder von der Versendart ab.

Quellformat Das Quellformat wird einerseits durch die gewählten Daten definiert und validiert, andererseits definiert diese Einstellung die Menge der möglichen Datenformate, die der Konstrukteur überhaupt an seinen Kontakt senden darf.

Zielformat Das Zielformat definiert sich abhängig vom Quellformat und den möglichen Datenkonvertierungen und wird durch die Menge der Datenformate validiert, die der Kontakt empfangen und bearbeiten kann.

Versendart Die Art des Datenversands wird durch die Austauschvereinbarung mit dem Projektpartner definiert. Die konkrete Versendart sollte von der Größe der erzeugten Daten abhängen, die vorab nicht verlässlich bestimmbar ist. Möglicherweise besitzen EDI-Systeme Obergrenzen für transportable Dateigrößen. Im Fall, daß das EDI-System den Transport ablehnt, modifiziert der Konstrukteur den Versendeauftrag.

Quelldaten Die Quelldaten werden abhängig vom eingestellten Quellformat zur Auswahl angezeigt. Häufig besitzen die CAD Modelle keinen vom Konstrukteur nachvollziehbaren Namen im Dateisystem, sondern werden aus einem Content-Management-System (CRM) oder aus einer Modelldatenbank oder Produktlebenszyklussystem (*product lifecycle management system, PLM*) ausgekoppelt.

Beschreibung Abhängig von der Versendart werden unterschiedliche Informationen zur Beschreibung der Daten benötigt.

Bei der obigen Beschreibung der Elemente fällt die mehrfache Speicherung von Informationen auf, beispielsweise werden CAD Systeme der Firma *und* dem Kontakt zugeordnet. Das Datenaustauschprogramm wählt die Daten abhängig vom Eingabezustand der Elemente aus. Dementsprechend werden häufig Neuberechnungen bei der Berechnung der Auswahlmengen benötigt. Das Datenaustauschprogramm implementiert folgende Reihenfolge der Dateneingaben nach Identifikation des Benutzers:

Sender: Abteilung, Name und Kürzel, sofern der Absender vom angemeldeten Benutzer abweicht

Empfänger: Firma, Abteilung, Name und Kürzel

Transformation: Quellformat, Zielformat, Optionen

Nutzdaten: Versendart, Beschreibung und Modellnamen

Für die Evaluierung konfigurierbarer Benutzerschnittstellen wurden Teile des Datenaustauschsystems der Firma Robert Bosch GmbH, Abteilung EAM5, Bühl (Baden) [24] nachgebildet. Zur Veranschaulichung der Vereinfachungsmöglichkeiten wurde der Dialog zur Beauftragung eines Datenaustauschprozesses prototypisch implementiert.

Nachfolgend werden typische Anwendungsszenarien beschrieben, bei denen mit dem herkömmlichen Datenaustauschprogramm bei fester Eingabereihenfolge kein Auftrag erzeugt werden kann. Das Problem besteht darin, daß ein Konstrukteur nicht

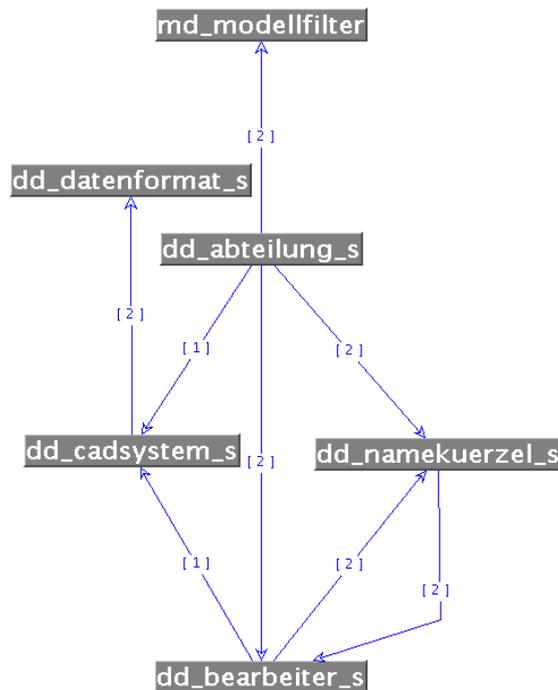


Abbildung 6.3: Abhängigkeitsgraph der Senderdaten für das Datenaustauschprogramm in [24]

aus allen Daten der Partnerbeziehung wählen kann, um seinen Austauschpartner im Programm zu bestimmen. Beispielsweise kommunizieren Konstrukteure per Telefon und kennen die Firma und den Namen ihres Gegenübers. Diese Informationen genügen zur gemeinsamen Projektarbeit, jedoch nicht zur Partnerbestimmung im Datenaustauschprogramm. Der Einsatz konfigurierbarer Benutzerschnittstellen ermöglicht durch geschicktes Vorbelegen der Daten dennoch die Eingabe eines vollständigen Auftrags. Die Benutzerschnittstelle enthält 13 Interaktionselemente. Die Abhängigkeiten der Elemente ergeben wenige eindeutige Vorbelegungen, so daß eine statische Analyse keine Vereinfachungen der Benutzerschnittstelle ermöglicht. Die Analyse der allgemeingültigen Struktur ergab eine bereits optimale Platzierung der Interaktionselemente. Häufig können in spezifischen Anwendungssituationen eindeutige Vorbelegungen abhängig von der Datenaustauschbeziehung vorausgesagt werden. Für diese Anwendungssituationen kann die Benutzerschnittstelle mit festen Elementwerten konfiguriert und mit vereinfachter Struktur gespeichert werden. Abbildung 6.3 zeigt die Gruppe Abhängigkeiten auf Senderseite. Nach Selektion der Abteilung (*dd_abteilung_s*) wird ein Datenbereich für die Auswahl von Modellen (*dd_modellfilter*) eindeutig definiert. Dieser Bereich enthält Konstruktionen der Abteilung des Absenders. Je nach Projekt kann der Benutzer andere Modellbereiche auswählen. An dieser Stelle können weitere Sicherheitsmechanismen mit Regeln (*UpdateRule*) (siehe Anhang A.1) implementiert werden. Diese beschränken den Zugriff auf Unternehmensdaten. Dies kann als Vorsichtsmaßnahme verstanden werden, jedoch bestehen häufig auch Konkurrenzsituationen zwischen Abteilungen des gleichen Unternehmens.

6.3 Austausch von CAD/CAM Konstruktionsdaten

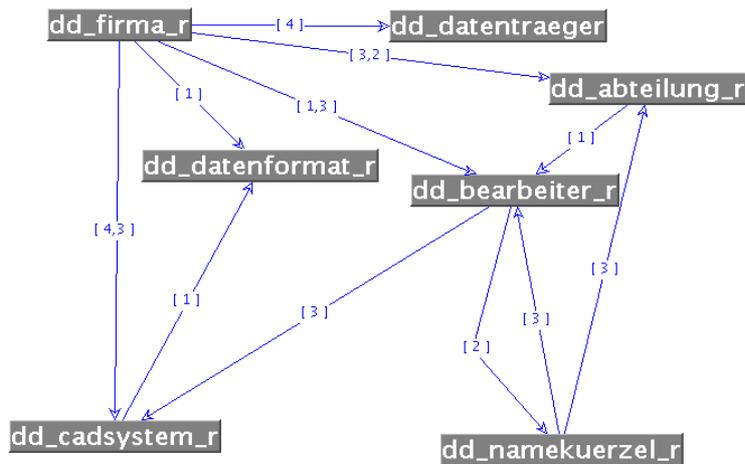


Abbildung 6.4: Abhängigkeitsgraph der Empfängerdaten für das Datenaustauschprogramm in [24]

Abbildung 6.5 zeigt den Dialog des Datenaustauschprogramms zur Eingabe der Sender-Empfänger Beziehung als konfigurierbare Benutzerschnittstelle. Die folgenden Abschnitte beschreiben typische Anwendungssituationen der Benutzer.

6.3.1 Austausch mit einem Konstruktionsbüro

Beim Austausch mit Konstruktionsbüros wird häufig nur ein einziger Kontakt für die Austauschbeziehung eingerichtet. Die Angabe der Firma reicht in diesem Fall aus, um Kontakt und Zielformat eindeutig zu bestimmen. Nach der Angabe des Absenders, der möglicherweise vom angemeldeten Benutzer abweicht, können Quellformat, Zielformat und Optionen angegeben werden. Konstruktionsbüros arbeiten gewöhnlich mit dem CAD System des Auftraggebers, d.h. es findet keine Konvertierung statt. Das Zielformat ist in diesem Fall ebenfalls eindeutig bestimmt.

Ein angemeldeter Benutzer der Daten seines CAD-Systems an ein Konstruktionsbüro sendet, kann zusätzlich Optionen der Datenkonvertierung fest einstellen. Die Dateneingaben des Benutzers erfolgen nach Anmeldung somit in dieser Reihenfolge:

1. Firma
2. Modelle
3. Versendeart
4. Beschreibung

Das Element *Optionen* wird vom Benutzer in der Konfiguration mit einem Wert fixiert, die übrigen Elemente besitzen lediglich informativen Charakter und können entsprechend angeordnet werden. Tauscht der Benutzer mit wenigen Konstruktionsbüros aus, kann der Aufwand weiter durch Speicherung einer Konfiguration für jedes Konstruktionsbüro auf die Eingabe der *Modelle* und der *Beschreibung* reduziert

CAD Data Exchange

Benutzer:	Hans-Peter Meier	Abteilung:	EAMS	E-Mail:	hans-peter.meier@de.bosch...	Kostenst.:	2222
Account:		Firma:	Robert Bosch GmbH	Telefon:	+49 72223 82 1488	Fax:	+49 72223 82 1489

Daten ausgeben	
Sender (sender)	Empfänger (destination)
Projekt	
Firma (Sender)	Firma(Empf.)
Abteilung(Sender)	Abteilung(Empf.)
Bearbeiter(Sender)	Bearbeiter(Empf.)
Adresscode(Sender)	Adresscode(Empf.)
CAD Sender	CAD(Empfänger)
Datenformat(Sender)	Datenformat(Empf.)
Option	Datenträger
Bemerkungen zum Austauschvorgang (informations for the exchange):	
3. Iteration	
Nachricht an den Austauschpartner (message to the customer / supplier):	
Projekt HCN-134-23.1	

Exit program

Abbildung 6.5: Sender–Empfänger Beziehung des Datenaustauschprogramms

6.3 Austausch von CAD/CAM Konstruktionsdaten

werden.

Die Interaktion mit der konfigurierbaren Benutzerschnittstelle läßt sich somit für den Austausch mit Konstruktionsbüros auf 3 oder 4 Dateneingaben vereinfachen.

6.3.2 Kürzel definiert den Kontakt

Häufig kennen nicht alle Mitglieder einer Projektgruppe die Kontakte des Projektpartners, sondern erhalten von einem unbekanntem Kontakt gesendete Daten, die sie modifizieren und mit dem richtigen Kürzel des Kontakts versehen wieder zurückschicken müssen. Ein Kürzel kann beim Austauschpartner verschiedene Bedeutungen bei der Datenverarbeitung besitzen, beispielsweise zur Identifikation

- eines Kontakts,
- einer Projektgruppe oder
- eines Datenformats.

Im dritten Fall enthalten die Kontaktdaten keine konstruktiven Informationen. Identifiziert das Kürzel eine Projektgruppe, so können entweder mehrere Kontakte das gleiche Kürzel besitzen oder ein stellvertretender Kontakt eingerichtet sein. In beiden Fällen spielen die Werte von *Abteilung* und *Name* keine Rolle.

Bei sequentieller Eingabe der Daten müßte der Benutzer die Abteilung und eine Kontaktperson (sofern mehrere Kontaktpersonen innerhalb der Abteilung existieren) benennen, um vom System eine Vorbelegung für ein Kürzel zu erhalten. Kürzel wurden in der Partnerbeziehung vereinbart und identifizieren die Daten auf Empfängerseite. Ein Kürzel besitzt dabei beliebige Bedeutungen. Die Dateneingaben des Benutzers erfolgen nach Anmeldung (z.B. an der Systemkonsole) somit in dieser Reihenfolge:

1. Firma
2. Kürzel
3. Modelle
4. Optionen
5. Versendeart
6. Beschreibung

Die sequentielle Dateneingabe würde bei vielen Datenaustauschprozessen aufgrund nicht kommunizierter Informationen über Abteilung und Name eines stellvertretenden Kontakts die Eingabe eines Versendeauftrages verhindern. Ein Benutzer würde in dieser Situation Kontakt zu einem Experten der Zentralabteilung oder seiner Projektgruppe suchen.

Die Interaktion mit der konfigurierbaren Benutzerschnittstelle läßt sich somit für den Austausch mit Konstruktionsbüros auf 6 Dateneingaben vereinfachen, sofern die Partnerbeziehung in der Datenbank existiert.

6.3.3 Geänderte Kontaktdaten

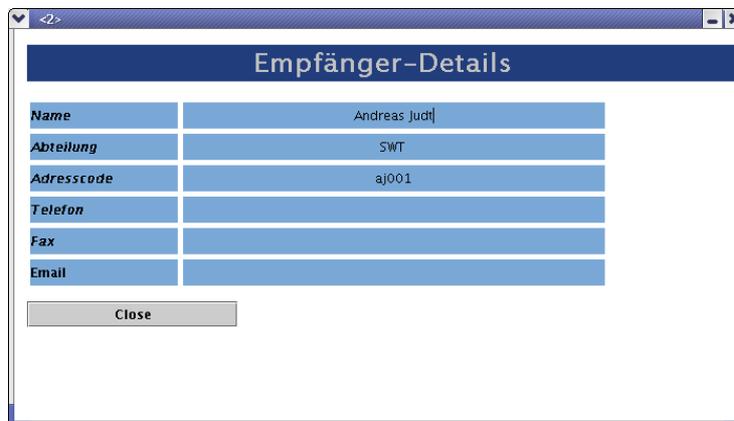
Im Zuge von Umstrukturierungen werden Projektgruppen, Abteilungen und Zuständigkeiten geändert. In einem unternehmensübergreifenden Entwicklungsprojekt führen diese Maßnahmen häufig dazu, daß ein Datenaustausch aufgrund geänderter Daten bei sequentieller Eingabe bzw. Berechnung der Vorbelegung nicht gestartet werden kann, da ein vom Benutzer erwarteter Wert nicht in der Auswahlmenge enthalten ist. Beispielsweise würde ein neuer Abteilungsname bei sequentieller Eingabe die Nachfragen bei Projektmitarbeitern oder der Zentralabteilung erfordern. Alternativ können dann Kontaktpersonen mittels Durchspielen der Abteilungen probiert werden. Dieses Vorgehen ist zeitaufwendig und ineffizient, führt aber üblicherweise zum Ziel.

Konfigurierbare Benutzerschnittstellen lösen das Problem durch dynamische Auswahl der Regel zur Berechnung der Auswahlmenge. Eine typische Lösung für den Benutzer wäre eine unscharfe Eingabe der vorhandenen Informationen. Dies kann durch Integration eines Filters (siehe Anhang A.1) erreicht werden. Der Benutzer gibt beispielsweise den bekannten Teil eines Namens, etwa einen Teil des Nachnamens, ein. Der Filter berechnet eine Liste möglicher Kontaktpersonen als Auswahlmenge. Der Benutzer selektiert eine Kontaktperson oder verfeinert den Filter. Nach Auswahl der Kontaktperson werden Abteilung und CAD System automatisch mit einer Selektion vorbelegt. Die Abteilung besitzt einen eindeutigen Wert, das CAD System enthält als Auswahlmenge die assoziierten Systeme. Somit kann der Prozeß fortgesetzt werden.

Als alternativer Ansatz wurde ein modales Fenster implementiert, das alle Kontaktdaten, wie beispielsweise Standort, Name, Abteilung, Telefon, Fax, E-Mail zur Auswahl stellt. Die konfigurierbare Benutzerschnittstelle bietet hier die Möglichkeit, die Partnerbeziehung aus genau den Parametern zu bestimmen, die dem Konstrukteur bekannt sind. Der Anwender stellt das Formular aus Abbildung 6.5 mit Formularfeldern aus Abbildung 6.6 zusammen, indem er die Felder über die Fenstergrenzen hinweg austauscht. Somit enthält das Formular zur Bestimmung der Partnerbeziehung nur die für den Spezialfall relevanten Formularfelder.

Wird jedes Element mit einem Filter und Regeln ausgestattet, kann der Benutzer über verschiedene Filtereingaben und Selektionen einen Kontakt auswählen. Für den Benutzer besteht weiterhin die Möglichkeit, Teile dieses modalen Dialogs in das unterliegende Fenster zu verlagern und so ein weiteres Interaktionselement zur Kontaktauswahl zu integrieren. Ein typisches Beispiel hierfür wäre die Integration der Telefonnummer des Kontakts in den unterliegenden Dialog. Dieses Vorgehen erzeugt eine große Gruppe, da die Kontaktinformationen Abhängigkeiten zu den Elementen des modalen Fensters besitzen. Die Implementierung der Strukturanalyse wurde alternativ so angepaßt, daß das modale Fenster eine eigene Gruppe definiert. Die Übernahme eines Kontaktes wird dann über ein Aktionselement implementiert angestoßen. Eine weitere Möglichkeit wird durch dynamisches Umketten bei Duplizieren des Elements vom modalen Fenster in den Kontaktdialog realisiert. Wird ein Element des modalen Dialogs auf das unterliegende Fenster kopiert, so kann die Erweiterung der Kontaktauswahl durch semantisch äquivalente Elemente (z.B. Abteilung, Name oder Adreßcode) erreicht werden. Abbildung 6.6 zeigt den modalen Dialog der alternativen Kontaktauswahl. Dieses Fenster wurde mit dem Aktionselement *details* verknüpft, das in Abbildung 6.5 rot markiert wurde.

6.3 Austausch von CAD/CAM Konstruktionsdaten



Empfänger-Details	
Name	Andreas Judt
Abteilung	SWT
Adresscode	aj001
Telefon	
Fax	
Email	

Close

Abbildung 6.6: Modaler Dialog für dynamische Kontaktauswahl

6.3.4 Projektbezogene Konfiguration

Abhängig vom jeweiligen Aufgabenfeld besitzen Konstrukteure nur wenige Austauschpartner. In diesem Fall bieten konfigurierbare Benutzerschnittstellen die Möglichkeit, die Kontaktdaten mit der Benutzerschnittstelle zu speichern. Der Konstrukteur erstellt eine Konfiguration für jeden seiner Austauschpartner¹. Um einen Datenaustausch-Prozess zu initiieren, startet der Konstrukteur das Programm mit der entsprechenden Konfiguration.

Beim Prototypen des Datenaustauschprogramms wird die Konfiguration als Kommandozeilenparameter angegeben. Alternativ kann zum Programmstart eine Liste von Konfigurationen zur Auswahl angeboten werden. Die gespeicherten Konfigurationen können zusätzlich modifiziert und gespeichert werden. Die Interaktionen nach Start der Benutzerschnittstelle reduzieren sich auf zwei Dateneingaben: die *Beschreibung* und die Auswahl der *Modelle*.

6.3.5 Ergebnis

Das originale Datenaustauschprogramm [24] wurde 1997 als Intranetanwendung mit Perl [59], Korn Shell [28] und HTML [61] implementiert. Eine korrespondierende monolithische Implementierung mit Java und Swing existiert für das System nicht. Die Komplexität beider Implementierungen konnte aus diesem Grund nicht verglichen werden.

Die im Abhängigkeitsgraphen dargestellten möglichen Vorbelegungen können mit herkömmlicher Programmierweise implementiert werden, wobei üblicherweise eine Interaktionsreihenfolge vorgegeben wird. Hat der Konstrukteur nicht alle notwendigen Informationen zur Hand, kann im originalen System kein Auftrag generiert werden. Zur Erreichung seines Ziels muß der Konstrukteur die notwendigen Informationen in seiner Projektgruppe erfragen. Alternativ informiert er sich bei der Zentralabteilung für Datenaustausch, um den Kontakt eindeutig zu identifizieren.

¹in diesem Fall üblich: 1 bis 3 Partner

Die dargestellten Anwendungssituationen zeigen mögliche Vereinfachungen, die aus der statischen Prozeßanalyse nicht erkennbar sind. Die Identifikation äquivalenter Semantik von Elementen verschiedener Gruppen ermöglicht zusätzlich ein dynamisches Umketten von Elementen über Gruppengrenzen hinweg, wobei die Effizienz der Berechnung von Auswahlmengen und Konsistenzprüfungen erhalten bleibt. Aufgrund fortschreitender Änderungen der Datenaustauschbeziehungen würde eine Mehrfachimplementierung der Benutzerschnittstelle für die gezeigten Fälle eine kurzfristige Lösung darstellen, aber nicht zukünftige Anwendungssituationen vereinfachen können.

6.4 Das Verwaltungssystem IHKdezent2

Das Modul Faktura ist Teil des Verwaltungssystems IHKdezent2, das von der TMG Systemhaus GmbH, Lauf a. d. Pegnitz, für Industrie- und Handelskammern entwickelt wurde. Faktura vereint komplexe Komponentenstrukturen und Spezialimplementierungen verschiedener Interaktionselemente. Die Look_{AS}^ILike Implementierung evaluiert die Fähigkeit des Konzepts, beliebige serialisierbare Komponenten in die Struktur einer konfigurierbaren Benutzerschnittstelle integrieren zu können. Die TMG Systemhaus GmbH entwickelte eine Programmbibliothek verschiedener spezialisierter Swing-Komponenten und Datenbankabfragen zur Initialisierung des Dialogs. Diese Komponenten unbekannter Implementierung wurden zunächst als Kontrollkomponenten spezialisiert. Die Vergleichsimplementierung generiert darauf aufbauend Fensterelemente, Markenelemente, Aktionselemente und Datenelemente und erzeugt ein äquivalentes Verhalten des Dialogs. Die Programmierung der konfigurierbaren Elemente wird anhand ausgewählter Komponenten der TMG Bibliothek dargestellt. Abbildung 6.7 zeigt die implementierte konfigurierbare Benutzerschnittstelle.

6.4.1 Spezialimplementierungen

Der Dialog des Moduls Faktura wurde dem Ursprungssystem, das etwa 1988 entstand, und einer wesentlich älteren Papierform nachempfunden. Die Benutzerschnittstelle enthält verschiedene Elemente, in denen aus einem Schlüssel ein Textfragment der Formalkorrespondenz erzeugt wird. Beispielsweise erzeugt der Schlüssel „I“ im Element *Anrede* den Wert „Herrn“. Dieser Text darf vom Benutzer selbstverständlich nicht verändert werden. Zur Anzeige dieser Texte wurden Datenelemente implementiert, die Textanzeigen (*javax.swing.JLabel*) als einzige Gestalt nutzen. Abhängig von den Eingaben des Benutzers werden Teile des Dialogs dynamisch ausgetauscht. Beispielsweise ändert die Selektion „Institution“ des Elements *Obj.Art* die Elementstruktur der rechten Hälfte des Dialogs. Für eine effiziente Auswahl eines Schlüssels wurden drei Interaktionselemente und ein Markenelement zu einer Spezialimplementierung einer *ComboBox* vereint. Abbildung 6.9 zeigt die TMG-*ComboBox* für das Element *Nam.Titel*. Eine solche *ComboBox* besteht aus einem Schlüssel, dem dazugehörigen Feld zur Anzeige des Klartexts und einem Knopf zur Anzeige einer Auswahlliste. Das Drücken des Knopfes (rot markiert) bewirkt die Anzeige eines modalen Fensters (grün markiert), das gültige Schlüssel-Klartext Paarungen in einer Tabelle (*javax.swing.JTable*) zur Auswahl anbietet. Die Selektion einer Tabellenzeile definiert einen neuen Wert für das Element. Gleichzeitig wird das modale Fenster entfernt. Die Positionierung dieses Fensters orientiert sich an der Position des Schlüsselfeldes im

6.4 Das Verwaltungssystem IHKdezent2

Elementname	Beschreibung
anrsl	Anredeschlüssel; Werte: Person, Institution, Sonstige
znamezus	Nam.Zusatz; Zusätze zum Namen, z.B. familiäre Titel
zbranr	Briefanrede
geschl	Geschlecht der Person; Werte: M(ännlich), W(eiblich), U(nbekannt)
ort2	Ort; Angabe für Großkundenadressen
plz2	Postleitzahl; Angabe für Großkundenadressen
name1	Nachname
name2	Vorname
alphasort	Sortierschlüssel; aus Name und Vorname berechnet.
strasse	Straße
plz	Postleitzahl
ort	Ort
zzztfon	Telefonnummer; inklusive Vorwahl

Tabelle 6.3: Programminterne Elementbezeichnungen des Faktura Moduls

Dialog: das modale Fenster wird genau unterhalb der TMG-ComboBox positioniert und ist nicht vom Benutzer modifizierbar.

6.4.2 Machbarkeit und Aufwand

Abbildung 6.8 zeigt den Abhängigkeitsgraphen des Faktura Moduls. Die programminterne Bezeichnung der Elemente wurde vom Originalsystem IHKdezent2 übernommen. Tabelle 6.3 erläutert die Abbildung der Elementnamen auf Interaktionselemente des Dialogs. Für die Eingabe der Telefonnummer (*zzztfon*) wird die Vorwahl aus dem Wert des Ortes (*ort*) berechnet und zur Erleichterung der Eingabe vorbelegt. Dies hat zur Folge, daß für die Konsistenzprüfung die Vorwahl aus dem Elementwert *zzztfon* extrahiert werden muß. Die Extraktion wurde in der Regel zur Vorbelegung der Vorwahl implementiert.

6.4.3 Ergebnis

Trotz relativ weniger Abhängigkeiten bietet die konfigurierbare Benutzerschnittstelle ein hohes Potential an Vereinfachungen durch Fixierung von Elementwerten und Auswahlmengen. Beispielsweise können die Abhängigkeiten der ersten Gruppe (*anrsl*, *znamezus*, *zbranr*, *geschl*) auch in entgegengesetzter Richtung gelten. Im Originalsystem hat sich gezeigt, daß die Entwickler diese Berechnung aufgrund der erwarteten Unübersichtlichkeit des Programms nicht implementiert haben. Es wären

6.5 Zusammenfassung

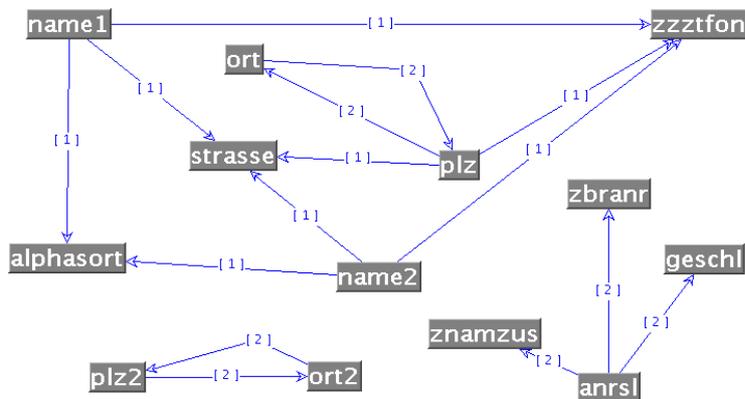


Abbildung 6.8: Abhängigkeitsgraph des IHKdezent2 Moduls Faktura

viel mehr Abhängigkeiten umsetzbar gewesen, als in Abbildung 6.8 gezeigt wurden. Die notwendige strukturierte Programmierung von Vorbelegungen und Konsistenzprüfungen in Regeln erlaubte eine übersichtliche Umsetzung der Verhaltensvorgabe des Auftraggebers, die als Textdokument vorlag.

Die Integration von Spezialimplementierungen der TMG Systemhaus GmbH verdeutlicht die Fähigkeit, beliebige Komponenten zur Implementierung der Benutzerschnittstelle verwenden zu können. Durch Migration der Spezialimplementierungen, z.B. des modalen Fensters der TMG-ComboBox wären weitere Effizienzsteigerungen bei der Entwicklung möglich.

6.5 Zusammenfassung

Die Evaluierung konfigurierbarer Benutzerschnittstellen zeigt die Machbarkeit des Konzepts und Möglichkeiten zur Vereinfachung von Dateneingaben. Die Vergleichbarkeit des Aufwandes zwischen konfigurierbaren Benutzerschnittstellen und ursprünglichen Systemen wurde argumentiert und beispielhaft dargestellt. Die Implementierung des Moduls Faktura evaluierte die Flexibilität des Konzepts konfigurierbarer Benutzerschnittstellen, beliebige Implementierungen grafischer Komponenten als Gestalten konfigurierbarer Elemente verwenden zu können. Situationsbedingte eindeutige Selektionen bzw. Vorbelegungen von Elementen ermöglichen in vielen Anwendungssituationen erhebliche Vereinfachungen der Dateneingabe. Mit der Implementierung des Auftragsdialogs des Datenaustauschsystems aus [24] als konfigurierbare Benutzerschnittstelle wurden übliche Anwendungssituationen theoriebasiert evaluiert, bei denen Vereinfachungen durch statische Analyse im Entwurf schwer zu entwickeln sind. Bei großen Gruppen lassen sich Knoten im Abhängigkeitsgraphen über Gruppengrenzen hinweg umketten, wenn semantisch äquivalente Elemente in verschiedenen Gruppen existieren. In einer Erweiterung der Implementierung kann ein Strukturoptimierer Elemente aus einer Komponentenbibliothek in die konfigurierbare Benutzerschnittstelle und insbesondere in die

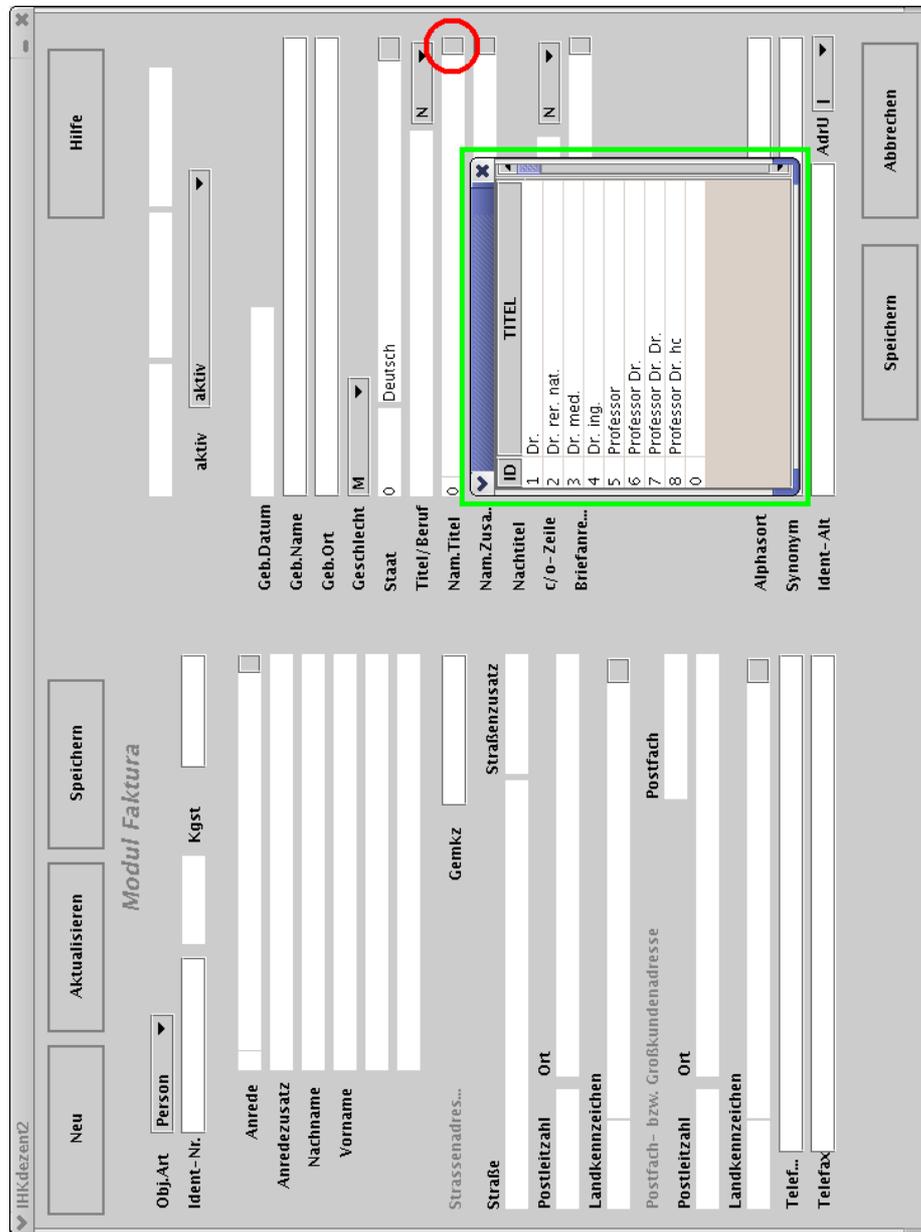


Abbildung 6.9: Integration von Spezialimplementierungen des Originalsystems

6.5 Zusammenfassung

Berechnung der Auswahlmengen integrieren, um eine Vereinfachung der Bedienung durch Flexibilisierung des Dialogs zu erreichen.

Konfigurierbare Benutzerschnittstellen lassen sich genau dann speichern, wenn alle beteiligten Komponenten serialisierbar sind. Insbesondere gilt dies für Hörer und Berechnungen. Die Implementierung der Programmbibliothek Look_{AS}^ILike zeigt, daß die Programmbibliotheken JAXB [35] und Swing [51] nicht fehlerfrei sind. Abschnitt 7.1.1 erläutert eine mögliche Vorgehensweise zur Lösung des Problems.

Kapitel 7

Zusammenfassung und Ausblick

Benutzerschnittstellen können Anwender bei der Vereinfachung von Dateneingaben unterstützen. Das dazu erforderliche Prozeßwissen befindet sich im mentalen Modell des Anwenders. Vereinfachungen reflektieren häufig Anwendungsfälle, die nicht allgemeingültig sind. Die so erreichbare Steigerung der Effizienz kann in der Praxis üblicherweise nicht durch Programmierung umgesetzt werden, sondern muß vom Benutzer vorgenommen werden. Die Vereinfachungen von Dateneingaben kann auch jenseits fensterbasierter Interaktion bei der Implementierung von Benutzerschnittstellen genutzt werden.

In Kapitel 6 wurden die Vereinfachungsmöglichkeiten formularbasierter Benutzerschnittstellen evaluiert. Für eine konkrete Umsetzung des Konzepts müssen technische Anpassungen und Erweiterungen für die Integration in ein geeignetes Anwendungsumfeld vorgenommen werden. Abschnitt 7.1 diskutiert verschiedene Möglichkeiten zur technischen Weiterentwicklung des Konzepts. Diese Arbeit konzentriert sich auf die technischen Aspekte bei der Vereinfachung formularbasierter Benutzerschnittstellen. Aspekte wie Verwendbarkeit der Anwenderunterstützung und Akzeptanz der Konfigurationsmöglichkeiten blieben unberücksichtigt. Abschnitt 7.2 diskutiert offengebliebene wissenschaftliche Fragestellungen des Konzepts.

7.1 Technische Weiterentwicklung

Um grafische Benutzerschnittstellen komfortabel zu entwickeln, werden heute sogenannte WYSIWYG¹ Editoren verwendet. Diese grafischen Editoren bauen häufig auf dem Komponentenprinzip der unterliegenden Programmbibliothek auf. Abschnitt 7.1.1 argumentiert, wie die Komponenten der konfigurierbaren Benutzerschnittstelle in WYSIWYG Editoren integriert werden können. Die Komplexität der Strukturanalyse ist abhängig von der Größe der Elementgruppen. Abschnitt 7.1.2 diskutiert die in der Praxis zu erwartenden Gruppengrößen.

¹„what you see is what you get“

7.1.1 Integration erweiterter Komponenten in Entwicklungsplattformen

Komponenten können aufgrund der Analysierbarkeit ihres Zustandes in Entwicklungswerkzeuge integriert werden. Auch konfigurierbare Benutzerschnittstellen können als Komponenten eingebettet werden. In Java findet das nachfolgend beschriebene Verfahren Anwendung.

Mit Hilfe des *PropertySheet* können die Eigenschaften (*Properties*) einer Java-Bean [33] grafisch eingesehen und modifiziert werden. Es zeigt nur die *Properties* an, zu denen *PropertyEditor* Implementierungen existieren. Ein *PropertySheet* wird mit dem Standardkonstruktor erzeugt und mit einem beliebigen JavaBeans-Objekt konfiguriert (*setBeanObject*). Die angezeigte JavaBean kann zu jeder Zeit mit dieser Methode ausgetauscht werden. Nachdem eine JavaBean konfiguriert wurde, kann ein erzeugtes *JPanel*-Objekt mit *getSheetPanel* erfragt werden und beispielsweise in einem eigenen Fenster angezeigt werden. Auf die angezeigten Eigenschaften kann dadurch Einfluß genommen werden, daß ein Filter eingesetzt wird (*setFilter*), der aus der Gesamtmenge der Eigenschaften eine Untermenge extrahiert. Hierfür wird die Schnittstelle *PropertyFilter* implementiert. Sie definiert die Basisfunktionalität aller Filter.

Für jede Eigenschaft gibt es einen eigenen Eintrag, der in der inneren Klasse *PropertyEntry* abgebildet ist. Jeder Eintrag besitzt einen Zeiger auf den *PropertyDescriptor*, der durch die Klasse *BeanIntrospector* erfragt wird und Informationen über die Eigenschaft angibt, einen *PropertyEditor* und eine Komponente (*Component*), die auf den grafischen Editor der Eigenschaft verweist. Üblicherweise werden nur *PropertyEditor* Implementierungen für einfache Datentypen bereitgestellt, sowie für *Color* und *Font*. Dabei stehen nur für die letzten beiden auch eine graphische Komponente zur Verfügung, die direkt angezeigt werden kann. Für alle anderen muß zunächst ein *JTextField* mit dem entsprechenden Hörer vom Typ *ActionListener* erzeugt werden, der den *PropertyEditor* durch *setValue* über eine Wertänderung informiert. Nur für boolesche Werte wird eine nicht editierbare *ComboBox* Implementierung mit äquivalenter Funktionalität bereitgestellt. Allgemein stellt ein *PropertyEditor* nur einen Wertespeicher dar, der den aktuellen Wert einer Eigenschaft verwaltet. Dieser Wert kann mit *getValue* erfragt und mit *setValue* modifiziert werden. Bei jedem *PropertyEditor* wird ein Hörer vom Typ *PropertyChangeListener* registriert, der über Veränderungen durch *setValue* informiert wird und die entsprechende *set*-Methode der JavaBean auslöst.

Die Integration der Look_AS^ILike Implementierung in Entwicklungswerkzeuge wie die NetBeans Plattform [34] erfolgt durch die Implementierung eines *PropertySheet*. JavaBeans lassen sich mit *PropertySheet* Implementierungen konfigurieren. Abbildung 7.1 zeigt das automatisch generierte *JPanel* der Kontrollkomponente *JComboBoxControl*. Beispielhaft wurden hier die Methoden *editable* und *enabled* der Spezialisierung grün markiert. Kontrollkomponenten können also wie Komponenten der unterliegenden Programmbibliothek in einem automatisch erzeugten Editor konfiguriert werden. Konfigurierbare Elemente, Gestalten und Berechnungen lassen sich somit als JavaBeans in grafische Entwicklungswerkzeuge integrieren.

7.1 Technische Weiterentwicklung



Abbildung 7.1: WYSIWYG Editor einer Gestalt

7.1.2 Begrenzung des Rechenaufwandes

Die Unterstützung des Anwenders basiert auf der Beurteilung potenzieller Interaktionsfolgen einer Benutzerschnittstelle. Der Aufwand zur Berechnung aller Permutationen steigt mit zunehmender Elementzahl jedoch stärker als die Entwicklung der Rechenleistung. Es ist daher nicht anzunehmen, daß sich die maximale Größe einer analysierbaren Gruppe wesentlich erhöht. Für den praktischen Einsatz konfigurierbarer Benutzerschnittstellen könnten Daten zur Beurteilung möglicher Elementstrukturen zentral gespeichert und mittels effizienter Suchverfahren mit geringer Latenzzeit zur Verfügung gestellt werden. Die Pfadmenge des Abhängigkeitsgraphen könnte beispielsweise von einer Datenbank erfragt werden, anstatt in der Initialisierungsphase der konfigurierbaren Benutzerschnittstelle berechnet zu werden. Relevante Pfade, die zur Beurteilung eines vom Anwender modifizierten Formulars benötigt werden, können

effizient gesucht werden. Ein Formular mit n Feldern besitzt $n!$ Varianten. Sortiert und indiziert man die Pfadmengen nach ihren Varianten, so besitzt der Zugriff die Komplexität $O(\log_2(n!))$. Der Index der Sortierung könnte beispielsweise ein Hashcode sein,

n	$n!$	$\log_2(n!)$
3	6	2,6
4	24	4,6
5	120	6,9
6	720	9,5
7	5040	12,3
8	40320	15,3
9	362.880	18,5
10	3.628.800	21,8

Tabelle 7.1: Aufwand der Datensuche abhängig von der Gruppengröße

der aus der Reihenfolge der Formularfelder berechnet wurde. Tabelle 7.1 zeigt die Entwicklung der Komplexität abhängig von der Gruppengröße n . Selbst bei einer Gruppe mit 10 Formularfeldern benötigt die Datenbank 22 Zugriffe bei der Suche. Der Aufwand steigt also bei geeigneter Organisation der Datenbank quasi linear mit der Größe der Elementgruppe.

7.2 Ausblick

Die Akzeptanz durch den Anwender stellt die wichtigste Herausforderung für den Erfolg einer Benutzerschnittstelle dar. Sie wird unter anderem durch Nützlichkeit (*engl. usefulness*) und Verwendbarkeit (*engl. usability*) erreicht. Um das Potenzial der Änderungsmöglichkeiten einer konfigurierbaren Benutzerschnittstelle nutzen zu können, muß der Anwender zunächst aus den Zusammenhängen innerhalb des Formulars einfache Vereinfachungsmöglichkeiten für seinen Anwendungsfall verstehen. Abschnitt 7.2.1 diskutiert die hierfür notwendigen kognitiven Fähigkeiten. Die Ergebnisse dieser Arbeit können nicht nur für grafische Benutzerschnittstellen verwendet werden. Vielmehr werden heute Daten in verschiedenen Formen erfaßt. Abschnitt 7.2.2 zeigt ein Beispiel, wie konfigurierbare Benutzerschnittstellen zur Vereinfachung der Datenerfassung mit einem Spracherkennungssystem angewendet werden können. Abschnitt 7.2.3 diskutiert abschließend Aspekte der Verwendbarkeit der konfigurierbaren Benutzerschnittstelle.

7.2.1 Nutzung kognitiver Fähigkeiten des Anwenders

Strategien zur Vereinfachungen von Dateneingaben basieren auf dem Prozeßmodell des Anwenders. Insbesondere entwickelt der Benutzer aus seinem Verständnis ein System von Abhängigkeiten der Daten. Die Frage, wie Anwender aus ihrem mentalen Modell die Vorschläge der konfigurierbaren Benutzerschnittstelle zur Modifikation der

7.2 Ausblick

Struktur nachvollziehen können, bleibt hier unbeantwortet. Ein erster Ansatz bestünde darin, erfahrenen Anwendern durch Trainieren der Vorgehensweise Hilfestellung zu leisten. Hierbei muß der Anwender erlernen, ein mentales Modell eines Formulars auf eine Formularstruktur abzubilden und Abhängigkeiten zu erkennen.

7.2.2 Post-WIMP: flexible Spracherkennung

Die Vereinfachung von Dateneingaben mit konfigurierbaren Benutzerschnittstellen kann auch für Post-WIMP [39] Benutzerschnittstellen verwendet werden. Eine telefonische, computergesteuerte Fahrplanauskunft erfragt benötigte Verbindungsdaten wie Abfahrtsbahnhof, Zielbahnhof, Klasse, Beförderungsmittel, etc. Das System legt somit eine Interaktionsfolge fest, die der Formulareingabe mit einer grafischen Benutzerschnittstelle entspricht. Fokuswechsel zwischen Interaktionselementen besitzen bei Spracherkennungssystemen kein Äquivalent. Die dynamische Vorbelegung abhängiger Elemente, die Nutzung von Filtern und Konsistenzprüfung erhaltener Informationen ermöglichen gezieltes Rückfragen fehlender Daten. Diese Technik kann in Verbindung mit flexibler Analyse beliebiger Informationsfolgen die Implementierung eines intuitiv bedienbaren Auskunftssystems ermöglichen.

Beispiel

Folgender Dialog zwischen Kunde und Spracherkennungssystem liefert mit zwei Sätzen (Interaktionen) die notwendigen Informationen für Fahrplanauskunft.

Kunde: *Ich möchte am Mittwoch nächster Woche früh morgens rauchfrei und bequem von Karlsruhe nach Siegen fahren.*

Auskunft: *Möchten Sie ICE fahren?*

Kunde: *Nein.*

Die erforderliche Rechenleistung liegt sicherlich im wesentlichen bei der flexiblen Spracherkennung. Im optimalen Fall erkennt das System, daß der Kunde Nichtraucher ist und erste Klasse fahren möchte. Früh morgens impliziert eine Abfahrtszeit zwischen 6 und 9 Uhr. Für eine Auskunft erfragt das System abschließend nur noch die Zugkategorie. Der wesentliche Vorteil dieses Prinzips besteht darin, daß der Anwender Informationen seines mentalen Modells in einer selbstdefinierten Interaktionsfolge angeben und so das System nach seinen Vorstellungen bedienen kann.

7.2.3 Aspekte der Verwendbarkeit

Die Verwendbarkeit und Nützlichkeit konfigurierbarer Benutzerschnittstellen hängt unmittelbar von ihrer Umsetzung für eine konkrete Technologie in ein konkretes Anwendungsumfeld ab. Für den Prototypen Look_{AS}^ILike wurden Werkzeuge entwickelt, um die Machbarkeit mit theoriebasierter Analyse zu evaluieren.

Die Verwendbarkeit der Werkzeuge sollte ergänzend zu dieser Arbeit in einer umfassenden empirischen Untersuchung geprüft werden. Hierfür ist es notwendig, die Werkzeuge zunächst an die Fähigkeiten der Anwender anzupassen. Bei der Untersuchung muß weiterhin evaluiert werden, wie die Anpassung der Formulare von Anwender leicht vorgenommen werden kann. Die Anpassung in der laufenden Anwendung mittels Drag-and-Drop kann die Anwender hier durchaus überfordern. Eine

Variante wäre das Vorschlagen neuer Strukturen mit Miniaturisierungen. Anwender könnten sich so ein Bild davon machen, wie alternative Formulare aussehen würden. Dabei muß auch ein Verfahren gefunden werden, Anwendern diese Änderungsvorschläge nachvollziehbar zu erläutern. Im Mißerfolgsfall entartet die konfigurierbare Benutzerschnittstelle zu einer selbstanpassenden Benutzerschnittstelle. Für selbstanpassende Benutzerschnittstellen wurden die Gründe der mangelnden Akzeptanz bereits in anderen Arbeiten evaluiert².

Eine wichtige Grundlage zur Beurteilung der Formularstruktur ist die Annahme der Eingabefolge. In dieser Arbeit wurde eine Folge berechnet, die sich an der Position der Formularfelder orientiert. Eine Verbesserung der Strukturanalyse könnte durch intensives Beobachten der Benutzung erreicht werden. Insbesondere sollte die Analyse von Interaktionsmustern mit der visuellen Gestaltung der Benutzerschnittstelle verknüpft werden. Eine weitere Rolle spielt bei dieser Betrachtung die Entfernung einzelner Intaktionen mit der Benutzerschnittstelle.

Der Erfolg einer konkreten Umsetzung sollte empirisch untersucht werden. Manber [30] hat mit der Evaluierung des MyYahoo! Portals eine ähnliche Untersuchung vorgenommen.

²vgl. Abschnitt 2.1.4.2.

Anhang A

Details der Implementierung

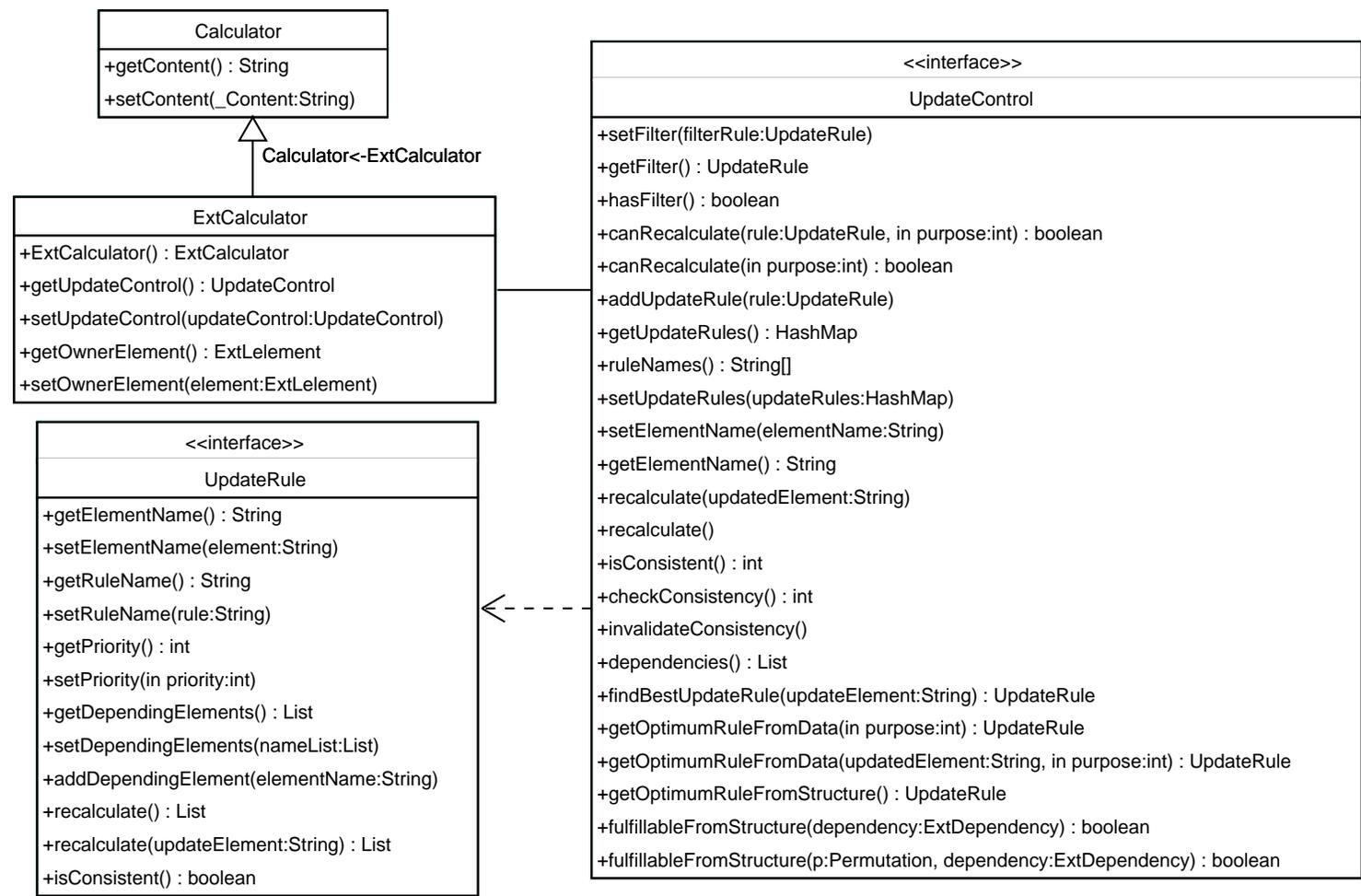
Eine Implementierung konfigurierbarer Benutzerschnittstellen für eine konkrete komponentenbasierte Programmibliothek erfordert die Spezialisierung des technologie-neutralen Klassenmodells. Die Programmierung erfolgt dann vergleichbar mit der ursprünglichen Technologie. Insbesondere können Elemente und Gestalten in Entwicklungswerkzeuge integriert werden. Der Prototyp `LookAsLike` implementiert konfigurierbare Benutzerschnittstellen für Java und Swing [51]. Diese Umsetzung benötigt zunächst eine verwaltende Instanz, das Dokument. Diese koordiniert Spezialisierungen von Element, Gestalt und Berechnung – insbesondere das Speichern und Laden der Benutzerschnittstelle. Die Bewertung der Struktur auf Basis des Interaktionsverhaltens und die daraus resultierende Unterstützung des Anwenders hängt im wesentlichen von der Benutzerkategorie und dem Einsatzgebiet der konfigurierbaren Benutzerschnittstelle ab. Techniken zur Analyse und Unterstützung werden anhand eines beispielhaften Werkzeugs erläutert. In der praktischen Anwendung werden die Analyseergebnisse in Systeme zur Gestaltung und Komplexitätsbewertung integriert.

A.1 Auswahlmenge und Konsistenzprüfung

Die Berechnung wird als Komponente in das konfigurierbare Element eingebettet. Sie besteht aus einem Verwalter `UpdateControl` und den Implementierungen der Abhängigkeiten, sog. Regeln (`UpdateRule`). Diese enthalten Algorithmen zur Berechnung der Auswahlmenge und der Konsistenzprüfung eines konfigurierbaren Elements und bilden die Grundlage zur Erzeugung des Abhängigkeitsgraphen aus der Implementierung heraus. Die Klasse `Calculator` enthält die Rohdaten der Berechnung im Attribut `content`. Beim Erzeugen des `UpdateControl` Objekts werden für die Elementstruktur verwaltende Informationen benötigt, die beim Instanzieren der Berechnung bereitgestellt werden. Diese Informationen werden in der Unterklasse `ExtCalculator` implementiert. Diese besitzt einen Verweis auf das zugehörige konfigurierbare Element (`ExtLelement`) und eine Referenz auf die erzeugte Implementierung des Verwalters.

Die Schnittstelle `UpdateControl` verwaltet die Regeln der Berechnung von Auswahlmenge und Konsistenzprüfung. Aus einer Regel `UpdateRule` wird die Liste vorausgesetzter Elemente extrahiert (`getDependingElements`). `addDependingElement` und `setDependingElements` werden zur Erzeugung der Implementierung aus den

Abbildung A.1: Die Berechnung der Auswahlmenge und der Konsistenzprüfung



A.1 Auswahlmenge und Konsistenzprüfung

Rohdaten verwendet. Das Attribut *priority* definiert die Güte der Abhängigkeit, die als Kantengewicht im Abhängigkeitsgraphen¹ notiert wird. Eine Regel besitzt einen logischen Namen (*getRuleName*, *setRuleName*) und einen Verweis auf den logischen Namen des assoziierten Elements (*setElementName*, *getElementName*). *recalculate* berechnet die Auswahlmenge allgemein oder mit der Information, welches Element einen neuen Wert erhalten hat. Damit kann der neue Elementwert gezielt in die Konsistenzprüfung einbezogen werden, da nur Regeln, die das geänderte Element voraussetzen, eine unterschiedliche Auswahlmenge bzw. Konsistenzaussage besitzen können. Eine Regel ist berechenbar, wenn ihre vorausgesetzten Elemente einen Wert besitzen. *isConsistent* liefert die Konsistenzaussage des aktuellen Eingabezustands. Im Falle eines konsistenten Elements müssen alle zugehörigen Regeln die Konsistenzfrage positiv beantwortet haben. Die Auswahlmengen können je nach Eingabezustand variieren. Insbesondere gibt die Güte keine Auskunft über die Größe der Ergebnismenge. Im Falle von einfach selektierbaren vorausgesetzten Elementen liefert die Wertemenge für ein einzeln selektierbares abhängiges Element für jede Selektion eine positive Konsistenzaussage. Für Mehrfachselektion gilt diese Aussage nicht, sondern hängt von der konkreten Implementierung der Regeln ab.

Die Schnittstelle *UpdateControl* wählt die zu verwendenden Regeln automatisch anhand der Güteaussagen der Regeln. Um die bestmögliche Regel zu identifizieren wird für die Güte eine Kleiner-Relation definiert. Die Methode *canRecalculate* prüft, ob eine berechenbare Regel existiert, bzw. eine konkrete Regel berechenbar ist. Die Konsistenzaussagen *isConsistent* und *checkConsistency* kennen drei Zustände:

- konsistent,
- nicht konsistent und
- nicht berechenbar.

Die Methoden liefern das Ergebnis *nicht berechenbar*, wenn keine Regel berechenbar ist. Die Methode *checkConsistency* löst eine Neuberechnung aus, während *isConsistent* das Ergebnis der letzten Konsistenzprüfung liefert und erst dann die Berechnung auslöst, wenn kein früheres Ergebnis existiert. Die Konsistenz kann explizit mit *invalidateConsistency* invalidiert werden. Die Methoden *addUpdateRule*, *setUpdateRules* und *getUpdateRules* werden zur Instanziierung, Speicherung und Analyse des Verwalters benötigt. Der Verwalter berechnet die bestmögliche Regel *getBestUpdateRule*, die bestmögliche Regel für die Struktur *getOptimumRuleFromStructure* und die bestmögliche Regel der aktuellen Eingabesituation *getOptimumRuleFromData*. Analoge Prüfungen können für Abhängigkeiten des Abhängigkeitsgraphen aus den Regeln berechnet werden (*fulfillableFromStructure*, *fulfillableFromData*). *recalculate* des Verwalters löst *recalculate* der bestmöglichen Regel *getOptimumRuleFromData* aus. Die Methode *getRuleNames* liefert die Namensliste der implementierten Regeln. Eine Regel des Verwalters nimmt eine Sonderstellung ein: der Filter. Für jeden Verwalter kann ein Filter implementiert werden, der die Auswahlmenge auf eine Teilmenge einschränken kann, ohne dafür Werte anderer Elemente zu verwenden. Die Konsistenzaussage *isConsistent* eines Filters liefert selbstverständlich nicht konsistent (*false*). Mit *hasFilter* kann geprüft werden, ob ein Filter eingebettet wurde. *setFilter* und *getFilter* werden zur Instanziierung bzw. Speicherung des Filters benötigt. Für die Regel kann ein Platzhalter definiert werden. Ist im neuen Wert eines Elements ein Platzhalter vorhanden, wird die

¹Beispiel: siehe Abschnitt 4.3

Auswahlmenge mit dem Filter neu berechnet. Andernfalls wurde die Neuberechnung der Auswahlmenge der abhängigen Elemente bei einem Verwalter ausgelöst.

A.2 Die Erzeugung des Klassenmodells

Die *Java Architecture for XML Binding (JAXB)* [35] bietet eine einfache, bidirektionale Abbildung zwischen XML Daten und Java-Klassen. Der zugehörige Übersetzer generiert aus einer gegebenen Document Type Definition (DTD) [9] und einer Schema Datei, einem sogenannten Binding Schema [10], die Hilfestellung bei der Erzeugung leistet, eine vollständige Klassenhierarchie zum Lesen, Bearbeiten, Validieren und Schreiben eines XML Dokuments. Mit dieser Abbildung lassen sich XML Daten ohne Programmieraufwand als Java-Objekte mit Daten und zugehörigen Methoden behandeln. Die Korrespondenz zwischen Klassenmodell und DTD zeigte sich in der Praxis wesentlich performanter und speichereffizienter als ereignisbasierte Zerstückeler wie *Simple API for XML (SAX)* [38] oder *Document Object Model (DOM)* [8]. Das Binding Schema erlaubt, Daten auf korrespondierende Datentypen abzubilden, Aufzählungstypen zu erzeugen, Schnittstellen zu implementieren und Regeln zur Erzeugung von Klassen einzuhalten. Spezialisierungen können als Unterklassen in die Bearbeitung der XML Daten integriert werden. Im Fall eines trivialen (leeren) Schemas erzeugt der Übersetzer eine separate Klasse für jeden Knoten. Innerhalb einer Klasse erzeugt der Übersetzer sog. *Properties*, die Methoden für den Zugriff auf die Kindelemente und deren Daten bereitstellen. Die Datentypen dieser Methoden hängen von den im XML Schema angegebenen Einstellungen zur Erzeugung der Kindelemente ab. Alle Elemente, die *#PCDATA* [9] enthalten, werden auf Zeichenketten abgebildet. Namensräume in XML Dokumenten werden nicht unterstützt.

A.3 Das Klassenmodell der konfigurierbaren Benutzerschnittstelle

Die Modellierung konfigurierbarer Elemente wird im folgenden zum Klassenmodell der konfigurierbaren Benutzerschnittstelle kombiniert. Hierfür wird eine Verwaltung der Elemente und der Strukturanalyse benötigt – das Dokument. Abbildung A.2 zeigt seine Modellierung. Es erzeugt selbständig den Kontrollflußgraphen und den Abhängigkeitsgraphen aus der Implementierung der konfigurierbaren Elemente.

Analyse und Interaktion mit dem Benutzer wurden in der Klasse *StructureOptimizer* implementiert. Die Analyse der Regeln und des Eingabezustands der Benutzerschnittstelle liefert der *PreemptionAnalyzer*.

Das Dokument verwaltet alle in der konfigurierbaren Benutzerschnittstelle enthaltenen Elemente. Diese wurden für Look_As^ILike in *ExtLelement* für Swing [51] spezialisiert. Das Dokument läßt sich leer oder mit Angabe eines Dateinamens erzeugen. Im ersten Fall erzeugt der Programmierer in der Entwicklungsphase eine konfigurierbare Benutzerschnittstelle aus einer Implementierung und speichert sie in einer Konfigurationsdatei. Im zweiten Fall lädt der Benutzer eine konfigurierbare Benutzerschnittstelle aus dieser Datei. Das vom Programmierer erzeugte Dokument generiert initial ein Eingangs- und ein Ausgangselement. Der Programmierer erzeugt

A.3 Das Klassenmodell der konfigurierbaren Benutzerschnittstelle

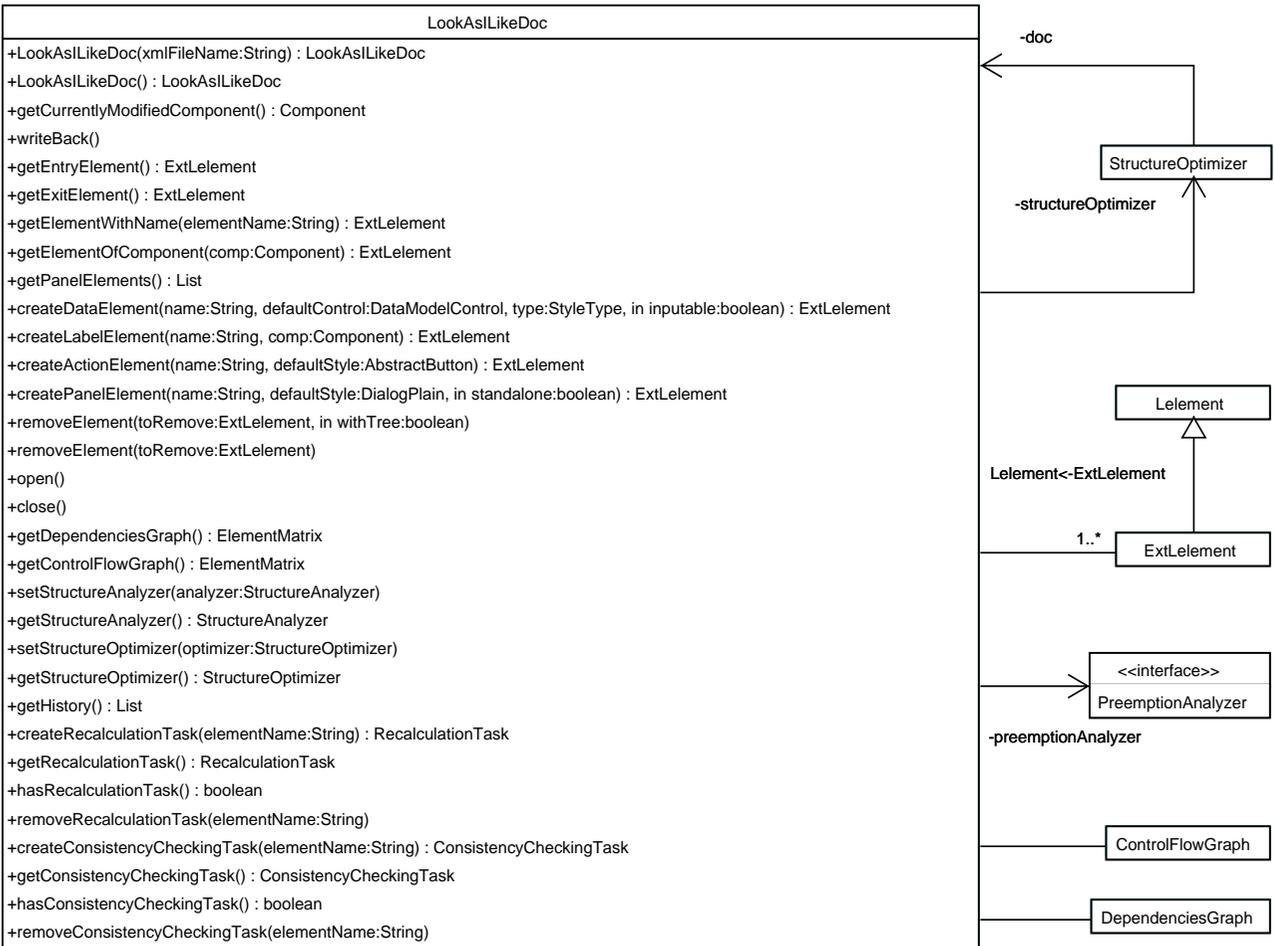


Abbildung A.2: Dokument einer konfigurierbaren Benutzerschnittstelle

die weitere Elementstruktur, die an Eingang und Ausgang geknüpft wird (*getEntryElement*, *getExitElement*). Für jeden Elementtyp wird eine Methode zur Erzeugung angeboten: *createDataElement*, *createActionElement*, *createPanelElement*, *createLabelElement*. Alle *create*-Methoden erhalten einen logischen Namen und eine Gestalt. Datenelemente können nach Erzeugung noch weitere Gestalten über Methoden von *ExtLelement* erhalten. Elemente werden mit *removeElement* aus der Struktur gelöscht. Hierbei werden entweder alle Kindelemente gelöscht oder die Lücke in der Struktur geschlossen. Die Methode *open* zeigt die Benutzerschnittstelle an, *close* zerstört sie. Mit *writeBack* wird die momentane Konfiguration der Benutzerschnittstelle zu einem beliebigen Zeitpunkt in einer Datei gespeichert. Die Methode *getPanelElements* liefert Referenzen zu allen Fensterelementen der Benutzerschnittstelle. Das Dokument besitzt eine Registratur aller Elemente der Benutzerschnittstelle. Diese werden entweder eindeutig über ihre logischen Namen (*getElementWithName*) oder jede ihrer Gestalten (*getElementOfComponent*) identifiziert. Gestalten müssen auf Interaktionen reagieren können und das Element benachrichtigen, das sie repräsentieren. *getElementOfComponent* nimmt diese Abbildung vor.

A.4 Konfigurierbare Elemente mit Swing

Abbildung A.3 zeigt das Binding-Schema eines konfigurierbaren Elements. Die semantische Prüfung der Datenstruktur muß vom Dokument (*LookAsILikeDoc*) geleistet werden. Das Binding Schema konfiguriert den JAXB-Übersetzer wie nachfolgend beschrieben. Die erzeugten Klassen werden dem Paket *xmlstructure.jaxbgenerated* zugeordnet, d.h. es wird der entsprechende *package*-Eintrag in der Klasse vorgenommen und die benötigte Verzeichnisstruktur erzeugt. Aus dem Element *elementlist* wird eine Klasse erzeugt, die die Wurzelklasse der Datenstruktur bildet. Aus dem Attribut *type* des XML Elements *lelement* wird der Aufzählungstyp *IElementType* erzeugt. Dieser Typ klassifiziert die konfigurierbaren Elemente und kennt die Werte *entry*, *exit*, *data*, *panel*, *action* und *label*. Für sie wird die Klasse *IElementType* mit entsprechenden Konstanten erzeugt. Für das XML Element *behaviour* wird eine Klasse erzeugt, die das Attribut *standalone* auf den Typ *ChoiceType* und *standalone* auf *OpeningType* abbildet. *ChoiceType* implementiert eine mehrfach verwendete ja/nein-Entscheidung. *OpeningType* definiert das Öffnen eines Fensters: parallel, modal oder ersetzend. Für das XML Element *configuration* wird das Attribut *format* auf den Typ *ConfFormatType* abgebildet, der Elementwert in Text- oder Zahlenwerte unterscheidet. Die Position wird in Form von Koordinaten gespeichert. Hierfür werden X- und Y-Koordinate in den primitiven Datentyp *int* gewandelt. Die Gestalten (*style*) werden mit dem Attribut *type* definiert. Der Aufzählungstyp *StyleType* definiert drei Gestalten: *single*, *medium* und *large*.

Die Speicherung von Rohdaten (*#PCDATA*) wird weder in der DTD noch im Binding-Schema genauer definiert. Ihre Interpretation von *style* und *calculator* unterliegt explizit der Implementierung des Dokuments.

Das technologieneutrale Klassenmodell wurde in Abbildung A.16 gezeigt. JAXB erzeugt in diesen Klassen zusätzlich Methoden zum Speichern, Laden und zur Prüfung der XML Daten. Jede von JAXB erzeugte Klasse erhält diesen Methodensatz. Beim Laden, Speichern oder Prüfen der Daten werden im Objektbaum, der äquivalent zum Elementbaum der XML Daten ist, die entsprechenden Methoden *marshal*, *unmarshal*

A.4 Konfigurierbare Elemente mit Swing

```
<?xml version="1.0" encoding="UTF-8" ?>
<xml-java-binding-schema version="1.0-ea">
  <options package="xmlstructure.jaxbgenerated"/>
  <element name="elementlist" type="class" root="true"/>
  <element name="lelement" type="class">
    <attribute name="type" convert="lElementType"/>
  </element>
  <element name="behaviour" type="class">
    <attribute name="standalone" convert="ChoiceType"/>
    <attribute name="openingmode" convert="OpeningType"/>
  </element>
  <element name="configuration" type="class">
    <attribute name="format" convert="ConfFormatType"/>
    <attribute name="multiselect" convert="ChoiceType"/>
    <attribute name="inputable" convert="ChoiceType"/>
    <attribute name="enabled" convert="ChoiceType"/>
    <attribute name="fixeditems" convert="ChoiceType"/>
  </element>
  <element name="item" type="class">
    <attribute name="default" convert="ChoiceType"/>
  </element>
  <element name="position" type="class">
    <attribute name="xpos" convert="int"/>
    <attribute name="ypos" convert="int"/>
  </element>
  <element name="style" type="class">
    <attribute name="type" convert="StyleType"/>
    <attribute name="width" convert="int"/>
    <attribute name="height" convert="int"/>
    <attribute name="default" convert="ChoiceType"/>
    <attribute name="minimumitemnumber" convert="int"/>
  </element>
  <element name="calculator" type="class"/>
  <enumeration name="lElementType"
    members="entry exit data panel action label"/>
  <enumeration name="ConfFormatType" members="text number"/>
  <enumeration name="StyleType" members="large medium single"/>
  <enumeration name="OpeningType"
    members="modal parallel replace"/>
  <enumeration name="ChoiceType" members="yes no"/>
</xml-java-binding-schema>
```

Abbildung A.3: Binding Schema für konfigurierbare Elemente

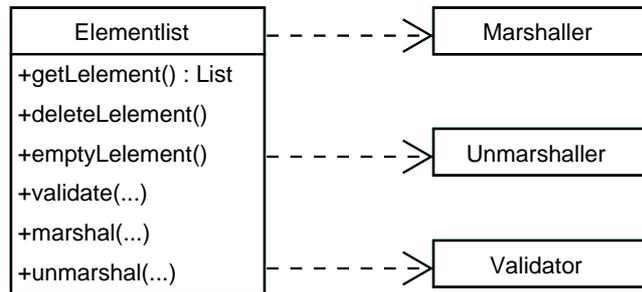


Abbildung A.4: JAXB-Methoden für Laden, Speichern und Prüfen

und *validate* aufgerufen. Abbildung A.4 zeigt die Methoden und referenzierten JAXB-Klassen am Beispiel der Klasse *Elementlist*.

A.4.1 Die Klasse *ExtLelement*

Abbildung A.5 zeigt das spezialisierte Modell eines konfigurierbaren Elements. *ExtLelement* spezialisiert *Lelement* um Methoden zur Verknüpfung mit dem Dokument, der Programmlogik und mit verschiedenen Gestalten. *getLookAsILikeDoc* liefert einen Zeiger auf das zugeordnete Dokument. Mit *addChild* werden Elemente miteinander verknüpft. Der Parameter *openingMode* wird für Fensterelemente verwendet und kann mit *getOpeningMode* erfragt werden. *getChildren* liefert die Liste der Kindelemente, *getParent* das Väterelement. Dieses kann der Eingang, ein Fensterelement oder ein Aktionselement sein. Baut man die Elementstruktur aus der Vaterbeziehung der Elemente auf, so erhält man einen zyklfreien Strukturbaum, der der Komponentenstruktur der korrespondierenden Swing-Benutzerschnittstelle entspricht. *removeChild* entfernt ein Kindelement und schließt dabei entweder die entstandene Lücke oder löscht den anhängenden Elementbaum. *setActive* aktiviert bzw. deaktiviert die Interaktion mit dem Element, *show* macht es sichtbar bzw. unsichtbar. Die Größe wird für die aktuell angezeigte Gestalt oder für einen konkreten Typ parametrisiert. Da die Gestalten unterschiedlich groß sein können, wird die Größe für jede Gestalt separat gespeichert. Die Position wird für alle Gestalten einheitlich festgelegt, um den Benutzer beim Wechsel der Gestalt nicht zu verwirren. *sortChildren* sortiert die Liste der Kindelemente nach Typ und Position wie in Abschnitt 4.4 beschrieben. Für den Vergleich der Elemente wurde die Klasse *ElementComparator* implementiert.

Die weiteren Methoden der Klasse *ExtLelement* werden für Datenelemente verwendet. Jedes Datenelement wird bei seiner Erzeugung vom Konstruktor mit einem Datenmodell ausgestattet, das die Auswahlmenge des Elements, seinen Selektionszustand, Mehrfachselektion und den Datentyp repräsentiert. Werte werden der Auswahlmenge mit *addItem* hinzugefügt und mit *removeItem* gelöscht. *emptyItems* löscht die Auswahlmenge, *updateItems* tauscht sie aus. *getModel* liefert das Datenmodell.

Außer dem Eingangs- und Ausgangselement wird jedem Elementtyp eine Gestalt, Datenelementen bis zu drei Gestalten zugeordnet. Diese spezialisieren beliebige

A.4 Konfigurierbare Elemente mit Swing

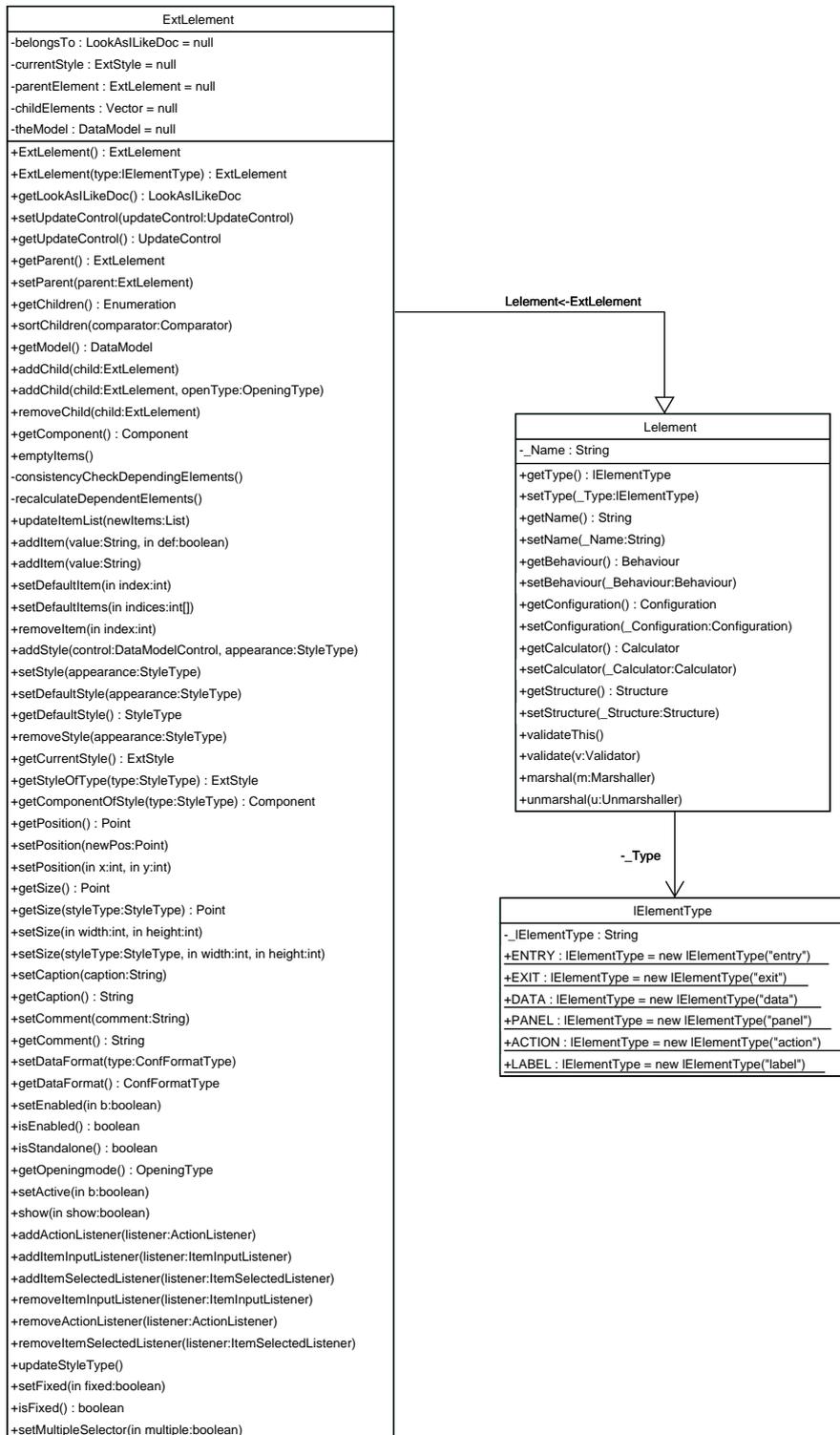


Abbildung A.5: Klassendiagramm konfigurierbarer Elemente

Swing-Komponenten und implementieren *DataModelControl*. *addStyle* fügt eine neue Gestalt mit Angabe seines Typs (*StyleType*) hinzu. Eine Gestalt wird als Standard erklärt und kann beim Element erfragt (*getDefaultStyle*) und modifiziert werden (*setDefaultStyle*). *getCurrentStyle* liefert die aktuell angezeigte Gestalt. Umgekehrt können Gestalten der einzelnen Typen (*getStyleOfType*) erfragt werden. *setStyle* ändert die Anzeige auf den übergebenen Gestalttypen. *updateStyleType* wechselt auf eine Gestalt, die der Größe der Auswahlmenge entspricht. Die Swing-Komponente der Gestalt wird mit *getComponent* bzw. *getComponentOfStyle* erfragt. Sie wird zur Anzeige in den Komponentenbaum der Swing-Benutzerschnittstelle eingefügt. Die Klasse *UpdateControl* implementiert die Berechnung des Elements. Sie wird zur Speicherung von *ExtCalculator* in ihre Rohdaten gewandelt. Elemente können für die Unterstützung des Benutzers eine Färbung und einen Hinweistext erhalten. *markElement* bzw. *unmarkElement* markieren die Gestalten.

Die Programmlogik kann Hörer (*listener*) bei jedem Element registrieren. Um unabhängig von der Implementierung der Gestalten zu sein, bietet Look_{AS}^ILike drei Hörer, die bei allen Implementierungen registrierbar sind. Swing bietet als allgemeinen Hörer den *ActionListener*. Er kann bei allen Elementen registriert werden. Zusätzlich kann auf die Neueingabe eines Wertes durch Registrierung eines *ItemInputListener* reagiert werden. Analog bietet der *ItemSelectedListener* die Reaktion auf die Selektion eines neuen Wertes der Auswahlmenge.

A.4.2 Die Klasse ExtStyle

Abbildung A.6 zeigt die Modellierung einer Gestalt. *ExtStyle* erweitert *Style* um Speichern, Validieren und Laden der Swing-Komponenten, die als Serialisierung im XML Element *style* gespeichert werden². Der Zugriff auf die Swing-Komponente (*graph-Component*) erfolgt mit *setComponent* bzw. *getComponent*. *ownerElement* referenziert das zugehörige Element. *syncXMLString* erzeugt die XML Serialisierung aus dem momentanen Zustand der Swing-Komponente. *syncComponent* erzeugt eine Swing-Komponente aus der XML Serialisierung. Hat sich der Zustand der Komponente in der Benutzerschnittstelle geändert, wird die XML Serialisierung invalidiert *markInvalid* und direkt oder beim Speichern der Benutzerschnittstelle als XML Datei neu erzeugt. *ExtStyle* überschreibt *validate*, *marshal* und *unmarshal*, um die XML Serialisierung bzw. Deserialisierung der Swing-Komponente auszulösen.

A.4.3 Die Klasse ExtCalculator

Abbildung A.7 zeigt das Modell der Berechnung. *ExtCalculator* erweitert *calculator* analog zu *ExtStyle*. Die Berechnung speichert den Verwalter (*UpdateControl*) und die zugehörigen Regeln (*UpdateRule*) als Serialisierung.

A.5 Die XML Serialisierung von Komponenten

Der Zustand eines Java-Objekts kann persistent gespeichert werden. Hierfür wird als Indikator die Schnittstelle *java.io.Serializable* implementiert. *Serializable* definiert weder Konstanten noch Methoden. Die tatsächliche Serialisierbarkeit eines Objekts hängt von der Serialisierbarkeit seiner Attribute ab. Die Serialisierung eines Objekts

²siehe Abschnitt A.5

A.5 Die XML Serialisierung von Komponenten

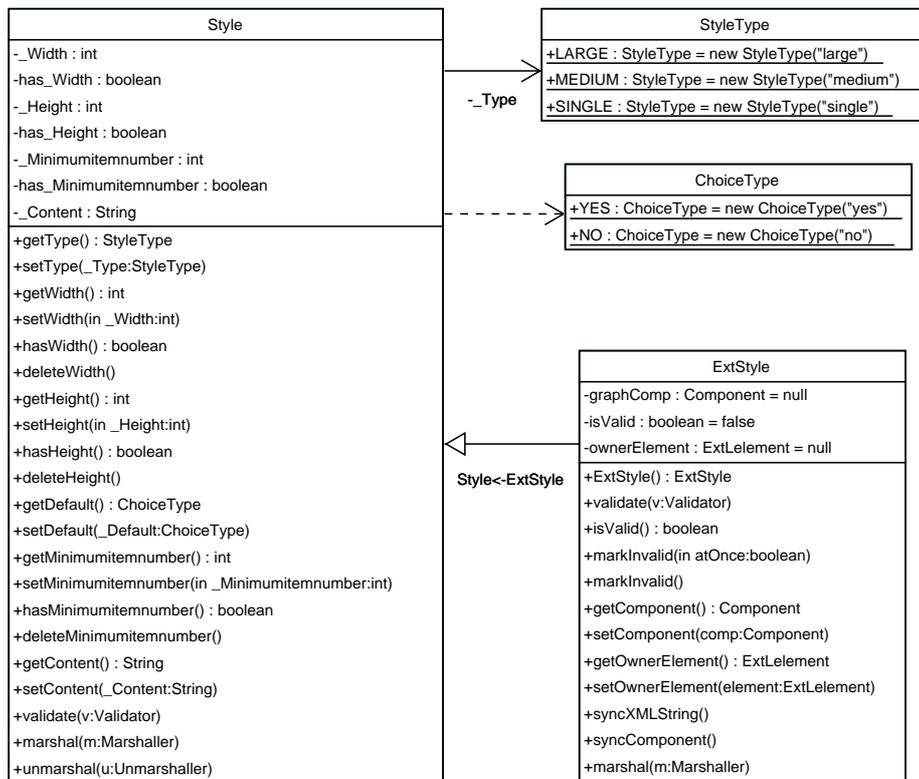


Abbildung A.6: Modell einer Gestalt

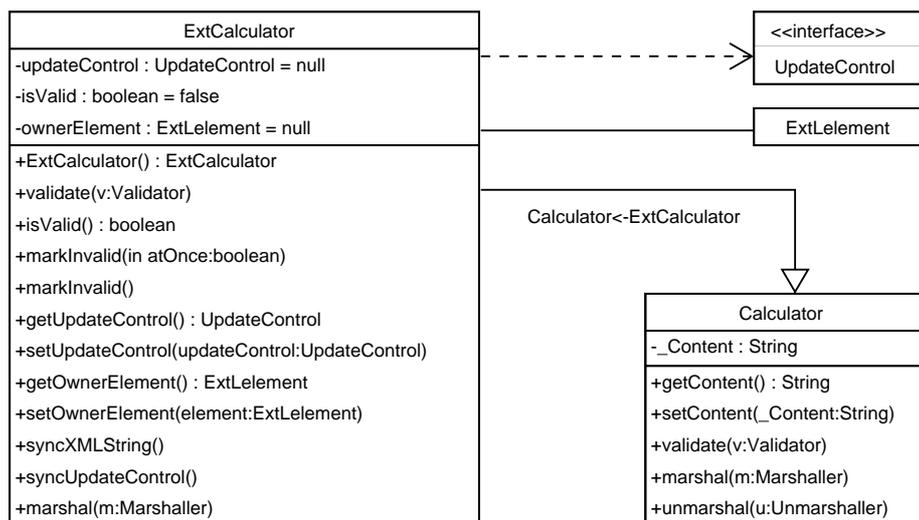


Abbildung A.7: Modell einer Berechnung

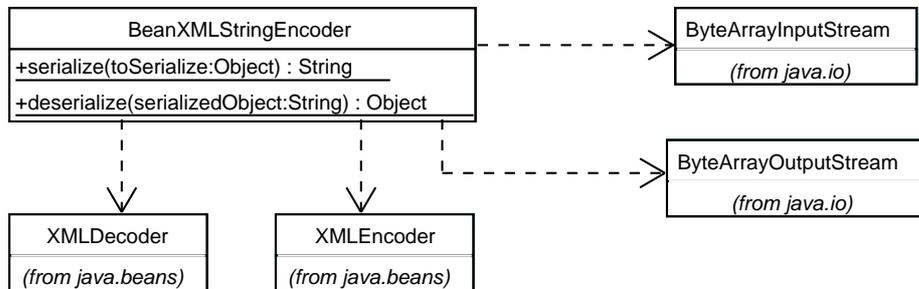


Abbildung A.8: XML Serialisierung für JavaBeans

```

<style type="medium" width="150" height="100"
default="no" minimumitemnumber="2">
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.4.2" class="java.beans.XMLDecoder">
  <object class=
    "lookasilikegui.controlcomponents.JListControl">
    <void property="selectionMode">
      <int>0</int>
    </void>
    <void method="addItemSelectedListener">
      <object class=
        "xmlstructure.datamodel.UpdateControlListener">
        <void property="elementName">
          <string>Wohnort</string>
        </void>
      </object>
    </void>
  </object>
</java>
</style>

```

Abbildung A.9: Beispiel einer XML-serialisierten Gestalt

A.6 Ein Programmbeispiel

speichert den Objekttyp und die Serialisierung seiner Attribute. JavaBeans [33] verwenden Serialisierung und verfeinern die Speicherung eines Objektzustandes durch eine wohldefinierte Zugriffsmethodik. Damit können Attribute anhand ihrer Zugriffsmethoden ohne Kenntnis der Implementierung verwendet werden. Java bietet zur Betrachtung von Objekten das Paket *Reflection API*.

Mit der Version 1.4 wurde die Programmbibliothek JavaBeans (*java.beans*) erheblich erweitert. `LookAsILike` nutzt die XML Serialisierung dieser Version. Die Klasse *XMLEncoder* speichert den Zustand einer JavaBean ähnlich wie eine Objektserialisierung, *XMLDecoder* erzeugt ein Objekt aus einer XML Serialisierung. Die `LookAsILike`-Klasse *BeanXMLStringEncoder* erzeugt eine *#PCDATA*-Zeichenkette aus der Serialisierung des XML Kodierers. Dafür werden die Sonderzeichen inklusive der Klammern der XML Elemente maskiert. Abbildung A.8 zeigt das Klassenmodell der XML Serialisierung, Abbildung A.9 die Serialisierung einer Gestalt. Diese wurde aus einem Objekt der Klasse *lookaslikegui.controlcomponents.JListControl* erzeugt. Bei der Deserialisierung wird ein Objekt mit dem Standardkonstruktor generiert. Die Konfiguration erfolgt durch formulierte Methodenaufrufe oder der Definition von Eigenschaften, die über entsprechende *set*-Methoden hergestellt werden.

Der Kalkulator verkörpert eine JavaBean. Objekte der Klasse *UpdateControl* und alle Regeln (*UpdateRule*) sind serialisierbar und mit dem Standardkonstruktor erzeugbar. Alle Eigenschaften der Berechnung lassen sich mit *get*-, *has*- und *is*-Methoden erfragen und *set*-Methoden konfigurieren.

A.6 Ein Programmbeispiel

Die Programmierweise von Swing und `LookAsILike` lässt sich miteinander vergleichen. `LookAsILike` bietet zusätzlich eine strukturierte Implementierung abhängiger Dateneingaben.

Eine `LookAsILike` Benutzerschnittstelle wird in folgenden Schritten implementiert:

Schritt 1: Neues `LookAsILike` Dokument erzeugen.

Schritt 2: Elemente implementieren und verknüpfen.

Schritt 3: Regeln implementieren und verknüpfen.

Schritt 4: Dokument speichern bzw. anzeigen.

Abbildung A.10, A.11, A.12 und A.13 zeigen einen Auszug aus der Implementierung der konfigurierbaren Benutzerschnittstelle in Abbildung 4.2.

In Abbildung A.10 wird ein neues Dokument, ein Fensterelement, ein Aktions-element und ein Markenelement implementiert. Ein `LookAsILike` Dokument wird mit dem Standardkonstruktor der Klasse *xmlstructure.LookAsIlikeDoc* erzeugt. Für das Fenster wird ein Objekt der Unterklasse von *jaxax.swing.JFrame* erzeugt: *GJFrame* erweitert *JFrame* um eine *Glasspane* Implementierung [51]. Die gläserne Schicht (*Glasspane*) ermöglicht dem Benutzer, konfigurierbare Elemente auf jedem Fenster beliebig zu positionieren und Konfigurationen der Gestalten vorzunehmen. Das erzeugte

```
// Create document and set filename

LookAsILikeDoc doc = new LookAsILikeDoc();
doc.setXMLFileName(myFilename);

// Create a frame

GJFrame gf1 = new GJFrame();
gf1.setLookAsILikeDoc(doc);
ExtLelement panell = doc.createPanelElement(
    "Adresseingabe 1", gf1, true);
panell.setPosition(200,200);
panell.setSize(600,400);
doc.getEntryElement().addChild(panell);

// Create a label

JLabel lb = new JLabel();
lb.setText("Wohnort");
ExtLelement label = doc.createLabelElement(
    "WohnortLabel",lb);
label.setPosition(250,420);
label.setSize(100,30);
panell.addChild(label);

// Create a button

JButton exit = new JButton("Ende");
ExitButtonActionListener exitListener =
    new ExitButtonActionListener();
exitListener.setLookAsILikeDoc(doc);
ExtLelement exitButton =
    doc.createActionElement("exitButton", exit);
exitButton.addActionListener(exitListener);
exitButton.setPosition(50,50);
exitButton.setSize(110,25);
exitButton.addChild(doc.getExitElement());
panell.addChild(exitButton);
```

Abbildung A.10: Dokument, Fenster, Marke und Aktionselement erzeugen

A.6 Ein Programmbeispiel

```
// Style SINGLE

TextFieldControl tc = new JTextFieldControl();
tc.setLookAsILikeDoc(doc);
ExtLelement el = doc.createDataElement(
    "Wohnort", tc, StyleType.SINGLE, false );
el.setPosition(250,450);
el.setSize(StyleType.SINGLE,170,25);

// Style MEDIUM

JListControl lc=new JListControl();
lc.setLookAsILikeDoc(doc);
el.addStyle(lc , StyleType.MEDIUM);
el.setSize(StyleType.MEDIUM,150,100);

// Style LARGE

JPanelControl pc = new JPanelControl();
ComboBoxControl cc = new JComboBoxControl();
pc.setInnerComponent(cc);
pc.setEditorMode(true);
pc.setLookAsILikeDoc(doc);
el.addStyle(pc, StyleType.LARGE);
el.setSize(StyleType.LARGE,170,30);

// element configuration

el.getStyleOfType(
    StyleType.MEDIUM).setMinimumitemnumber(2);
el.getStyleOfType(
    StyleType.LARGE).setMinimumitemnumber(6);
el.setDefaultStyle(StyleType.SINGLE);
panell.addChild(el);
```

Abbildung A.11: Datenelement *Wohnort* mit Gestalten erzeugen

```
UpdateRule dur;
UpdateControl duc = new DefaultUpdateControl();
duc.setElementName("Wohnort");

dur = new Wohnort_A_UpdateRule();
dur.setRuleName("Wohnort_A");
dur.setElementName("Wohnort");
dur.setLookAsILikeDoc(doc);
dur.setPriority(
    MetricConstant.PRIORITY_PREEMPTION_SINGLE_LIMITING);
dur.addDependingElement("Postleitzahl");
duc.addUpdateRule(dur);

dur = new Wohnort_B_UpdateRule();
dur.setRuleName("Wohnort_B");
dur.setElementName("Wohnort");
dur.setLookAsILikeDoc(doc);
dur.setPriority(
    MetricConstant.PRIORITY_PREEMPTION_SINGLE_LIMITING);
dur.addDependingElement("Vorwahl");
duc.addUpdateRule(dur);

dur = new Wohnort_C_UpdateRule();
dur.setRuleName("Wohnort_C");
dur.setElementName("Wohnort");
dur.setLookAsILikeDoc(doc);
dur.setPriority(
    MetricConstant.PRIORITY_PREEMPTION_MULTIPLE_UNIQUE);
dur.addDependingElement("Postleitzahl");
dur.addDependingElement("Strasse");
dur.addDependingElement("Hausnummer");
duc.addUpdateRule(dur);

doc.getElementWithName("Wohnort").setUpdateControl(duc);

doc.writeBack();
doc.open();
```

Abbildung A.12: Berechnung; Dokument speichern und anzeigen

A.6 Ein Programmbeispiel

```
public class Wohnort_A_UpdateRule extends DefaultUpdateRule {
    private AddressDatabase db;
    public Wohnort_A_UpdateRule() throws ClassNotFoundException {
        db=new AddressDatabase();
    }
    public String[] recalculate() {
        return recalculate(getElementName());
    }
    public String[] recalculate(String updateElement) {
        try {
            Connection con = db.getConnection();
            ExtLelement element1 =
                lookAsILikeDoc().getElementWithName("Postleitzahl");
            String item1 = element1.getModel().getCurrentValue();
            PreparedStatement ps = con.prepareStatement(
                "SELECT DISTINCT wohnort FROM dinfo96 WHERE "
                + "postleitzahl=? ORDER BY wohnort;");
            ps.setString(1,item1);
            ResultSet res = ps.executeQuery();
            Vector v = new Vector();
            while (res.next()) {v.add(res.getString(1));}
            res.close();ps.close();con.close();
            String[] items = new String[v.size()];
            v.copyInto(items); return items;
        } catch (Exception e) {e.printStackTrace();}
        return null;
    }
    public boolean isConsistent() {
        try {
            Connection con = db.getConnection();
            ExtLelement element1 =
                lookAsILikeDoc().getElementWithName("Wohnort");
            ExtLelement element2 =
                lookAsILikeDoc().getElementWithName("Postleitzahl");
            String item1 = element1.getModel().getCurrentValue();
            String item2 = element2.getModel().getCurrentValue();
            PreparedStatement ps = con.prepareStatement(
                "SELECT count (wohnort) FROM "
                + "dinfo96 WHERE wohnort=? and postleitzahl=?;");
            ps.setString(1,item1); ps.setString(2,item2);
            ResultSet res = ps.executeQuery(); res.next();
            int rows = res.getInt(1);
            res.close();ps.close();con.close();
            if (rows > 0) return true;
        } catch (Exception e) {e.printStackTrace();}
        return false;}}
}
```

Abbildung A.13: Die Regel *Wohnort_A*

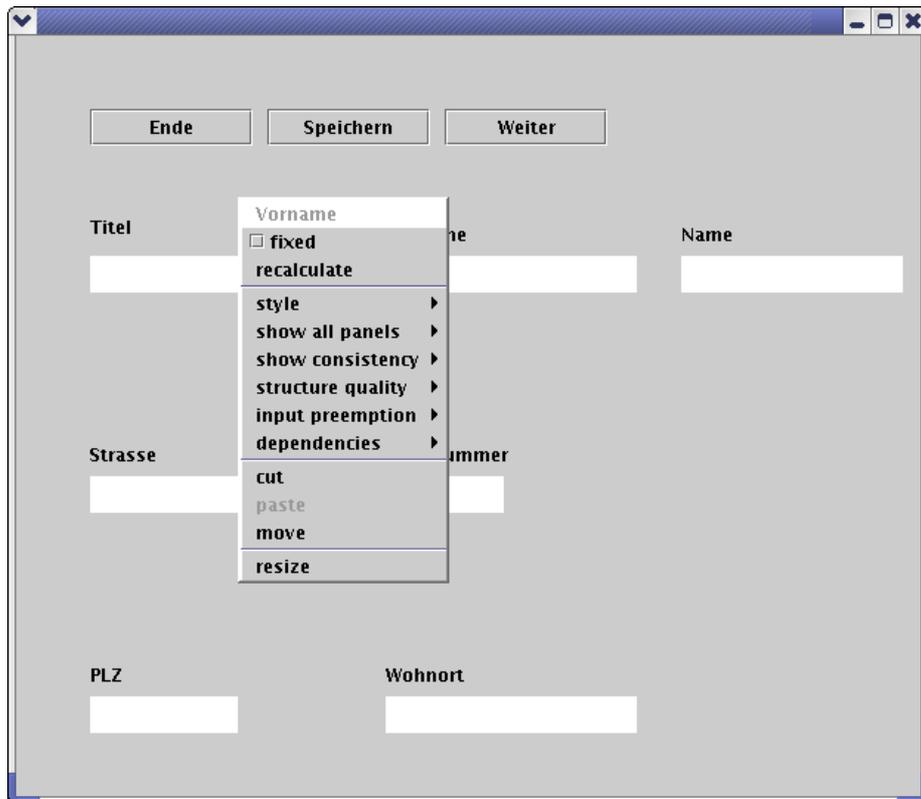


Abbildung A.14: Konfiguration eines Elements

A.6 Ein Programmbeispiel

Element *Adresseingabe 1* wird mit Größe und Position konfiguriert. Anschließend wird es mit dem Eingangselement verknüpft. Das Markenelement *WohnortLabel* erhält eine Gestalt vom Typ *JLabel*. Die Gestalt wird nach seiner Erzeugung mit dem Anzeigetext, das Markenelement mit Größe und Position konfiguriert und an das Fensterelement geknüpft. Das Aktionselement *Ende* wird analog implementiert. Zusätzlich erhält es einen Hörer vom Typ *ExitButtonListener*. Der Hörer wird mit einer Referenz auf das Dokument konfiguriert. Wird dieser Knopf gedrückt, beendet der Hörer die Benutzerschnittstelle. Das Ausgangselement dient zur Plausibilitätskontrolle für den Kontrollfluß und führt im Falle von Swing keine weitere Aktion aus.

Abbildung A.11 zeigt die Implementierung des Datenelements *Wohnort*. Zunächst wird das Datenelement mit einem Textfeld als Standardgestalt vom Typ *single* erzeugt. Anschließend werden eine Liste und ein Kombinationsfeld als Gestalten *medium* bzw. *large* hinzugefügt. Alle Gestalten des Datenelements implementieren *DataModelControl* für den Zugriff auf das Datenmodell. Das Kombinationsfeld wurde aufgrund seiner Implementierung in einem unsichtbaren Fenster eingebettet³. Dieses Fenster vom Typ *JPanelControl* implementiert *DataModelControl* und ermöglicht die Verknüpfung mit dem Konfigurationsmenü des Elements. Es agiert als Kontrollkomponente und delegiert alle Methodenaufrufe an die eingebettete Kontrollkomponente. Nach Einfügen der Gestalten werden die Schwellwerte für das automatische Umschalten der Gestalt konfiguriert (*setMinimumitemnumber*) und das Element an das Fenster geknüpft. Ändert sich die Auswahlmenge des Elements durch Neueingabe oder Berechnung, wird geprüft, welche Gestalt zur Anzeige verwendet werden muß.

Abbildung A.12 zeigt die Implementierung der Berechnung. Das Dokument wird mit *writeBack* gespeichert und mit *open* geladen und angezeigt. Für die Berechnung wird zunächst ein Objekt des Typs *DefaultUpdateControl* erzeugt und dem Element zugeordnet. Es implementiert die Verwaltung (*UpdateControl*) und wird von der Programmbibliothek als Standardimplementierung zur Verfügung gestellt. Im Anschluß werden die Implementierungen der Regeln instanziiert und der Verwaltung mit *addUpdateRule* hinzugefügt. Sie werden mit folgenden Daten konfiguriert: Regelname, Elementname, Güte der Vorbelegung und vorausgesetzte Elemente. *setPriority* klassifiziert die Güte der Regeln.

Abbildung A.13 zeigt die Implementierung der Regel *Wohnort_A*. Die Klasse *Wohnort_A_UpdateRule* erweitert *DefaultUpdateRule* um eine konkrete Implementierung der Auswahlmenge und Konsistenzprüfung des Elements *Wohnort*. Die Klasse *DefaultUpdateRule* wird von der Programmbibliothek zur Verfügung gestellt und verkörpert den Verwalter der Regeln. Die Methode *recalculate* erfragt den Wert des vorausgesetzten Elements und erzeugt aus der Adressdatenbank die Auswahlmenge durch eine Datenbankabfrage. Die Konsistenzprüfung *isConsistent* prüft in der Datenbank die Existenz einer Tabellenzeile, die den Elementwert und die Werte der vorausgesetzten Elemente enthält. Existiert eine solche Zeile nicht, wird der Elementwert als nicht konsistent angenommen.

Abbildung A.14 zeigt einen beispielhaften Konfigurationsdialog der Elemente. Der Benutzer kann die Auswahlmenge und die Selektion eines Elements *recalculator* mit *fixed* fixieren, so daß bei Wertänderung eines vorausgesetzten Elements keine

³Grund: JPanel erzeugt kein Ereignis bei Interaktionen mit der rechten Maustaste.

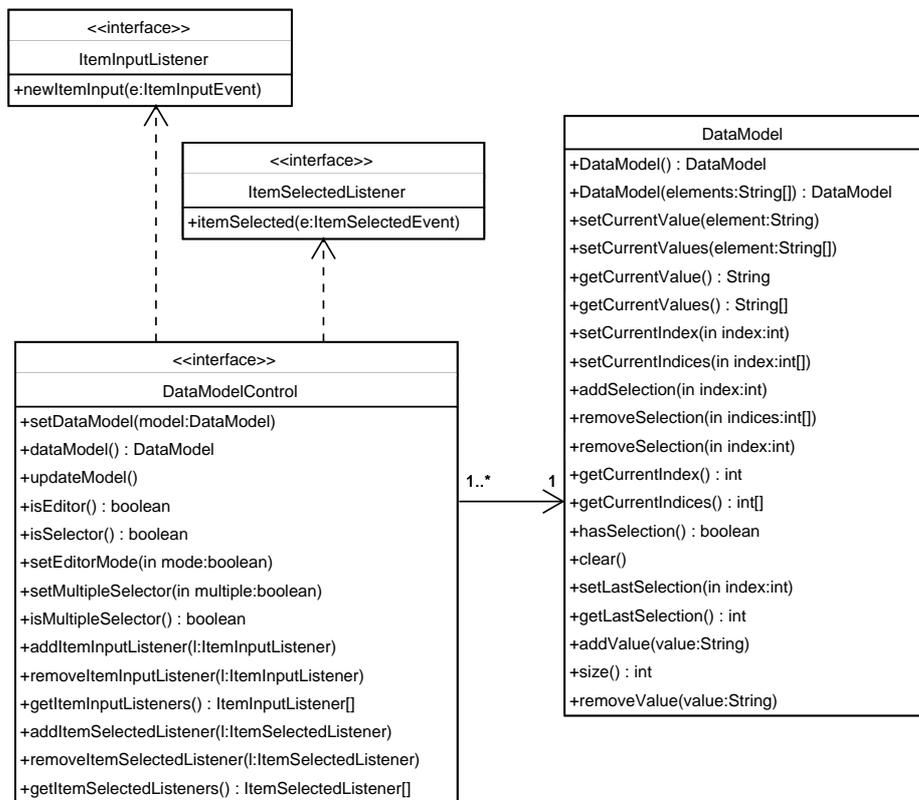


Abbildung A.15: Die Schnittstelle DataModelControl

Neuberechnung stattfindet. Weiterhin können Elemente verschoben (*move*), vergrößert (*resize*) und zwischen Fenstern bewegt (*cut* und *paste*) werden. Alle übrigen Menüpunkte dienen der Informationsanzeige des Prototyps und sind im Produktivbetrieb für den Benutzer unsichtbar. Sie können vom Programmierer für Tests der Struktur genutzt werden.

A.7 Der Zugriff auf das Datenmodell

Komponenten einer spezifischen Programmbibliothek müssen zur Kommunikation mit Datenmodell und Element um die Schnittstelle *DataModelControl* spezialisiert werden. Abbildung A.15 zeigt das Modell der Kontrollkomponente. Diese bietet zwei technologieneutrale Hörer. Der *ItemInputListener* wird ausgelöst, wenn ein neuer Wert vom Benutzer eingegeben wurde. Der *ItemSelectedListener* wird ausgelöst, wenn die Selektion geändert wurde. Beide Hörer können in der Programmlogik verwendet werden, um technologieneutral auf Interaktionen zu reagieren. *DataModelControl* bietet die entsprechenden Methoden (*addItemInputListener*, *addItemSelectedListener*, *getItemInputListeners*, *getItemSelectedListeners*, *removeItemInputListener*, *removeSelectedInputListener*) zur Verwaltung der Hörer, die nicht bei jeder Gestalt, sondern beim konfigurierbaren Element registriert werden. Das konfigurierbare Element

A.8 Die Strukturbeschreibung mit erweiterten Komponenten

verwaltet neutrale Hörer der Gestalten.

Datenelemente besitzen ein Datenmodell (*DataModel*), das die Auswahlmenge und die Selektion des Benutzers repräsentiert. Das Datenmodell wird beim Erzeugen des konfigurierbaren Elements aus der initialen Auswahlmenge und der initialen Selektion generiert. Die möglichen Werte der Auswahlmenge über ihren Index oder Wert selektiert bzw. deselektiert. Weiterhin besitzt das Datenmodell Methoden zur Behandlung von Mehrfachselektionen. Die Methode *size* liefert die Größe der Auswahlmenge. *addValue* fügt einen Wert hinzu, *addSelection* selektiert ihn, *clear* löscht die Auswahlmenge. *getCurrentIndex* bzw. *getCurrentIndices* liefert die Indizes, *getCurrentValue* bzw. *getCurrentValues* liefert die Werte der aktuellen Selektion. *hasSelection* zeigt an, ob eine Selektion vorliegt. Die konfigurierbare Benutzerschnittstelle stellt mit dieser Methode fest, ob ein Element einen Wert besitzt. *removeSelection* entfernt die Selektion einer Wertemenge, *removeValue* entfernt Werte aus der Auswahlmenge. *setCurrentIndex* bzw. *setCurrentIndices* liefert die Indizes, *setCurrentValue* bzw. *setCurrentValues* definiert die Werte der aktuellen Selektion. *size* liefert die Größe der Auswahlmenge.

Die Verknüpfung des Datenmodells mit der Gestalt erfolgt über seine Registrierung (*setDataModel, dataModel*). *setEditorMode* definiert, ob die Auswahlmenge vom Benutzer verändert werden darf, *isEditor* liefert die vorgenommene Einstellung. *setMultipleSelector* definiert, ob die Auswahlmenge mehrfach selektiert werden darf, *isMultipleSelector* liefert entsprechend die vorgenommene Einstellung. Die Methode *updateModel* aktualisiert die Gestalt aus dem Datenmodell. Diese wird bei Änderung der angezeigten Gestalt ausgelöst.

A.8 Die Strukturbeschreibung mit erweiterten Komponenten

Das in Abbildung A.16 gezeigte Modell wurde für Java mit dem Java XML Bindung (JAXB) Prozessor [35] erzeugt. Attribute der XML Elemente erhalten eine *set*- und eine *get*-Methode. Bei optionalen Attributen wird zusätzlich eine *has*-Methode erzeugt, um ihre Existenz zu prüfen. Mehrfach vorkommende XML Elemente werden in den generierten Klassen als Listen⁴ gespeichert. Da Listen Objekte sind und ihren Objektzeiger bei Modifikation der Listenelemente nicht verändern, werden für sie ausschließlich *get*-Methoden erzeugt. An dieser Stelle sei noch einmal bemerkt, daß das erzeugte Modell mit seinen Operationen nicht bindend ist, sondern Entwurfsregeln von JAXB reflektiert.

Die Abbildungen A.17 und A.18 zeigen einen Auszug der Elementstruktur der Adresseingabe aus Abbildung 4.2. Die Einbettung der Gestalten (*style*) und der Berechnung (*calculator*) wird in Kapitel 5 und Anhang A.5 beschrieben und hier nur symbolisch vorgenommen. Das Element *Eingang* verkörpert das Eingangselement. Es definiert, welche Fenster beim Aufbau der konfigurierbaren Benutzerschnittstelle initial geöffnet werden. Außer der Strukturdefinition besitzt das Element keine Konfiguration. Das Fenster *Adresseingabe 1* wird beim Start des Programms geöffnet. Es ist ein eigenständiges Fenster, das an der Bildschirmposition (200|200) erzeugt

⁴siehe Schnittstelle `java.util.List`

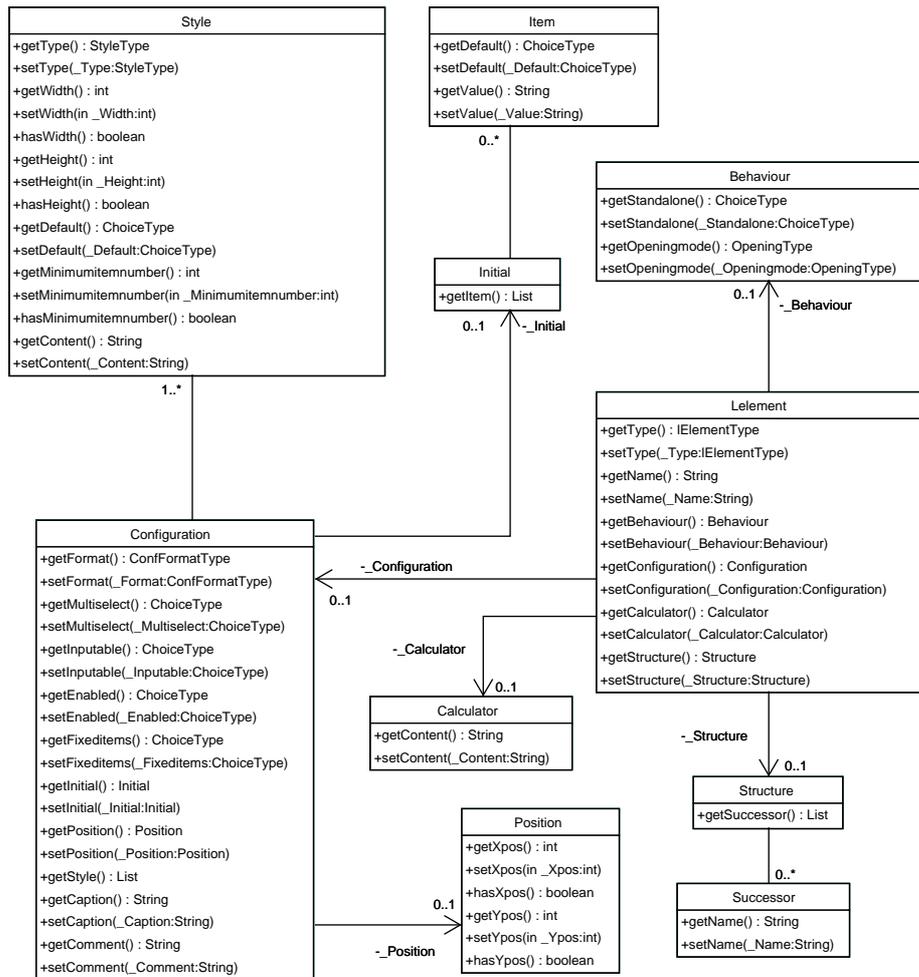


Abbildung A.16: Modell eines konfigurierbaren Elements

A.8 Die Strukturbeschreibung mit erweiterten Komponenten

```
<?xml version="1.0" encoding="UTF-8"?>
<elementlist>
  <lelement type="entry" name="Eingang">
    <structure><successor name="Adresseingabe 1"/></structure>
  </lelement>
  <lelement type="exit" name="Ausgang"><structure/></lelement>
  <lelement type="panel" name="Adresseingabe 1">
    <behaviour standalone="yes"/>
    <configuration>
      <initial/><position xpos="200" ypos="200"/>
      <style type="single" width="600" height="400"
        default="yes">
        <!-- Rohdaten -->
      </style>
    </configuration>
    <structure>
      <successor name="Ende"/>
      <successor name="Speichern"/>
      <successor name="Weiter"/>
      <successor name="Titel"/>
      <successor name="TitelMarke"/>
      <successor name="Vorname"/>
      <successor name="VornameMarke"/>
      <successor name="Name"/>
      <successor name="NameMarke"/>
      <successor name="Strasse"/>
      <successor name="StrasseMarke"/>
      <successor name="Hausnummer"/>
      <successor name="HausnummerMarke"/>
      <successor name="Postleitzahl"/>
      <successor name="PostleitzahlMarke"/>
      <successor name="Wohnort"/>
      <successor name="WohnortMarke"/>
      <successor name="Vorwahl"/>
      <successor name="VorwahlMarke"/>
      <successor name="Telefon"/>
      <successor name="TelefonMarke"/>
    </structure>
  </lelement>
```

Abbildung A.17: Elementstruktur der Adresseingabe (Auszug 1)

```
<lelement type="action" name="Ende">aus Abbildung 4.2
  <behaviour/>
  <configuration>
    <initial/>
    <position xpos="50" ypos="50"/>
    <style type="single" width="110" height="25" default="yes">
      <!-- Rohdaten -->
    </style>
  </configuration>
  <structure><successor name="Ausgang"/></structure>
</lelement>
<lelement type="data" name="Titel">
  <configuration multiselect="no">
    <initial>
      <item default="no" value="Dr."/>
      <item default="yes" value="Prof. Dr."/>
      <item default="no" value="Dipl.-Inform."/>
    </initial><position xpos="50" ypos="150"/>
    <style type="single" width="100" height="25"
      default="yes">
      <!-- Rohdaten -->
    </style>
    <style type="large" width="100" height="25">
      <!-- Rohdaten -->
    </style>
  </configuration>
  <structure/>
</lelement>
<lelement type="label" name="TitelMarke">
  <configuration multiselect="no">
    <initial/><position xpos="50" ypos="120"/>
    <style type="single" width="50" height="20"
      default="yes">
      <!-- Rohdaten -->
    </style>
  </configuration>
  <structure/>
</elementlist>
```

Abbildung A.18: Elementstruktur der Adresseingabe (Auszug 2)

A.9 Ein Beispiel der Anwenderunterstützung

wird und (600|400) Pixel groß ist. Da nur Datenelemente eine Auswahlmenge besitzen, bleibt *initial* leer. *Adresseingabe 1* stellt folgende Elemente dar: *saveButton*, *nextButton*, *Titel*, *TitelMarke*, *Vorname*, *VornameMarke*, *Name*, *NameMarke*, *Strasse*, *StrasseMarke*, *Hausnummer*, *HausnummerMarke*, *Postleitzahl*, *PostleitzahlMarke*, *Wohnort*, *WohnortMarke*, *Vorwahl*, *VorwahlMarke*, *Telefon* und *TelefonMarke*. Das Aktionselement *Ende* beendet die konfigurierbare Benutzerschnittstelle. Ein Fensterelement besitzt genau eine Gestalt. Das Beenden des Programms wird durch die Nachfolgerbeziehung von Aktionselement *Ende* und Ausgangselement erreicht. Das Datenelement *Titel* besitzt eine Auswahlmenge mit drei Werten, wobei *Prof. Dr.* beim Erzeugen des Elements *initial* selektiert wird und Werte nur einfach selektierbar sind. Hierbei wurden zwei Gestalten konfiguriert. Das Markenelement *TitelMarke* visualisiert eine Beschreibung für das Element *Titel*.

Der Strukturgraph wird aus der Nachfolgerbeziehung (*successor*) der Elemente aufgebaut und besitzt diese als Knotenmenge. Die Kanten des Graphen sind gerichtet, zyklisch, ungewichtet und stellen die Nachfolgerbeziehung dar. Die Elementstruktur der Abbildungen A.17 und A.18 besitzt beispielsweise die Kanten

- *Eingang* \rightarrow *Adresseingabe1*,
- *Adresseingabe1* \rightarrow *Titel*,
- *Adresseingabe1* \rightarrow *Ende* und
- *Ende* \rightarrow *Ausgang*.

Der Strukturgraph definiert das Aussehen und die Fensterreihenfolge der konfigurierbaren Benutzerschnittstelle. Besitzt ein Aktionselement oder der Eingang mehrere Fensterelemente als Nachfolger, so werden diese nach der Aktion geöffnet. Ein Fenster kann eingebettete Fensterelemente enthalten, die ihrerseits wieder eingebettete Fenster, Daten-, Marken- oder Aktionselemente enthalten können. Ein unzusammenhängender Graph besäße eine Elementmenge, die sich mit keiner Aktion der konfigurierbaren Benutzerschnittstelle anzeigen ließe.

Die Verknüpfung der Fenster-, Daten-, Marken- und Aktionselemente erfolgt analog zur Verknüpfung der Interaktionselemente. Die von den Elementen verwendeten Gestalten werden beim Erzeugen aus den Rohdaten extrahiert und nach der Strukturbeschreibung ihrer konfigurierbaren Elemente miteinander verknüpft.

Die in diesem Kapitel vorgestellte Elementstruktur läßt sich so zur Speicherung beliebiger konfigurierbarer Benutzerschnittstellen verwenden. Für die Modifizierbarkeit durch den Anwender ist es erforderlich, daß die Struktur erzeugt gespeichert und wiederhergestellt werden kann. Änderungen der Struktur bedeuten Änderung der Nachfolgerbeziehungen. Die Art und Weise der Änderbarkeit der konfigurierbaren Benutzerschnittstelle hängt von den technischen Möglichkeiten der grafischen Benutzerschnittstelle ab, die die Gestalten implementiert.

A.9 Ein Beispiel der Anwenderunterstützung

Die Struktur der konfigurierbaren Benutzerschnittstelle wird anhand des Kontrollflußgraphen, des Abhängigkeitsgraphen und des Eingabeverhaltens des Benutzers

analysiert. Die Unterstützung des Anwenders basiert auf Ergebnissen der Strukturanalyse (vgl. Abschnitt 4.4). Abbildung A.19 zeigt das zugehörige Klassenmodell. Elementgruppen werden als Objekte der Klasse *PermutationGroup* formuliert. Jede Gruppe wird mit den assoziierten Elementen, ihren Abhängigkeiten, einer Identifikation (*GID*) und der bevorzugten Interaktionsfolge des Benutzers (*setFilter*) konfiguriert. Für jede Gruppe wird die Menge ihrer Permutationen berechnet (*permute*) und anhand einer Metrik (*PermutationMetric*) beurteilt. Die favorisierten Permutationen werden vom Analysewerkzeug erfragt (*getBestPermutations*). Jede Permutation erhält eine Identifikation (*ID*) und wird mit ihrer Gruppenzugehörigkeit, Elementreihenfolge und den Abhängigkeiten der Gruppe generiert. Die Methode *setRelevant* erhält die Interaktionsfolge des Benutzers und liefert als Ergebnis, ob die Permutation der Folge entspricht. *dependenciesFulfilled* liefert die Menge der erfüllten Abhängigkeiten, *preemptableElements* die Liste der Elemente, deren Vorbelegungen mit der Struktur der Permutation berechenbar sind. Permutationen sind anhand ihrer Beurteilung vergleichbar. Für jede Permutation lassen sich Erreichbarkeiten der Elemente prüfen (*pathsFromControlflow*). Hierfür wird aus der Elementreihenfolge ein Kontrollflußgraph berechnet.

Für jede Gruppe wird die Menge der bestgeeigneten Permutationen bestimmt. Für jedes Element einer Permutation kann durch Vergleich des Kontrollflußgraphen und der Permutation festgestellt werden, welche Elemente der Benutzerschnittstelle nicht der Reihenfolge der Permutation entsprechen. Die Klasse *StructureOptimizerView* implementiert die Sicht auf die Strukturanalyse. Abbildung A.20 zeigt das Beispielwerkzeug für die Adresseingabe aus Abbildung 4.2. In diesem Werkzeug wird anhand die bestmögliche verbleibende Eingabefolge anhand der bereits erfolgten Formulareingaben angezeigt. Der Benutzer kann damit die Qualität der von ihm geänderten Eingabefolge betrachten und die Struktur der Benutzerschnittstelle anpassen. Das Benutzermodell gibt die Reihenfolge {*Name, Postleitzahl, Vorwahl*} vor. Gruppe 2 ist vom Benutzermodell nicht betroffen, da nur ein Element der Gruppe in der Interaktionsfolge vorkommt. Damit wird für diese Gruppe die bestmögliche Permutation für eine optimale Struktur gewählt: die Folge {*Name, Vorname*}. Die Eingabe des Namens ermöglicht im Kontrollfluß die Vorbelegung des Elements *Vorname* durch das Element *Name*. In der Evaluierung hat sich herausgestellt, daß selbst bei großen Datenbanken häufig nur wenige Vornamen zur Auswahl stehen. In den verbleibenden Fällen kann der Vorname als Freitext eingegeben werden. Für die Gruppe 3 werden lediglich Permutationen betrachtet, die die Elementreihenfolge {*Postleitzahl, Vorwahl*} enthält. Die entsprechenden Permutationen besitzen die Identifikationen 31,21 und 35. Für die Permutation 31 wird vorgeschlagen, das Element *Postleitzahl* als erstes Element der Gruppe in den Kontrollfluß zu integrieren. Es wird davon ausgegangen, daß keine Auswahlmenge berechnet wird. Für diese Permutation wird angezeigt, daß die Benutzerschnittstelle für alle Elemente mit Ausnahme der *Strasse* eine optimale Position aufweist. Das Werkzeug schlägt vor, das Element *Strasse* im Kontrollfluß zwischen den Elementen *Wohnort* und *Hausnummer* zu platzieren. Die tatsächliche Position in der Benutzerschnittstelle hängt von der Berechnung des Kontrollflußgraphen ab. Im hier vorgestellten Beispiel wurde ein YX-Algorithmus⁵ verwendet, also müsste *Strasse* in visueller Folge zwischen *Wohnort* und *Hausnummer* angeordnet werden.

Das Problem großer Gruppen wurde bereits in Abschnitt 4.4 diskutiert. Der Aufwand

⁵siehe Abschnitt 4.4

A.9 Ein Beispiel der Anwenderunterstützung

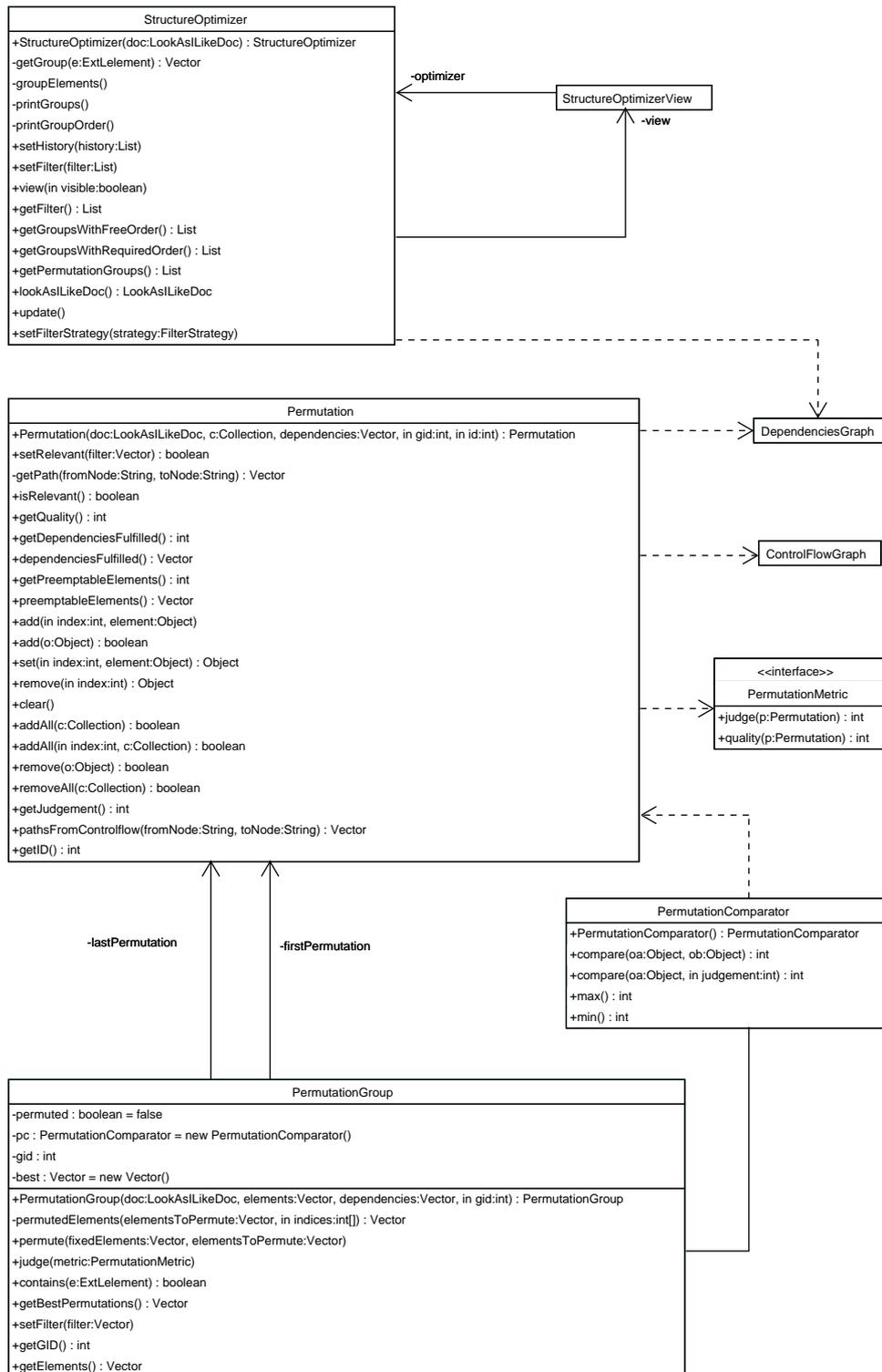


Abbildung A.19: Klassendiagramm der Strukturoptimierung

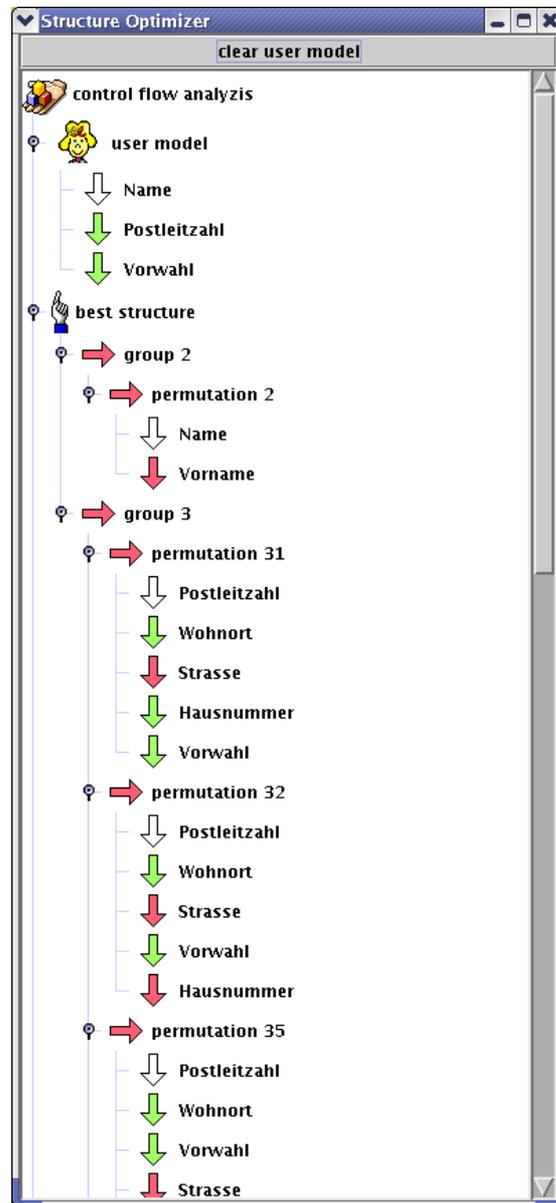


Abbildung A.20: Werkzeug zur Strukturoptimierung

A.10 Probleme mit Java Swing

zur Beurteilung aller n -elementigen Permutationen wächst mit der Ordnung $O(n!)$. Für die systematische Erzeugung aller Permutationen wurde der Algorithmus aus [29] als Iterator (Schnittstelle *java.util.Iterator*) implementiert. Abhängig von der verfügbaren Rechenleistung⁶ muß eine maximale Gruppengröße definiert werden. Diese Größe sollte zur Laufzeit ohne signifikante Wartezeiten für den Anwender analysierbar sein. Für den Prototypen wurde eine maximale Größe von 7 Elementen definiert. Die Unterstützung des Benutzers wird bei großen Gruppen mit folgender Annahme erreicht: Die optimale allgemeingültige Elementfolge wird vom Entwickler konfiguriert. Eine optimierte Struktur positioniert üblicherweise vorausgesetzte Elemente mit wenigen Abhängigkeiten an den Beginn der visuellen Folge. Für den Programmstart wird daher angenommen, daß die Struktur der konfigurierbaren Benutzerschnittstelle bereits optimiert wurde. Die Strukturanalyse beurteilt Permutationen erst mit der Verfügbarkeit einer bevorzugten Interaktionsfolge. Diese schränkt die Freiheitsgrade der Gruppe und damit die zu permutierende Teilmenge ein. Diese durch die bevorzugte Interaktionsfolge auf eine handhabbare Größe reduzierten Permutationen ermöglichen die Unterstützung bei den verbleibenden Dateneingaben.

Die Optimierung der Struktur kann vom System auch automatisch vorgenommen werden. Es kann den Benutzer darauf hinweisen, daß eine konkrete Umstrukturierung aufgrund seines Interaktionsverhaltens sinnvoll ist. Um eine geeignete Benutzerschnittstelle automatisch erzeugen zu können, sollten Aspekte wie Effizienz und Verwendbarkeit betrachtet werden (vgl. Kapitel 2). Hierfür kann eine Beurteilung mit Gestaltungsrichtlinien und Komplexitätsmessung hinzugezogen werden.

A.10 Probleme mit Java Swing

Die Serialisierbarkeit von Objekten hängt von seinen Attributdaten ab. Selbst wenn die implementierende Klasse als serialisierbar gekennzeichnet wurde, läßt sich die Entscheidung, daß Serialisierbarkeit vorliegt, erst durch die Serialisierung selbst feststellen. Die Speicherbarkeit einer konfigurierbaren Benutzerschnittstelle kann somit nicht durch erfolgreiche Programmübersetzung, sondern nur durch geeignete Testläufe verifiziert werden. Vergleichbare Komponentenkonzepte [52, 35] bieten hierfür Prüfwerkzeuge (*verifier*) an.

Die Serialisierbarkeit von Kontrollkomponenten hat sich bei der Implementierung des Prototypen als komplexes Problem dargestellt, da neben der Serialisierbarkeit der Spezialisierung die der originalen Programmbibliothek als Fehlerquellen identifiziert wurden. Abhängig von der Konfiguration der grafischen Interaktionselemente (als Gestalten) waren diese durchaus nicht mehr serialisierbar. Als Beispiel sei hier die Verwendung des Hinweistextes (*ToolTipText*) in Swing [51] erwähnt. Diese Implementierung im Java Development Kit (JDK [51]) wird jeder Komponente vererbt und verhindert die Serialisierung, sobald sie erstmals aktiviert wurde. Für die Entwicklung konfigurierbarer Benutzerschnittstellen sollten technologieabhängige Werkzeuge geschaffen werden, die den Ablauf einer konfigurierbaren Benutzerschnittstelle simulieren und Fehlerquellen aufzeigen.

⁶z.B. Intel Pentium 4, 2GHz, 60GByte Festplatte, 768 MByte RAM

A.11 Zusammenfassung

Der Prototyp Look_{AS}^ILike spezialisiert konfigurierbare Benutzerschnittstellen für Java und Swing. Er wurde aus der XML Struktur der konfigurierbaren Element generiert. Die Unterstützung des Benutzers basiert auf der Bildung von Elementgruppen des Abhängigkeitsgraphen. Große Gruppen erzeugen dabei hohe Aufwände in der Analyse. Aus den Gruppen können dennoch effizient Strukturen mit weniger Dateneingaben gefunden werden⁷. Damit kann die Benutzerschnittstelle Anwendern entweder durch einen Automatismus oder ein Werkzeug Unterstützung bei der Vereinfachung der Dateneingabe leisten und seine Berechnung der Auswahlmengen bzw. der Konsistenzprüfungen automatisch anpassen.

⁷vgl. Abschnitt 7.1.2

Anhang B

Glossar

Abhängiges Element Ein abhängiges Element enthält Berechnungen, die Werte weiterer Elemente zur Berechnung der Auswahlmenge und Konsistenzprüfung benötigen.

Abhängigkeitsgraph Im Abhängigkeitsgraphen einer konfigurierbaren Benutzerschnittstelle werden die Abhängigkeitsbeziehungen zwischen den konfigurierbaren Elementen der Benutzerschnittstelle notiert.

Anpassende Systeme Anpassende Systeme (*adaptive systems*) versuchen, die Benutzerschnittstelle für die Prozesse, die Fähigkeiten und die Expertise der Anwender zu adaptieren.

Benutzerkategorie Generische Klassifizierung des Benutzers.

Benutzermodell Informationen, die eine Benutzerschnittstelle zur Analyse des Anwenders sammelt.

Dokument Verwaltende Instanz für konfigurierbare Elemente.

Element Konfigurierbares Element.

Filter Ein Filter stellt eine spezielle Regel dar. Er schränkt die Vorbelegung auf eine Teilmenge ein, ohne dafür Werte anderer Elemente zu verwenden.

Gestalt Die Gestalt eines Formularfelds stellt die Schnittstelle einer Dateneingabe der Benutzerschnittstelle mit dem Benutzer dar.

Gruppe, Elementgruppe Sei E_0 die nichtleere Menge der Elemente einer konfigurierbaren Benutzerschnittstelle. Eine nichtleere Menge $E \subseteq E_0$ heißt Elementgruppe, wenn gilt:

1. Kein Element aus E besitzt ein vorausgesetztes Element in $E_0 \setminus E$.
2. Kein Element aus $E_0 \setminus E$ besitzt ein abhängiges Element in E .
3. Keine Teilmenge $E_1 \subset E$ erfüllt 1 und 2.

Berechnung Eine Berechnung berechnet die Auswahlmenge und die Konsistenzprüfung eines Elements aus den Werten von mindestens einem konfigurierbaren Element der Benutzerschnittstelle. In einer Berechnung werden Referenzen auf die vorausgesetzten Elemente gespeichert.

Konfigurierbare Benutzerschnittstelle Formularbasierte Benutzerschnittstelle, mit der Anwender Dateneingaben selbständig vereinfachen können.

Konfigurierbares Element Ein konfigurierbares Element erweitert die Implementierung eines beliebigen Formularfelds um eine beliebige Zahl von Berechnungen und besitzt mindestens eine Gestalt.

Konsistenz Konsistenz kann verschiedene Aspekte der Benutzerschnittstelle adressieren: Datenkonsistenz, Bezug zur realen Welt, Dialogformen, Interaktionsparadigmen, etc.

Mensch-Maschine Interaktion Die Forschung der Mensch-Maschine Interaktion (*Human-Computer Interaction, HCI*) beschäftigt sich neben technischen Aspekten der Maschine auch mit psychosozialen Fähigkeiten des Menschen.

Güte einer Abhängigkeit Die Güte einer Abhängigkeit ist ein Maß für die Auswahlmenge, die aus einer Abhängigkeit heraus berechnet werden kann.

Güte eines abhängigen Elements Die Güte eines abhängigen Elements ist ein Maß für die Auswahlmengen, die aus allen seinen Abhängigkeiten berechnet werden können.

Regel Regeln implementieren die Abhängigkeitsbeziehungen der konfigurierbaren Elemente. Aus ihnen wird der Abhängigkeitsgraph automatisch erzeugt.

Serialisierbarkeit Fähigkeit, einen Objektzustand persistent zu speichern und zu beliebigen Zeitpunkten wiederherstellen zu können.

Triviale Gruppe Sei E_0 die nichtleere Menge der Elemente einer konfigurierbaren Benutzerschnittstelle. Eine einelementige Gruppe $G \subseteq E_0, G = \{g\}$ heißt triviale Gruppe.

Vorausgesetztes Element Ein vorausgesetztes Element ist ein konfigurierbares Element, das in mindestens einer Berechnung eines anderen Elements referenziert wird.

Wert eines konfigurierbaren Elements Eine beliebige, nichtleere Teilmenge der Auswahlmenge eines Elements wird als Wert bezeichnet. Der Wert eines konfigurierbaren Elements wird von der Gestalt als Selektion in der Auswahlmenge dargestellt.

WIMP Windows, Icons, Menus and a Pointing device.

Index

- Abhängigkeit
 - Attribute, 40
 - Güte, 40
- Abhängigkeitsgraph, 39
 - DTD, 55
 - Güte, 40
 - Modell, 55
 - optimaler, 51
- Adaptive Forms, 24
- Adaptive Systems, 21
- Adresseingabe, 42
- Analyse
 - theoriebasiert, 20
- Anpassende Systeme, 21
- Anwenderunterstützung, 37, 115
- Attribute, 40
- Auswahlmenge
 - Berechnung, 9
- Benutzer
 - Bedürfnisse, 21
 - Experte, 20
 - Kategorien, 10
 - Kategorisierung, 64
- Benutzerbeteiligung
 - im Entwurf, 18
- Benutzerinteraktion, 10, 63
- Benutzermodell, 12, 13, 16, 21, 48, 116
 - ereignisbasiert, 27, 35
- Benutzermodellierung
 - ereignisbasiert, 34
- Benutzerschnittstelle
 - automatische Anpassung, 21
 - benutzerangepaßt, 21
 - direkte Manipulation, 22
 - formularbasierte Datenerfassung, 9
 - Komplexität, 16, 119
 - konfigurierbare, 10
 - Metrik, 16
 - theoriebasierte Analyse, 20
- Benutzerverhalten, 17
- Berechnung, 39, 91
 - Implementierung, 109
 - Regel, 91
 - Verwalter, 91
 - Zyklen, 65
- Datenmodell, 111
- Dokument, 91
 - Verhinderung zyklischer Berechnung, 65
- Dynamic Forms, 24
- Eingabesequenz
 - bevorzugte, 18
- Element, 39
 - abhängiges, 39
 - Güte, 40
 - Aktionselement, 52
 - Ausgang, 52
 - Datenelement, 52, 111
 - Eingang, 52
 - Fensterelement, 52
 - Güte, 40
 - Attribute, 40
 - Gruppe, 41
 - konfigurierbares, 39, 52
 - Konsistenzprüfung, 91
 - Markenelement, 52
 - Name, 52
 - Tooltip, 55
 - Typ, 52
 - vorausgesetztes, 24, 39
 - Vorbelegung, 91
 - Wert, 39
- Endbenutzer Programmierung, 22
- Entwicklungsplattform, 86
- Entwurf
 - benutzerzentriert, 18
- Ereignisse
 - asynchrone, 28
 - Dauer, 28

- Frequenzbänder, 28
- Grammatiken, 29
- synchrone, 28
- Ereignisströme, 30
- Evaluierung, 63
 - Adresseingabe, 65
 - benutzerzentriert, 19, 21
 - IHKdezent2, 78
 - Konstruktionsdatenaustausch, 69
 - Thesen, 63
- Güte, 40
- Gestalt, 24, 39
 - Kontrollkomponente, 110
- Gestaltung
 - konsistente, 16
 - Richtlinien, 119
- Gruppe, 41, 115
 - dynamisches Umketten, 76
 - Freiheitsgrade, 49
 - große, 76, 116
 - Annahme zur Analyse, 119
 - Permutation, 115
 - triviale, 41
- Hörer
 - ActionListener, 100
 - ItemInputListener, 100, 110
 - ItemSelectedListener, 100, 110
- Human Computer Interaction, 15
- Interaktionsfolge, 45
 - Annahmen, 45
- JavaBeans, 103
 - Properties, 86
 - Property Sheet, 86
 - Serialisierung, 103
 - XML, 103
- JAXB, 94, 111
 - Binding Schema, 94
 - Klassenmodell, 94
- Klassenmodell
 - Spezialisierung, 91
- Knotensortierung, 45
- Komponente
 - Kontrollkomponente, 110
 - Serialisierung, 100
 - unbekannte, 78
 - Zustand, 53
- Konsistenz
 - der Benutzerschnittstelle, 16
 - externe, 16
 - interne, 16
 - von Dateneingaben, 16
 - Zustände, 93
- Kontrollfluß, 57
 - optimistischer, 58
- Kontrollflußgraph, 57
 - Knotensortierung, 45
 - optimistischer, 45
- Mensch-Maschine Interaktion, 15
- Modell
 - konzeptuelles, 17
 - mentales, 16, 25
- Nützlichkeit, 11, 15, 18, 20
- Permutationen
 - Beurteilung, 48
- Power User, 11, 20
- Prozeßwissen, 10, 63
- Regel, 91
 - berechenbar, 93
 - bestmögliche, 93
 - Filter, 93
 - Implementierung, 109
 - Qualität, 93
- Softwareergonomie, 9
- Strukturanalyse
 - Meßgrößen, 46
 - Metrik, 46
- Strukturgraph, 55, 115
- Thesen der Arbeit, 10
 - Evaluierung, 63
- UIML, 23
- Verwendbarkeit, 11, 15, 18–20
 - Analyse, 28
 - eines Produkts, 19
 - Evaluierung, 28
- XAML, 23
- XForms, 23
- XIML, 23
- XML, 22, 111

INDEX

YX-Nummerierung, 45

Zielgruppe, 10

Abbildungsverzeichnis

1.1	Konzept der Vereinfachung formularbasierter Datenerfassung	12
1.2	Strukturdefinition konfigurierbarer Benutzerschnittstellen	13
3.1	Frequenzspektrum von Ereignissen (aus [48])	29
3.2	3D-Darstellung der Mausklick-Dichte in einer grafischen Benutzerschnittstelle (aus [20])	33
3.3	Prinzip der ereignisbasierten Benutzermodellierung	34
3.4	Frequenzbereich zur Berechnung des Benutzermodells (aus [20])	35
4.1	Beispiel einer Abhängigkeit	41
4.2	Benutzerschnittstelle für Adresseingabe	42
4.3	Abhängigkeitsgraph für Adresseingabe	43
4.4	Algorithmus zur Berechnung der Eingabefolge	46
5.1	Prinzip der Entwicklung einer konfigurierbaren Benutzerschnittstelle	51
5.2	Struktur eines konfigurierbaren Elements	53
5.3	Document Type Definition (DTD) der Komponentenstruktur	54
5.4	Klassenmodell der Komponentenstruktur	56
5.5	Document Type Definition (DTD) des Abhängigkeitsgraphen	56
5.6	Klassenmodell des Abhängigkeitsgraphen	57
5.7	Document Type Definition (DTD) des Kontrollflußgraphen	58
5.8	Klassenmodell des Kontrollflußgraphen	59
5.9	Struktur eines konfigurierbaren Elements	60
6.1	Selektion und Konsistenz nach Eingabe der Postleitzahl	67
6.2	Selektion und Konsistenz nach Eingabe von Wohnort und Straße	68
6.3	Abhängigkeitsgraph der Senderdaten für das Datenaustauschprogramm in [24]	72
6.4	Abhängigkeitsgraph der Empfängerdaten für das Datenaustauschprogramm in [24]	73
6.5	Sender–Empfänger Beziehung des Datenaustauschprogramms	74
6.6	Modaler Dialog für dynamische Kontaktauswahl	77
6.7	Dialog des IHKdezent2 Moduls Faktura	80
6.8	Abhängigkeitsgraph des IHKdezent2 Moduls Faktura	81
6.9	Integration von Spezialimplementierungen des Originalsystems	82
7.1	WYSIWYG Editor einer Gestalt	87
A.1	Die Berechnung der Auswahlmenge und der Konsistenzprüfung	92

A.2	Dokument einer konfigurierbaren Benutzerschnittstelle	95
A.3	Binding Schema für konfigurierbare Elemente	97
A.4	JAXB-Methoden für Laden, Speichern und Prüfen	98
A.5	Klassendiagramm konfigurierbarer Elemente	99
A.6	Modell einer Gestalt	101
A.7	Modell einer Berechnung	101
A.8	XML Serialisierung für JavaBeans	102
A.9	Beispiel einer XML-serialisierten Gestalt	102
A.10	Dokument, Fenster, Marke und Aktionselement erzeugen	104
A.11	Datenelement <i>Wohnort</i> mit Gestalten erzeugen	105
A.12	Berechnung; Dokument speichern und anzeigen	106
A.13	Die Regel <i>Wohnort_A</i>	107
A.14	Konfiguration eines Elements	108
A.15	Die Schnittstelle <i>DataModelControl</i>	110
A.16	Modell eines konfigurierbaren Elements	112
A.17	Elementstruktur der Adresseingabe (Auszug 1)	113
A.18	Elementstruktur der Adresseingabe (Auszug 2)	114
A.19	Klassendiagramm der Strukturoptimierung	117
A.20	Werkzeug zur Strukturoptimierung	118

Tabellenverzeichnis

3.1	Indikatoren zur Sammlung von Verwendbarkeitsdaten (aus [21]) . . .	29
3.2	Abstraktionsebenen hochfrequenter Ereignisse	30
6.1	Auszug bester Interaktionsfolgen der Adresseingabe	66
6.2	Legende der Konsistenzanzeige von Elementwerten	67
6.3	Programminterne Elementbezeichnungen des Faktura Moduls	79
7.1	Aufwand der Datensuche abhängig von der Gruppengröße	88

Literaturverzeichnis

- [1] The Emacs Editor. <http://www.emacs.org>.
- [2] Marc Abrams and Jim Helms. User Interface Markup Language (UIML) Specification Draft Language Version 3.0. <http://www.uiml.org/specs/docs/uiml30-revised-02-12-02.pdf>, 2002.
- [3] Robert B. Allen. Mental models and user models. In Martin G. Helander, Thomas K. Landauer, and Prasad V. Prabh, editors, *Handbook of Human-Computer Interaction*, pages 49–63. Elsevier Science B.V., 1997.
- [4] Apple Computer, Inc. The Macintosh Platform. <http://www.apple.com>, 1984.
- [5] David Benyon and Dianne Murray. Developing adaptive systems to fit individual aptitudes. In *Proceedings of the 1st international conference on Intelligent user interfaces*, pages 115–121. ACM Press, 1993.
- [6] Borland. JBuilder. <http://www.borland.com/jbuilder>, 2002.
- [7] I. N. Bronstein, K. A. Semendajajew, G. Musiol, and H. Mühlig. *Taschenbuch der Mathematik*. Harri Deutsch, 1995.
- [8] World Wide Web Consortium. Document Object Model. <http://www.w3.org/DOM/>, 1998.
- [9] World Wide Web Consortium. Extensible Markup Language. <http://www.w3.org/XML/>, 1998.
- [10] World Wide Web Consortium. XML Schema. <http://www.w3.org/XML/Schema/>, 2001.
- [11] Microsoft Corporation. Microsoft .NET Platform. <http://www.microsoft.com/dotnet>, 2000.
- [12] Ernest Edmonds. The future of intelligent interfaces: not just how?, but what? and why? In *IUI 93: Proceedings of the 1st international conference on Intelligent user interfaces*, pages 13–17, New York, NY, USA, 1993. ACM Press.
- [13] Michael Eisenberg. End-User Programming. In Martin G. Helander, Thomas K. Landauer, and Prasad V. Prabh, editors, *Handbook of Human-Computer Interaction*, pages 1127–1146. Elsevier Science B.V., 1997.
- [14] XIML Forum. XIML: A Universal Language for User Interfaces. <http://www.ximl.org/Docs.asp>, 2001.

- [15] Martin R. Frank and Pedro A. Szekely. Adaptive forms: An interaction paradigm for entering structured data. In *Intelligent User Interfaces*, pages 153–160, 1998.
- [16] Donald M. Frohlich. Direct Manipulation and Other Lessons. In Martin G. Helander, Thomas K. Landauer, and Prasad V. Prabhhu, editors, *Handbook of Human-Computer Interaction*, pages 463–488. Elsevier Science B.V., 1997.
- [17] A. Girgensohn, B. Zimmermann, A. Lee, B. Burns, and M. Atwood. Dynamic forms: An enhanced interaction abstraction based on forms. In *Fifth IFIP Conference on Human-Computer Interaction*, pages 362–367. Chapman & Hall, 1995.
- [18] Jonathan Grudin. The case against user interface consistency. volume 32, pages 1164–1173. ACM Press, 1989.
- [19] Jonathan Grudin. Utility and Usability: Research issues and development contexts. *Interaction with computers*, 4(2), 1992. Elsevier B. V.
- [20] M. Guzdial, P. Santos, A. Badre, S. Hudson, and M. Gray. Analyzing and visualizing log files: A computational science of usability. *Presented at HCI Consortium Workshop*, 1994.
- [21] David M. Hilbert and David F. Redmiles. Extracting usability information from user interface events. *ACM Comput. Surv.*, 32(4):384–421, 2000.
- [22] IBM. The Eclipse Platform. <http://www.eclipse.org>, 2002.
- [23] Autodesk Inc. AutoCAD. <http://www.autodesk.com>.
- [24] Andreas Judt. *Ein systemunabhängiges, netzwerkübergreifendes Graphical User Interface (GUI) für den Austausch von CAD/CAM Konstruktionsdaten*. University of Kaiserslautern, Germany, 1997. Diploma Thesis.
- [25] John Karat. User centered design: quality or quackery? *interactions*, 3(4):18–20, 1996. ACM Press.
- [26] John Karat. User-centered software evaluation methodologies. In Martin G. Helander, Thomas K. Landauer, and Prasad V. Prabhhu, editors, *Handbook of Human-Computer Interaction*, pages 689–704. Elsevier Science B.V., 1997.
- [27] R. Kass and T. Finin. A general user modelling facility. In *Proceedings of the SIG-CHI conference on Human factors in computing systems*, pages 145–150. ACM Press, 1988.
- [28] David G. Korn. The KornShell Language. <http://www.kornshell.com>, 1996.
- [29] Udi Manber. *Fundamentals of Algorithmics*. Prentice-Hall Inc., 1996.
- [30] Udi Manber, Ash Patel, and John Robison. Experience with personalization of yahoo! *Commun. ACM*, 43(8):35–39, 2000.
- [31] Ji-Ye Mao, Karel Vredenburg, Paul W. Smith, and Tom Carey. User-centered design methods in practice: a survey of the state of the art. In *Proceedings of the 2001 conference of the Centre for Advanced Studies on Collaborative research*, page 12. IBM Press, 2001.

LITERATURVERZEICHNIS

- [32] Microsoft Corporation. Microsoft Extensible Application Markup Language (XAML). <http://xml.coverpages.org/ms-xaml.html>, 2004.
- [33] Sun Microsystems. JavaBeans Component Architecture. <http://java.sun.com/products/javabeans/>, 1999.
- [34] Sun Microsystems. NetBeans Platform. <http://www.netbeans.org>, 2000.
- [35] Sun Microsystems. Java Architecture for XML Binding. <http://java.sun.com/xml/jaxb/index.html>, 2001.
- [36] Sun Microsystems. Sun ONE Studio. <http://www.sun.com/software/sundev/index.html>, 2002.
- [37] Michael J. Muller, Jean Hallowell Haslwanter, and Tom Dayton. Participatory practices in software lifecycle. In Martin G. Helander, Thomas K. Landauer, and Prasad V. Prabhu, editors, *Handbook of Human-Computer Interaction*, pages 255–297. Elsevier Science B.V., 1997.
- [38] Peter Murray-Rust, Tim Bray, David Megginson, and et. al. Simple API for XML. <http://www.saxproject.org/>, 1998.
- [39] Brad Myers, Scott E. Hudson, and Randy Pausch. Past, present, and future of user interface software tools. *ACM Trans. Comput.-Hum. Interact.*, 7(1):3–28, 2000.
- [40] Ramond S. Nickerson and Thomas K. Landauer. Human-computer interaction: Background and issues. In Martin G. Helander, Thomas K. Landauer, and Prasad V. Prabhu, editors, *Handbook of Human-Computer Interaction*, pages 3–31. Elsevier Science B.V., 1997.
- [41] Jakob Nielsen, editor. *Coordinating User Interfaces for Consistency*. Academic Press Inc., 1989.
- [42] Jakob Nielsen. *Usability Engineering*. Academic Press, Cambridge, MA., 1993.
- [43] Oasis. XML Markup Languages for User Interface Definition. <http://xml.coverpages.org/userInterfaceXML.html>, 2004.
- [44] Bernd Oesterreich. *Objektorientierte Softwareentwicklung*. Oldenbourg Verlag, 2001.
- [45] ProCAEss GmbH. Design Data eXchange (DDX). <http://www.procaess.com>, 1997.
- [46] ProSTEP GmbH. Data eXchange Manager (DXM). <http://www.prostep.de>, 1996.
- [47] Stephanie Rosenbaum, Janice Anne Rohn, and Judee Humberg. A toolkit for strategic usability: results from workshops, panels, and surveys. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 337–344. ACM Press, 2000.
- [48] P. M. Sanderson and C. Fisher. Exploratory Sequential Data Analysis: Foundations. *Human-Computer Interaction Special Issue on ESDA*, 9, 1994.
- [49] Ahmad Shahinniya. *Aufzählung von Wegen in unzusammenhängenden, gerichteten, zyklischen Graphen*. University of Karlsruhe, Germany, 2005. Studienarbeit.

-
- [50] Ben Shneiderman. *Designing the User Interface*. Addison-Wesley Longman Publishing Co., Inc., 1998.
- [51] Sun Microsystems. Java Development Kit and API specification. <http://java.sun.com/products>, 1996.
- [52] Sun Microsystems. Java 2 enterprise edition. <http://java.sun.com/products/j2ee>, 1998.
- [53] Sun Microsystems. The swing connection. <http://java.sun.com/products/jfc/tsc>, 2003.
- [54] M. Sweeny, M Maguire, and B. Shackel. Evaluating Human-Computer Interaction: A Framework. *International Journal of Man-Machine Studies*, 38, 1993. Academic Press.
- [55] The Mozilla Project. XML User Interface Language (XUL). <http://www.mozilla.org/projects/xul>, 2003.
- [56] Shari Trewin, Gottfried Zimmermann, and Gregg Vanderheiden. Abstract user interface representations: how well do they support universal access? In *Proceedings of the 2003 conference on Universal usability*, pages 77–84. ACM Press, 2003.
- [57] Thomas S. Tullis. Screen Design. In Martin G. Helander, Thomas K. Landauer, and Prasad V. Prabhu, editors, *Handbook of Human-Computer Interaction*, pages 503–531. Elsevier Science B.V., 1997.
- [58] Karel Vredenburg, Ji-Ye Mao, Paul W. Smith, and Tom Carey. A survey of user-centered design practice. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 471–478. ACM Press, 2002.
- [59] Larry Wall. *Programming Perl*. O'Reilly & Associates, 2000.
- [60] Donald O. Walter. Programs that do what they think you want. *SIGCAS Comput. Soc.*, 30(3):22–22, 2000.
- [61] World Wide Web Consortium. HyperText Markup Language (HTML). <http://www.w3.org/MarkUp/>, 1997.
- [62] World Wide Web Consortium. Extensible HyperText Markup Language (XHTML). <http://www.w3.org/TR/xhtml1>, 2000.
- [63] World Wide Web Consortium. XForms 1.0. <http://www.w3.org/TR/2002/CR-xforms-20021112>, 2002.
- [64] World Wide Web Consortium. Voice Extensible Markup Language (VoiceXML). <http://www.w3.org/TR/voicexml20>, 2004.
- [65] X Consortium. The X Window System. <http://www.x.org>, 1988.
- [66] G. Zimmermann, G. Vanderheiden, and A. Gilman. Prototype implementations for a universal remote console specification. In *CHI 2002 Conference on Human Factors in Computing Systems*, pages 510–511. MN, April 2002.

Lebenslauf: Andreas Judt

Persönliche Angaben

Geburtsdatum	26. Oktober 1970
Geburtsort	Siegen, NRW
Staatsangehörigkeit	deutsch

Ausbildung

1997	Diplom-Informatiker, Universität Kaiserslautern
1991 bis 1997	Studium der Allgemeinen Informatik, Universität Kaiserslautern
1990 bis 1991	Grundwehrdienst, Stabssoldat
1981 bis 1990	Allgemeine Hochschulreife, Gymnasium Auf der Morgenröthe, Siegen, NRW

Berufserfahrung

seit 2002	Forschungszentrum Informatik an der Universität Karlsruhe Softwaretechnik, Abteilungsleiter
2000 bis 2002	Forschungszentrum Informatik an der Universität Karlsruhe Softwaretechnik, Wissenschaftlicher Mitarbeiter
1998 bis 2000	ProCAEss GmbH – professional CAE software solutions Projektleiter und Gesellschafter
1997 bis 1998	CAE-Beratung Markus Latz & Siegfried Heinz Softwareentwickler
1996 bis 1997	Robert Bosch GmbH, Bühl (Baden), Praktikant und Diplomand