Institut für Informatik
der Technischen Universität München

Modellierung grafischer Dialogsysteme mit Ereignisstellen-Diagrammen in WYSIWYG-orientierter Darstellung

Alfred Zeidler

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr. Peter Paul Spies
Prüfer der Dissertation:	
	1. Univ.-Prof. Dr. Dr. h.c. Jürgen Eickel
	2. Univ.-Prof. Dr. Helmut Seidl

Die Dissertation wurde am 15. Januar 2004 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 22. Juli 2004 angenommen.

Danksagung

Herrn Professor Eickel gilt herzlicher Dank zunächst dafür, dass er mir spontan die Gelegenheit zur Durchführung dieser Arbeit an seinem Lehrstuhl eröffnete, dann für die Geduld, die er über die langen Jahre bis zum Durchbruch und Abschluss aufbrachte und selbstverständlich nicht zuletzt für die kritische Betrachtung und die zahlreichen fruchtbaren Hinweise.

Herzlichen Dank an Herrn Professor Seidl als zweitem Gutachter, auch für die Anregungen in der Schlussphase, die die Aufnahme weiterer interessanter Aspekte in die Arbeit zur Folge hatten.

Frau Remmes gilt ganz besonderer Dank für ihr so freundliches Engagement, mit dem sie die durch meine externe berufliche Tätigkeit bedingte räumliche Entfernung zum Lehrstuhl stets äußerst herzlich und hilfreich überbrückte.

Ebenfalls herzlichen Dank an die Herren Professoren Endres und Struss, über die der Kontakt zum Lehrstuhl zustande kam.

Mein freundschaftlicher Dank geht an meine Kollegen Theo Bodensteiner, Eberhard Dittmar, Brian Hill und Dirk Nasri-Roudsari, die mir über die lange Zeit sowohl moralisch als auch inhaltlich mit einer Reihe von Anmerkungen und Tipps zur Seite standen.

Nicht zuletzt aber danke ich vor allem meiner Familie, ganz besonders meiner Frau Marianne und meinen Kindern. Ohne ihre Aufmunterung, Unterstützung und den, wie ich doch annehme, allen schwerfallenden Verzicht auf gemeinsam verbrachte Zeit wäre die Arbeit nicht entstanden.

Ursensollen, 31. Dezember 2003

Inhaltsverzeichnis

Abbildungsverzeichnis

0 Einführung

0.0 Allgemeine Problemstellung

Mit der Einführung von Dialogsystemen und insbesondere mit der Entwicklung grafischer Bedienoberflächen seit nunmehr gut zwei Jahrzehnten – einer der Meilensteine war wohl nach Arbeiten bei Rank-Xerox die Einführung des Apple-Macintosh Anfang der achziger Jahre – geht die Entwicklung von Techniken, Methoden und Werkzeugen zum Entwurf und zur Implementierung im Bereich der computerunterstützten Software-Entwicklung einher. Zahlreiche Arbeiten zum Thema Bedienoberflächen insbesondere unter dem Aspekt der benutzerfreundlichen Gestaltung des Dialogs sind erschienen.

Die immer noch ansteigende weltweite Nutzung des Internets in den letzten Jahren brachte weitere neue Sprachen und Techniken der Implementierung und mit der Anwendung eines immer größer werdenden Kreises von Benutzern zunehmende Anforderungen an die Gestaltung und Realisierung der mit dem Internet verbundenen Kommunikationstechniken. Neben sogenannte Standalone-Anwendungen mit grafischen Oberflächen – realisiert etwa auf Basis von Windows, X-Windows, Java-Swing – traten immer mehr browsergestützte Anwendungen mit zunächst primitiven Point-and-click-Oberflächen. Dabei hat die rasante Entwicklung der Basistechnik mit dem Schwerpunkt einer einfachen global gültigen Definition des Austausches der Information über allgemein anerkannte Sprachen und Protokolle (HTML, XML, HTTP..) sowie serverseitige Implementierungstechniken (Servlets, Active Server Pages (ASP, JSP),...) die Bemühungen um eine systematische, benutzergerechte und auch für den Entwickler effiziente Modellierung der grafischen Oberflächen in den Hintergrund gedrängt. Lag der Schwerpunkt der Funktionalität der angebotenen Entwicklungsumgebungen anfangs in der Unterstützung der Realisierung der Dialogoberfläche (UIMS-Systeme), so konzentrierte sich in letzter Zeit der Mehrwert neuer Freigaben sogenannter integrierter Entwicklungsumgebungen (Integrated Development Environments) auf Aspekte der Verteilung von Anwendungen im Netz etwa auf der Grundlage standardisierter Software-Architekturen (Stichwort J2EE, .net; Application Server, EJB, ...).

Erwiesen sich bereits in der Vergangenheit zahlreiche Dialoganwendungen, stand-alone wie auch im Client-Server-Betrieb, als relativ komplex, dürfte die Komplexität des Dialogs bei Internetanwendungen mit immer größerem Spektrum der Anwendungen ebenso zunehmen. Umso mehr ist es erforderlich, Verfahren, Techniken und Werkzeuge anzubieten, die es erlauben, unabhängig von den sich relativ schnell ändernden Basistechniken der Implementierung ein benutzerkonformes, anwendungsgerechtes und „logisches“ Design der Bedienoberfläche durchzuführen. Nicht zuletzt unter dem Druck immer kürzerer Entwicklungszyklen jedoch soll mit dem Entwurf der logischen Dialogoberfläche für den Entwickler eine effiziente, schnelle und komfortable Umsetzung in die jeweils aktuelle Implementierungstechnik möglich sein.

Die Vergangenheit zeigt, dass bei Entwicklern und Managern von IT-Projekten eine prinzipielle Bereitschaft existiert, zur Steigerung der Produktivität und Qualität Werkzeuge für Design und Implementierung grafischer Dialogsysteme einzusetzen. Anfängliche Euphorie weicht jedoch häufig einem Stadium der Ernüchterung. Dies gilt sowohl für Systeme zum Rapid-Prototyping und Rapid-Development als auch für Systeme, die einen

streng modellbasierten Ansatz zur Top-down-Analyse mit mehr oder weniger automatischer Umsetzung in die Implementierung erlauben. Auch sogenannte Round-Trip-Systeme, die iterativ eine Umsetzung des Modells auf die Implementierungsebene und eine Rückführung implementierter Codesequenzen auf die Modellebene versprechen, haben bisher keinen Durchbruch in der breiten Anwendung gefunden.

Die Gründe dafür sind vielseitig.
Einer davon ist sicherlich, dass keines der auf dem Markt verfügbaren Systeme eine Sprache bzw. Technik bietet, mit der ein logisches Design des Dialogs, insbesondere auch der Dialogdynamik, unabhängig von einer konkreten Wahl der spezifischen Implementierung der Oberfläche möglich wird. Die zur Zeit als der Standard der Modellierung geltende UML (Unified Modelling Language) bietet zwar eine Reihe verschiedenster Diagramme für ein objektorientiertes Design der Anwendung. Auf die spezifische Notwendigkeit einer Darstellung des Dialogs wird aber wenig Wert gelegt. Die dafür manchmal empfohlenen State-Charts stellen keine befriedigende Lösung dar (siehe später).

In der Regel verlässt man sich im Projekt auf eine dann meist unvollständige prototypische Realisierung ohne vorangehendes systematisches Design, das auch mit potentiellen künftigen Nutzern abgestimmt ist. Auch nach Fertigstellung des Systems und Freigabe für Produktion und Wartung existiert kein Überblick über die Struktur des Dialogs, keine Gesamtsicht des Dialogs, die auch dem Designer oder Entwickler den Zugang zu relevanten Stellen der Implementierung zum Zwecke der Änderung und Wartung erleichtern würde. Eine mühsame Suche im Code wird erforderlich, wenn nicht wider Erwarten manuell erzeugte Dokumentation zum aktuellen Stand der Entwicklung vorhanden ist.

In gängigen Entwicklungswerkzeugen für grafische Oberflächen fehlen grafische Darstellungsmöglichkeiten der Dialogdynamik völlig oder sind nur ansatzweise und dann nur auf der Ebene konkreter Oberflächenelemente vorhanden**.**
Einige wenige Systeme zum Rapid-Prototyping/Rapid-Development unterstützen die Darstellung der Dialogdynamik rudimentär, z.B. mit Wizards zur Codierung einfacher Ereignisroutinen oder ersten, noch unvollständigen Ansätzen einer grafischen Darstellung dynamischer Relationen zwischen Objekten der Oberfläche.

Aber auch die Entwicklung der statischen Elemente der Oberfläche lässt Wünsche offen. Denn in der Phase des Designs ist die Wahl eines konkreten Dialogelementes häufig noch nicht entschieden, obwohl vielleicht Aspekte der dynamischen Anbindung des Objektes bereits festliegen. Der Entwickler oder Designer ist dann dennoch gezwungen eine bestimmte Wahl für ein konkretes Objekt zu treffen. Eine spätere Änderung seiner Wahl wird schwierig. Er ist in der Regel gezwungen, sämtliche Information zum alten Objekt zu beseitigen und auch die dynamische Anbindung des dann neuen Objektes erneut zu formulieren.

Modellbasierte Systeme bieten grundsätzlich die Möglichkeit, zunächst auf einer implementierungsneutralen Ebene diese Objekte darzustellen. Sie haben aber damit wiederum den Nachteil, dass bei Situationen, in denen eine Konkretisierung für den Entwickler klar auf der Hand liegt, erst eine meist umständliche Abbildung erfolgen muss. Das Modell liegt zudem in einer zunächst sehr abstrakten Darstellung vor, die keinen unmittelbaren Eindruck des konkret resultierenden Dialogs vermittelt. Dies ist eine Darstellung, die in der Entwurfsphase etwa zur Diskussion mit einem zukünftigen Benutzer ungeeignet ist und natürlich auch dem Designer oder Entwickler keinen Eindruck vom echten Dialog zur Laufzeit vermittelt.

0.1 Zielsetzung der Arbeit

In der vorliegenden Arbeit wird eine Technik vorgestellt, die es erlaubt eine benutzerzentrierte Modellierung einer grafischen Dialogoberfläche durchzuführen, in der konkrete Dialogelemente mit abstrakten aber intuitiven Sprachelementen kombiniert in einer Modell-Darstellung zur Anwendung kommen können. Ein wesentlicher Bestandteil der Arbeit ist somit die Entwicklung einer integrierten Darstellung von Dialog, die es erlaubt, auf unterschiedlichen Ebenen der Abstraktion die Modellierung des Dialogs durchzuführen.
Es entfällt damit der Druck, in der Designphase bereits mühevoll konkrete Elemente auszuarbeiten, bei gleichzeitiger Möglichkeit, wo gewünscht, konkrete Elemente des Dialogs, z.B. Widgets, einzuführen.
In der Modellsprache sind Sprachelemente enthalten, die insbesondere auch die Dynamik des Dialogs modellierbar macht und einen Überblick über den möglichen Ablauf des Dialogs bietet. Die gewählte Form der grafischen Darstellung dieser Sprachelemente gibt einen intuitiven Eindruck der späteren zur Laufzeit vorliegenden grafischen Dialogoberfläche. Dennoch kann die Sprache so abstrakt gewählt werden, dass eine Umsetzung in andere Dialogarten (z.B. Kommandodialog) ebenso möglich ist. Die entwickelte Modellsprache rückt die Objekte des Dialogs und die Ereignisse darauf in den Vordergrund und zeigt die Wirkung dieser Ereignisse auf die Objekte (bzw. Stellen) auf. Im Gegensatz zu anderen abstrakten Modellansätzen (Zustandsübergangsdiagramme, Petrinetze, ...) wird eine intuitive „dialogobjektorienterte“ Darstellung möglich, die zudem auch komplexen dynamischen Anforderungen genügt.
Die Entwicklung dieser Art von Darstellung geschieht in einem „Bottom-Up-Ansatz“. Ein wesentliches Ergebnis der Entwicklung sind die Sprachelemente des sogenannten Ereignisstellendiagramms und die Einbettung in eine praxisorientierte Modellierungsumgebung mit der Integration abstrakter Diagrammkomponenten und konkreter Darstellungselemente.
Ein wesentlicher Nachteil der Darstellung grafischen Dialogs in Zustandsübergangsdiagrammen beispielsweise, nämlich die Notwendigkeit einer kaum handhabbaren Vielzahl von Zustandsübergängen, wird durch eine besondere Definition von Dialogstellen und die Einführung von „Metaereignissen“ (Öffnen, Schließen von Stellen) beseitigt. Im Vergleich zu State Charts (z.B. [HAREL87], [OESTER97]) ist die Darstellung nebenläufiger Dialoge noch flexibler gelöst. Im Gegensatz zum Einsatz von Petrinetzen (z.B. [JANSS96]), die ursprünglich für andere Anwendungsgebiete geschaffen wurden, bietet das Ereignisstellendiagramm einen natürlicheren, mehr dem spezifischen Problem angepassten und mächtigeren Zugang zur Darstellung der Dialogdynamik.
Das Ziel ist unter anderem, mit möglichst wenigen und eingängigen Sprachelementen die Schnittstelle zwischen Benutzer und System zu beschreiben, die auch dem Modellier-„Laien“, also etwa dem künftigen Anwender des Systems, das Modell verständlich und handhabbar macht.
Neben einem systematischen „Top-down-Entwurf“ wird z.B. mit der unmittelbaren Integration konkreter Dialogobjekte in Ereignisstellendiagramme ein „Bottom-up-Ansatz“ im Vorgehen des Designs ermöglicht. D.h. der Designer kann im Entwurf mit einer konkreten Darstellung beginnen und diese bei Bedarf sukzessive mit abstrakteren verallgemeinernden Sprachmitteln fortführen. Er kann u.a. eine exemplarische Darstellung wählen, die er durch eine Beschreibung der Variation dieser Elemente zu einer vollständigen Darstellung der allgemeinen Programmlogik erweitert.
Die Beschreibungsmöglichkeit der dynamischen Variation von Ereignissen bzw. Stellen durch Einführung von Variationsnotationen an der Modellstelle steigern den praktischen Nutzen der Modelldarstellung. Ähnliche Ansätze finden wir in constraint-basierten Systemen und Systemen zum „Programming by Demonstration“ (z.B. [VANDER95], [MYERS88]).

Eine Implementierung des grafischen Dialogs aus dem Modell heraus ist im Sinne von Rapid-Development auf sehr kurzem Weg, auf Wunsch weitestgehend automatisch, machbar.

0.2 Kapitelübersicht

In **Kapitel 1** werden drei gängige methodische Ansätze der Realisierung von Dialogsystemen, nämlich der Einsatz von Systemen zum Rapid-Development/Rapid-Prototyping, die Nutzung abstrakter Modellsysteme sowie die direkte Programmierung im Überblick diskutiert und deren wesentliche Vor- und Nachteile besprochen.
Kapitel 2 stellt die Kernziele des zu erarbeitenden integrierten Ansatzes vor, der eine logische Dialogmodellierung unter Einbeziehung der Dialogdynamik und einer WYSIWYG-orientierten Darstellung erlaubt.
Kapitel 3 schafft die Grundlage zur Erarbeitung des integrierten Ansatzes mit der Einführung eines einfachen allgemeinen Modells (Grundmodells) zum Verständnis von Dialog, das als Ausgangspunkt für die zu erarbeitende Modellsprache dient.
Kapitel 4 zeigt, wie in einer Bottom-up-Vorgehensweise, ausgehend vom Grundmodell und einer naiven, konkreten und extensiven Darstellung in mehreren Schritten eine intensivierte abstrahierte Darstellung gewonnen werden kann. Dabei werden die wesentlichen Sprachelemente des Ereignisstellendiagramms erarbeitet und Grundüberlegungen für die Entwicklung dieser Sprachelemente ausgeführt.
Kapitel 5 stellt das Ergebnis der in den vorangegangenen Kapiteln vorgestellten Überlegungen dar, das Ereignisstellenmodell. Es wird dort zunächst in seiner primitiven Variante, dann mit Einbeziehung einer Hierarchierelation formal eingeführt, wobei die Semantik in Form einer Übergangsfunktion eines Transitionssystems definiert wird. Es folgt die Abbildung eines deterministischen endlichen Automaten auf die Ereignisstellendarstellung und umgekehrt die Abbildung einer eingeschränkten Ereignisstellendarstellung auf einen endlichen Automaten. Danach werden die grafischen Sprachelemente eines Ereignisstellendiagrammes vorgestellt. In diesem Zusammenhang wird ein Konzept von Modellsichten erarbeitet, das eine flexible Aufteilung eines Modells in verschiedenen Diagrammen bzw. Diagrammsichten erlaubt.
Kapitel 6 diskutiert die Einbeziehung konkreter Widgetdarstellungen in das zunächst in Kapitel 5 vorgestellte Ereignisstellendiagramm, das dort nur abstrakte Dialog-Elemente (Dialog-Stellen) enthält.
Kapitel 7 behandelt die Einführung des Begriffes der Exemplarvariation. Mit Hilfe der exemplarischen Darstellung von Dialogen in Ereignisstellendiagrammen und der Möglichkeit mit zusätzlichen Sprachelementen die dynamische Variation der Dialoge zu beschreiben, wird das Ereignisstellenmodell in seiner Mächtigkeit wesentlich erweitert.
Kapitel 8 beschreibt ein Modellbeispiel aus der unmittelbaren Praxis des Autors im Abriss und zeigt den Einsatz der vorher vorgestellten Elemente von Ereignisstellendiagrammen inklusive Exemplarvariation.
Kapitel 9 bringt den Vergleich des Ereignisstellenmodells mit verschiedenen bekannten und relevanten Ansätzen der Dialogmodellierung. Es werden zunächst klassische Ansätze wie das Koroutinenkonzept von Jacob, das Ereignismodell von Green, Janssens Dialognetze und State Charts von Harel, dann modernere Ansätze, wie die HIT-Technik von Schreiber und Sketchingsysteme aufgeführt. Es werden außerdem Parallelen und Gegensätze zu Modellen konkurrierender Prozesse anhand eines Vergleichs mit Milners Pi-Kalkül angesprochen.
Kapitel 10 stellt die prototypische Realisierung eines Systems zur Modellierung mit Ereignisstellendiagrammen vor.
Zusammenfassung und Ausblick in **Kapitel 11**, Glossar in **Kapitel 12**, Literaturverzeichnis in **Kapitel 13** sowie **Anhang A-D** schließen die Arbeit ab.

1 Gängige Ansätze der Realisierung von Dialogsystemen

Im Folgenden werden drei grundsätzlich unterschiedliche Ansätze der Methodik bei Entwurf und Realisierung von Dialogsystemen beschrieben, wie sie in der Regel in Projekten verfolgt werden, nämlich

- der Einsatz eines Werkzeuges zum Rapid-Prototyping oder Rapid-Development,
- der Einsatz eines modellbasierten Systems zur Top-down-Modellierung und
- die Realisierung rein auf Ebene der Programmierung, wie z.B. in Java, C++ oder einer Skriptsprache ohne Einsatz eines der erstgenannten Hilfsmittel.

Sie sollen als Ausgangspunkte für die Entwicklung der vorgestellten neuen Methodik dienen.

Die ersten beiden Ansätze werden, wenn überhaupt, manchmal alternativ, manchmal nebeneinander sich ergänzend im selben Projekt angewendet. Die eine Methodik ist, unter Verwendung eines Werkzeuges zum Rapid-Prototyping oder Rapid-Development, ein grafisches Dialogsystem prototypisch zu entwerfen, bzw. im Fall von Rapid-Development-Systemen mit oder nach ersten Entwurfsschritten die Entwicklung für den produktiven Einsatz durchzuführen. (Wir wollen im Folgenden in der Regel ohne Einschränkung der Allgemeinheit von Rapid-Development-Systemen (RDS) sprechen und Rapid-Prototyping-Systeme, wenn nicht besonders hervorgehoben, mit einschließen.) Wir können derartige RD-Systeme in die Kategorie der sogenannten Lower-Case-Tools einordnen.
Die alternative Methodik ist der Einsatz eines sogenannten Upper-Case-Tools zum Aufbau eines zunächst abstrakten Modells unter Verwendung einer entsprechenden abstrakten Modellierungssprache. Wir wollen in diesem Fall von modellbasierten Systemen (kurz Modellsystemen) sprechen.
Wir werden im Folgenden wesentliche Vor- und Nachteile beider Ansätze im Überblick aufzeigen. Auf einzelne Systeme bzw. Modellansätze soll in späteren Abschnitten noch eingegangen werden (siehe Kap.9).
Beide Extreme stellen zwei Ausgangspunkte für eine Integration in der später vorgestellten Systematik zur Modellierung und Realisierung in Ereignisstellendiagrammen dar.

1.0 Einsatz von Systemen zum Rapid-Development

Rapid-Development-Systeme zur Realisierung von Dialoganwendungen zielen im Allgemeinen darauf, mittels spezifischer Werkzeuge (Spezialeditoren, …) und problemspezifischer, oft auch deskriptiver Sprachelemente, häufig auch mit grafischen Hilfsmitteln, eine Dialoganwendung zu beschreiben, um diese dann möglichst schnell und automatisch in entsprechenden Programmcode meist einer universellen Programmiersprache umzusetzen.
Eine umfassende, systematische Klassifizierung derartiger Systeme soll hier nicht vorgenommen werden. Wir wollen hier jedoch einige wesentliche Merkmale aufzeigen, die für die Zielsetzung dieser Arbeit von Bedeutung sind. Insbesondere sollen wesentliche Ausdrucksmittel zur Darstellung von Dynamik in gängigen Systemen zum Rapid-Development angesprochen werden.

1.0.0 WYSIWYG-Darstellung statischer Dialogkomponenten

Eine Vielzahl von Systemen zum Rapid-Development von Dialogsystemen enthält eine Komponente, mit der die statischen Dialogobjekte am Bildschirm im sogenannten WYSIWYG-Modus erstellt werden können. WYSIWYG (What You See Is What You Get) bedeutet in diesem Falle, dass die Objekte durch den Entwickler mithilfe eines entsprechen-

den Werkzeuges, eines Dialogobjekteditors (Screenpainters, ...) am Bildschirm bereits so dargestellt werden können, wie sie dann später zur Laufzeit, also im Echteinsatz ebenfalls für den Benutzer der Anwendung erscheinen. Einzelne Objekttypen werden beispielsweise in sogenannten Toolbars angeboten, können dort vom Entwickler ausgewählt und auf einer Arbeitsfläche mithilfe direkter Manipulation, beispielsweise im Drag-and-Drop-Modus, zusammengestellt werden. Die einzelnen Komponenten bilden dabei eine Hierarchie. D.h. es gibt eine Reihe von sogenannten Container-Komponenten, in die wiederum Subcontainer oder letztlich primitive Komponenten eingefügt werden können. Die einzelnen Objektattribute können zum Teil alternativ zur direkten Manipulation oder in Ergänzung zu dieser mit Hilfe spezieller Attributeditoren in ihren Werten textuell angezeigt, erfasst und geändert werden.
Es handelt sich hier zunächst um die Montage statischer Komponenten. Statisch sei hier so verstanden, dass ein aus Teilkomponenten zusammengestelltes Objekt auch zur Laufzeit mit all seinen Komponenten über eine bestimmte Zeit existiert. Aus Sicht des Benutzers vom Zeitpunkt des Öffnens am Bildschirm bis zum Schließen, aus Sicht des Programms vom Zeitpunkt des Kreierens des Objektes bis zum Löschen aus dem Arbeitsspeicher oder einem Sekundärspeicher.
Allerdings steckt implizit in vielen der so definierten Objekte bereits dynamisches Verhalten (siehe unten).

1.0.1 Darstellung der Dynamik

1.0.1.0 Dynamik in vorgefertigten Komponenten

Implizit ist in vielen der im Dialogobjekt-Editor vorgegebenen Objekte bereits dynamisches Verhalten enthalten. Dieses wird jedoch eben nicht explizit gesondert beim Aufbau der Komponenten angegeben, sondern steckt bereits von vorne herein in der Definition der vorgegebenen Komponente. Dies gilt für dynamisches Verhalten einzelner primitiver Objekte im Detail („Mikrodialog“), z.B. bei einem editierbaren Textfeld, wie für Dynamik in

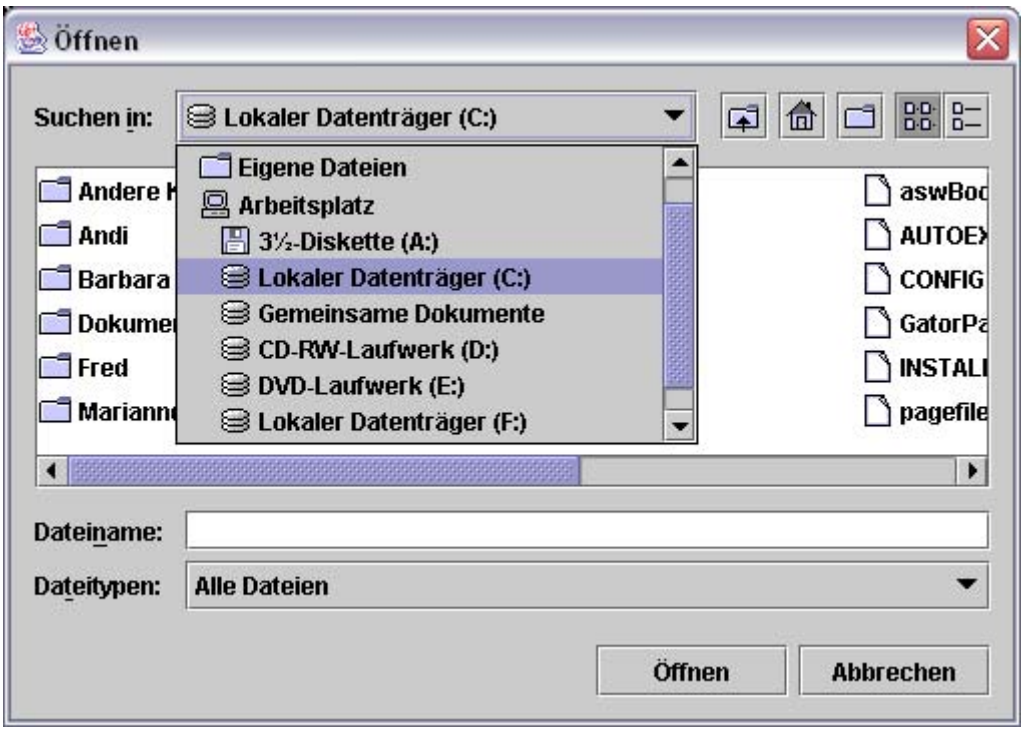


Abbildung 1-1 FileSelectionBox: Beispiel einer vorgefertigten Dialogkomponente zur Realisierung eines Dialogs mit hohem Anteil an Dialogdynamik.

zusammengesetzten Komponenten („Makrodialog“), z.B. bei einer Kombobox oder besonders bei einer bereits komplett vorgefertigten Komponente zur Dateiauswahl (FileSelectionBox). Jedem editierbaren Textfeld ist standardmäßig bereits ein Editor „einprogrammiert", der die Veränderung des Textfeldes während des Editionsvorgangs steuert. Ebenso muss das Aufklappen der Auswahlliste einer Kombobox, die Selektion einzelner Listeneinträge, das Nachtragen des selektierten Wertes im editierbaren Feld der Box nicht neu modelliert oder programmiert werden. Ähnliches gilt für den Dialog, der in der FileSelectionBox vorgegeben ist und letztlich zur Darstellung eines Dateinamens nach durchgeführter Auswahl über einige Mausklicks führt.

1.0.1.1 Attributgesteuerte Dynamik

Der objektbasierte Ansatz der meisten Systeme ermöglicht es, Dialogobjekte als Instanzen bestimmter Objektklassen zu definieren. Über die in einer Klasse zusammengefassten Attribute können nun statische, aber auch dynamische Aspekte einer Objektinstanz beschrieben werden. Unter zunächst eher statischen Aspekten verstehen wir beispielsweise die Höhe und Breite eines Widgets, seine Farbe oder die Zuordnung zu seiner übergeordneten Komponente. Dynamische Aspekte sind z.B. Angaben darüber, wann ein Widget geschlossen werden soll: z.B. kann eine Dialog-Box beim Drücken seines Ok-Buttons per Attributangabe wahlweise entweder automatisch geschlossen werden oder aber geöffnet bleiben (Autoclose-Attribut in Motif). Genauso kann jedoch auch die Größe eines Widgets als dynamisch veränderbar definiert werden: Dies gilt z.B. dann, wenn die Widgetgröße dynamisch berechnet werden soll. Dies und die Art der Layoutberechnung kann nun wiederum über Attribute gesteuert werden (Layoutmanagerzuordnung in Visual Cafe für Java, Constraints in Motif). Über die Zuordnung eines Models (Model-View-Conroller-Konzept (MVC-Konzept)) kann z.B. auch der Inhalt eines Widgets, etwa die Anzahl der Einträge einer Liste, dynamisch gesteuert werden. Ein dem MVC-Konzept zugeordneter definierter Eventing-Mechanismus sorgt dann automatisch dafür, dass etwa bei Änderungen des Models das Widget als View-Objekt benachrichtigt und aktualisiert wird.
Generell gibt es eine Reihe von Attributen, über die das dynamische Verhalten von Dialogobjekten bei Eintreten bestimmter Ereignisse gesteuert werden kann. (Z.B. auch Bewegung des Vollbilds oder Bewegung nur eines Schattenrahmens, etc.) Die eigene Erstellung von Programmcode ist dabei in der Regel nicht erforderlich, da entweder das Entwicklungssystem die Interpretation der Attribute und die Umsetzung in entsprechenden Code übernimmt oder die Interpretation der Attribute bereits in den jeweiligen Klassen der Dialogobjekte implementiert ist.

Auch in HTML-basierten Systemen kann Dialogdynamik über Attribute definiert werden, die den Tags mitgegeben werden. Abbildung 1-2 (nächste Seite) zeigt im System Dreamweaver die Definition eines Hyperlinks, der bewirkt, dass bei Mausklick auf das Feld „(See Details)“ eine HTML-Seite namens „index.htm“ geöffnet wird. In der Ellipse rechts oben sieht man die WYSIWYG-Darstellung, im Fensterausschnitt darunter (mittlere Ellipse) den entsprechenden HTML-Code und im Eigenschaftenfenster des Editors die Attribute des Hyperlink-Tags mit dem Attribut Hyperlink (Ellipse unten links).

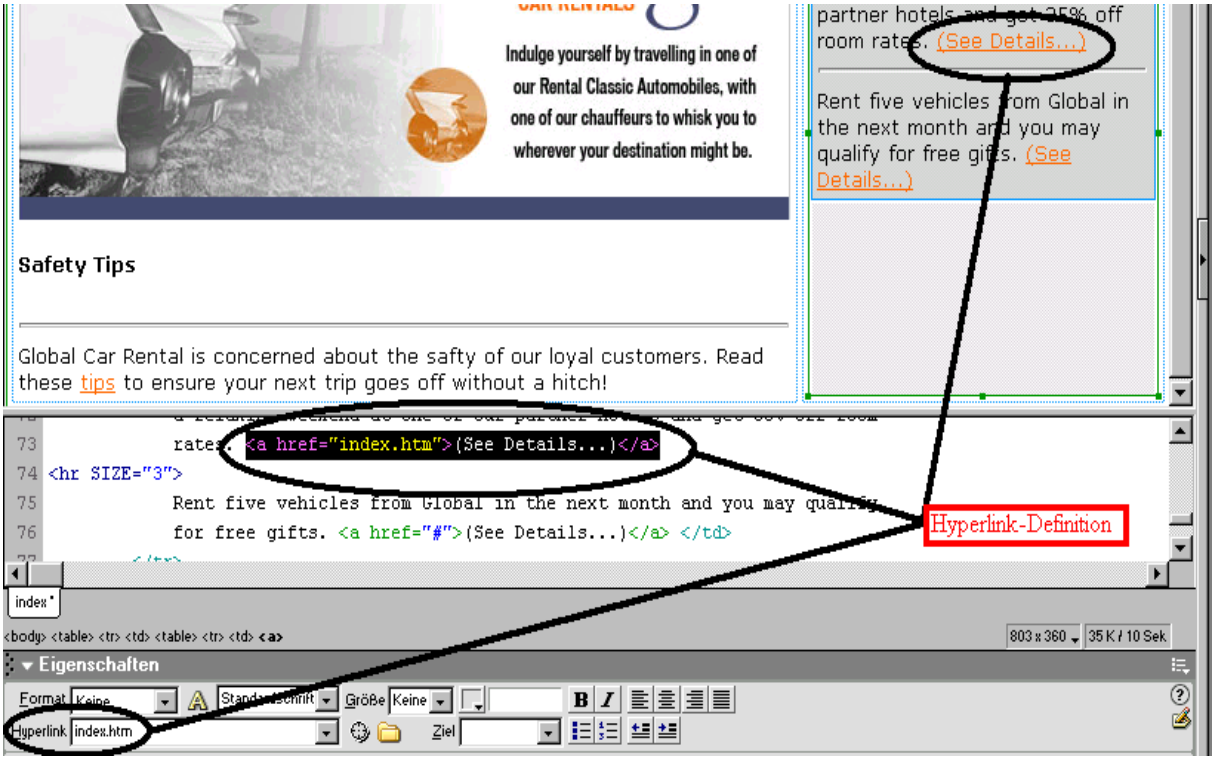


Abbildung 1-2 Attribute zur Definition der Dialogdynamik in HTML-basierten Systemen am Beispiel Dreamweaver der Firma Macromedia

1.0.1.2 Ereignisgesteuerte Routinen

Die wohl am meisten verwendete Art der Darstellung des dynamischen Verhaltens eines Dialogsystems ist die Formulierung von Programmcode in einer spezifischen Skriptsprache oder einer gängigen Sprache der dritten Generation (Java, C++, C#, J#, Visual Basic, Perl, PHP …). Es werden dabei Routinen geschrieben, die bei Eintreten bestimmter Ereignisse angestoßen werden (Callback-Routinen, Eventhandler in Motif, Eventlistener in Java, …). Über entsprechende Methoden der Dialogobjektklassen können die Widgets beispielsweise kreiert, geöffnet, geschlossen, und gelöscht werden. Viele Attribute können ebenso über Methoden dynamisch abgefragt und gesetzt werden.

1.0.1.3 Wizards zur Codegenerierung

Ein weiteres Hilfsmittel zur Beschreibung von Dynamik sind so genannte Wizards oder Programmassistenten, die den Programmierer bei der Erzeugung von Code unterstützen. Es handelt sich dabei um Dialogsequenzen, in denen der Benutzer, vom System geführt, in einer Art Frage-Antwort-Dialog eine Aufgabe löst. Dabei handelt es sich hier beispielsweise um die Beschreibung der notwendigen Reaktionen auf ein bestimmtes Ereignis. In mehreren Schritten wird der Benutzer durch den Assistenten gefragt, auf welches Ereignis welche Aktionen an welchen Objekten durchgeführt werden sollen. Dazu werden dem Benutzer die möglichen Antworten bereits zur einfachen Auswahl angeboten. Das System setzt dann die Antworten des Benutzers meist sofort automatisch in entsprechenden Programmcode um. Der Ausgangspunkt der Abfrage ist in der Regel das Objekt, an dem ein Ereignis stattfindet. Entsprechend ist der generierte Code dann Teil einer Routine, die bei Eintreten des definierten Ereignisses angesprungen wird (siehe oben ereignisgesteuerte Routinen).

1.0.1.4 Grafische Ansätze

Einige Systeme bieten erste Ansätze zu einer grafischen Darstellung der Beschreibung des dynamischen Verhaltens von Dialogobjekten. Eines der ersten Systeme dieser Art war z.B. ein Editor im System NeXT [SHEBA93]. Mithilfe von Pfeilen zwischen einzelnen Objekten konnten bestimmte Aktionen angedeutet werden. Ähnliches gilt auch für die grafische Darstellung dynamischer Bezüge, etwa in der Entwicklungsumgebung Visual Cafe. Dort führt ein Aktivieren des Pfeils zum Aufruf eines wie oben beschriebenen Code-Wizards.

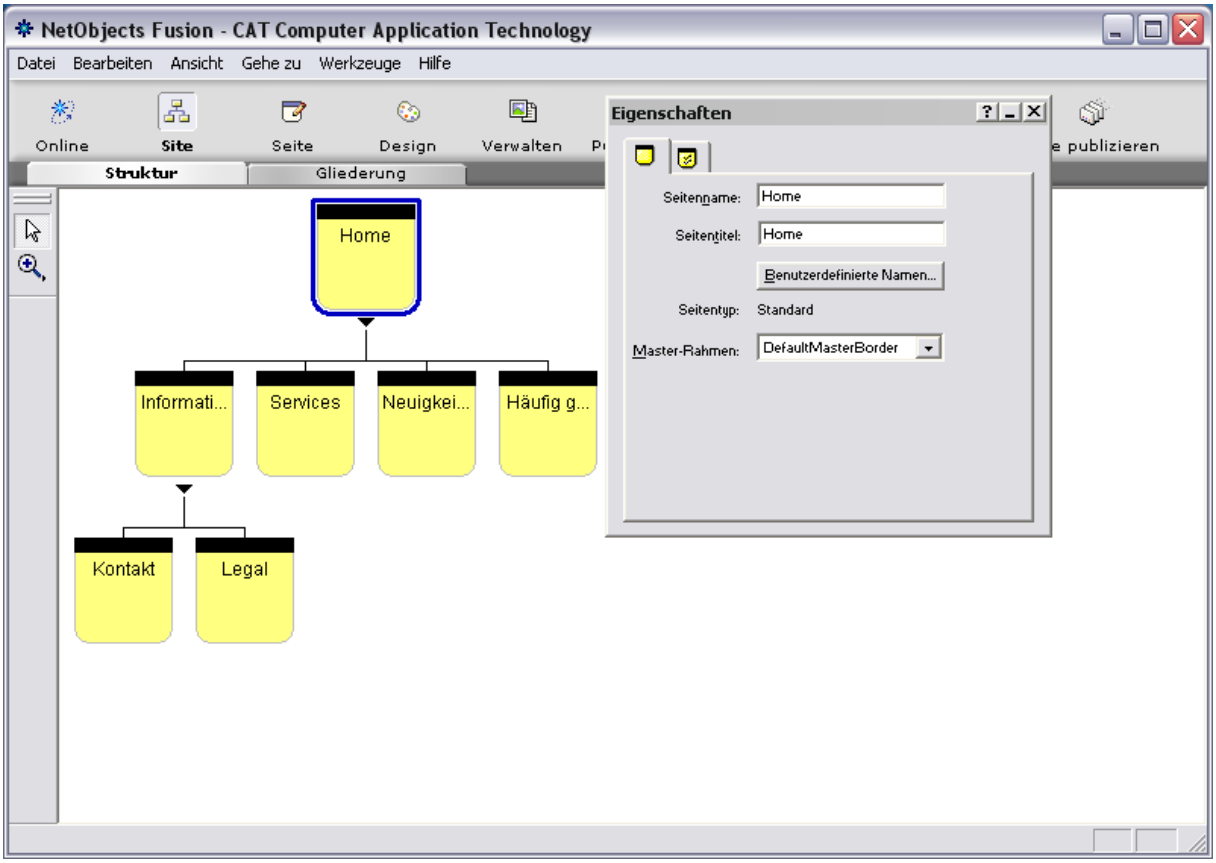


Abbildung 1-3: Webseitenübersicht in NetObjects Fusion. Grafische Darstellung einer hierarchischen Aufrufstruktur.

1.0.2 Vorteile

Die Vorteile des Einsatzes von Rapid-Development-Systemen mit WYSIWYG-Darstellung sind sofort evident:

- Verkürzung des Entwicklungszyklus

Die Erstellung eines Prototyps (bei Rapid-Prototyping-Systemen) bzw. die Erstellung eines ersten einsatzfähigen Systems ist zumindest in einfachen Fällen relativ schnell mit wenig Aufwand machbar.

- Frühzeitige Erkennung von Fehlern

Der relativ kurze Entwicklungszyklus ermöglicht ein iteratives Vorgehen im Projektablauf. Fehler oder Mängel können schnell erkannt und behoben werden.

- Frühzeitige Abstimmung mit potentiellen Nutzern

Nicht nur der Entwickler erhält in einem frühen Stadium der Entwicklung des Dialogsystems einen unmittelbaren Eindruck vom zu erwartenden Endergebnis sondern auch der spätere Benutzer. Ist dieser in die Entwurfs- und Entwicklungsphase aktiv mit einbezogen, erhöht dies seine Akzeptanz gegenüber dem zu erstellenden System erheblich.

- Standardisierung für Oberfläche und technische Implementierung

Die automatische Codegenerierung bringt in gewissem Grad eine Standardisierung sowohl was die Benutzeroberfläche betrifft als auch im Hinblick auf die erzeugte technische Implementierung. Ein eventuell später notwendiger Eingriff in den generierten und damit relativ „normierten" Code dürfte in der Regel leichter vorzunehmen sein als ein Eingriff in unter Umständen durch mehrere Entwickler individuell erstellten Code.

1.0.3 Nachteile

So evident die Vorteile erscheinen, so gravierend sind die Nachteile beim Einsatz derartiger Systeme, zumindest was den aktuellen Leistungsstand der Tools betrifft. Die Nachteile werden insbesondere dann sichtbar, wenn die Komplexität der Anwendung steigt, bzw. dynamische Aspekte eine verstärkte Rolle spielen:

- WYSIWYG nur für statische Komponenten

Die einfache Erstellung der Dialoge in „WYSIWYG-Manier" kann im Allgemeinen nur für a priori in Struktur und Inhalt festliegende Dialogobjekte durchgeführt werden. Dies schränkt die Anwendbarkeit der Tools gerade da stark ein, wo eigentlich ihr entscheidender Vorteil erwartet wird.

- Rudimentäre, unvollständige Darstellung der Dynamik

Nur in wenigen Werkzeugen und dann nur rudimentär ist eine Darstellung der Dialogdynamik mittels deskriptiver und/oder grafischer Sprachelemente bisher unterstützt. In vielen Fällen muss zur Darstellung der Dynamik auf eine Skriptsprache oder allgemeine Programmiersprache zurückgegriffen werden.

- Fehlende Gesamtsicht des Makrodialogs, der Dialogstruktur

Weder der Benutzer des Systems noch der Entwickler erhalten insbesondere eine Gesamtsicht auf die Struktur des Dialogs. Allerdings kann der Ablauf meist sehr schnell simuliert werden.

- Keine abstrahierende logische Darstellung

Rapid-Development-Tools sehen keine Möglichkeit einer abstrakten, von der konkreten Implementierung losgelösten Darstellung des Dialogs.

- Verfrühte Festlegung auf konkrete Objekte

Der Entwickler ist gezwungen, sich sofort auf ein konkretes Dialogobjekt, eine bestimmte Widgetklasse für die Realisierung des Dialogs festzulegen. Dies birgt die Gefahr in sich, dass der Entwickler sich auf Implementierungsdetails konzentriert, bevor überhaupt wichtige logische Aspekte des Dialogs geklärt sind. Häufig muss dann mit der Implementierung von neuem begonnen werden, wenn eine auch nur leichte Korrektur auf der Ebene der logischen Anforderungen an die Funktionalität des Dialogs nötig wird. Z.B. erfordert die nachträgliche Erkenntnis, dass an einer Stelle im Dialog mit einer variablen Anzahl von Werten statt mit einer fixen Anzahl zu rechnen ist, in vielen Fällen die Wahl einer neuen Widgetklasse und damit das Verwerfen der kompletten Implementierung: Das Widget muss u.U. anders kreiert werden, Werte müssen mit anderen Funktionen gesetzt und abgefragt werden, usw.

- Fehlende Flexibilität bei Umsetzung auf andere Dialogstile, Dialogarten

Die sofortige Festlegung auf konkrete Dialogobjekte bereits in einer frühen Phase der Entwicklung erzwingt die frühe Wahl eines bestimmten Dialogstils bzw. einer bestimmten Dialogart. Eine automatische Umsetzung wird durch Rapid-Development-Systeme wenn überhaupt nur insofern unterstützt, dass ggf. eine Transformation in ein anderes Basissystem der Implementierung und/oder in ein anderes „Look-and-Feel" (z.B. von Motif zu Windows) gewechselt werden kann. Eine Umsetzung in unterschiedliche Dialogarten, etwa ein Wechsel von grafischem Dialog in Kommandodialog auf einer Kommandozeile ist damit nicht möglich.

1.1 Abstrakte Modellsysteme

Als Alternative zum Einsatz eines Werkzeuges zum Rapid Development oder Rapid Prototyping, also eines so genannten Lower-Case-Tools, bietet sich prinzipiell der Einsatz einer abstrakten Modellsprache, das eventuell hinsichtlich Modellspezifikation und Generierung einer konkreten Implementierung durch so genannte Upper-Case-Tools unterstützt wird.
Wir wollen im Folgenden auf einige zu Grunde liegende Modell-Ansätze kurz eingehen. Es sind dies Ansätze, die zum einen die statische Modellierung der Dialogkomponenten und/oder die Modellierung der Dialogdynamik unterstützen.

1.1.0 Statische Objektmodellierung

Zur Darstellung und Erzeugung von Dialogkomponenten kann ein statisches Objektmodell oder Datenmodell herangezogen werden. Es kann nahe liegend zum einen dazu dienen, die Statik von Oberflächenelementen am Bildschirm festzulegen, zum anderen aber auch, allerdings in eingeschränktem Maße, zur automatischen Ableitung der mit den Oberflächenelementen verbundenen Dialogdynamik und Applikationslogik. Dies gilt insbesondere für Datenbank-Applikationen.

Generierung von Statik und Dynamik aus Datenmodell
So gibt es zum Beispiel Systeme, die für die Generierung der Dialogobjekte Klassendiagramme einer objektorientierten Modellierung zu Grunde legen (Beispiel: Janus-System [BALZE94]). Andere Systeme verwenden das durch ein Datenbankschema vorliegende Datenmodell zum Aufbau der Oberflächenelemente, wie z.B. der Formulare, und zur Realisierung der zugehörigen Dialoge und Applikationslogik zum Erfassen, Ändern und Suchen der Daten (Beispiel: DialogBuilder-Dialoggenerator [SIEMEN94]).
In der Regel ist das Ergebnis von Modellierung und Generierung eine relativ schematische, stereotype Umsetzung an der Oberfläche, die aber in einfachen Anwendungsfällen ausreichen mag. Das statische Datenmodell bzw. Objektmodell genügt allerdings nicht für eine umfassende und flexible Definition der Dynamik des Dialogs im allgemeinen Fall.

1.1.1 Modellierung der Dynamik

Für eine spezifische Darstellung der Dialogdynamik gibt es eine Reihe von Modellansätzen, die zum Großteil auf allgemeinen Modellansätzen zur Darstellung von Dynamik beruhen, zum Teil auch aus anderen Anwendungsgebieten übernommen und für die spezifischen Anforderungen der Dialogmodellierung erweitert bzw. angepasst wurden.
Wir wollen im Folgenden nur kurz auf diese Ansätze eingehen, da sie aus der Literatur als bekannt vorausgesetzt werden können. Es sollen nur die wesentlichen Merkmale im Hinblick auf Vor- und Nachteile für die Dialogmodellierung angesprochen werden, die diese Arbeit, insbesondere die Entwicklung des Ereignisstellenmodells beeinflusst haben oder mit ihr vergleichbar sind. Auf in diesem Sinne wichtige Beispiele wird im Kapitel 9 noch näher eingegangen.

1.1.1.0 Zustandsübergangsdiagramme

Basis einer ganzen Reihe von Modellen ist die Darstellung der Dynamik mithilfe von Zustandsübergangsdiagrammen (ZÜD) [FOLE90][GREEN86].

Einfache Zustandsübergangsdiagramme bestehen aus Knoten (grafisch dargestellt durch Kreise) und gerichteten Kanten (Pfeile). Die Knoten definieren die Zustände, die Kanten die Zustandsübergänge. An den Kanten können Ereignisse und/oder Aktionen notiert werden, Ereignisse, die den Zustandsübergang auslösen bzw. Aktionen, die beim Übergang ausgeführt werden. Die Aktionen können auch in den Knoten eingetragen werden.
Eine Erweiterung der einfachen Zustandsübergangsdiagramme stellen so genannte rekursive Zustandsübergangsdiagramme (RTN´s, Recursive Transition Networks) und Augmented Transition Networks (ATN´s) dar. In einem RTN repräsentieren Knoten ggf. den rekursiven Aufruf eines Subdiagramms. In einem ATN sind zusätzlich Variablen und Funktionen erlaubt, mittels derer z.B. Bedingungen für Zustandsübergänge und Wertzuweisungen formuliert werden können. Manchmal wird ein ATN als Erweiterung eines RTN gesehen, in anderer Literatur werden ATN´s mit rekursiven Subdiagrammen auch als Augmented Recursive Transition Networks bezeichnet.
Generell können wir die erweiterten Zustandsübergangsdiagramme in eine Klasse verallgemeinerter Transitionssysteme einordnen [EICKEL01].
Ein Vorteil bei einfachen Zustandübergangsdiagrammen liegt in dem klaren Konzept und der Möglichkeit einer intuitiv eingängigen grafischen Darstellung. Durch die Erweiterungen in den Modellen wird die Mächtigkeit signifikant gesteigert. Dabei nimmt der Anteil der Grafik jedoch ab und die Darstellung verlagert sich immer mehr hin zur textuellen Notation. Der Anwender ist gezwungen, eine eigene Modellsprache zu erlernen, die zunächst nicht mehr sofort einsichtig ist.
Ein Kernproblem bei der Darstellung von Zuständen und Übergängen ist, dass schon bei nicht allzu komplexen Anwendungen die Anzahl der Zustände und die Anzahl der Übergänge extrem ansteigt. Dies wirkt sich ganz besonders negativ aus, wenn nebenläufige Vorgänge beschrieben werden sollen.
Im Kapitel 9 werden wir auf zwei herausragende Ansätze näher eingehen, die das Problem der Nebenläufigkeit in Zustandsübergangsdiagrammen adressieren, nämlich das Koroutinenkonzept von Jacob [JACOB86] und die State Charts [HAREL87][OESTER97], wobei letztere ihren Eingang in die Unified Modelling Language (UML) für objektorientierte Modellierung und Design gefunden haben. Es sei jedoch bereits hier angemerkt, dass beide Modelle für eine benutzergerechte und auch entwicklergerechte Modellierung von grafischen Dialogsystemen nur bedingt tauglich sind. Bei Jacobs Koroutinen-Konzept geht der Vorteil der grafischen Darstellung verloren und State Charts sind, wenn auch eine grafische Darstellung paralleler Zustände möglich ist, zu sehr der Zustandssicht verhaftet und dadurch z.B. für eine WYSIWYG-Darstellung wenig geeignet.

1.1.1.1 Petrinetze

Petrinetze [PETRI62][HERRT89][OBERQ87] wurden ursprünglich für die Darstellung von Produktionsprozessen eingeführt. Den Kern der verschiedenen Varianten von Petrinetz-Modellen (einfache Petrinetze, Plätze-Transitions-Netze, Prädikats-Transitionsnetze, RFA-Netze), bilden Stellen (auch Plätze genannt), Transitionen und gerichtete Kanten. Transitionen werden über die Kanten mit Eingangs- und Ausgangsstellen verbunden. Um dynamische Prozesse zu modellieren, werden den Stellen so genannte Marken zugeordnet. Eine Transition „feuert“ (in einem einfachen Petrinetz), wenn alle Eingangsstellen eine Marke aufweisen und reine Ausgangsstellen ohne Marke sind. Die Marken der Eingangsstellen werden mit dem Feuern beseitigt. Dafür wird jede Ausgangsstelle mit einer Marke versehen. Petrinetze sind insbesondere für die Darstellung nebenläufiger Vorgänge geeignet.
Auf eine Variante, die Dialognetze [JANSS96], wird in Kapitel 9 noch gesondert eingegangen. Ein Nachteil der Petrinetze liegt darin, dass die Begriffe der Welt der Dialogmodellierung teilweise erst in die Begriffswelt der Petrinetze umgesetzt werden muss

und somit die intuitive Darstellung leidet. Dieser Nachteil kann noch ausgemerzt werden, wenn eine Transformation in dialogspezifische Sprachelemente durchgeführt wird, wie dies z.B. bei den so genannten Dialoggraphen [SCHLU95] der Fall ist. Ein vielleicht schwerer wiegender Nachteil ist, dass mit Petrinetzen die Darstellung des Datenflusses nur mit zusätzlichen nicht grafischen Sprachmitteln erreicht werden kann.

1.1.1.2 Dialog-Grammatiken

Eine weitere bekannte Technik der Modellierung von Dialog ist die Beschreibung in Form einer Grammatik [GREEN86]. Die Struktur des Dialogs wird dabei mit Hilfe der Produktionsregeln unter Verwendung terminaler und nichtterminaler Symbole festgelegt. Über die Produktionsregeln kann die Analyse der Benutzer-Eingaben durchgeführt werden. Werden während der Analyse bestimmte Produktionen erkannt, wird den einzelnen Produktionen zugeordnete Information zum Aufruf von Anwendungsfunktionen und zur Ausgabe von Information an der Präsentationsschnittstelle genutzt. Eine formale Darstellung der Produktionsregeln kann textuell (etwa in bekannter BNF-Notation) erfolgen. Als grafische Variante der Darstellung dienen Syntaxdiagramme oder baumartige Darstellungen der Produktionsregeln [EICKEL01].
Eine Variante zur Dialogmodellierung mit Grammatiken bilden Attribut-Grammatiken, bei denen jedem nichtterminalen Symbol ein Attribut des Typs „Applikation“ zugeordnet ist [EICKEL01].
Ein Vorteil des Einsatzes von Grammatiken ist, dass es sich um eine insbesondere im Umfeld des Compilerbaus bewährte Technik handelt, die teilweise auch Anwendungsprogrammierern nicht unbekannt ist. Die Darstellung einer sequenziellen Dialogstruktur, wie diese etwa in einem Kommandodialog, Frage-Antwort-Dialog oder Menüdialog vorkommt, ist in Form von Produktionen nahe liegend. Für den Benutzer ist die Darstellung der Dialogstruktur in Form einzelner Produktionen allerdings etwa im Vergleich zu Zustandsübergangsdiagrammen weniger geeignet [JANSS96]. Ein weiterer Nachteil ist, dass mit Ausnahme einiger Erweiterungen die Darstellung von Nebenläufigkeit nicht unterstützt wird.

1.1.1.3 Ereignismodelle

Ereignismodelle basieren auf der Sicht, dass in einem Dialogsystem so genannte Eventhandler meist parallel auf bestimmte Ereignisse warten. Bei Eintreffen eines Ereignisses, in der Regel aber nicht notwendig eines hardwarenahen Eingabeereignisses (z.B. Mausklick), wird dieses Ereignis von einer zentralen Instanz aufgenommen und an die oder den entsprechenden Eventhandler verteilt. Der jeweilige Eventhandler arbeitet dann das Ereignis etwa mit Aufruf einer Applikationsfunktion, der Durchführung einer Ausgabe, dem Versenden eines weiteren Ereignisses oder dem Aktivieren oder Deaktivieren eines anderen Eventhandlers ab um dann ggf. die Kontrolle wieder an die zentrale Instanz zu übergeben (siehe auch Abschnitt 9.1 Ereignismodell nach Green).
Ereignismodelle heben sich von den bisher genannten Modellen in einigen Gesichtspunkten ab. Sie sind zunächst nicht für eine grafische Modellierung, wie dies bei Petrinetzen und Zustandsübergangsdiagrammen der Fall ist, gedacht. Ereignismodelle bilden vielmehr eher eine konzeptuelle Grundlage für eine Reihe von Basissystemen, aber auch User-Interface-Management-Systemen, gekoppelt mit direkter Programmierung (siehe unten) zur Implementierung von Dialogsystemen, wie dies z.B. in X-Windows/ Motif, Microsoft-Windows, Java/Swing, aber auch in Skriptsprachen wie Java-Script der Fall ist.

Wegen seiner technischen Orientierung und der Aufteilung des Systems auf verschiedene Eventhandler, deren Eventroutinen „auszuprogrammieren“ sind, ist das Ereignismodell

zunächst nicht für eine intuitive und übersichtliche Darstellung des logischen Dialogs geeignet.
Das Ereignismodell bildet jedoch eine wesentliche Grundlage auch für das in dieser Arbeit entwickelte Ereignisstellenmodell mit der grafischen Modellierung in Ereignisstellendiagrammen.

1.2 Direkte Programmierung

Als wesentliche dritte Kategorie von Ansätzen wollen wir die Ansätze mit direkter Programmierung bezeichnen. Dies sind Ansätze, bei denen relativ implementierungsnah, etwa mit einer Skriptsprache (Java-Script, Perl, …) oder einer allgemeinen Programmiersprache (Java, C++, C#, …) direkt vom Entwickler die Anwendung realisiert wird. Wir wollen dazu im weiteren Sinne auch die Verwendung spezifischer Protokollsprachen zählen, wie etwa HTML und darauf aufsetzender Techniken wie ASP (Active Server Pages), JSP (Java Server Pages) oder den Einsatz von Servlets im Umfeld der Realisierung von Netzanwendungen, ohne hier den Anspruch auf Vollständigkeit zu erheben.
Es sind dies generell die Ansätze, die unmittelbar das zur Verfügung stehende Basissystem zur Realisierung nutzen, das Sprachelemente auf einer eher technischen Ebene und weniger logischen Ebene zur Realisierung der Anwendung bietet. Eine exakte und vor allem umfassende Abgrenzung soll und kann im Rahmen dieser Arbeit nicht erfolgen.
Etwas pragmatisch können wir diese Ansätze auch als solche umschreiben, die weder auf Basis einer Sprache zur Beschreibung eines logischen Dialogmodells noch mit Hilfe eines Werkzeuges zur grafischen „WYSIWYG“-Darstellung arbeiten.
Häufig ist in derartigen Ansätzen der direkten Programmierung ein Ereignismodell in Form bestimmter Sprachelemente bzw. durch bestimmte (Klassen-)Bibliotheken (X-Windows Eventhandler, Java-Listener-Konzept, …) unterstützt.

1.2.0 Vorteile

Die Vorteile der direkten Programmierung werden vor allem von versierten Entwicklern geschätzt:

- Flexibilität wegen vollständigem Repertoire an Sprachmitteln des Basissystems

Der Entwickler ist in der Lage, alle Möglichkeiten des Basissystems auszuschöpfen, da er direkt in der Sprache des Basissystems arbeitet. Die Flexibilität des Systems erlaubt in der Regel, exakt die geforderte Funktionalität der Anwendung umzusetzen.

- Hoch dynamisch und potentiell generisch

Die Mächtigkeit der verfügbaren Sprache des Basissystems erlaubt im Allgemeinen das Programm hoch dynamisch und generisch zu gestalten. Der Entwickler kann die Architektur des Codes weitgehend selbst bestimmen und somit die Dynamik sowie die generischen Eigenschaften der Implementierung optimal steuern.

- Durchgängig einheitliche, verbreitete Sprache

Der Grad der Verbreitung der Sprache des Basissystems ist in der Regel größer als der eines speziellen Upper- oder Lower-case-Tools. Dies erleichtert auch dem Management mitunter die Entscheidung für die direkte Programmierung. Zum einen ist dann eine Entwicklungsmannschaft mit dem erforderlichen Know-how schneller zu rekrutieren, zum anderen ist zu erwarten, dass die Wartbarkeit langfristig ohne Abhängigkeit vom Tool-Hersteller gewährleistet ist.

1.2.1 Nachteile

Die direkte Programmierung hat allerdings entscheidende Nachteile, die vor allem aus Gesichtspunkten eines idealen, kostengünstigen Software-Engineerings die Entscheidung gegen diesen Ansatz und für die Verwendung eines Rapid-Development-Systems oder eines modellbasierten Systems nahe legen können:

- Wenig direkte Transparenz

Mit der direkten Programmierung ist in der Praxis die Codierung nach den individuellen Vorstellungen durch die einzelnen Entwickler verbunden, es sei denn im Projekt wird auf andere Weise mit Zusatzaufwand für die Einhaltung einer durchgängigen Architektur und die Umsetzung eines Konzeptes zur softwaretechnischen Qualitätssicherung gesorgt. Dies führt dazu, dass die Einarbeitung in die Funktionalität des Systems und deren Implementierung erfahrungsgemäß viel Aufwand erfordert.

- Fehlende Benutzernähe

Insbesondere wird durch die direkte Programmierung der Zugang für den Benutzer (und auch das Management) in keiner Weise unterstützt. Der Benutzer kann erst nach Fertigstellung das Ergebnis des Entwicklungsprozesses überprüfen, falls nicht vor der Entwicklungsphase eine gesonderte Phase der Abstimmung mithilfe zusätzlicher Dokumentation bzw. aufwändiger Spezifikation erfolgt.

- Keine Unterstützung für logisches Design

Die direkte Verwendung der Implementierungssprache des Basissystems bietet keine Unterstützung für ein systematisches logisches Design des Dialogs. Dies birgt die Gefahr noch stärker als bei Rapid-Development-Systemen in sich, dass das Design - falls es überhaupt durchgeführt wird – von technischen Details beherrscht wird und dass dabei wichtige logische Gesichtspunkte übersehen werden.

- Starke Abhängigkeit von der aktuellen Technik

Die komplette Implementierung mittels direkter Programmierung erhöht naturgemäß die Abhängigkeit vom jeweilig verwendeten Basissystem. Eine Umsetzung in ein anderes System wird dadurch sehr schwierig und führt in den meisten Fällen zu einer häufig unnötigen Neuentwicklung.

1.3 Resümee

Eine übersichtliche, systematische und logische Darstellung der dynamischen Strukturen des Dialogs fehlt in derzeitigen Rapid-Development-Systemen. Die Definition der Dynamik der Oberfläche findet im Wesentlichen im Programmcode statt und ist schwer zugänglich. Generell wird keine abstrakte, logische Modellierung des Dialogs unterstützt.
Rapid-Development-Systeme mit grafischen Dialogobjekt-Editoren ermöglichen eine schnelle konkrete Darstellung statischer Elemente des Dialogs und nur in einfachen Fällen eine rasche Implementierung von Dynamik.

Modellbasierte Systeme hingegen unterstützen im Prinzip einen abstrakten, systematischen Entwurf.
Einige modellbasierte Systeme bieten insbesondere den Ansatz der Darstellung der Dynamik. Die Möglichkeiten zur Darstellung der Dynamik sind jedoch auch bei bisherigen Systemen mit grafischer Darstellung stark eingeschränkt.
In modellbasierten Systemen fehlt zudem eine ausreichend intuitive Darstellung des Dialogs, die auch einem potentiellen Benutzer und EDV-Laien eine Vorstellung der Bedienoberfläche des zukünftigen Systems erschließt (kein WYSIWYG). Auch für den Entwickler ist häufig eine umfassende Gesamtdarstellung von Statik und Dynamik des Dialogs nicht möglich. Dies

gilt insbesondere weil einerseits nicht an eine Gesamtdarstellung der Zusammenhänge gedacht ist (Ereignismodell), eine Darstellung der Nebenläufigkeit in den meisten Fällen nicht praktikabel ist (z.B. Zustandsübergangsdiagramme) oder generell nicht unterstützt wird (Grammatiken).
Eine Umsetzung des Modells ist nicht zuletzt wegen einer eingeschränkten und manchmal auch nicht problemnahen Sprache in eine konkrete Oberfläche schwierig und umständlich.

Der große Vorteil der direkten Verwendung einer Programmiersprache ist hingegen, dass alle Möglichkeiten insbesondere zur Implementierung dynamischer Dialogteile unmittelbar zur Verfügung stehen. Wie bei Rapid-Development-Systemen ist allerdings eine übersichtliche. logische und leicht zugängliche Darstellung der Dynamik und in diesem Fall auch der statischen Teile nicht vorhanden.

Ein Einsatz aller drei geschilderten Ansätze führt zu Inkonsistenz und Doppelarbeit. Auch neuere Versuche zur Integration in sogenannten Round-Trip-Systemen, die einen nahtlosen Übergang vom Modell zum generierten Code und umgekehrt vom (teilweise) manuell erstellten oder geänderten Code zurück zur Modelldarstellung erlauben, haben sich bisher nicht durchgesetzt. Nach Meinung des Autors liegt dies zum Teil an den mangelnden und wenig intuitiven Darstellungsmöglichkeiten insbesondere der Dialogdynamik auf der logischen Modellebene.

2 Neuer integrierter Modellierungsansatz

2.0 Kernziele

Aus der oben geführten Diskussion über Vor- und Nachteile des Rapid-Development-Ansatzes, modellbasierter Ansätze und der Ansätze mit direkter Programmierung resultiert ein integrativer Ansatz, der die wesentlichen Vorteile der diskutierten Ansätze vereint. Aus ihm lassen sich zwei Kernziele für die Entwicklung von Dialogsystemen ableiten:

- Zum einen soll die Möglichkeit der logischen Modellierung des Dialogs aus dem modellbasierten Ansatz mit den Möglichkeiten zur Darstellung der Dynamik, wie er bei direkter Programmierung nahezu uneingeschränkt gegeben ist, verbunden werden. Damit sollen die bisher bestehenden Einschränkungen zur Darstellung der Dynamik in grafisch orientierten Modellen beseitigt werden.
- Zum anderen soll eine Integration logischer Modellierung mit layoutnaher WYSIWYG-Darstellung erfolgen, wie sie in grafischen Rapid-Development-Systemen geboten wird. Es soll einerseits eine intuitive Darstellung auch für den potentiellen Benutzer des Systems ermöglicht werden. Andererseits soll auch im Vorgang des Entwurfs ein zwangloses Umschalten von konkreter zu abstrakter Darstellung und umgekehrt ermöglicht werden.

2.0.0 Logische Dialogmodellierung unter Einbeziehung der Dynamik

Ein erstes Kernziel ist die Entwicklung einer problemnahen Modellsprache, die eine einfache, übersichtliche Modellierung der logischen Dialogoberfläche unter Einbeziehung der Dialogdynamik erlaubt.
Wir wollen im Folgenden informell die Begriffe, die das erste Ziel der Arbeit umschreiben, nämlich die Modellierung von „logischem Dialog“ und „Dialogdynamik“ etwas genauer umreißen.

2.0.0.0 Begriffsklärung logischer Dialog

Unter logischem Dialog wollen wir alle Merkmale oder Eigenschaften des Dialogs verstehen, die zur Erfüllung des Informationsaustausches in einer Anwendung im Sinne der Aufgabenstellung einer Anwendung wesentlich oder bedeutend sind. Es sind also Merkmale des Dialogs, die zur Entscheidung dafür herangezogen werden können, ob der Dialog für die Erfüllung der Aufgabe der Anwendung geeignet ist. Unwesentlich oder unbedeutend sind Merkmale, die zur Durchführung der Aufgabe unnötig sind oder Merkmale, die durch andere ersetzt werden können, ohne die Erfüllung der Aufgabenstellung zu beeinträchtigen.
Die Merkmale des logischen Dialogs ergeben sich beispielsweise aus einer systematischen Analyse der Aufgabenstellung einer Anwendung und insbesondere des sich daraus resultierenden notwendigen Informationsaustausches.
Unwesentlich sind besonders Merkmale, die eine bestimmte austauschbare Form oder Art des Dialogs, einen bestimmten austauschbaren Dialogstil, kurz ein bestimmtes austauschbares Layout (siehe unten) beschreiben.
Dabei nehmen wir an, dass die „eigentliche“ Aufgabenstellung einer Anwendung im allgemeinen nicht die Wahl eines bestimmten Layouts vorgibt.
Wesentlich sind in der Regel Merkmale, die den Inhalt der im Dialog übermittelten Information beschreiben.

Dabei muss davon ausgegangen werden, dass der Inhalt des Dialogs von der logischen Aufgabenstellung, dem Aufgabenmodell, einerseits und andererseits von der logischen Sicht der Benutzer, dem Benutzermodell, abgeleitet wird. Durch ein Benutzermodell wird die logische Sicht des Benutzers auf die Aufgabenstellung, die durchzuführenden Tätigkeiten und die zu bearbeitenden (Informations-)Objekte, beschrieben.
Entsprechend sind bei der Erarbeitung des logischen Dialogmodells die bekannten Forderungen der Softwareergonomie nach Aufgabenangemessenheit und Erwartungskonformität zu berücksichtigen.

Einordnung logischen Dialog in semantische Ebene des Schichtenmodells nach Foley
Das Sprachenmodell von Foley [FOLEY74] unterscheidet nach einer konzeptuellen, semantischen, syntaktischen und lexikalischen Ebene, in der die Realisierung eines Dialogsystems beschrieben werden kann. Wenn wir von dieser Schichtung durch Foley ausgehen, ist ein logisches Dialogmodell am ehesten der semantischen Ebene zuzuordnen. Erst in einer darunter liegenden Schicht wird die Art des Dialogs, der Dialogstil festgelegt.

In der konzeptuellen Ebene wird die Aufgabenstellung des Systems geklärt. Hier werden die Leistungen, Funktionen des zu erstellenden Systems beschrieben. Es wird definiert, *was* das Anwendungsprogramm zu tun hat. Die konzeptuelle Ebene spiegelt das Benutzermodell vom System wider. Es wird keine Aussage über das *Wie* der Realisierung gemacht.

In der semantischen Ebene wird eine Detaillierung dahingehend vorgenommen, dass Informationseinheiten definiert werden, die zwischen System und Benutzer auszutauschen sind. Die semantische Ebene kennt die Systemfunktionen, die für die Bedienoberfläche relevant sind und im Dialog angesprochen werden. Ebenso wird definiert, welche Objekte an der Oberfläche präsentiert werden sollen. Es wird festgelegt, welche Bezüge zwischen den auszutauschenden Einheiten im Sinne der Aufgabenstellung bestehen, etwa eine notwendige Reihenfolge beim Aufruf von Funktionen, etc. Die semantische Ebene kennt alle genannten Funktionen und Objekte in ihrer Bedeutung, nicht aber in der Form der Realisierung.

In der syntaktischen Ebene erfolgt letztlich die Festlegung der tatsächlich möglichen sprachlichen Einheiten (Token) und der konkret möglichen Folge der einzelnen Dialogschritte. Hier wird z.B. bestimmt, in welchem Zustand welche Eingaben erlaubt sind.

Erst in der lexikalischen Ebene werden die sprachlichen Einheiten der Eingabe-/ Ausgabesprache auf konkrete Hardware-Ressourcen (z.B. Eingabegeräte) bzw. -Ereignisse abgebildet.

2.0.0.1 Abgrenzung zum Layout

Logischer Dialog ist gegen das Layout des Dialogs abzugrenzen. Unter Layout verstehen wir u.a. die verschiedenen Dialogarten, in denen derselbe logische Dialog realisiert werden kann. Ebenso kann ein und der derselbe logische Dialog meist in unterschiedlichen Dialogstilen ausgearbeitet sein.

Dialogarten

Als klassische Dialogarten kennen wir z.B. den Frage-Antwort-Dialog, Kommandodialog, Menüdialog, Formulardialog sowie den grafischen Dialog [ZEIDL92]. Wir können diese ohne Anspruch auf Vollständigkeit durch die Kategorien Kiosksysteme und multimedialen Dialog erweitern.
Im Frage-Antwort-Dialog kann z.B. inhaltlich gesehen prinzipiell derselbe Informationsaustausch erfolgen wie im Kommandodialog. Ein wesentlicher Unterschied ist lediglich, dass im

Frage-Antwort-Dialog das System die Führung des Dialogs übernimmt, während im Kommandodialog der Benutzer die Initiative ergreift und im Frage-Antwort-Dialog sich der Dialog mitunter sehr lange hinzieht, bis die gewünschte Information übermittelt ist Allerdings sieht man an diesem Beispiel, dass die Wahl der Dialogart nicht in allen Anwendungsfällen unbedeutend für die Erfüllung der Aufgabenstellung ist. Es kann z.B. in bestimmten Fällen entscheidend sein, welcher Dialogpartner die Initiative des Informationsaustausches übernehmen kann. Denken wir z.B. an Dialogsysteme zur Steuerung von Produktionsprozessen, wo sorgfältig entschieden werden muss, in welchem Umfang in kritischen Situationen das System oder der Benutzer den Dialog steuert. (Eine Notbremse sollte immer für den Benutzer möglich sein. Gleichzeitig muss ausgeschlossen sein, dass sich der Benutzer dadurch selbst oder andere Mitarbeiter gefährdet.)

Dialogstile

Noch unbedeutender für die logisch übermittelte Information kann die Wahl eines bestimmten Dialogstils gesehen werden. Was in der Regel in so genannten Stylesheets definiert ist, wie etwa die Wahl eines bestimmten Zeichensatzes, einer Schriftgröße, einer bestimmten Hintergrundfarbe, etc. ist für den logischen Dialog meist irrelevant. Allerdings kann auch hier bei falscher Wahl manchmal die Qualitätsanforderung der Aufgabenangemessenheit verletzt sein. Denken wir z.B. an die Wahl der Schriftgröße bei Sehbehinderten, etc. (Der Vollständigkeit halber sei hier angemerkt, dass Stylesheets etwa bei XML auch dazu verwendet werden können, um den Inhalt der Information zu beeinflussen, z.B. zu filtern. Diese Funktionalität ist aber nicht als Merkmal eines bestimmten Dialogstils zu sehen.)

2.0.0.2 Definition der Dynamik

Unter Dynamik des Dialogs verstehen wir allgemein seine zeitliche Entwicklung. Dies umfasst im Wesentlichen die mögliche Abfolge von Eingaben und Ausgaben an der Dialogschnittstelle sowie den Ablauf des Dialogs in Abhängigkeit der spezifischen Situation, des aktuellen Kontexts der Anwendung.
Wir können hier wiederum zwischen einer Dynamik des Dialogs unterscheiden, die sich aus der logischen Aufgabenstellung ergibt, und einer Dynamik, die aus der Wahl eines bestimmten Layouts und der technischen Implementierung resultiert. In Abschnitt 4.2.1 wird noch näher auf Kriterien der Dialogdynamik eingegangen.
Als Beispiel für eine layoutbedingte Dynamik sei hier genannt, dass von der Dialogart abhängig der Ablauf des Dialogs bestimmt werden kann: Während in einem Formulardialog mehrere Datenfelder gleichzeitig am Bildschirm erscheinen und häufig in beliebiger Reihenfolge dort Werte eingegeben werden können, werden in einem Frage-Antwort-Dialog die Felder einzeln hintereinander ausgegeben und sukzessive in bestimmter Reihenfolge abgefragt. Dieses Beispiel zeigt, dass das, was im Formulardialog intuitiv als statische Komponente gesehen wird, nämlich das Formular mit seinen fix vorgegebenen Feldern, in einer anderen Dialogart in eine dynamische Komponente umgesetzt wird, nämlich im Beispiel in eine Dialogfolge mit Ausgabe der einzelnen Felder.

2.0.1 Integration logischer und layoutbehafteter Darstellung

Ein weiteres Kernziel ist die Verbindung des Ansatzes des modellbasierten Dialogentwurfs mit wesentlichen Vorteilen des Rapid-Development-Ansatzes, nämlich der konkreten WYSIWYG-Darstellung und der schnellen, direkten Umsetzung in ein ablauffähiges System. Die Erfahrung zeigt, dass in der Phase des Entwurfs eines Dialogsystems Vorstellungen über die Gestaltung der Bedienoberfläche auf unterschiedlich konkreter bzw. abstrakter Ebene vorliegen. In der Modellsprache bzw. im die Sprache unterstützenden Werkzeug sollen deshalb diese konkreten und abstrakten Vorstellungen erfasst und weiter entwickelt werden

können. Im Idealfall können dann je nach Zielsetzung sukzessive abstrakte Elemente konkretisiert und umgekehrt konkrete Elemente abstrahiert werden.
Liegt das Ziel darin, zu einem ablauffähigen System zu gelangen, werden schrittweise abstrakte Elemente durch ihre Konkretisierungen ersetzt bzw. den abstrakten Elementen werden konkrete „Implementierungen“ zugeordnet.
Ist umgekehrt das Ziel, zu einer abstrakten, logischen Darstellung des Systems zu gelangen, findet im Rahmen des Modells eine Ersetzung konkreter durch abstrakte Elemente statt. Dieser Weg wird beispielsweise dann beschritten, wenn in einer Phase der Analyse aus konkreten, exemplarischen Vorstellungen des Benutzers eine abstrakte, logische Darstellung gewonnen werden soll. Der Weg der Abstraktion kann auch dann gewählt werden, wenn eine Implementierung bereits vorliegt und eine nachträgliche Umsetzung in eine andere Konkretisierung erfolgen soll, z.B. eine Umsetzung in eine andere Dialogart.
Der Designer soll also nicht zu einer Vorgehensweise Top-down oder Bottom-up gezwungen sein.

2.1 Abgeleitete Ziele und Rahmenbedingungen

Die im Folgenden genannten Ziele vertiefen die oben genannten Kernziele bzw. stellen wesentliche Rahmenbedingungen für die Entwicklung des Modellsystems dar:

- Grafische Darstellung

Ziel ist, die Modellierung weitgehend in grafischer Darstellung durchführen zu können. Die grafische Darstellung wird angestrebt, weil wir annehmen, dass diese Darstellungsform für den Entwickler wie auch den Benutzer einen schnellen und umfassenden Zugang zum Modell bietet.

- Ergebnisorientierte Darstellung

Damit verbunden ist, dass wir in erster Linie (aber nicht notwendigerweise) die Darstellung grafischer Oberflächen anstreben und diese in WYSIWYG-Darstellung bereits im Modell erfassen wollen. Der Entwickler wie auch der potentielle Benutzer sollen möglichst früh einen konkreten Eindruck vom zu erstellenden System erhalten.

- Offenes System

Wir gehen davon aus, dass eine vollständige Darstellung des zu erstellenden Systems nicht mit rein grafischen Sprachelementen der zu entwickelnden Modellsprache möglich ist. Ziel ist deshalb eine Kombination der grafischen Sprachelemente mit textuellen Elementen sowie mit der Möglichkeit, komplexe Sachverhalte auch in einer Programmiersprache des Basissystems ausdrücken zu können, ohne dabei den grafisch definierten Teil des Modells verwerfen zu müssen. Grafische und textuelle Darstellung sowie die Darstellung in einer Programmiersprache des Basissystems sollen sich ergänzen.

- Freiheit im Entwurf

Das angestrebte System zur Modellierung soll Entwurf, Implementierung und Dokumentation unterstützen. Insbesondere soll die Phase des Entwurfs von Anfang an durch das angestrebte System begleitet werden. Dies bedeutet, dass z.B. zunächst skizzenhafte, unvollständige Modellinformation schrittweise ergänzt und vervollständigt werden kann. Besonders verfolgt wird in dieser Arbeit das Ziel, dass der Systemdesigner in einem Bottom-up-Ansatz ausgehend von konkreten Beispielen möglichst nahtlos zu einer vollständigen Beschreibung gelangen kann.

3 Einführung in ein allgemeines Grundmodell zur Beschreibung von Dialogstatik und Dynamik

Im Folgenden Kapitel wird eine Sicht des Dialogs entwickelt, die im weiteren Verlauf als Grundlage für die Diskussion der Modellierung dienen soll. Wir wollen diese Sicht als Grundmodell bezeichnen. Es werden darin zentrale Elemente des Dialogs und wichtige Relationen dieser Elemente zueinander festgelegt, die als Ausgangspunkt für die Entwicklung einer Modellsprache zur Beschreibung von statischer und dynamischer Dialogstruktur dienen soll. Das Grundmodell beschreibt den Dialog auf einer abstrakten, allgemeingültigen Ebene. D.h. die durch das Grundmodell vermittelte Sicht auf den Dialog kann auf unterschiedlichste Konkretisierungen abgebildet werden.

3.0 Zentrale Begriffe des Dialogmodells

3.0.0 Dialogereignis

Der Begriff des Ereignisses ist bekannt aus anderen formalen Modellansätzen, wie z.B. Zustandsübergangsdiagrammen (Zustandsautomaten) oder Ereignismodellen. Ein Ereignis drückt im Allgemeinen einen dynamischen Vorgang aus. D.h. mit einem Ereignis ist auf konkreter physikalischer Ebene Zeit und Ort verbunden, an dem das Ereignis festgestellt werden kann. Wenn wir von der konkreten physikalischen Ebene abstrahieren, wollen wir das Ereignis einer Stelle in einem virtuellen Stellenraum (siehe unten) zuordnen. Dialogereignisse tragen dem dynamischen Vorgang des Austausches von Information Rechnung.
Der Austausch der Information kann in Form von vier Ereignistypen beschrieben werden:

Eingabe
Ein Eingabeereignis ist dann zu verzeichnen, wenn der Benutzer Information an einer Dialogschnittstelle eingibt, bzw. verfügbar macht. Sehen wir das Ereignis als Ergebnis einer Operation, können wir diese im übertragenen Sinne auch als Schreiben des Benutzers bezeichnen.

Ausgabe
Entsprechend ist ein Ausgabeereignis dann zu verzeichnen, wenn das System Information an einer Dialogschnittstelle zur Verfügung stellt. Die zugrunde liegende Operation kann als Schreiben des Systems bezeichnet werden.
Wenn wir Eingabe und Ausgabe als Ereignisse in unser Modell aufnehmen, entspricht dies einer symmetrischen Sicht der Dialogschnittstelle zwischen den Dialogpartnern Benutzer und System bzw. Applikationslogik.

Die ersten beiden Typen von Dialogereignissen reichen in vielen Fällen aus, um auf einer mehr oder weniger abstrakten Ebene die Vorgänge des Informationsaustauschs hinreichend exakt zu beschreiben. Mit dem Vorgang der Eingabe wird meist implizit angenommen, dass damit die Entgegennahme der Information durch das System verbunden ist. Ebenso wird meist mit der Ausgabe von Information die Wahrnehmung derselben durch den Benutzer angenommen. Streng genommen müssen wir jedoch zwischen dem Vorgang des Schreibens auf der einen Seite und des Lesens der Information auf der anderen Seite unterscheiden. Dies ist zumindest dann notwendig, wenn wir ausdrücken wollen, dass die Abnahme der Information durch den Benutzer oder das System zu einem anderen Zeitpunkt erfolgt als das Zuführen der Information, also das Schreiben an der Schnittstelle. Wir führen deshalb zwei weitere Typen von Dialogereignis ein:

Lesen durch System
Übergang einer Informationseinheit von der Schnittstelle zu einer anderen Stelle im System. In bestimmten Fällen ist es sinnvoll, das Ereignis der Abnahme der Information von der Dialogschnittstelle zu beschreiben bzw. festzuhalten. Die Abnahme der Information kann ab dem Zeitpunkt der Verfügbarkeit in einem bestimmten Zeitraum an der Schnittstelle erfolgen. Mit der Abnahme von einer Stelle ist eine unmittelbare Wirkung an einer anderen Stelle verbunden, die wir ohne Beschränkung der Allgemeinheit als ein Schreiben von Information an einer anderen Stelle bezeichnen wollen. Dabei ist die andere Stelle nicht unbedingt eine Dialogstelle.

Wahrnehmung durch Benutzer
Zumindest aus Symmetriegründen führen wir diesen Typ ebenfalls ein, wenn er auch nicht Gegenstand unserer Modellierung sein wird. Die Wahrnehmung der Information an der Dialogschnittstelle ist das Pendant der Abnahme von Information durch die Applikationslogik-Komponente.

Der vorletzte Ereignistyp (Lesen durch System) wird besonders dann sinnvoll, wenn wir das System in eine Dialogkomponente und eine Restkomponente (Systemlogik oder Applikationslogik) unterteilen und wir den Übergang der Information vom Benutzer zur Dialogkomponente einerseits und von der Dialogkomponente zur Logikkomponente unterscheiden und beschreiben wollen.
Eine ähnliche Überlegung gilt schon aus Symmetriegründen für den vierten Ereignistyp, da genau genommen mit der Ausgabe von Information durch das System an der Dialogschnittstelle noch keine Wahrnehmung dieser Information durch den Benutzer gewährleistet ist. Der vierte Ereignistyp wird aber für unsere Modellierung kaum eine Rolle spielen, da die Wahrnehmung der Information durch den Benutzer nicht Gegenstand des (EDV-) Systemmodells sein kann und soll.

Der Vorgang des Lesens ist im Allgemeinen als ein aktiver Vorgang verstanden. Wir wollen in unserem Modell jedoch zwischen einem aktiven und passiven, bzw. einem asynchronen und synchronen Lesen unterscheiden. Im Falle des asynchronen Lesens steuert die Systemkomponente den Lesevorgang selbst an, so dass der Zeitpunkt des Lesens von der Dialogschnittstelle in der Regel nicht unmittelbar nach dem Eingabeereignis an der Schnittstelle erfolgt. Im Falle des synchronen Lesens handelt es sich um einen Vorgang, in dem das Eingabeereignis unmittelbar das Leseereignis bewirkt (siehe auch unten Ereignisfolgen).

Ereignisse können strukturiert sein und deren Struktur kann über Relationen (siehe unten) beschrieben werden.

3.0.1 Dialogstelle

Dialogstellen sind Elemente unseres Modells, denen jeweils eine Menge von Dialogereignissen zugeordnet werden kann. Diese Zuordnung wird dadurch definiert, dass umgekehrt jedem Ereignis eine Stelle zugeordnet wird.
Wir sagen, ein (Dialog-)Ereignis findet an einer (Dialog-)Stelle statt. Solange nichts darüber ausgesagt ist, wie diese Stelle konkretisiert wird, handelt es sich um einen abstrakten, virtuellen Begriff. Die abstrakte Stelle dient allerdings als Platzhalter für eine konkretere Stelle. Die abstrakten Stellen finden ihre Abbildung in konkreteren Stellen auf dem Wege der

Implementierung bzw. Realisierung der Dialogoberfläche. Sie können in Zwischenstufen der Realisierung zunächst noch relativ „virtuell“ sein. wie z.B. die Menge der Widget-Objekte innerhalb eines Programms oder aber konkret physikalisch erfassbar, wie die Pixel am Bildschirm, die Tasten einer Tastatur oder Frequenzbereiche eines Radiosenders.
Eine Stelle repräsentiert letztendlich (wird abgebildet auf) eine Ressource, an der das Ereignis erfolgt bzw. an der ein Ereignis erfolgen kann. Stellen können für bestimmte Ereignisse über einen bestimmten Zeitraum verfügbar gemacht (reserviert, geöffnet, sensitiv, aktiviert) werden. Ebenso können sie wieder unbrauchbar (gesperrt, geschlossen, insensitiv, deaktiviert) werden.
Mit einem Ereignis an einer Stelle ist die Darstellung einer Informationseinheit an eben dieser Stelle verbunden (siehe unten Dialogwert). An der Stelle wird bei Eintreten des Ereignisses die Information übergeben bzw. abgeholt.
Der landläufige Sprachgebrauch verbindet mit dem Begriff Stelle auch die Einordnung eines Objektes oder eines Ereignisses im räumlichen und/oder zeitlichen Sinne (wo und wann) oder die Einordnung innerhalb einer diskreten Menge von Objekten oder Ereignissen (im Sinne von: an welcher Stelle oder Position befindet sich ein Objekt oder geschieht ein Ereignis?). Dieser Aspekt der Einordnung spielt auch in unserem Modell eine wesentliche Rolle, da dort die statische und insbesondere die dynamische Struktur der Dialogereignisse beschrieben werden soll. Wir wollen beschreiben, wo und wann, also an welcher Stelle ein Dialogereignis geschieht bzw. geschehen kann (siehe auch unten Relationen des Dialogmodells).
Wie Ereignisse können Dialogstellen strukturiert sein und deren Struktur kann über Relationen beschrieben werden. Eine erste nahe liegende Strukturierung ist wiederum eine Hierarchie von Stellen und Teilstellen (siehe unten Relationen).

Stelle als Abstraktion von Zeit und Ort, Abbildung auf Zeitstellen oder Ortsstellen
Die Abstraktion durch den Begriff Stelle im Modell von einer bestimmten und in der letzten Stufe der Realisierung physikalischen Ressource, die im realen Ablauf des Dialogs zum Informationsaustausch verwendet wird, macht uns während des Modellierungsvorganges zunächst davon frei, uns frühzeitig auf eine bestimmte Ressource festzulegen. Insbesondere haben wir z.B. die Freiheit, zwischen einer Abbildung der Stelle auf die Dimension Zeit oder auf die Dimension Raum zu wählen: Fassen wir beispielsweise den Bildschirm als eine räumliche Ressource auf, können wir diese Ressource zur gleichzeitigen Darstellung mehrerer Dialogereignisse durch Aufteilung der Bildschirmfläche in mehrere Felder verwenden. Wir ordnen dann im realen Ablauf den Dialogereignissen unterschiedliche örtliche Stellen zu. Alternativ dazu ist es jedoch oft möglich, dieselben Dialogereignisse in nur einem Feld zeitlich hintereinander darzustellen. Damit ordnen wir den Ereignissen unterschiedliche zeitliche Stellen, genauer Zeitabschnitte, unter Verwendung einer einzigen örtlichen Stelle zu. Im einen Fall „verbrauchen“ wir mehr an Ressource Raum im anderen Fall mehr an Ressource Zeit. Im Modell sprechen wir auf höchster Abstraktionsstufe nur von einer wie auch immer gearteten Stelle, der ein Ereignis zugeordnet wird.

Triggerstellen
Werden an einer Stelle Ereignisse unmittelbar an andere Stellen weitergegeben, was wir oben bereits mit dem Vorgang des synchronen oder passiven Lesens an dieser Stelle in Verbindung gebracht haben, sprechen wir von einer Triggerstelle.

Speicherstellen
Um ein asynchrones Lesen von Eingaben an einer Dialogschnittstelle möglich zu machen, muss eine Stelle in der Lage sein, die eingegebene Information über einen ausreichend langen Zeitraum zu speichern. Wir bezeichnen derartige Stellen als Speicherstellen.

3.0.2 Dialogwert

Mit jedem Ereignis ist eine Informationseinheit verbunden, die entweder an die Dialogstelle abgegeben (Eingabe-/Ausgabeereignis) oder von der Stelle abgelesen wird (Lesen durch System bzw. Wahrnehmung durch Benutzer). Diese Informationseinheit können wir auch als Dialogwert des Ereignisses beschreiben. Ebenso können wir einer Dialogstelle einen Dialogwert zuordnen. Der Dialogwert der Dialogstelle wird durch den Wert eines Eingabe-Ereignisses oder Ausgabe-Ereignisses, das an der Stelle stattfindet, beeinflusst. D.h. durch die Eingabe eines Wertes an einer Stelle wird der Wert einer Dialogstelle ggf. verändert. Der Wert der Ereignisstelle kann beim Leseereignis abgegriffen werden.

Dialogwert von Triggerstellen
Bei einer Triggerstelle besteht unter Umständen der Dialogwert darin, dass lediglich die Information an andere Stellen weitergegeben wird, **dass** ein Ereignis eingetreten ist. Der Wert des synchronen Lesevorgangs besteht also im „Triggern“ von Ereignissen an anderen Stellen. Besitzt die Triggerstelle keine Speicherfunktion, ist nach dem Triggervorgang kein Ablesen eines Dialogwertes mehr möglich. (Im Ereignisstellenmodell Kap.5 ist das Triggern eines Ereignisses im Wesentlichen durch die Öffne-Relation ausgedrückt.)

3.0.3 Relationen des Dialogs

Zwischen Ereignissen, Stellen und Werten des Dialogs bestehen eine Reihe von Beziehungen, die im Modell festgehalten werden sollen. Es gibt dabei Beziehungen, die rein statischen Charakter haben und Beziehungen, die im Gegensatz zu diesen das dynamische Verhalten des Dialogs beschreiben bzw. beeinflussen. Wir nehmen in unser Grundmodell zwei Typen von Relationen auf, nämlich zum einen Relationen, die eine Bestandteilshierarchie ausdrücken, die wir jeweils zwischen Dialogereignissen, zwischen Dialogstellen und Dialogwerten definieren können. Der zweite Typ ist der einer Folgerelation, deren Ausprägungen jeweils Folgen ebenfalls zwischen Ereignissen, Stellen und letztlich auch Werten definieren. Besitzt einerseits die Bestandteilshierarchie einen eher statischen Charakter, dient andererseits die Folgerelation zur Beschreibung der Dynamik.

3.0.3.0 Hierarchie

Hierarchie der Ereignisse
Ereignisse können aus Ereignissen (so genannten Teilereignissen) bestehen und werden dann zusammengesetzte Ereignisse genannt. Diese Beziehung zwischen zusammengesetztem Ereignis und Teilereignis(sen) ergibt die übliche Bestandteilshierarchie. Der Dialog, der im Modell beschrieben werden soll, kann als zusammengesetztes Ereignis betrachtet werden, das alle anderen im Modell beschrieben Ereignisse als unmittelbare oder mittelbare Teilereignisse enthält. Ereignisse, die keine Teilereignisse enthalten, werden primitive Ereignisse genannt.

Hierarchie der Stellen
Stellen können ebenso in einer Stellenhierarchie geordnet werden. Durch die Stellenhierarchie können Hierarchien von Ereignismengen und Ereignissen induziert werden:
Alle an einer primitiven Stelle stattfindenden Ereignisse definieren eine der Stelle zugeordnete Ereignismenge. Alle an den Teilstellen zugeordneten Ereignismengen einer zusammengesetzten Stelle bilden eine der Stelle zugeordnete zusammengesetzte Ereignismenge.
Wie gerade oben erwähnt, können Ereignisse als Teilereignisse zu einem zusammengesetzten Ereignis zusammengefasst werden. So kann es Sinn machen, alle in einem Dialog an einer Stelle stattfindenden Ereignisse wieder als ein (dann zusammengesetztes) Ereignis zu

betrachten. Ist diese Stelle Teilstelle einer anderen Stelle bilden dann wiederum die zusammengesetzten Ereignisse Teilereignisse eines zusammengesetzten Ereignisses der übergeordneten Stelle. In diesem Sinne wird über die Stellen und Teilstellen eine Hierarchie zusammengesetzter Ereignisse definiert.
Beispiel: Die Eingabe eines Wortes in einem Formularfeld ergibt sich als zusammengesetztes Ereignis aus der Eingabe einer bestimmten Folge einzelner Buchstaben, deren einzelne Eingaben wir als Teilereignisse im Formularfeld sehen können. Das Formularfeld sei dabei eine primitive Stelle, die wir in unserem Modell nicht mehr weiter in Teilstellen aufteilen wollen. Die Eingabe im Formular, die sich aus der Eingabe der Worte in die einzelnen Formularfelder ergibt, stellt nun wiederum ein zusammengesetztes Ereignis dar.

3.0.3.1 Dialogfolge

Ereignisse können andere Ereignisse auslösen oder Ereignisse können andere Ereignisse ermöglichen. Ereignisse können also entweder potentiell oder aber unbedingt in Folge eines anderen Ereignisses oder in Folge anderer Ereignisse geschehen. Die Folgebeziehung zweier Ereignisse kann kausalen Ursprung haben, also in der Logik der Applikation begründet sein, etwa im Sinne einer notwendigen Reaktion auf eine vorangegangene Aktion. Sie kann auch aus dialogtechnischen Gründen heraus bestimmt sein, wenn wir etwa an die sukzessive seitenweise Ausgabe einer Liste von Werten denken, die sich nicht logisch gegenseitig bedingen. Wir haben dann eine logisch willkürliche Folge der Ausgabeereignisse, bedingt etwa durch eingeschränkte Verfügbarkeit von Ressourcen, wie z.B. der Bildschirmfläche (siehe dazu auch Überlegungen in Abschnitt 4.2.0.1).

3.1 Formale Darstellung des Grundmodells

Das oben informell dargestellte Grundverständnis des Mensch-Maschine-Dialogs soll im folgenden Abschnitt in einer etwas formaleren Darstellung zusammengefasst werden. Es sei hier darauf hingewiesen, dass das Grundmodell nur als ein Ausgangspunkt für die Erarbeitung eines für die Systementwicklung nutzbaren Modells dient.

Ein System primitiver Dialoge D ist ein Tupel (E, T, W, S, F, τ, σ, φ, ω)
mit
E: Menge von Ereignissen
T: Menge von Ereignistypen T = { I_B,O_S,L_S,L_B}
W: Menge von Werten
S: Menge von Dialogstellen (auch Ereignisstellen genannt)
F: Menge von Ereignisfolgen (auch Dialoge genannt) f : e_0, e_1, …, e_n
mit $e_i \in E$ für i = 0, 1, … n, und $n \in N$,
f ist also eine Funktion f | { 0, .. , n } $\rightarrow$ E
τ: Zuordnungsfunktion von Ereignistypen τ: E $\rightarrow$ T
σ : Funktion zur Zuordnung von Stellen zu Ereignissen: σ | E $\rightarrow$ S
φ: Funktion zur Zuordnung eines Wertes zu Ereignis: φ | E $\rightarrow$ W
ω: partielle Funktion zur Zuordnung eines Wertes zu einer Stelle nach einer Reihe von Ereignissen

$$\omega \mid (S, EHIST) \rightarrow W,$$

mit EHIST = { ehist, ehist $\in$ E* und $\exists$ f $\in$ F, i $\in$ N: (f(0), f(1), .. f(i)) = ehist }

Zum Grundmodell, insbesondere zur Funktion ω, noch einige Erläuterungen:

Einem Ereignis ist in unserem Grundmodell eine Stelle zugeordnet (Funktion σ). Wir sagen, das Ereignis e findet an der Stelle s statt, wenn gilt $\sigma(e) = s$. Wir ordnen ebenfalls jedem Ereignis einen bestimmten Wert zu (Funktion φ).

Wir wollen nun zusätzlich annehmen, dass prinzipiell auch an einer Stelle zu bestimmten Zeitpunkten Werte abgelesen werden können. Dass dies Sinn macht, ist evident, wenn wir uns vorstellen, dass nach dem Einwirken (Stattfinden) eines Ereignisses an einer Stelle an dieser auch ein bestimmter Wert vorliegt. Dieser Wert ändert sich im Allgemeinen in Abhängigkeit der an der Stelle stattfindenden Ereignisse. Der gerade gültige Wert der Stelle kann mit dem Wert des letzten Ereignisses an der Stelle übereinstimmen oder kann zumindest durch den Wert des letzten Ereignisses beeinflusst worden sein.

Beispiel: Jede Betätigung eines Druckknopfes an einem Zähler erhöhe den Wert des Zählers um 1. Betrachten wir den Zähler mit aktuellem Wert $n \in N$ samt Druckknopf als eine Eingabestelle, können wir den logischen Wert des Ereignisses „Knopf drücken“ mit 1 bewerten und den Wert des Zählers nach dem Eingabeereignis mit n+1.

Wir gehen auch davon aus, dass Speicherstellen in der Lage sind, ihren einmal zugeteilten Wert über einen gewissen Zeitraum beizubehalten, so dass dieser Wert auch nach Eintreten weiterer Ereignisse an anderen Stellen an besagter Stelle abgelesen werden kann. Außerdem nehmen wir an, dass bestimmte Stellen bereits zum Startzeitpunkt eines Dialogs einen ablesbaren Wert besitzen.

Dies führt zur Annahme der Existenz einer partiellen Funktion ω zur Zuordnung eines Wertes zu einer Ereignisstelle s nach Eintreten einer Teilfolge von Ereignissen $e_0, e_1, \ldots, e_i$. Sinnvollerweise können wir annehmen, dass diese Funktion zumindest dann nicht definiert ist, wenn eine Ereignisteilfolge in keiner der Ereignisfolgen $f \in F$ als Anfang vorkommt. Dass die Funktion als partielle eingeführt wird, geschieht aber deshalb, weil wir davon ausgehen, dass manche Stellen nicht über den gesamten Zeitraum des Dialogs verfügbar sind.
ω erlaubt uns also nur teilweise, nach einem bestimmten Ereignis in einer Ereignisfolge f an einer Ereignisstelle s einen Wert „abzulesen“. Wir fordern diese Funktion, ohne zunächst eine Aussage darüber zu treffen, wie diese Funktion in einer Implementierung realisiert wird. Ziel unseres Grundmodells ist es auch generell nicht, dieses in eine konkrete Implementierung umzusetzen, sondern eine Basis für die Entwicklung anderer konkret einsetzbarer Modelle zu schaffen.

Bemerkung zum Ereignisbegriff:
Der Begriff Ereignis kann auf unterschiedlichen Ebenen der Konkretisierung Verwendung finden. Manchmal ist es nötig, zwischen einem konkreten Ereignis und einem abstrakten Ereignis zu unterscheiden. Einem konkreten Ereignis kann z.B. ein genauer Zeitpunkt zugeordnet werden. Bei einem etwas abstrakteren Verständnis können wir beispielsweise vom konkreten Zeitpunkt absehen. Es handelt sich dann bei einem so verstandenen Ereignis eher um eine Klasse konkreter Ereignisse bzw. umgekehrt bei den konkreten Ereignissen eher um Ereignisvorkommen des abstrakten Ereignisses.
Entsprechend dieser unterschiedlichen Abstraktionsebene und dem unterschiedlichen Verständnis von Ereignis ist in einem Fall dann die Funktion $f \mid \{ 0, .. , n \} \rightarrow E$ eine injektive Funktion, deren inverse f^{-1} zumindest auf $f(\{ 0, .. , n \})$ definiert ist und jedem Ereignis genau ein $i \in \{ 0, .. , n \}$ zuordnet.
(Die Zuordnung eines Index in der Folge der Ereignisse ist allerdings bereits eine Abstraktion von einem konkreten Zeitstempel.)
Auch die Funktion $\sigma \mid E \rightarrow S$ ist entsprechend zu interpretieren, je nachdem, ob wir die Stellen als Stellen betrachten, die den Zeitpunkt des Ereignisses mit einbeziehen. Ist dies der Fall, gehen wir notwendigerweise in unserem Modell davon aus, dass mit Ereignis der konkrete Ereignisbegriff verstanden ist.
Wenn wir in einer Bottom-up-Vorgehensweise im WYSIWYG-Ansatz (siehe Kap.4) von Ereignissen sprechen, meinen wir zunächst den konkreten Ereignisbegriff: Im extensiven WYSIWYG-Ansatz wird zunächst jedes konkrete Ereignis aufgezählt und jedes Ereignis entspricht auch genau einer Ereignisstelle im Modell. Erst durch

die Reduktion der Stellen fassen wir die konkreten Ereignisse zusammen und abstrahieren damit auch in der Darstellung des Modells die konkreten Ereignisse zu abstrakteren Ereignissen.

Zusammengesetzter Dialog

Ein System zusammengesetzter Dialoge HD liegt vor, wenn obiges Tupel modifiziert und um eine zweistellige Relation $H \subset E \otimes E$ erweitert wird, die hierarchische Zusammensetzungen der Ereignisse beschreibt.

$$HD = (E, T, W, S, F, \tau, \sigma, \varphi, \omega, H)$$

Falls $(e_1,e_2) \in H$ sagen wir Ereignis e_2 ist Teilereignis von Ereignis e_1.
H* sei die transitive und reflexive Hülle von H.

Eine hierarchische Relation von Ereignissen in einer Ereignisfolge wird induziert, wenn wir beispielsweise eine Hierarchie von Stellen $X \subset (S \otimes S)$ einführen.
Falls $(s_1, s_2) \in X$ sagen wir s_2 ist Teilstelle von s_1.
X* sei wiederum die transitive und reflexive Hülle von X.

Eine formale Darstellung der durch die Stellenhierarchie induzierten Ereignishierarchien findet sich im Anhang C. Für die Ausführungen in den folgenden Kapiteln ist nur von Bedeutung, dass über eine Stellenhierarchie die an den Teilstellen einer zusammengesetzten Stelle stattfindenden Ereignisse zu einem Ereignis an der zusammengesetzten Stelle zusammengefasst werden können.

Wenn wir Ereignishierarchien mit zusammengesetzten Ereignissen einführen, ist es sinnvoll, die Menge T der Ereignistypen um die Kombinationen aus den vier Grundtypen zu erweitern. Wir erhalten dann eine Menge $T' \subset T^*$:

$$T' = \{ I_B, O_S, L_S, L_B, I_BO_S, I_BL_S, I_BL_B, O_SL_S, O_SL_B, L_SL_B, I_BO_SL_S, I_BO_SL_B, I_BL_SL_B, O_S,L_S,L_B, I_BO_SL_BL_S\}$$

Entsprechend haben wir dann auch eine Funktion

$$\tau' : E \rightarrow T'$$

τ' ordnet zusammengesetzten Ereignissen die entsprechenden zusammengesetzten Typen zu.

4 Bottom-up-Modellierung und Abstraktion

Im nun folgenden Kapitel wird auf Basis des oben vorgestellten Verständnisses von Dialog, das wir in einem sogenannten Grundmodell des Dialogs zusammengefasst haben, ein Bottom-up-Ansatz der Dialogmodellierung diskutiert. Der Bottom-up-Ansatz geht von der terminalen, konkreten Erscheinungsform des Dialogs zur Laufzeit aus und versucht so weit wie möglich diese Erscheinungsform im Modell direkt widerzuspiegeln (im Folgenden WYSIWYG-Darstellung, auch extensive Darstellung genannt).
Da wir bei einer konsequenten WYSIWYG-Darstellung erwartungsgemäß sehr schnell an Grenzen unter anderem bezüglich der Vollständigkeit stoßen, werden einige wenige aber wesentliche Schritte von der konkreten zu einer abstrakteren, reduzierten Darstellung im Modell vorgestellt. Diese Abstraktions- oder Reduktionsschritte, die auch zur Einführung weiterer Sprachelemente im Modell führen, ermöglichen uns eine kompaktere (intensivere) Darstellung des Dialogs, erlauben aber immer noch eine weitgehend intuitive Form. Diese Sprachelemente gehen letztendlich im Kapitel 5 und folgende in die Darstellungsmöglichkeiten des Ereignisstellendiagrammes ein.

4.0 WYSIWYG-Ansatz

Der WYSIWYG-Ansatz im Modell erlaubt uns eine weitgehend realistische und damit auch anschauliche Darstellung des Dialogs. Er kann deshalb als Ausgangspunkt für eine benutzerzentrierte Modellierung dienen. Ein Ziel ist dann, aus der WYSIWYG-Darstellung sukzessive eine Darstellung zu gewinnen, die es auch erlaubt, den nötigen Programmcode für die Realisierung der Dialogkomponente zu erzeugen. Der WYSIWYG-Ansatz liefert zunächst eine auf ein bestimmtes Layout fixierte und in vielen Fällen noch unvollständige Darstellung des Systems. Von der konkreten Darstellung ausgehend kann in einigen Modellierungsschritten jedoch ein vom Layout abstrahierendes, logisches und vollständiges Modell gewonnen werden.

Als wichtige Charakteristika einer konsequenten WYSIWYG-Darstellung wollen wir zwei Merkmale festhalten:

a) WYSIWYG-Darstellung der Dynamik, d.h. die möglichst zeitgetreue Darstellung des tatsächlichen, realen Dialogablaufs, eine möglichst zeitgetreue Folge der Dialogereignisse. Da es in der Regel keinen eindeutig fixierten einzigen Ablauf des zu realisierenden Dialogs gibt, bedeutet dies die explizite Darstellung aller möglichen alternativen Dialogfolgen, sprich das Aufzeigen aller in unserem obigen Grundmodell definierten Folgen.
b) Darstellung der Ereignisse an den realen Dialogstellen. WYSIWYG bedeutet auch eine Darstellung des Dialogs so, wie er sich tatsächlich zur Laufzeit visuell, akustisch, taktil ... darstellt. Nach unserem Grundmodell bedeutet dies eine konkrete Darstellung aller Dialogstellen und der Ereignisse an ihnen, d.h. die Abbildung des Dialogs mit realen auch zur Laufzeit verwendeten Ressourcen.

Ansatz 1. Explizite Darstellung der Dialogdynamik in (echt-)zeitlicher Abwicklung
Im Extremfall der Bottom-up-Vorgehensweise, d.h. in einer äußerst konsequenten Lösung der Aufgabe, den Dialog bei der Modellierung bereits „laufzeitähnlich“ darzustellen, muss nicht nur das statische Aussehen der Dialogstellen, sondern auch das dynamische Verhalten der Schnittstelle möglichst explizit und konkret dargestellt werden.

Der Extremfall bedeutet: Wir nehmen unser Endprodukt, nämlich die Darstellung des Dialogs zur Laufzeit als unser Modell. Hier beißt sich prinzipiell die Katze natürlich in den Schwanz: In den meisten Fällen ist der Dialog, so wie er sich zur Laufzeit darstellen soll, nicht fertig vorhanden. Wir wollen ihn ja gerade erst definieren.
Aber wir können uns diesem Extremfall der exakten Laufzeitdarstellung bereits zum Definitionszeitpunkt weitgehend annähern. D.h. die Definition des dynamischen Verhaltens geschieht durch ein „Vorspielen", ein Durchlaufen des dynamischen Verhaltens in möglichst echter Zeit. Der Modellierer des Systems erzeugt also bei der Definition nicht zunächst ein statisches Programm. Sondern er spielt Schritt für Schritt, Ereignis für Ereignis an simulierten realen Stellen den (echt-)zeitlichen Ablauf des gewünschten Systems durch. (Ähnliche Ansätze sind in der Methode Programming by Example [CYPHE91] oder bei der Makroaufzeichnung in Excel [MICRO01] wieder zu erkennen.)
Dieser zunächst vielleicht etwas extrem anmutende Ansatz ist für viele Fälle durchaus realisierbar: Stellen wir uns einen Modelleditor vor, der neben dem Zeichnen statischer Widgets auch erlaubt, Kommandos abzusetzen, die Ereignisse auf diesen Widgets bewirken, wie z.B. Öffnen, Setzen oder Löschen eines Wertes, etc., Ereignisse, die dann sofort am Bildschirm umgesetzt und aber auch protokolliert werden.
Es handelt sich dann bei diesem Modelleditor um im Wesentlichen nichts anderes als einen Kommandointerpreter, der auf Basis mehr oder weniger primitiver konkreter Dialogereignisse auf konkreten Dialogstellen operiert. Im Idealfall können die Kommandos mit Hilfe der Technik der direkten Manipulation an der Dialogstelle durchgeführt werden. Z.B. kann das Setzen eines Wertes in einem Textwidget durch direktes „Hineinschreiben" geschehen, ein sich öffnendes Widget wird einfach direkt am Bildschirm an die gewünschte Stelle „hingemalt", etc. Der Editor speichert in einem Aufzeichnungsmodus bei Durchführung der Kommandos nicht nur die statische Information der Widgets, sondern auch den Ablauf der einzelnen Dialogschritte.

Vorteile:
- Der Modellierer kann bereits unmittelbar in der Definitionsphase Schritt für Schritt exakt das Aussehen und Verhalten der Oberfläche verifizieren.
- Er arbeitet mit primitiven, überschaubaren Operationen, die den Ablauf Schritt für Schritt vorantreiben, also ohne komplexe Logik.
Nachteile: siehe unten

Ansatz 2: Explizite, statisch räumliche Darstellung der Dialogdynamik bzw. generell der Dialogereignisse (räumlich extensiv (explizit, extensional)):
Anstelle der zeitgetreuen Darstellung können wir im Modell eine statische Darstellung wählen. D.h. wir bilden das Zeitverhalten, die Dynamik statisch ab. Dies geschieht beispielsweise dadurch, dass wir eine Zeichenfläche nutzen und dort unsere Dialog-Ereignisse verteilt an aufeinander folgenden Stellen aufzeichnen. Wir zeichnen eine Bilderfolge mit den konkreten Ereignissen auf die Fläche. Auf diese Weise gewinnen wir eine nach wie vor, mit Ausnahme des Echtzeitverhaltens, konkrete Darstellung.
Der Vorteil der statischen Darstellung ist, dass der Betrachter „auf einen Blick" eine Folge oder Teilfolge des Dialogs überschauen kann. Ihm wird keine zeitliche Reihenfolge der Beobachtung aufgezwungen. Er kann die dargestellte Folge nach eigenem Gutdünken vorwärts und rückwärts verfolgen. Die zeitliche Abfolge ist für ihn aber dennoch erkennbar. Zum anderen sind wir in unserer Modellierung nicht an ein Medium gebunden, das Dynamik unterstützt.

4.0.0 Ein einfaches Beispiel der WYSIWYG-Modellierung

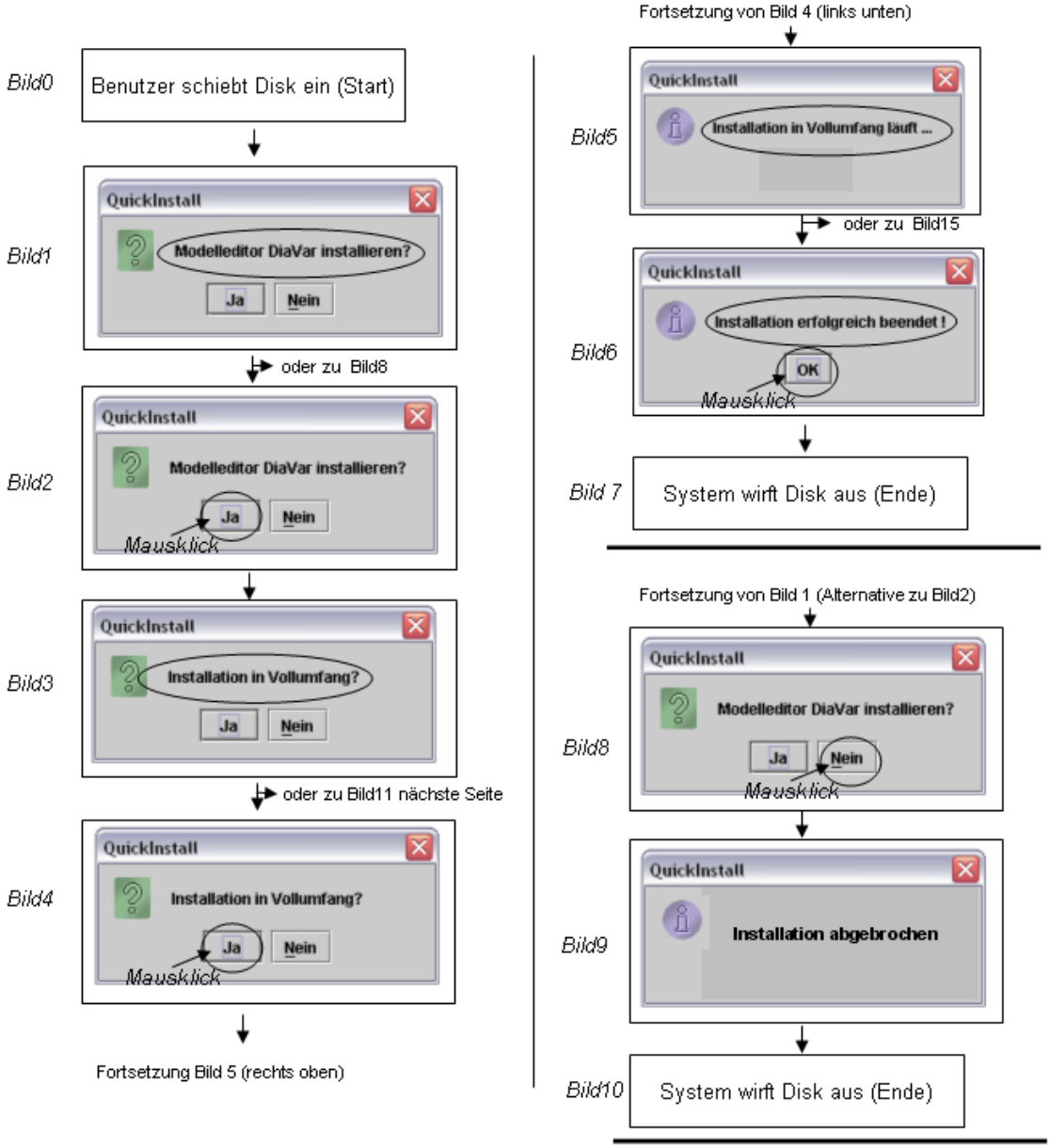


Abbildung 4-1 Beispiel für WYSIWYG-Modellierung (Teil 1, Teil 2 in Abbildung 4-2)
Jedes Ausgabe- und Eingabeereignis wird Schritt für Schritt in einem eigenen Bild festgehalten.

Abbildung 4-1 und Abbildung 4-2 zeigen einen einfachen Dialog zur Installation eines Softwarepaketes in einer Folge von Bildern. Jedes Bild in den Abbildungen zeigt eine „Momentaufnahme“ des Dialogs, die ein Eingabe- oder Ausgabeereignis repräsentieren soll. Im Standardfall zeigt die Bildreihenfolge auch die Reihenfolge der Ereignisse. Im Dialog ergeben sich jedoch alternative Verzweigungen. Um diese darstellen zu können, sind die Bilder durchnumeriert und gegebenenfalls mit entsprechenden Fortsetzungshinweisen versehen.
Wir erleben bei Betrachtung des statischen Bildes natürlich nicht konkret das echtzeitliche dynamische Erscheinen des Ereignisses. Ein Ereignis wird letztlich nur durch sein Ergebnis im Bild erkennbar. So führt z.B. das Ereignis der Ausgabe eines Widgets dazu, dass wir es am Bild wahrnehmen können. Die Eingabe des Benutzers stellen wir ebenfalls durch ein Ergebnis der Eingabe fest, also z.B. anhand eines Wertes, den wir in einem Eingabewidget feststellen können.

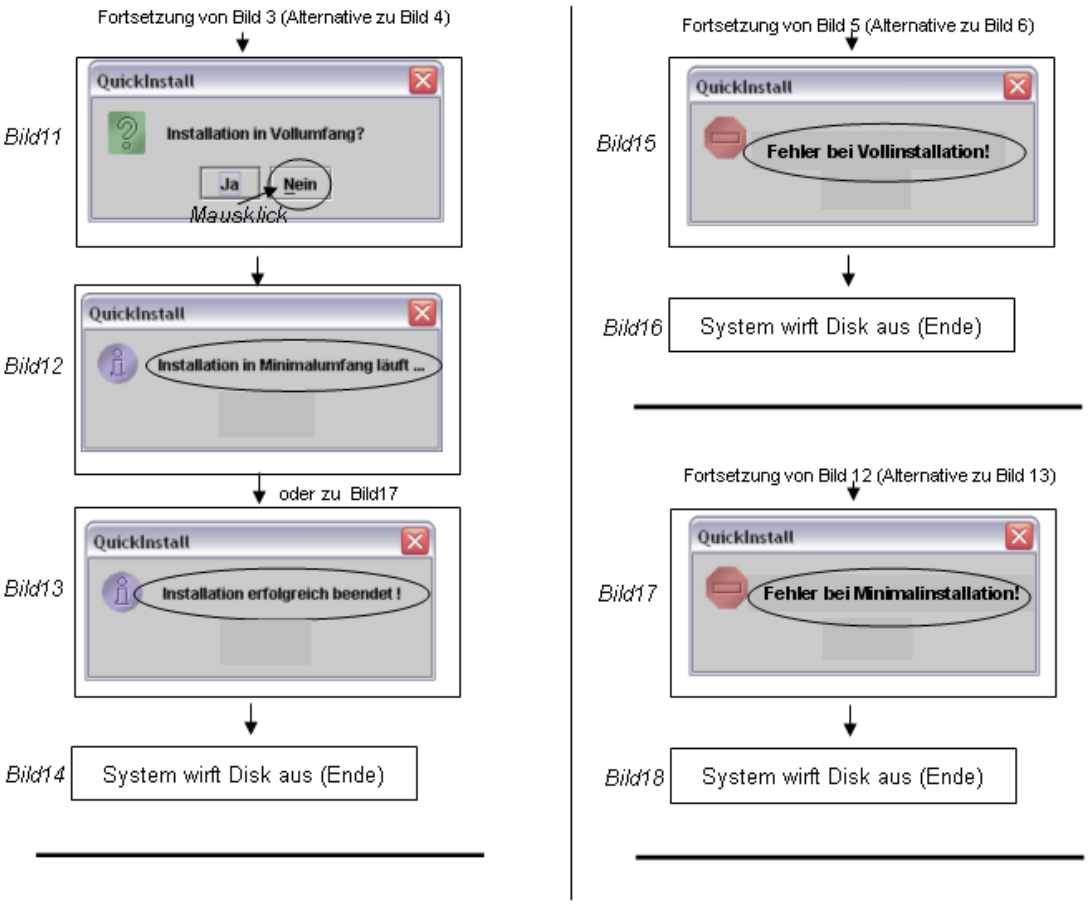


Abbildung 4-2 WYSIWYG-Beispiel Teil2 (Fortsetzung von Abbildung 4-1)

Da in unserem Beispiel in einem Bild nicht nur das Ergebnis des aktuell letzten Ereignisses zu sehen ist, sondern auch zum Teil die Ergebnisse bereits vorangegangener Ereignisse, benötigen wir ein Merkmal zur Unterscheidung zwischen den alten Ereignissen und dem neuen. Wir erkennen das neue Ereignis einmal durch einen Vergleich mit dem Vorgängerbild oder dadurch, dass wir es besonders grafisch hervorheben (siehe z.B. ovale Umrahmung in den Abbildungen).
Für den menschlichen Betrachter der Bilderfolge wird der Ablauf des Dialogs in dieser Form der Darstellung intuitiv klar. Dabei setzen wir natürlich dennoch immer einen gewissen Grad an Vorkenntnis und Abstraktionsfähigkeit voraus. Um den Dialog letztlich als Benutzer durchführen zu können, muss dieser etwa bei unserem Beispiel wissen, was unter Mausklick zu verstehen ist. Wir hätten in unsere Darstellung allerdings auch die Eingabe Mausklick bildlich mit aufnehmen und noch näher spezifizieren können, dass standardmäßig die linke Maustaste gemeint ist.
Wir machen in unserem Modell also implizite Annahmen über einen bereits bekannten Kontext oder aber wir lassen bestimmte Beschreibungsmerkmale im Sinne einer späteren Konkretisierung offen.
Für die Zielsetzung der benutzerzentrierten Modellierung des Dialogs erscheint die gewählte Art der Darstellung des Dialogs wegen seiner intuitiven und klaren Form, zumindest was dieses einfache Beispiel betrifft, sehr gut geeignet.
Die Umsetzung einer derartigen Modelldarstellung in ein ablauffähiges Programm benötigt eine Formalisierung der oben gezeigten Darstellung, die von einer Maschine entsprechend interpretiert werden kann.
Als Ansatz dient das oben bereits vorgestellte Grundmodell. Nach Grundmodell entspricht jedem Ereignis eine Stelle, hier ein Bild. Umgekehrt ist, wie oben erwähnt, in jedem Bild ein

Ereignis enthalten. Ausgehend vom ersten Bild erhalten wir über die Folgebilder unter Berücksichtigung der alternativen Verzweigungen eine Menge von Dialogereignisfolgen. In der Modelldarstellung dienen die einzelnen Bilder als Stellen (Modellstellen).

Nach unserem Grundmodell haben wir ein Tupel (E, T, W, S, F, τ, σ, φ, ω)

Als Ereignismenge E haben wir:

E = { e0, … , e18 },
T = { I_B, O_S } ,
W = { "schiebt Diskette ein", "Programm installieren?", „Ja", „Nein",
„Installation in vollem Umfang?", „Installation in Vollumfang",
„Installation in Minimalumfang", „Installation erfolgreich beendet",
„Fehler bei Vollinstallation", „Fehler bei Minimalinstallation"
„Installation abgebrochen", „gibt Diskette aus" }
S = { Bild0, .. , Bild18 }
F = { f1: e0, e1, e2, e3, e4, e5, e6, e7
f2: e0, e1, e8, e9, e10
f3: e0, e1, e2, e3, e11, e12, e13, e14
f4: e0, e1, e2, e3, e4, e5, e15, e16
f5: e0, e1, e2, e3, e11, e12, e17, e18 }

Die Funktionen τ, σ, φ für die einzelnen Ereignisse in E lauten:

Für	e0:	I_B,	Bild0,	"schiebt Diskette ein"
	e1:	Os,	Bild1,	„Programm installieren?"
	e2:	I_B,	Bild2,	„Ja"
	e3:	O_S,	Bild3,	„Installation in vollem Umfang?"
	…			
	e18:	O_S,	Bild18,	"gibt Diskette aus"

Die Funktion ω liefert den Wert einer Stelle identisch zum Wert des Ereignisses an der Stelle:

Also ω(Bild0) = φ(e0) = „schiebt Diskette ein"
ω(Bild1) = φ(e1) = „Programm installieren?"
…
ω(Bild18) = φ(e18) = „gibt Diskette aus".

Wir haben hier eine bijektive Abbildung zwischen den Stellen im Modell und den Ereignissen.
Jedes Bild zeigt auch konkret, wie ein Ereignis zur Laufzeit mit Hilfe eines Widgets realisiert wird. Wir haben hier also auch eine Abbildung der Ereignisse bzw. der Modellstellen auf Laufzeitstellen. Hier ist zunächst noch nichts darüber ausgesagt, ob es sich bei den Laufzeitstellen an den Modellstellen um jeweils unterschiedliche Instanzen von Widgets handelt oder ob für unterschiedliche Modellstellen dieselben Instanzen verwendet werden sollen.

Mithilfe der formalen Darstellung können wir einen Algorithmus definieren, der den oben beschriebenen Dialog zur Laufzeit realisiert. Der Algorithmus sei im Folgenden kurz skizziert (Kommentar in kursiver Schrift):

1. Setze den aktuellen Folgenindex i auf 1. Markiere alle Folgen als aktiv.
2. Falls alle Ereignisse im aktuellen Glied i aller aktiven Folgen Ausgabeereignisse:

Falls mehrere alternative Ausgabeereignisse in aktuellen Folgegliedern:
Frage die Applikationslogik, welches der Ausgabeereignisse durchzuführen ist.
Markiere alle Folgen mit anderen Ausgabeereignissen bzw. Modellstellen als nicht aktiv.
Führe Ausgabe durch:
Schließe alle im aktuellen Bild nicht mehr vorkommenden Widgets des Vorgängerbilds (falls vorhanden).
Öffne alle Widgets im zugeordneten Bild, falls nicht schon vorher geschehen, und gib im gekennzeichneten Widget den Ausgabewert aus.

Andernfalls: Falls alle Ereignisse im aktuellen Glied aller aktiven Folgen Eingabeereignisse:
Schließe alle im Bild nicht mehr vorkommenden Widgets des Vorgängerbildes.
Öffne alle Widgets aller aktuellen Folgeglieder (falls nicht bereits geschehen) und ermögliche damit die potentiellen Eingaben aller aktuellen Folgeglieder. Warte auf Eingabe.
Sobald Eingabe erfolgt, lösche Aktivmarkierung aller aktiven Folgen mit alternativen Eingaben im aktuellen Folgeglied.
Speichere Identifikator der aktuellen Modellstelle der noch aktiven Folgen und aktuellen Eingabewert.
Kommentar: Speichern dient zum Festhalten der Historie für spätere Entscheidungen durch Applikationslogik
Andernfalls: Fehler
Kommentar: Ein Mischfall, nämlich dass in einer Situation aktive Folgen im aktuellen Folgeglied Eingaben und Ausgaben ausweisen, sei nicht erlaubt. Dies impliziert eine Einschränkung in der Darstellung: Die Folgeglieder einer „Verzweigung" des Dialogs (im Sinne einer Baumdarstellung) dürfen nur ausschließlich Eingaben oder ausschließlich Ausgaben ausweisen.
Erhöhe aktuellen Gliedindex um 1.

3. Falls Anzahl der aktuellen Folgeglieder der aktiven Folgen = 0, Programmende
   Andernfalls: Falls Anzahl der aktuellen Folgenglieder kleiner als vor dem Hochzählen:
   Kommentar: Eine der alternativen Folgen ist dann zu Ende
   Frage Applikationslogik, ob Programm zu Ende.
   Falls ja, Ende
   Andernfalls: Fahre fort mit 2
   Andernfalls:
   Fahre fort mit 2

Voraussetzung für das Funktionieren des obigen Algorithmus ist, dass in bestimmten Situationen die Applikationslogik um eine Entscheidung bezüglich des weiteren Verlaufs des Dialogs gefragt werden kann. Um dabei die Applikationslogik zu unterstützen, werden, wie oben angedeutet, Identifikatoren für die durchlaufenen Modellstellen und Eingabewerte gespeichert, auf die die Applikationslogik zugreifen kann.
Eine weitere Voraussetzung ist, dass die gewählten Widgets dazu geeignet sind, die mit den Ereignissen verbundenen Werte der Eingabe und Ausgabe in definierter Weise darzustellen.

4.0.1 Wahl von Widgets als Stellenmenge zur Laufzeit

Wie oben erwähnt, wird letztlich jedes Ereignis einer Stelle zur Laufzeit zugeordnet. Betrachten wir beispielsweise den Bildschirm als unsere Stellenmenge während des Ablaufs des Dialogprogrammes: Dann können wir die Pixel als primitive Teilstellen interpretieren, die mittels Eingabe- und Ausgabeereignissen mit Pixelwerten belegt werden.

Betrachten wir hingegen die Menge aller denkbaren Widgetinstanzen als unsere Stellenmenge, so können wir ein Widget als eine einem Dialogereignis zugeordnete Dialogstelle interpretieren. Diese Widgetinstanzen werden wiederum auf die Pixelstellen am Bildschirm abgebildet.

Eine Dialogstelle ist häufig zusammengesetzt aus Ausgabe- und Eingabestellen. Erlaubt das Widget keine Eingabe, handelt es sich um reine Ausgabeereignisse an der Stelle und damit um eine reine Ausgabestelle. Handelt es sich um ein Widget zur Eingabe, repräsentiert das Widget neben Eingabeereignissen im Allgemeinen auch Ausgabeereignisse. Dabei beinhalten die Ausgabeereignisse oft nur den Benutzer unterstützende Information, die auf die mögliche Eingabe hinweist (Prompting).

Abhängig von der verfügbaren Stellenmenge im Modell können wir die Ereignisse auf die Stellen aufteilen. Mittels einer Vorschrift werden die Ereignisse und Stellen des Modells dann auf Ereignisse und Stellen zur Laufzeit übertragen, d.h. die Laufzeitstellen mit ihren Ereignissen werden realisiert.

Wird zur Laufzeit für jedes Ereignis ein eigenes Widget verwendet, haben wir eine bijektive Abbildung der Laufzeitstellen auf die Modellstellen. Wird ein Widget zur Laufzeit für mehrere Ereignisse hintereinander verwendet, d.h. finden an ein und derselben Laufzeitstelle hintereinander mehrere Ereignisse statt, zeichnen wir dasselbe Widget an mehreren Modellstellen, da wir ja (bis jetzt jedenfalls) jedes Ereignis einer eigenen Modellstelle zuordnen. Wir haben in diesem Fall eine redundante Darstellung der Laufzeitstellen im Modell und es wird notwendig, eine Zuordnung von Modellstellen auf Identitäten von Laufzeitstellen vorzunehmen. Wir haben dann eine surjektive Abbildung der Ereignisstellen im Modell auf die Ereignisstellen zur Laufzeit.
Dazu ein Beispiel: zur Laufzeit kann generell innerhalb einer Applikation ein und dasselbe Textwidget für Fehlermeldungen verwendet werden. Jede Fehlermeldung stellt ein Ausgabeereignis dar, das auf demselben Widget passiert. Im Modell haben wir aber mit der expliziten Ereignisdarstellung das Textwidget an mehreren Modellstellen. Die Kennzeichnung, dass es sich zur Laufzeit um ein und dasselbe Widget handelt, wird z.B. über einen der Modellstelle zusätzlich zugeordneten eindeutigen Laufzeitstellennamen vorgenommen.

Darstellung der Dialogfolgen in einem Grafen

Die möglichen Dialogfolgen können auch in Form eines gerichteten Grafen dargestellt werden.
Die Bilder bzw. Modellstellen sind die Knoten des Grafen. Die Kanten repräsentieren jeweils den Übergang von einem Bild zum Folgebild, so dass ein Pfad des Grafen eine der alternativen Dialogfolgen darstellt.
Im Folgenden wollen wir aus Gründen der Platzersparnis von der Darstellung konkreter Widgets im Grafen abstrahieren. Für die nun geführte Diskussion zur Darstellung der Dialogfolgen ist die Wahl konkreter Widgets ohne Belang. Wir weisen die Knoten des Grafen meist dann in Form von beschrifteten und annotierten Ellipsen (oder Rechtecken) aus. Dabei bezeichnet die Beschriftung innerhalb der Ellipse entweder einen Kommentar, der das Ereignis an der jeweiligen Stelle umschreibt, oder den mit dem Ereignis verbundenen Ausgabewert oder Eingabewert in Textform. Die Annotation steht für einen Identifikator der Stelle (Modellstelle) und den Typ des Ereignisses an der Stelle (Eingabe (E bzw. I) oder Ausgabe (A bzw. O)).

Nach dem oben vorgestellten Ziel der WYSIWYG-Darstellung wollen wir nach Möglichkeit alle Ereignisse des Dialogs in allen möglichen Ablauffolgen explizit aufzeigen. In einer extrem extensiven Darstellung bedeutet dies, dass wir alle Ablauffolgen voneinander getrennt an getrennten Modellstellen nachweisen. In der statischen Darstellung in einem Grafen kann das so erfolgen, dass jede Dialogfolge in einem eigenen unabhängigen Teilgrafen dargestellt ist.
Wir verlieren allerdings keine nennenswerte Information, wenn die alternativen Folgen im Grafen in Baumform dargestellt werden. Erst wenn die Folgenglieder der alternativen Folgen unterschiedliche Ereignisse aufweisen, wird im Grafen über unterschiedliche Kanten verzweigt und die Ereignisse werden in unterschiedlichen Knoten des Grafen dargestellt (Abbildung 4-3). Die bisher im Dialog erfolgte Ereignisfolge wird eindeutig durch den erreichten Knoten im Baumgrafen festgehalten.
Betrachten wir den Dialog als ein in sich geschlossenes deterministisches System, wissen wir die eindeutige Fortsetzung des Dialogs, also das nächste Dialogereignis, wenn wir wissen, in welchem Knoten wir uns aktuell befinden. Wir benötigen kein zusätzliches „Gedächtnis“ bezüglich des bisher durchgeführten Dialogs, um die Fortsetzung zu steuern bzw. das nächste Dialogereignis zu bestimmen. Erst die Beeinflussung des Systems durch unbestimmbare Ereignisse von außen führt zu möglichen alternativen Verzweigungen, etwa alternative Eingaben des Benutzers oder alternative Ausgaben der Applikationslogik.

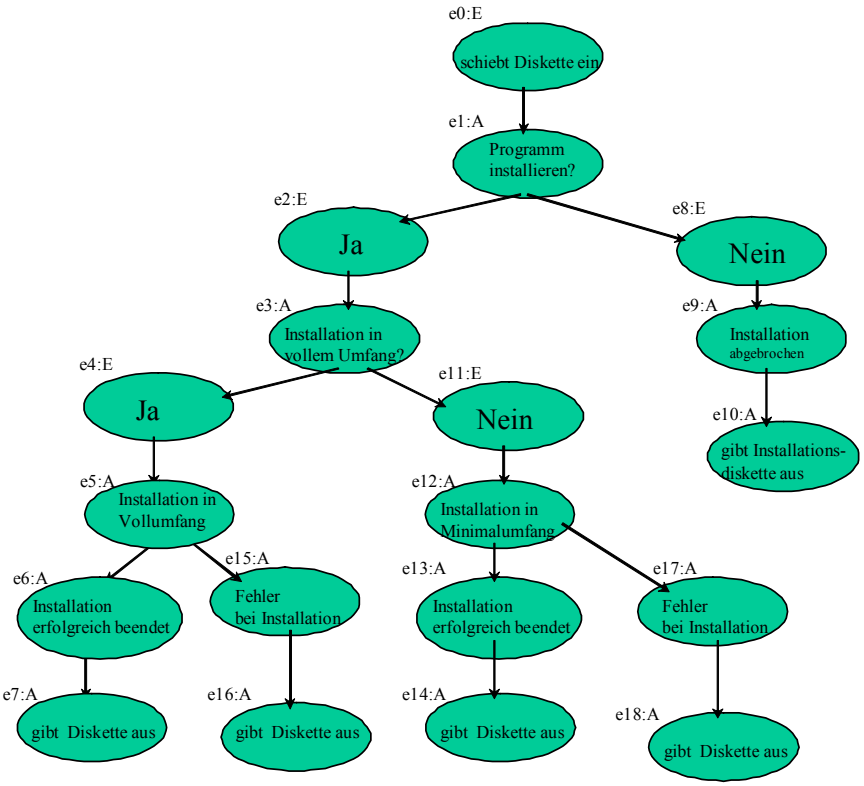


Abbildung 4-3: Beispieldialogfolgen als Pfade eines Baums in einem Grafen.

Jeder Knoten im Baum repräsentiert einen eindeutigen Pfad und damit eine eindeutige Ereignisfolge und Dialogsituation im Sinne der Historie des bisher durchgeführten Dialogs.

4.0.2 Grenzen der WYSIWYG-Modellierung

Der obige Beispieldialog ist derart einfach, dass er in all seinen alternativen Abläufen in einem Grafen relativ übersichtlich vollständig zu sehen ist. Insbesondere ist die Anzahl seiner alternativen Folgen endlich, ebenso die Anzahl Dialogschritte innerhalb einer Alternative. Der Graf stellt ein vollständiges Modell des Dialogs aus Sicht des Benutzers dar.
Wie nun bereits aus der Literatur bekannt (z.B. [JACOB86][JANSS96]…), ist die Darstellung eines Dialogsystems in Form von Zuständen und Zustandsübergängen im Allgemeinen nur schwer handhabbar. Dies gilt insbesondere dann, wenn im System sehr viele Zustände und/oder mögliche Zustandsübergänge vorkommen. Die oben beschriebene Form der extensiven Darstellung stellt eine derartig potentiell pathologische Beschreibungsform dar. Jede Stelle der extensiven Darstellung kann als Zustand, jeder Übergang von einer Stelle zur anderen als Transition interpretiert werden. Da wir die Darstellung jeder möglichen Ereignisfolge und damit aller möglichen Zustände und Zustandsübergänge vorsehen, kommen wir in bereits relativ einfachen Fällen zu einer sehr großen Anzahl von Zuständen und Transitionen.

4.1 Reduktion von Stellen und Transitionen

Um den wesentlichen Nachteil der extensiven Darstellung, die sehr hohe und häufig unendliche Anzahl von Stellen zu beseitigen, müssen wir trivialerweise die Anzahl der benötigten Stellen reduzieren. Dasselbe gilt für die Übergänge zwischen den Stellen. Die Reduktion der Stellen und Transitionen wollen wir auch als Intensivierung der Darstellung bezeichnen.

Zusammenführung identischer Teilfolgen alternativer Dialoge

Mit der Darstellung des Dialogablaufes in Baumform sind wir bereits von einer extrem extensiven Darstellung aller alternativen Folgen mit einer Aufzählung aller einzelnen Folgen mit allen einzelnen Ereignissen an jeweils eigenen Modellstellen abgewichen. Eine weitere einfache Reduktion der Knoten und Kanten bzw. der Modellstellen und ihrer Übergänge erreicht man, wenn nun nicht nur identische Teilfolgen am Anfang (wie bei der Baumdarstellung der Fall), sondern generell identische Teilfolgen alternativer Folgen zu einem einzigen Teilpfad im Grafen zusammengefasst werden. Dabei fassen wir Teilfolgen als identisch auf, wenn die einzelnen Folgenglieder in Typ und Wert des dargestellten Ereignisses übereinstimmen.
Dies geschieht nun wiederum problemlos, wenn die Zusammenfassung nicht zu einem Informationsverlust führt, der eine Fortsetzung nach der zusammengefassten Teilfolge mit den bisher gegeben Mitteln der Darstellung in unserem Grafen unmöglich macht.
Ein Informationsverlust liegt dann vor, wenn es nach den zusammengefassten Teilfolgen aufgrund des vorher abgelaufenen Dialogs jeweils alternative Fortsetzungen der Ereignisfolgen gibt. Ohne den Aufbau eines zusätzlichen Gedächtnisses, das uns sagt, aus welcher der alternativen Dialogabläufe wir ursprünglich zum aktuellen Knoten gelangt sind, können wir dann keine Entscheidung mehr bezüglich der Fortsetzung unseres Dialoges fällen. D.h. aus der Kenntnis des aktuellen Knotens im Grafen allein können wir die Fortsetzung des Dialogs nicht mehr bestimmen.
Wie man im gewählten Beispiel schnell sieht, können dort die Teilfolgen am Ende zu einem einzigen Teilpfad zusammengefasst werden (Abbildung 4-4). Die Ereignisfolgen am Ende der zusammengeführten Alternativpfade sind unabhängig von den vorher eingeschlagenen alternativen Wegen.

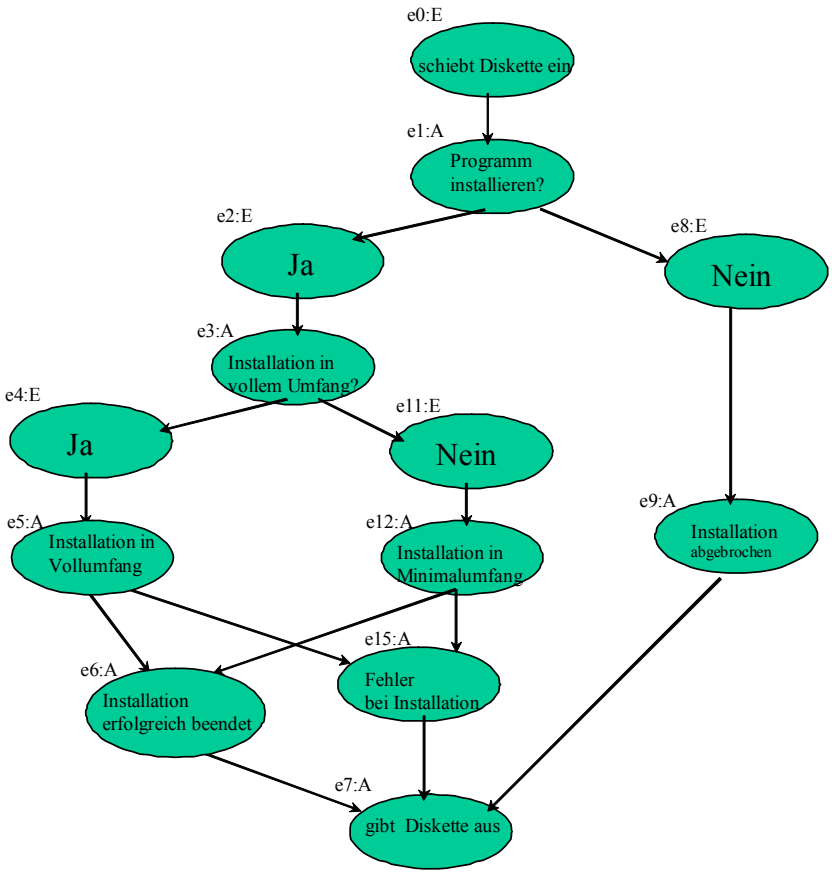


Abbildung 4-4: Zusammenführung identischer Folgen bzw. Stellen

Im Beispiel ist der Dialogschritt „System gibt Diskette aus“ allen drei Pfaden gemein und somit generell von der Vorgeschichte im Dialog unabhängig. Die Dialogschritte „Fehler bei Installation“ und „Installation erfolgreich“ sind von der vorhergehenden Auswahl des Installationsumfangs unabhängig. „Unabhängig“ ist hier so zu verstehen, dass die möglichen Dialogschritte in den jeweiligen Pfaden identisch sind.
Dies bedeutet natürlich nicht, dass beim tatsächlichen Ablauf die Vorgeschichte des Dialogs nicht dafür ausschlaggebend sein kann, welche der möglichen Schritte eingeschlagen wird. Beispielsweise kann nach Auswahl der Vollinstallation ein Fehler auftreten, weil der vorhandene Speicherplatz nicht ausreicht und deshalb die Fehlermeldung ausgegeben wird, während nach Wahl der Minimalinstallation die Installation erfolgreich verläuft und entsprechend die Erfolgsmeldung am Bildschirm erscheint. D.h., dass die Gesamtsituation im System z.B. nach der Wahl einer Alternative zum Installationsumfang durchaus unterschieden werden muss. Betrachtet man jedoch lediglich die Teilkomponente „Dialogsystem“ und lässt die Applikationslogik-Komponente außer Acht, sind die möglichen Folgeschritte nach der alternativen Auswahl dieselben. Das Dialogsystem hat also in beiden Fällen, gleich ob Minimalumfang oder Vollinstallation gewählt wurde, eine Reaktion auf den Fehlerfall und eine Reaktion auf den erfolgreichen Ausgang vorzusehen. Welche der beiden Meldungen dann ausgegeben werden, ist von der Applikationslogik-Komponente zu entscheiden (siehe Einführung von Conditions 4.1.3).

Dies bedeutet umgekehrt, dass für die Applikationslogik selbstverständlich relevant sein kann, welcher Pfad von alternativen Dialogpfaden eingeschlagen wurde. Wenn wir in unserem Dialogmodell also alternative Dialogpfade wieder auf einen gemeinsamen Pfad zusammenführen, setzen wir voraus, dass die Applikationslogik-Komponente nötigenfalls rechtzeitig ein eigenes „Gedächtnis“ bezüglich der aktuellen Situation aufbaut. Denn alleine aus der Stelle im Dialog heraus kann nicht mehr auf die aktuelle Gesamtsituation geschlossen werden (siehe wiederum Entscheidungsmechanismen in Abschnitt 4.1.3).

Im Gegensatz zur Darstellung in Abbildung 4-3 sind nun einige identische Stellen bzw. Übergänge zusammengefasst: e13 wird auf e6 zurückgeführt; e17 auf e15; e10, e14, e16, e18 auf e7; e8 auf e11. Damit sind auch Übergänge ersetzt: (e13, e14) durch (e6, e7); (e17, e18) durch (e6, e7); (e12, e17) durch (e12, e6) usw.

Wenn nun identische Folgen alternativer Pfade in der Darstellung im Grafen auf denselben Teilpfad zusammengeführt werden, kann in vielen Fällen nicht davon ausgegangen werden, dass die nachfolgenden Dialogschritte von der Vorgeschichte, also dem Verlauf des Dialogs vor Durchlauf der zusammengefassten Pfade unabhängig sind. In diesem Fall ist in der bisher gewählten Darstellung dann ein echter Informationsverlust bezüglich der Dialogsteuerung gegeben.

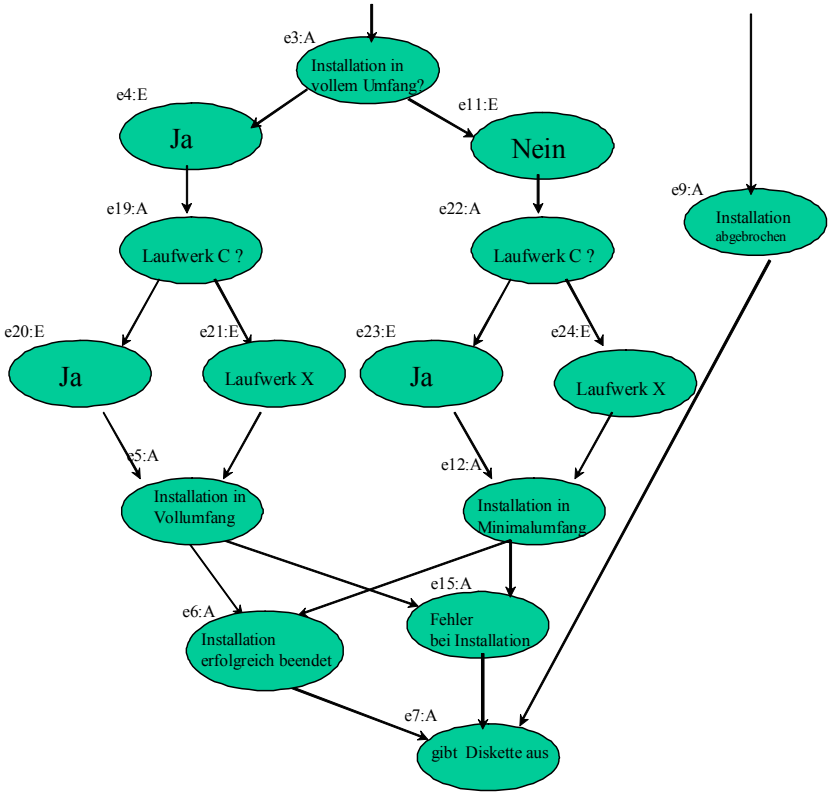


Abbildung 4-5: Das Beispiel des Installationsdialogs wird um einen Dialog zur Auswahl eines Laufwerks erweitert.
Der Laufwerksdialog wird an zwei Stellen, nämlich nach der Wahl der Vollinstallation und nach der Entscheidung für eine Installation im Minimalumfang dargestellt.

In unserem Installationsbeispiel werden zur Erläuterung dieses Falls hinter die Dialogschritte zur Auswahl des Installationsumfangs Dialogstellen zur Auswahl des Laufwerkes, auf das die neue Software installiert werden soll, eingeführt. Danach soll weiterhin die Meldung erfolgen, dass der Start der Installation erfolgt ist und in welchem Umfang die Installation erfolgt.

Nach der Zusammenfassung der in Abbildung 4-5 noch zweifach dargestellten Laufwerksauswahl zu gemeinsamen Stellen in Abbildung 4-6 erfolgt eine von der Vorgeschichte abhängige Verzweigung in unterschiedliche Startmeldungen. Die Vorgeschichte ist in Abbildung 4-6 nicht mehr unmittelbar aus den Stellen, an denen die Verzweigung erfolgen soll (e20 und e21) abzuleiten. D.h. die Situation des Dialogsystems wird durch diese Stellen alleine nicht mehr ausreichend repräsentiert. Es wird an den Stellen zusätzliche Information erforderlich, die für die Entscheidung des weiteren Dialogablaufs abgefragt werden kann. Um diesen Mechanismus zur Abfrage der Historie in unserem Modell zu beschreiben, benötigen wir neben den bisher vorgestellten Elementen des Grafen nun ein neues Sprachmittel, das wir in Form so genannter Conditions einführen wollen (siehe 4.1.3).

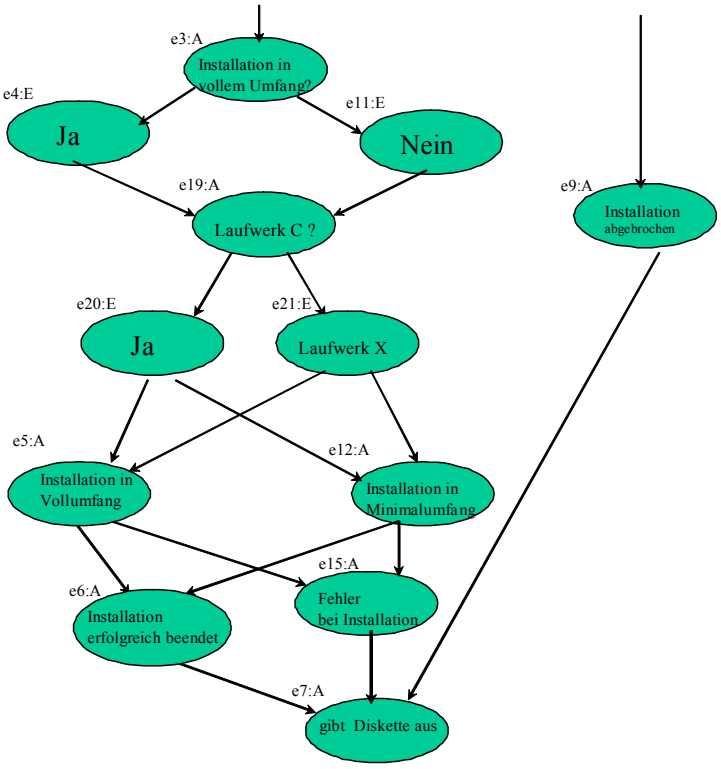


Abbildung 4-6: Zusammenführung der beiden Darstellungen des Laufwerksdialogs mit anschließender historienabhängiger Verzweigung.

Die Darstellung ist hier so gewählt, dass der Laufwerksdialog mit all seinen Dialogstellen nur einmal gezeigt wird und er aus verschiedenen Stellen heraus angesprochen wird.

4.1.0 Zusammenfassung von Werten

Eine übliche Technik, Stellen und Übergänge zu reduzieren, wird (meist intuitiv) in der Modellierung angewandt, indem alternative Stellen, also Stellen, die von einer Stelle aus durch alternative Übergänge erreicht werden können, zu einer Stelle mit unterschiedlichen

alternativen Wertebelegungen gebündelt werden. Es erfolgt dann im Modell nur mehr ein einziger Übergang zu einer Stelle, die als Wertebereich das ganze Bündel der einzelnen Werte der vormaligen Alternativstellen beinhaltet.
Beispiel: Dialog zur Eingabe eines Datums
Aus der Sicht der möglichen Eingaben eines Datumswertes gibt es sehr viele konkrete Möglichkeiten eines Datumsdialogs, die wir alle konkret im Modell hinmalen müssten, wenn wir sie durch primitive Dialoge mit konkreten Werten ausdrücken wollten; nämlich für jeden möglichen Wert eine Stelle mit entsprechenden Übergängen von den Ausgangsstellen und zu den Folgestellen. Stattdessen stellen wir eine Dialogstelle dar, die im Wertebereich alle möglichen Datumswerte beinhaltet.
Wir abstrahieren in diesem Fall also vom konkreten Wert, was wiederum dazu führt, dass wir an späteren Stellen in der Folge unter Umständen Übergänge in Abhängigkeit vom konkret eingegebenen Datumswert durch explizite Abfrage unterscheiden müssen. Durch die Abstraktion geht also zunächst im Modell Information verloren, die wir durch Einsatz von Sprachelementen zur Abfrage des konkreten Wertes ersetzen werden (siehe 4.1.3).

Wir wollen zur weitergehenden Erläuterung ein neues Dialogbeispiel einführen, nämlich den Dialog mit einem Bankautomaten, der in Abbildung 4-7 wiederum in Form eines Grafen dargestellt ist. Hier gibt es im Modell zwei Stellen, an denen die Eingabe unterschiedlicher Werte an ein und derselben Stelle möglich ist, nämlich die Eingabe einer Geheimzahl zur Authentifizierung des Kartenbesitzers und die Eingabe eines vom Kunden bestimmten beliebigen nicht standardmäßig vom Automaten angebotenen Betrages, der vom Konto abgehoben werden soll.
Bei der Eingabe einer Geheimzahl ist es nahe liegend, dass diese der Benutzer eingeben muss und dass nicht der Automat alle gültigen oder alle potentiell möglichen Werte zur Auswahl anbietet. Es ist aus Sicht des Dialogsystems also zunächst ungewiss, welcher konkrete Wert für die Geheimzahl eingegeben wird.
(Die beliebige Angabe eines abzuhebenden Geldbetrags allerdings wird bei manchen Automaten auch durch die Auswahl einer Menge vorgegebener Beträge ersetzt oder manchmal auch ergänzt.)
Um nun die verschiedenen Werte der Eingabe, also die verschiedenen Eingabeereignisse im Modell darzustellen, nutzen wir die bereits in unserem Grundmodell nicht ausgeschlossene Möglichkeit, einer Stelle eine Ereignismenge und damit auch mehrere Werte zuzuordnen. Die Menge der zugelassenen Werte können wir als Wertebereich der Stelle bezeichnen.

Wir haben also etwas formaler betrachtet eine Funktion, die jeder Stelle eine Menge von Werten, ihren so genannten Wertebereich zuordnet. Diese ist häufig definiert als die Funktion, die die Stelle auf die Menge aller Werte aller möglichen Eingabeereignisse an der Stelle abbildet.
Wie bereits bei der Darstellung des Grundmodells erwähnt, muss jedoch der Wert einer Stelle nach Eintreten eines Ereignisses an der Stelle nicht unbedingt mit dem Wert dieses Ereignisses übereinstimmen. Somit kann beim Zusammenfassen von Stellen der Wertebereich der zusammengefassten Stelle sich auch von der Wertemenge unterscheiden, die durch die Ereignisse an der Stelle definiert ist.

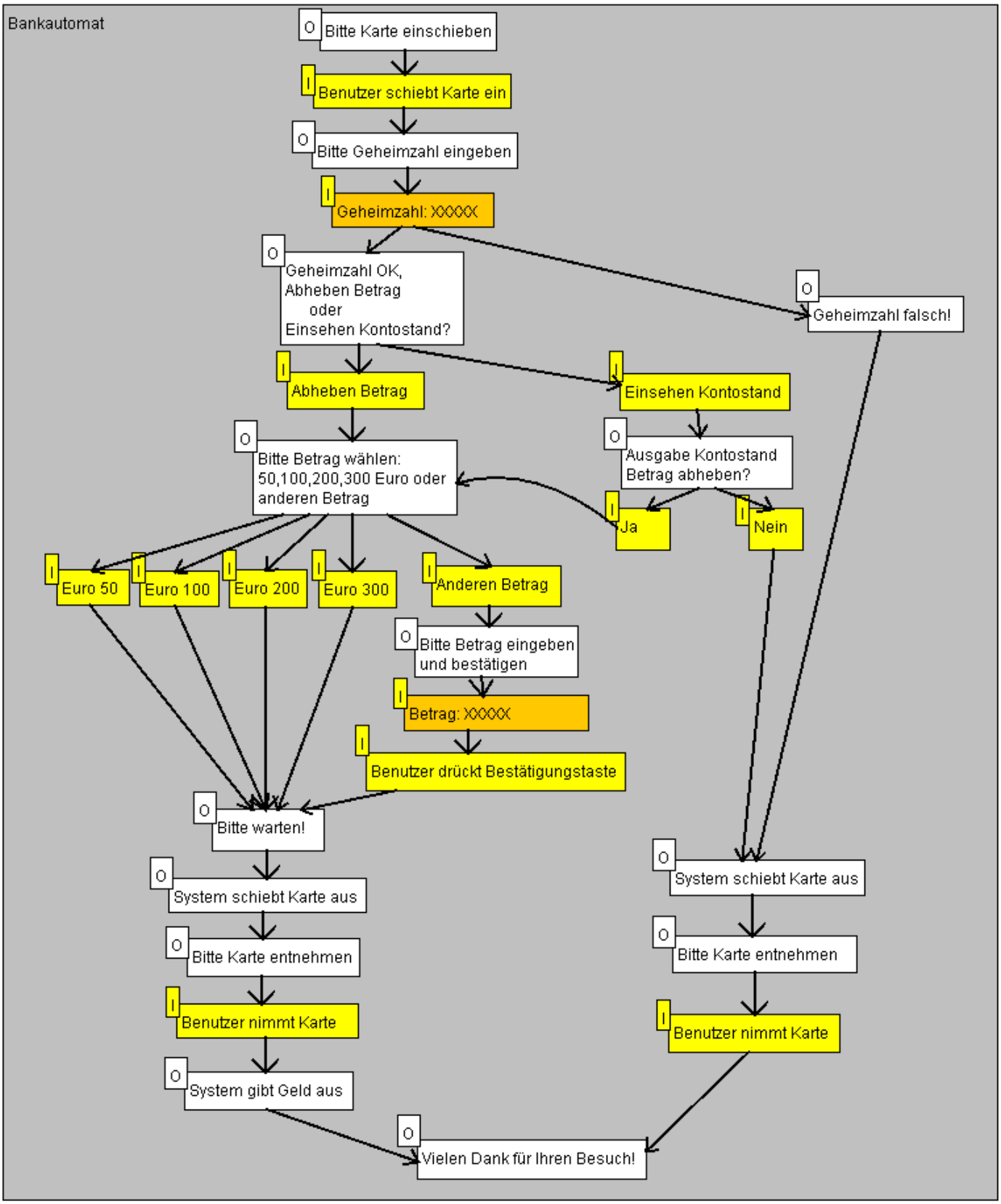


Abbildung 4-7: Graf zur Darstellung des Dialogablaufs eines Bankautomaten (ohne Zyklen)

Die Knoten des Grafen (hier als Rechtecke dargestellt) sind mit Ereignistypen (I=Input, O=Output) annotiert. Die in Orange hinterlegten Knoten repräsentieren Eingabestellen, denen ein Wertebereich zugeordnet werden muss, der die möglichen Werte der Eingabeereignisse festlegt. Alle anderen Dialogstellen repräsentieren jeweils ein Ereignis mit einem einzigen Wert.

Allgemeiner formuliert haben wir deshalb eine Funktion ω_{WEB}, die einer Stelle s genau die Menge aller Werte als Wertebereich zuordnet, auf die die Funktion ω die Paare (s, ehist) mit beliebigen Ereignisteilfolgen ehist $\in$ EHIST abbildet:
Die Wertebereichszuordnung ist eine Funktion

$$\omega_{WEB} : S \rightarrow \text{Potenzmenge}(W) ,$$

dabei gilt $WB_s := \omega_{WEB}(s) := \{ w \mid w \in W \text{ mit } \exists \text{ ehist} \in \text{EHIST mit } \omega(s, \text{ehist}) = w \}$

(s, ehist) steht dabei (wie bereits bei Definition des Grundmodells erwähnt) für eine Stelle s in einer bestimmten Situation des Dialogs, nämlich nach Eintreten einer Teilfolge ehist = f(0), … f(i) von Ereignissen in einer Folge f für ein i $\in$ N .

Wenn wir eine neue Stelle s durch Zusammenfassung einer Menge von Stellen S' konstruieren, können wir den Wertebereich wie folgt festlegen:

$$WB_s = \omega_{WEB}(s) = \{ w \mid w \in W \text{ mit: } \exists (s' \in S', \text{ ehist} \in \text{EHIST}) \text{ mit } \omega(s', \text{ehist}) = w \},$$

wobei S' die Menge der zusammengefassten Stellen bezeichnet.

An beiden genannten Eingabestellen unseres Bankautomaten-Beispiels erfolgt nach der Eingabe eine Prüfung des eingegebenen Wertes. Die eingegebene Geheimzahl muss selbstverständlich mit der der eingeschobenen Karte zugeordneten Geheimzahl übereinstimmen und bei dem eingegebenen Geldbetrag darf unter Umständen ein gesetztes Limit nicht überschritten werden.
Die entsprechenden Prüfungen durch das System sind zunächst in unserem Modell nicht dargestellt, da sie an der Dialogoberfläche nur durch die nach der Prüfung erfolgende Reaktion des Systems festgestellt werden kann. (Das Ziel der Darstellung in unserem Modell ist zunächst nur die Darstellung der Oberfläche.)
Aus nicht im Modell dargestellten Ursachen erfolgt nach der Eingabe der Geheimzahl eine Verzweigung in alternative Dialogstellen, nämlich in eine Stelle mit Fehlermeldung, die eine nicht erlaubte Eingabe anzeigt, oder aber in eine Stelle, die eine erfolgreiche Fortsetzung des Dialogs repräsentiert. Wir wissen, dass die unterschiedliche Fortsetzung jeweils von unterschiedlichen Eingabewerten an der Eingabestell abhängt. Um diesen Sachverhalt im Modell ausdrücken zu können, benötigen wir wiederum wie im Falle der Zusammenlegung identischer Teilfolgen ein Sprachmittel, das über die Darstellung im bisherigen Grafen hinausgeht.
Die Zusammenfassung von Stellen mit alternativen Werten zu einer Stelle führt selbstverständlich ebenfalls wie die Zusammenlegung identischer Teilfolgen zu einem Verlust der Information im Grafen. Wir können aus der Stelle selbst keinen eindeutigen Verlauf des Dialogs hinsichtlich der eingegeben Werte und damit der erfolgten Dialogereignisse mehr feststellen. Wollten wir dies alleine mit den Sprachmitteln des Grafen, also mit Knoten (Stellen) und Kanten erreichen, müssten wir wie vor der Zusammenfassung jeden der möglichen Eingabewerte durch einen Knoten (eine Stelle) repräsentieren.
Um den Verlauf des Dialogs mithilfe unseres Modells auch nach Zusammenfassung von Stellen alternativer Wertebelegungen steuern zu können, verwenden wir wieder als Sprachmittel die oben bereits erwähnte Condition (siehe auch unten 4.1.3).

4.1.1 Hierarchienbildung

Nach der Zusammenfassung mehrerer Stellen zu einer Stelle können wir diese neue Stelle in unserem Modell nach wie vor als eine primitive Stelle betrachten. Wir können aber auch die Zusammenfassung selbst in unserem Modell festhalten, indem wir diese neue Stelle als eine zusammengesetzte Stelle im Modell definieren und die zusammengefassten Stellen als

Teilstellen der zusammengesetzten Stelle aufführen. Dies machen wir im Allgemeinen dann, wenn z.B. nicht allzu viele Teilstellen aufzuzeigen sind oder wenn die Teilstellen später für die Implementierung ganz bestimmten unterschiedlichen Laufzeitstellen zugeordnet werden sollen. Z.B. kann ein Datum mit drei konkreten Teilstellen Tag, Monat und Jahr auch auf drei Felder am Bildschirm abgebildet werden. In Point-and-click-Dialogen sehen wir häufig, dass dann konkrete Tage, Monate und Jahre auch explizit zur Auswahl z.B. in Pop-up-Menüs angeboten werden.
Zwischen den Teilstellen und der zusammengesetzten Stelle besteht eine hierarchische Beziehung, die wir häufig als eine has-a-Beziehung (obiges Beispiel Datum hat Tag, Monat und Jahr) oder als eine is-a-Beziehung (Beispiel: Tag ist 1-ter, oder 2-ter .. , oder 31-ter) betrachten können.
Es werden häufig alternative Stellen zusammengefasst, wenn sie in ihren Eigenschaften, in ihrer Bedeutung Ähnlichkeiten aufweisen, die die Zusammenfassung als sinnvoll erscheinen lassen, oder auch, wenn der nachfolgende Dialogfluss durch die Zusammenfassung kaum berührt ist, also nahezu oder vollständig unabhängig ist.
Wir erreichen durch die Zusammenfassung nicht nur eine rein technische Verringerung der Stellen sondern auch häufig eine semantische Strukturierung.

4.1.1.0 Zusammenfassung von Dialogfolgen

Eigentlich trivial: eine Sequenz von Ein-/Ausgabedialogen wird zu einem Dialog zusammengefasst. Wir stellen in unserem Grafen die Sequenz nur mehr als eine Stelle dar. Wollen wir die Detailinformation über die Einzelschritte nicht verlieren, definieren wir diese Stelle als zusammengesetzte Stelle, die einen Subdialog repräsentiert, in dem die ursprüngliche Dialogsequenz dargestellt ist.
Zum Gedächtnisverlust bezüglich Historie: Solange wir nur eine bestimmte Teilsequenz zu einer Stelle mit Subdialog zusammenfassen und der entstandene Subdialog einzig dieser zusammengefassten Stelle zugedacht ist, benötigen wir noch kein zusätzliches Gedächtnis für die richtige Fortsetzung nach Ablauf des Subdialogs.
Erst wenn der gleiche Subdialog – und das ist ein weiterer Schritt der Zusammenfassung – an mehreren Stellen Verwendung findet, benötigen wir wiederum ein Gedächtnis, aus dem wir ermitteln können, an welcher Stelle der Subdialog „aufgerufen“ wurde und wo demnach fortzusetzen ist.
Bei einer Hierarchisierung über mehrere Stufen gelangen wir letztlich zu einem Gedächtnis, das als so genannter Keller organisiert ist.

4.1.1.1 Zusammenfassung von Teilgrafen

Neben der Zusammenfassung einfacher Sequenzen bilden wir im Allgemeinen Hierarchien, indem wir ganze Teilgrafen zu einer Stelle reduzieren. D.h. in die Zusammenfassung werden auch alternative Dialogpfade mit einbezogen. Sobald jedoch Alternativen der Dialogführung im Subdialog enthalten sind, kann es nach Abschluss des Subdialogs und Rückkehr zur „Aufrufstelle“ erforderlich sein, zu unterscheiden, welche der Alternativen im Subdialog gewählt wurde. Die Fortsetzung einer zusammengefassten Dialogstelle ist dann durch den inneren Ablauf (Zustand) der Dialogstelle bestimmt. Dieser ist zunächst aus der Darstellung des Dialogs als gefaltete Stelle mit Übergängen zu Folgestellen nicht ersichtlich. Um diesen Mangel an Information auszugleichen, gibt es beispielsweise die Möglichkeit

1. der besonderen Darstellung der Aufrufstelle des Subdialogs mit mehreren alternativen Anschlüssen: d.h. wir sehen im Subdialog nicht eine einzige Rückkehrstelle vor, sondern eine Rückkehrstelle für jede der Alternativen. Im Grafen des Subdialogs bedeutet dies für jede Alternative einen Endknoten. An der Aufrufstelle ist dann für jeden dieser Endknoten eine alternative Anschlussstelle vorzusehen.

2. einer einzigen Rückkehrstelle und dann der Formulierung von Bedingungen an den Übergängen von der Aufrufstelle zu Folgestellen (siehe auch Abbildung 4-8).

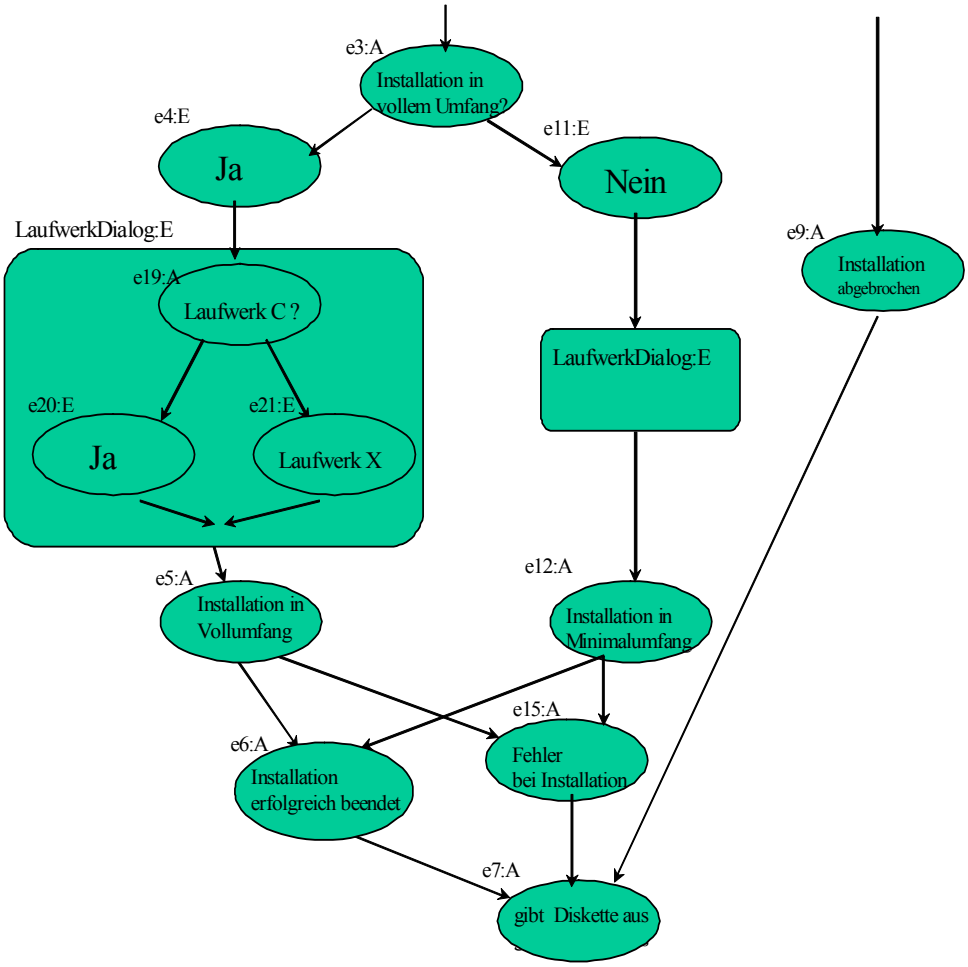


Abbildung 4-8: Hierarchische Darstellung am Beispiel des Laufwerksdialogs.

Statt der Reduktion von Stellen und Stellenübergängen in Abbildung 4-6 durch einmalige Darstellung wird hier ein Dialog mit Namen „LaufwerkDialog“ zweimal gezeigt. Der Dialog fasst mehrere „Subdialoge“ zusammen. An der einen Stelle wird der Dialog „entfaltet“ an der anderen „gefaltet“ gezeigt. Der Laufwerk-Dialog hat nur einen Ausgang. Welcher der alternativen internen Stellen durchlaufen wurde, ist aus der Fortführung des Grafen über eine Ausgangsstelle nicht mehr ersichtlich. Wir benötigen ein zusätzliches „Gedächtnis“, das wir im späteren Verlauf abfragen können (siehe später Hilfsstellen und Conditions).

Die Zusammenfassung kann, wie oben bereits erwähnt, in mehreren Stufen erfolgen. Letztlich kann der gesamte Dialog einer Stelle in einem Grafen zugeordnet werden, der nur diese Stelle als Knoten beinhaltet (Graf der Hierarchiestufe null). Die Stelle repräsentiert das gesamte zusammengesetzte Dialogereignis.

Grad der notwendigen oder sinnvollen Verfeinerung:
Statt die oben beschriebene Bündelung durchzuführen, nehmen wir umgekehrt im Modellierungsprozess häufig auch eine Verfeinerung in mehreren Hierarchiestufen vor. Im Sinne einer Granulierung des Dialogs kommen wir zu einem Feinheitsgrad, der aus aufgabenlogischer und/oder aus dialogtechnischer Sicht für die Modellierung irrelevant wird. Das obige Beispiel der Modellierung einer Datumseingabe kann z.B. im Extremfall im Sinne einer weiteren Detaillierung bis auf eine Ebene fortgesetzt werden, in der auch noch die einzelnen Ziffern einer Jahreszahl als Teilstellen modelliert werden. In vielen Fällen ist aber bereits die Detaillierung der Eingabe eines Datums in Tag, Monat und Jahr uninteressant. Zum einen ist es in vielen Fällen (nicht in allen) aufgabenlogisch unwichtig, wie ein Datum und welches Datum eingegeben wird. In vielen Fällen bleibt insbesondere eine Fortsetzung des Dialogpfades von der Eingabe eines bestimmten Datumswertes unbeeinflusst. D.h. unser Modellgraf ändert sich in der Darstellung der potentiell nachfolgenden Dialoge nicht mehr, wenn wir eine höhere Feinheit der Dialogdarstellung der Ausgangsstelle wählen.
Die notwendige Verfeinerung ergibt sich auch aus der Ebene, ab der in konkrete vorgefertigte Laufzeitstellen abgebildet werden kann. Wenn z.B. bereits auf der Ebene der Implementierung ein komfortables Datumswidget existiert, ist eine detaillierte Darstellung des Datumseingabedialogs auf Modellebene unnötige Doppelarbeit.

4.1.2 Schleifenbildung und Rekursion

4.1.2.0 Zyklen

Die Einführung von Zyklen (Schleifen) in unserem Graphen geschieht üblicherweise, indem wir einen Pfad an einer Stelle fortsetzen, indem wir eine gerichtete Kante auf eine Stelle „zurückführen“, die bereits im selben Pfad enthalten ist.
Mit der Bildung von Zyklen wird die bisherige Eigenschaft im Modell aufgegeben, dass eine Stelle nur höchstens einmal erreicht wird. Eine Stelle kann mehrmals „durchlaufen“ werden, d.h. eine Stelle repräsentiert im Sinne der konkreten Abfolge von Ereignissen nicht mehr nur genau ein (potentielles) Ereignis im Dialogablauf sondern mehrere (potentielle) konkrete Ereignisse. Die Zyklenbildung führt also zu einer weiteren Art Zusammenfassung oder Abstraktion eines Ereignisses bzw. einer Ereignisstelle: die Identität der Stelle an sich macht keine Unterscheidung mehr bezüglich des eindeutigen Vorgängerpfades möglich. Der potentiell ab der Stelle nachfolgende Verlauf des Pfads ist unabhängig von unterschiedlichen Vorgängerpfaden. Wollen wir den Folgepfad von der Vorgeschichte abhängig machen, insbesondere davon, wie oft der Zyklus bereits durchlaufen wurde, müssen wir dies wiederum mit gesonderten Sprachmitteln bewerkstelligen (etwa zum Test einer Abbruch-Bedingung).
Das Einführen von Zyklen bedeutet das wiederholte Aufrufen „derselben“ Dialogsequenz (desselben Teilgrafen) aus einer unterschiedlichen Historie, also im Sinne der durchlaufenen Historie aus unterschiedlichen Zuständen. Sehen wir den Modellgrafen allerdings als Zustandsübergangsdiagramm, befinden wir uns nach jedem Übergang in die Anfangsstelle des Zyklus im selben Zustand. Wir abstrahieren also hier von der Anzahl des Durchlaufens des Zyklus. Wenn wir also eine Unterscheidung des möglichen Folgedialogs in Abhängigkeit von der Anzahl der Zyklenwiederholung treffen wollen, benötigen wir wiederum ein Gedächtnis zusätzlich zum vorliegenden Modellgrafen mit entsprechenden Sprachmitteln zur Abfrage.
Semantisch kann die Zusammenfassung von Teildialogen durch die Einführung von Schleifen unterschiedlich interpretiert werden: So kann das nochmalige Durchlaufen einer Dialogstelle bedeuten, dass vorher an der Dialogstelle gemachte Aussagen im Sinne einer Korrektur oder Ergänzung überschrieben oder erweitert werden sollen. Es handelt sich dann um eine Art Wiederaufnahme und Fortsetzung des alten Dialogs. Es kann aber auch z.B. bedeuten, dass

ein völlig neuer Dialog begonnen wird, der nur in seiner Bedeutung, in Form und Inhalt ähnlich zu verstehen ist, aber eine neue Aussage darstellt, die die vorher gemachten Aussagen an der Dialogstelle nicht berührt. Entsprechend kann auch in der späteren Implementierung dieselbe Dialogstelle (z.B. dasselbe Formular) bei Neudurchlauf der Stelle mit der gleichen Inkarnation eines Widgets realisiert werden oder es kann eine neue Inkarnation (weiteres Formular) kreiert werden.

4.1.2.1 Rekursion

Wie durch Einführung von Zyklen, kann eine Stellenreduktion erreicht werden, wenn wir Rekursion in unserer Darstellung im Modellgrafen erlauben:
Rekursion liegt im hierarchischen Modellgrafen dann vor, wenn eine zusammengesetzte Stelle in der durch die Hierarchie-Relation definierten transitiven Hülle als Teilstelle von sich selbst auftritt. Dies bedeutet bezüglich der Darstellung im Modellgrafen, dass die Stelle als Knoten des der Stelle selbst zugeordneten Grafen oder an einem Knoten eines Subgrafen dieses Grafen (usw.) aufgeführt ist.

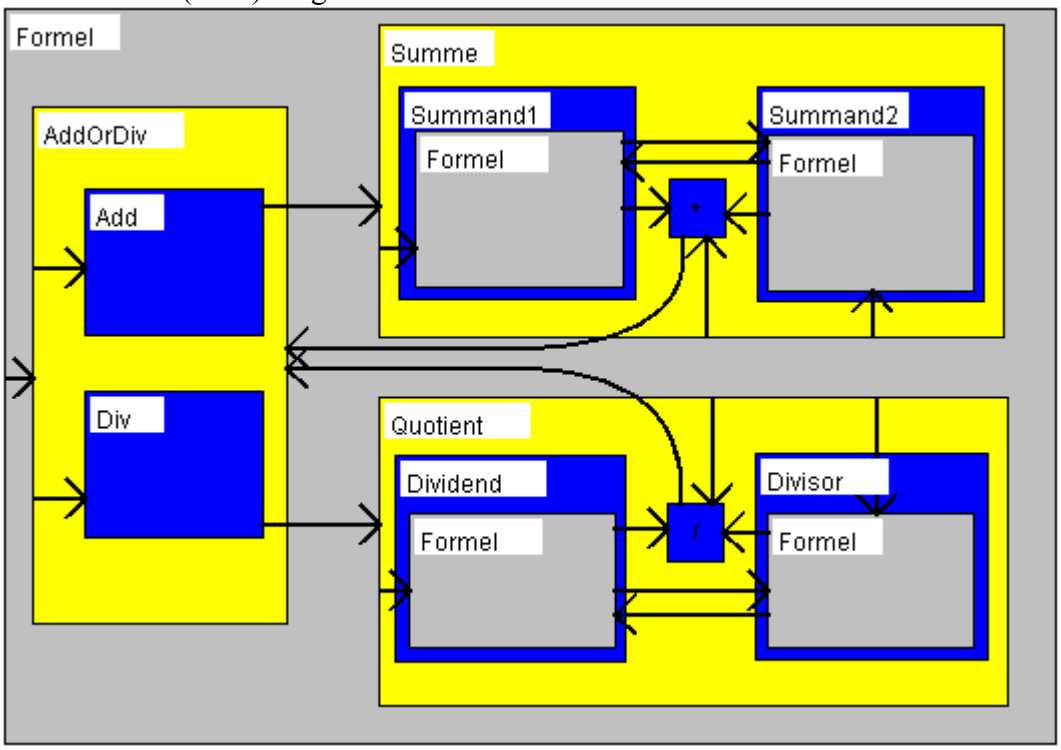


Abbildung 4-9: Beispiel für Rekursion im hierarchischen Dialogmodellgrafen

Die Abbildung zeigt das Modell eines Formeleditors (in Anlehnung an [EICKEL01]). Die über die dargestellten Kanten begehbaren Dialogpfade beinhalten einen rekursiven Aufruf des Formeldialogs. In der Darstellung ist auf eine Annotation der Ereignistypen verzichtet: Alle Knoten können o.B.d.A. als Eingabestellen betrachtet werden. Die vom Rand eines Rechtecks (=Knotens) nach innen gehenden Kanten definieren den Übergang zu (u.U. alternativen) Startknoten im Subgrafen. So erfolgt z.B. bei Eingabe von „Add“ ein Übergang zum Summendialog mit den Alternativen: Eingabe Formel für Summand1 oder Eingabe Formel für Summand2 oder Eingabe „Plus“.

Wir können dann sagen, die Stelle tritt rekursiv auf.
In der Grafhierarchie wird also dieselbe Stelle durch zwei Knoten repräsentiert, die in unterschiedlichen Hierarchieebenen dargestellt sind und hierarchisch einander zugeordnet sind.

Wird die Hierarchie aufgelöst und werden die Stellen durch untergeordnete Teilstellen ersetzt, benötigt man bekanntermaßen - ähnlich wie bei Zyklen - ein Abbruchkriterium, um irgendwann den Ersetzungsmechanismus (die Produktion von Teilstellen) der rekursiv auftretenden Stelle zu beenden.
Wir benötigen in unserem Grafen also eine Bedingung, die ausdrückt, dass die durch die Stelle repräsentierten Dialogereignisse in einer bestimmten Dialogsituation nicht mehr nachfolgen können.

4.1.3 Entscheidungsmechanismen für Stellenübergänge und Wertebelegungen

In den vorangehenden Abschnitten haben wir festgestellt, dass durch die Reduktion der Stellen in der Modelldarstellung in manchen Fällen Information über die Fortführung des Dialogs verloren geht. Wir benötigen dann einen zusätzlichen Entscheidungsmechanismus, der die Auswahl aus alternativen Übergängen zu anderen Dialogstellen vornimmt. Dieser Entscheidungsmechanismus wird generell benötigt, wenn in unserem Modell alternative Fortführungen vorhanden sind, die vom System, also sowohl von der Dialogkomponente als auch von der Applikationslogik gesteuert werden sollen. Dies gilt also insbesondere für die Fortführung des Dialogs zu alternativen Stellen, die eine Systemausgabe darstellen. Bereits im ersten Beispiel des Installationsdialogs hatten wir die Situation, dass eine alternative Verzweigung für den Fall einer gelungenen und einer fehlerhaften Installation vorgesehen war. In diesem Fall ist es die Aufgabe der Applikationslogik, die Entscheidung zwischen den Alternativen zu fällen und an die Dialogkomponente weiterzugeben, die letztlich dann die Entscheidung für die Fortführung der Darstellung an der Dialog-Oberfläche trifft.
Gibt es im Dialog alternative Eingabestellen des Benutzers, liegt die Entscheidung für die zu wählende Eingabestelle beim Benutzer. Hier ist das System nur insofern gefordert, als es von vornherein entscheiden muss, ob es alle zunächst im Modell dargestellten alternativen Fortführungen zu diesen Eingabestellen tatsächlich in einer bestimmten Dialogsituation verfügbar macht.

4.1.3.0 Bedingungen (Conditions)

Um nun ein Hilfsmittel zur Verfügung zu stellen, mit dem das System entscheiden kann, ob ein Übergang zu einer potentiellen Fortsetzungsstelle erfolgen soll, führen wir eine Funktion mit booleschem Rückgabewert ein, die einem Übergang zwischen zwei Stellen zugeordnet werden kann. Liefert die Funktion den Wert TRUE, soll der Übergang von der Ausgangsstelle zur Zielstelle erfolgen, d.h. das Ereignis an der Zielstelle findet statt. Im FALSE-Fall erfolgt entsprechend kein Übergang. Ohne Beschränkung der Allgemeinheit besitzt die jeweilige Funktion eine Reihe von Eingangsparametern in Form von Werten an definierten Stellen.
Die boolesche Funktion soll im Folgenden als Condition oder Prädikat (siehe [Eickel01]) bezeichnet werden. In der formalen Darstellung unseres Modells führen wir somit ein:

FS: Menge der durch die Ereignisfolgen induzierten Stellenfolgen fs,
fs: $N_f \rightarrow S$, wobei fs durch eine Ereignisfolge $f \in F$ (F Menge der Dialogereignisfolgen) wie folgt definiert ist:
$fs(j) := \sigma(e_j)$ für $e_j \in f$,
und $j \in N_f = \{ 0,..,n \}$, N_f Definitionsbereich von f.

Ü: Relation der Stellenübergänge, $Ü \subseteq S \times S$, wobei

$(s_i,s_j) \in Ü \Leftrightarrow \exists\, i \in N, fs \in FS$ mit $fs(i) = s_i \wedge f_s(i+1) = s_j$,

also s_i Vorgänger von s_j für ein $fs \in FS$.

C: Menge der Conditions $\beta_{(si,sj)}$, als Funktionen der Form

$$\beta_{(si,sj)} : WS_{\beta(si,sj)} \rightarrow \{ TRUE, FALSE \},$$

wobei $WS_{\beta(si,sj)}$ Menge aller Wertezuordnungen auf einer definierten Stellenmenge $S_{\beta(si,sj)} \subseteq S$, also:

$$WS_{\beta(si,sj)} = \{ \omega s, \omega s \text{ ist Funktion } \omega s \mid S_{\beta(si,sj)} \rightarrow W \}.$$

Ein Übergang von Stelle s_i nach Stelle s_j soll nur dann stattfinden, wenn die Condition $\beta_{(si,sj)}$ angewandt auf die aktuell gültige Wertebelegung ωs_{akt} an den Stellen in $S_{\beta(si,sj)}$ den Wert TRUE liefert, also wenn gilt: $\beta_{(si,sj)}(\omega s_{akt}) = TRUE$.
Die aktuelle gültige Wertebelegung wird definiert durch:
$\omega s_{akt}(s) = \omega(s, ehist)$, für $s \in S_{\beta(si,sj)}$ mit $ehist \in EHIST$ als aktuell abgelaufener Teilfolge von Ereignissen.

Auf diese Weise wird beispielsweise ermöglicht, dass die Historie der Dialogführung für die Entscheidung der Fortsetzung herangezogen werden kann. Dies geschieht so, dass über Werte an bestimmten Stellen die Historie des Dialogablaufs aufgezeichnet wurde (siehe dazu unten auch Werteübertragungen) und von diesen Stellen nun zur Entscheidungsfindung mittels Condition die Werte abgelesen werden.

4.1.3.1 Hilfsstellen

Um im System auch Schreibereignisse zuzulassen, die weder als Benutzer-Eingabe noch als Systemausgabe gedacht sind, führen wir einen neuen Ereignistyp M_S (Memorize by System oder Merken durch System) ein. Ereignisse des Typs M_S, die auch als „verborgene" Schreibereignisse des Systems charakterisiert werden können, dienen z.B. dazu, Werte zur Beschreibung der Dialoghistorie an Stellen zu speichern, die nicht an der Dialogoberfläche wahrgenommen werden. Die diesen Ereignissen zugeordneten für den Benutzer verborgenen Stellen wollen wir als Hilfsstellen oder auch als „reine Speicherstellen" bezeichnen. Auf diese Hilfsstellen kann nun zusätzlich mittels Conditions (s.o.) oder Übertragungsfunktionen (s.u.) zugegriffen werden.
Formal erweitern wir also unsere Menge von Ereignis-Typen T:

$$T = \{ I_B, O_S, M_S, L_B, L_S, \}$$

(Die Menge der möglichen Ereignistypen T' für zusammengesetzte Ereignisse erweitert sich um entsprechende Kombinationsmöglichkeiten mit Typ M_S.)
Ereignisfolgen können sich nunmehr aus „reinen Dialogereignissen" gemischt mit verborgenen Schreibereignissen des Typs M_S zusammensetzen.
Eine sinnvolle Einschränkung sei noch gefordert: An derselben primitiven Stelle seien Ereignisse der Typen O_S oder I_B gleichzeitig mit Ereignissen des Typs M_S nicht erlaubt. Eine nicht zusammengesetzte Stelle, an der das System schreibt, ist also eindeutig entweder eine verborgene Hilfsstelle oder eine Stelle an der Dialogoberfläche.

4.1.3.2 Übertragungsfunktionen (Propagationsfunktionen)

Da wir in unserer Modelldarstellung in der Regel alternative Ereignisse an ein und derselben Stelle zusammengefasst haben, benötigen wir nicht nur einen Entscheidungsmechanismus für

die Wahl alternativer Stellenübergänge, sondern auch für die Wahl alternativer Ereignisse. D.h., ist z.B. im Dialog eine Situation erreicht, in der das System eine Ausgabe an einer Stelle durchführen soll, an der potentiell unterschiedliche Ausgaben gemacht werden können, braucht das System eine Handhabe, um zu entscheiden, welches Ereignis propagiert werden soll. Diese Entscheidungsnotwendigkeit gilt wiederum nicht nur für Ausgabeereignisse, sondern generell für alle Ereignisse, die vom System zu realisieren sind und für die Alternativen bestehen, also auch für Schreibvorgänge an Nicht-Dialogstellen (Hilfsstellen der Dialogkomponente).
Die Entscheidung für ein Ereignis alternativer Ereignisse an einer Schreibstelle ist äquivalent zur Entscheidung, welcher Wert an die Stelle übertragen werden soll. Dies resultiert daraus, dass wir in unserem Modell Ereignisse nur nach Typ, Stelle und Wert unterscheiden. Wenn Typ und Stelle bereits aus der aktuellen Situation heraus gegeben sind, bleibt nur noch, den Wert zu bestimmen. Für die Wertebestimmung führen wir nun Funktionen ein, die ähnlich zu den oben beschriebenen Conditions auf Basis von aktuellen Werten an bestimmten Stellen einen der möglichen Alternativwerte bestimmen, der für die Realisierung des anstehenden Folgeereignisses benötigt wird. Eine derartige Funktion wollen wir Übertragungsfunktion oder Propagationsfunktion nennen, da sie für die Werteübertragung bzw. für die Propagation eines bestimmten Ereignisses durch das System verwendet wird.
Formal bedeutet dies folgende Modellerweiterung:

P: Menge von Übertragungsfunktionen (Propagationsfunktionen) $p_{(si,sj)}$ der Form:

$$p_{(si,sj)} \mid WS_{P(si,sj)} \rightarrow W,$$

$p_{(si,sj)}$ ist jeweils einem Stellenübergang $(s_i,s_j) \in Ü$ (siehe oben) zugeordnet.
$WS_{P(si,sj)}$ ist ähnlich zur Definitionsmenge von Conditions (siehe oben) eine Menge von Wertezuordnungen auf einer für die Übertragungsfunktion definierten Stellenmenge $S_{P(si,sj)}$.

Mit einer Übertragungsfunktion wird es nun dem System ermöglicht, zu entscheiden welches der möglichen alternativen Schreibereignisse des Systems an einer bestimmten aktuellen Folgestelle propagiert werden soll.

Dabei bestimmt die Propagationsfunktion den Wert des Ereignisses. Typ und Stelle des Ereignisses ergeben sich aus der übrigen Modelldarstellung: die Propagationsfunktion ist eindeutig einem Stellenübergang (si,sj) zugeordnet. Also handelt es sich um ein Ereignis an der Stelle sj. Außerdem kann es sich nur um ein Schreibereignis des Systems handeln, also entweder vom Typ O_S oder M_S. Wegen oben gemachter Einschränkung ist an einer Stelle nur einer der beiden Typen erlaubt und damit eine eindeutige Zuordnung des Typs möglich:
$\sigma^{-1}(s_j)$ liefert uns alle Ereignisse an der Stelle und damit auch das Ereignis vom Typ O_S oder M_S mit dem von der Propagationsfunktion gelieferten Wert.

Gemeinsam mit den oben eingeführten Conditions wird damit die komplette Steuerung der Dialogereignisse, soweit sie in der Hand des Systems liegt, ermöglicht. Hier nehmen wir allerdings eine etwas idealisierte Sicht im Modell ein. Wir gehen davon aus, dass mit den Hilfsstellen letztlich der komplette Speicher repräsentiert wird, der für die Realisierung einer Anwendung benötigt wird, und dass wir durch Anwendung entsprechender Propagations-funktionen diesen Speicher abfragen und mit Werten belegen. In der Praxis bedeutet dies, dass eine oder mehrere „sehr große“ Hilfsstellen benötigt werden, die bedeutende Teile des Speichers der Applikation abdecken.
Wenn wir beispielsweise eine Datenbankfrage absetzen, müssen wir die Datenbank mit all ihrer Nutz- und Verwaltungsinformation als Hilfsstelle betrachten, auf die eine Propagationsfunktion zugreift, um das Suchergebnis zu ermitteln. Umgekehrt gilt beim

Abspeichern von Werten in der Datenbank, dass eine Propagationsfunktion die Hilfsstelle „Datenbank“ so beschreibt, dass dort die neuen Werte eingefügt werden. Wir werden derartige Hilfsstellen natürlich nicht vollständig beschreiben, d.h. „ausmodellieren“ wollen, wenn wir ein Dialogmodell für eine Datenbankapplikation erstellen wollen. Wir können z.B. im Allgemeinen nicht den Anfangswert einer derartigen Hilfsstelle für den Start des Systems angeben.
In der Praxis werden wir uns also damit behelfen, dass wir beispielsweise als Wert der Hilfsstelle einen Bezeichner für die Datenbank angeben, wobei wir uns dessen bewusst sind, dass die formal korrekte Propagationsfunktion bei jeder ihrer Anwendungen auf den dynamisch sich ändernden, aktuellen Inhalt der Datenbank und nicht nur auf den konstanten Bezeichner zugreift und ein entsprechendes aktuelles Ergebnis für die Ausgabe liefert. Für das Abspeichern in der Datenbank gilt ähnliches: Formal korrekt berechnet die Propagationsfunktion einen neuen Wert der Hilfsstelle „Datenbank“, der dort abgelegt wird. In der Praxis geben wir jedoch die Datenbank nicht als Ausgabeparameter bzw. Hilfsstelle an. Es handelt sich dann sozusagen um eine verborgene nicht im Modell dargestellte Stelle, deren Beschreiben durch die Implementierung der Propagationsfunktion realisiert wird.
Um die verborgene Stelle im Modell sichtbar zu machen, könnte man beispielsweise einen besonders ausgezeichneten Typ von Hilfsstelle (etwa Typ „blind“) einführen, der bei der automatischen Umsetzung in die Implementierung nicht eingeht.

4.2 Logische Abstraktion zeitlicher Folgen

Die oben angeführten zum Teil trivialen, manchmal intuitiv oder auch bewusst durchgeführten Schritte zur Reduktion der Ereignisstellen in unserem Modell reichen bei weitem noch nicht aus, um in einer Vielzahl von Anwendungen das Modell auf ein vernünftiges Maß von Modellstellen einzuschränken.
Durch die Zusammenfassung identischer Stellenfolgen, das Einführen von Stellen mit Wertebereichen, die Mehrfachverwendung von Subgrafen, das Einführen von Zyklen und Rekursion ist es zwar gelungen, die Darstellung der konkreten Ereignisse an einzelnen Stellen im Modell zurückzuführen auf eine Darstellung von Ereignismengen. Jede Modellstelle repräsentiert eine Menge von Ereignissen. Durch die Stellen des Modells wird die Menge aller Ereignisse auf Ereignisklassen reduziert.
Wir gehen bisher aber immer noch davon aus, alle möglichen Übergänge von einem Dialogereignis zu einem anderen auch als Übergang in unserem Modell explizit auszuweisen. Dies geschieht dadurch, dass wir alle möglichen Übergänge von einer Ereignisklasse zu einer anderen Ereignisklasse im Modell in Form einer Kante unseres Grafen repräsentieren.
Dabei gibt es nun Übergänge, die im Sinne der logischen Modellierung unseres Dialogs irrelevant sind. D.h., es gibt Übergänge, die einfach aus der Annahme in unserem Grundmodell resultieren, dass alle Ereignisse im Dialogablauf in einer bestimmten Reihenfolge einzuordnen sind, wobei aber die Reihenfolge dieser Dialogschritte logisch unbedeutend ist.
Ziel des folgenden Abschnittes ist es, zu einer weiteren Reduktion der notwendigen Darstellung von Übergängen in unserem Modell zu kommen, um damit eine Modellierung in der Praxis zu ermöglichen bzw. diese signifikant zu verbessern.

4.2.0 Differenzierung der Stellentransitionen

Bisher haben wir den Dialogfluss in homogener Weise durch die Ereignisfolge beschrieben. Im Grundmodell wird jeder mögliche Schritt von einem Ereignis zum nächsten in gleicher

Weise durch aufeinander folgende Folgenglieder (e_i, e_{i+1}) in einer Folge f dargestellt. Im bisher entwickelten Modell-Grafen bedeutet dies, zwei Ereignisstellen als Knoten mit einer gerichteten Kante zu verbinden. Der Übergang von einer Stelle zur anderen bedeutet das Eintreten eines Ereignisses aus der Menge der Ereignisse an der Folgestelle nach dem Eintreten eines Ereignisses aus der Menge der Ereignisse der Vorgängerstelle.
Die Darstellung ist für jeden Übergang gleich, d.h. es wird keine Aussage über die Ursache der Ereignisfolge, d.h. den Zusammenhang der beiden nachfolgenden Ereignisse bzw. Ereignismengen gemacht, ob beispielsweise der Übergang von einem Ereignis zum anderen rein zufällig ist oder ob es sich um einen unausweichlichen (z.B. modalen oder vom System synchron ausgelösten) Übergang handelt.
Wenn wir uns nun von einer indifferenten Darstellung aller möglichen (mehr oder weniger willkürlichen) Ereignisfolgen lösen und etwa nur die vom System oder nur vom Dialogsystem verantworteten unmittelbar bewirkten Übergänge beschreiben und die vom Benutzer und ggf. auch der Applikationslogik bewirkten Übergänge nur soweit beschreiben, als sie von der Dialogkomponente ermöglicht werden müssen, gewinnen wir eine wesentlich effizientere und übersichtlichere Darstellung.

4.2.1 Kriterien der Dialogdynamik

Wir wollen im Folgenden einige informelle Grundüberlegungen dazu anstellen, wieweit es möglich bzw. notwendig oder sinnvoll ist, im Dialogmodell unterschiedlich begründete Zusammenhänge der einzelnen Dialogschritte darzustellen und welche unterschiedlichen Sprachelemente dazu von Nutzen sein könnten. Die Untersuchung der Zusammenhänge dient dem Ziel, diese beschreiben zu können und damit in einem Modell zu erfassen. Letztlich resultiert auch daraus, welche Sprachelemente wir im Modell zur Beschreibung der Dynamik benötigen.
Insbesondere ist zu prüfen, inwieweit der bisher eingeführte Ereignisbegriff, den wir anstelle des gerade benutzten Begriffs Dialogschritt im Modell verwenden, durch andere Sprachelemente ergänzt oder ersetzt werden muss, um die Kontrolle und Steuerung des Dialogablaufes zu beschreiben.

4.2.1.0 Ereignisse bewirken Ereignisse

Wenn wir vom Grundmodell ausgehen, so sprechen wir dort von Folgen von Dialogereignissen, nämlich Eingaben und Ausgaben, die sich aus nicht weiter beschriebenen Gründen aneinanderreihen. In Anbetracht des Aspektes der Reihenfolge sprechen wir bei Dialogereignissen auch von Dialogschritten, wohl im Sinne von einzelnen Gliedern, die eine Dialogfolge ergeben. Ein Dialogschritt kann somit verstanden werden als ein einzelnes in einer Folge befindlicher Dialogereignisse.
Dass wir den Ereignisbegriff verwenden, ist nahe liegend nicht zuletzt, weil er in gängigen Modellen zur Beschreibung der Dialogdynamik verwendet wird. So ist er ein zentraler Begriff bei den so genannten Ereignismodellen. [GREEN86]. Aber auch beispielsweise bei Zustandsübergangsdiagrammen sprechen wir von Ereignissen, die einen Übergang von einem Zustand in einen anderen bewirken. Der Anstoß für das Ausführen von Aktionen wird in herkömmlichen Modellsystemen an Ereignisse oder das Eintreten von Bedingungen geknüpft.
Benötigen wir nun aber neben dem Ereignisbegriff einen weiteren Begriff, der eben den Dialogschritt, im Sinne einer Reaktion auf ein eingetretenes Ereignis oder eine eingetretene Bedingung beschreibt, etwa den Begriff Aktion? Oder kommen wir mit weniger Sprachmitteln genauso aus, um den Ablauf des Dialogs, die Steuerung, die Reihenfolge zu

beschreiben? Reicht es nicht aus, lediglich die Dialogschritte als Ereignisse darzustellen und sie selbst zu ordnen, wo es applikationslogisch oder dialogtechnisch gesehen erforderlich ist? Diese Frage hat sich teilweise in der bisherigen Darstellung unseres Grundmodells durch die Gleichsetzung des Begriffs Dialogschritt mit dem Begriff Dialogereignis beantwortet. Denn der Anstoß eines Dialogschritts aufgrund eines Ereignisses ist begrifflich bereits auf den Anstoß eines Dialogereignisses durch ein anderes Dialogereignis zurückgeführt. Der Dialog setzt sich also im Wesentlichen in unserem Modell ausschließlich aus einer Folge von Ereignissen, die in gleicher Weise auch als Aktionen gesehen werden können, zusammen.

Wird die Modellsprache dazu verwendet, um tatsächlich durchgeführte Dialoge lediglich zu protokollieren, sind die Ereignisse, die einen Dialogschritt auslösen, die aber selbst keine Dialogschritte sind, nicht unbedingt zur Darstellung nötig. Es kann auch ausreichen, nur die durchgeführten Dialogschritte ohne die auslösenden Ereignisse bzw. die auslösenden Ursachen darzustellen. Ebenso können die auslösenden Ereignisse fehlen, wenn nur prinzipiell (d.h. ohne Betrachtung der Bedingungen/Ereignisse) potentiell mögliche Dialogschritte aufgezeigt werden sollen.
Für den Benutzer wie für den Modellierer kann es sinnvoll sein, den potentiellen Übergang von einem Dialogschritt zum anderen im Überblick darzustellen, ohne dabei zunächst die Bedingungen, Ereignisse bzw. Aktionen, die diese Übergänge bewirken, aber außerhalb der Dialogschnittstelle liegen, zu zeigen. Um allerdings die Kontrolle des Dialogablaufs für die Implementierung des Systems zu spezifizieren, müssen dann die systemseitigen Ursachen für Dialogschritte, also Ereignisse bzw. Aktionen im System, die zu Ereignissen bzw. Aktionen an der Dialogschnittstelle führen, hinzugefügt werden.

Im Folgenden sollen einige Kriterien zur Beschreibung der zeitlichen Reihenfolge von Dialogereignissen herausgearbeitet werden. Wir werden sehen, dass es logisch und technisch begründete Reihenfolgen gibt, Reihenfolgen, die sich aus der Darstellung echtzeitlichen Geschehens ergeben und zufallsbedingte Reihenfolgen, bzw. Reihenfolgen, die nicht von vorneherein im Dialogmodell erfassbar sind.

4.2.1.1 Logisch begründete Dialogreihenfolgen

Wenn wir sehr weit vom konkret durchgeführten Dialog abstrahieren und nur die grundlegende Zielsetzung einer Dialoganwendung betrachten, also eine rein aufgabenlogische Sicht (etwa resultierend aus der Phase der Aufgabenanalyse) einnehmen, können bereits Anforderungen definiert werden, die die zeitliche Reihenfolge einzelner Dialogschritte bei der Dialogmodellierung bestimmen.

Logisch bedingte Abläufe von Teilaufgaben
Aus der Aufgabenanalyse heraus ergeben sich häufig Abhängigkeiten einzelner Teilaufgaben, die den Dialogablauf in eine bestimmte Reihenfolge zwingen.
Beispielsweise kann bei einem Dialog zur Auftragserfassung aufgrund der Definition der logischen Anforderungen an den Ablauf der Auftragsannahme festgelegt sein, dass die Festlegung eines Preises erst dann erfolgen soll, wenn die im Auftrag enthaltenen Leistungen definiert sind und auch erst dann, wenn der Kunde bekannt ist. Da der Preis durch das System automatisch berechnet werden soll, muss daher die Eingabe der Leistungsdaten des Auftrags und die Erfassung der Kundendaten durch den Mitarbeiter vor der Ausgabe des automatisch errechneten Preises liegen.

Ablaufoptimierende Reihenfolge
Gehen wir etwa davon aus, dass die Zielsetzung einer Dialoganwendung, wie die anderer Anwendungssysteme auch, dadurch definiert ist, möglichst effizient, kostenoptimierend bestimmte Aufgaben durchzuführen und Ergebnisse zu liefern. Dann ist entscheidend, dass die beteiligten Instanzen oder Teilkomponenten des Systems, die die Aufgaben zu erledigen haben, rechtzeitig mit der gewünschten oder benötigten Information versorgt werden, um ihre

Arbeit ohne Zeitverzögerung ausführen zu können. Bei einer Dialoganwendung haben wir als beteiligte Instanzen zum einen den Benutzer, zum anderen die Applikationskomponente, die über die Dialogkomponente bzw. Dialogschnittstelle mit Information zu versorgen sind.

Daraus ergibt sich als Ziel der Modellierung des Dialogs, den notwendigen Austausch der Information an der Dialogschnittstelle diesem Ziel der Ablaufoptimierung unterzuordnen. Wodurch wird dann der Zeitpunkt der einzelnen Dialogschritte bestimmt?

Um beispielsweise Verzögerungen im Ablauf zu vermeiden, muss spätestens dann die Information verfügbar sein, wenn ein Dialogpartner für die Durchführung seiner Aufgabe diese benötigt. Es ist dann primär nicht entscheidend, wann die Information von einem Kommunikationspartner an der Schnittstelle zur Verfügung gestellt wurde, sondern nur entscheidend, dass sie dann verfügbar ist, wenn die die Information abnehmende Komponente die Information braucht. Entscheidend ist also in diesem Fall nicht der Zeitpunkt der Eingabe, sondern der der notwendigen Abnahme der Information.

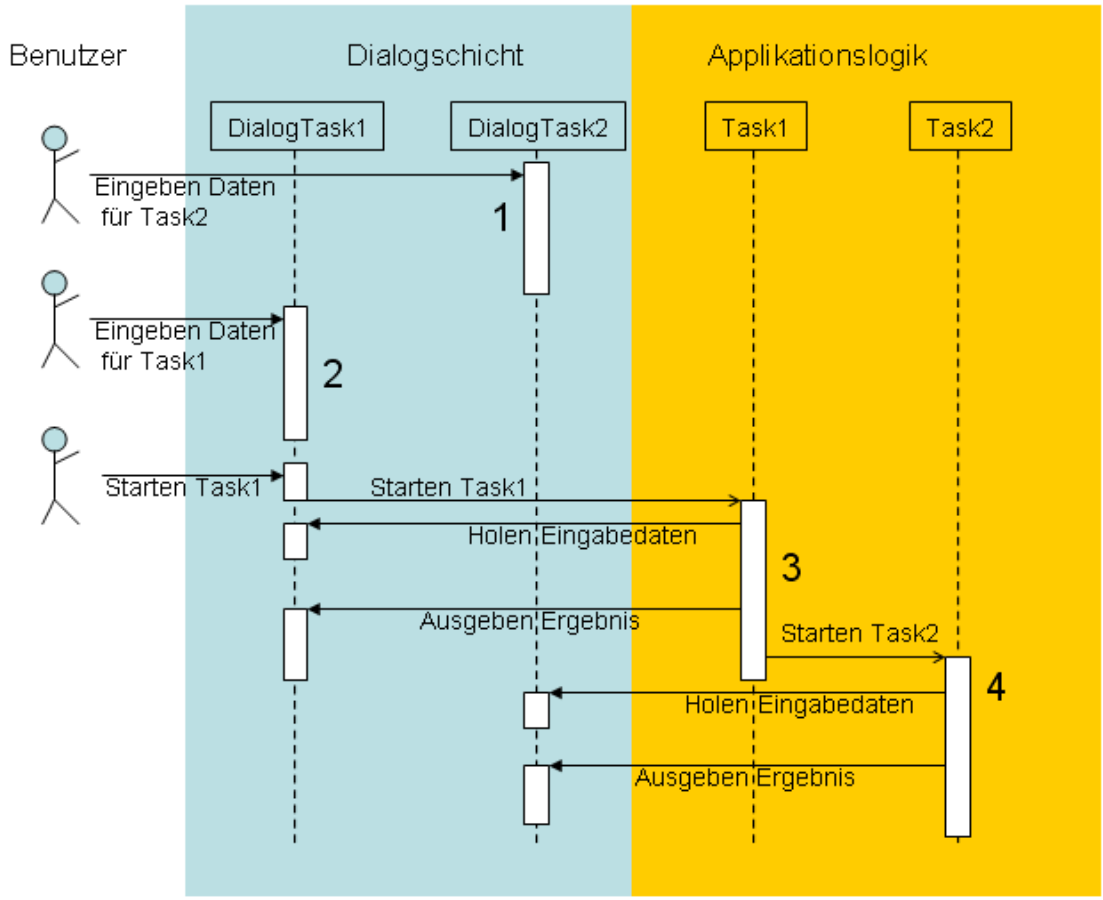


Abbildung 4-10: Entkoppelung von Eingabe der Daten und Durchführung der Aufgabe
Das Sequenzdiagramm zeigt ein Szenario, in dem mithilfe der Trennung von Dialog und Applikationslogik der Eingabedialog für die Durchführung der Aufgabe „Task2“ (Schritt 1) vor dem Eingabedialog für Aufgabe „Task1“ (Schritt 2) erfolgen kann, obwohl aufgabenlogisch Task2 (Schritt 4) von Task1 (Schritt 3) abhängt.

Nehmen wir zwei Arbeitsschritte A1 und A2 an, denen jeweils Eingabedialoge E(A1) und E(A2) sowie Ausgabedialoge A(A1) und A(A2) zugeordnet sind:
Beispiel1: A1 ist vor A2 logisch notwendig. Daraus folgt nicht automatisch, dass E(A1) vor E(A2) erfolgen muss. Die Eingabe für A1 kann auch nach der für A2 erfolgen, falls nicht

A(A1) für E(A2) benötigt wird. Entscheidend ist nur, dass bei Start der Aufgabe A1 die Eingabe zur Verfügung steht. Der Zeitpunkt der Informationseingabe zur Durchführung der Aufgabe kann aber vom Start der eigentlichen Durchführung der Aufgabe entkoppelt sein. Beispiel2: A1 dauert länger als A2. Dann ist A2 nicht notwendig logisch von A1 abhängig, aber dennoch ist es evtl. aus Performanzgründen sinnvoll, A1 eher zu starten als A2. Es macht dann auch Sinn, E(A1) vor E(A2) zu erledigen.

Wir wollen unser obiges Beispiel der Auftragserfassung ausweiten und erlauben, dass der Mitarbeiter, der einen Auftrag erfasst, auch alternativ zur automatischen Preisberechnung den Preis selbst festlegen kann, der Preis aber anschließend vom System auf Plausibilität geprüft werden soll. Dann ist zunächst die Reihenfolge der Eingabe von Preis-, Leistungs- und Kundendaten irrelevant. Und zwar ist sie irrelevant dann, wenn die Prüfung des Preises durch das System von der Eingabe der Preisdaten durch den Benutzer entkoppelt ist und das Lesen der Preisdaten durch die Applikationslogik erst zu einem späteren Zeitpunkt durchgeführt wird. Zum Zeitpunkt der Prüfung der Preisdaten müssen dann zusätzlich die Leistungsdaten und die Kundendaten des Auftrags vorliegen. Erst dann kann die entsprechende Buchung des Auftrags vorgenommen werden. Wir haben in diesem Beispiel also eine Trennung der Eingabenzeitpunkte von den Zeitpunkten des Lesens und der Verarbeitung der Information durch die Applikationslogik und somit die Befreiung von applikationslogischen Abhängigkeiten auf der Eingabeseite.

Fazit: Die Reihenfolge der einzelnen Dialogschritte kann durch logische, den Ablauf optimierende Gesichtspunkte vorgegeben sein. Dabei sind Abhängigkeiten auf der Seite der Informationsabnahme, also der Seite der Leseereignisse zum einen und Abhängigkeiten auf der Seite der Versorgung der Dialogschnittstelle, also der schreibenden Seite, zum anderen mitunter getrennt zu betrachten.

Echtzeitliche Abhängigkeiten
Eine besondere Art von Abhängigkeit in der Reihenfolge bzw. in den Zeitpunkten der Darstellung von Dialogereignissen liegt vor, wenn die Dialogschritte Ereignisse in Echtzeit darstellen sollen. Dies ist z.B. dann der Fall, wenn die absolute Zeit oder relative Zeitereignisse kontinuierlich oder in bestimmten Zeitintervallen anzuzeigen sind. Hier ist die Reihenfolge der Zeitereignisse untereinander zum einen durch die Echtzeit diktiert. Zum anderen finden die Ereignisse weitgehend unabhängig von anderen Ereignissen im System statt. Diese Eigenschaft echtzeitlicher Ereignisse legt somit nahe, sie in die Klasse der aus Sicht des Dialogsystems asynchronen Ereignisse (siehe unten) einzuordnen, deren Eintreten durch eine externe Ereignisquelle, nämlich eine extern vorhandene Uhr bestimmt wird.
Eine Entkoppelung der Darstellung von Echtzeitereignissen von ihrem realen Eintreten selbst macht in vielen Fällen wenig Sinn. D.h. wir müssen in diesen Fällen dafür sorgen, dass die Darstellung an der Dialogschnittstelle möglichst unmittelbar mit dem Eintreten des Ereignisses übereinstimmt. Besonders ist es bei Echtzeitdarstellung wenig sinnvoll, die Reihenfolge der Darstellung der Zeitereignisse zu vertauschen.
Auch im Dialogsystem synchron dargestellte Echtzeitereignisse (im Dialogsystem Echtzeitverhalten simulierende bzw. im Dialogsystem erzeugte und Echtzeit darstellende Ereignisse) können als Ereignisse betrachtet werden, deren Reihenfolge vorbestimmt ist. Mit anderen Worten: Echtzeitereignisse sind per se zeitkritisch und liegen in Zeitpunkt und Reihenfolge der Darstellung fest. Reale nicht simulierte Echtzeitereignisse sind aber weitgehend entkoppelt von anderen Ereignissen im Dialogsystem zu sehen und können deshalb als asynchrone extern gesteuerte Ereignisse (so paradox das auch klingt) betrachtet werden. Dies muss im Modell entsprechend ausgedrückt werden können.

Logische, nicht zeitlich gegebene Ordnungen
Neben den echt durch die Zeit gegebenen Reihenfolgen der Dialogschritte gibt es Reihenfolgen, die aufgrund irgendwelcher anderen vorgegebenen Ordnungskriterien definiert sind, die nicht durch die Zeit bestimmt sind. Im Dialog darzustellende Elemente oder Objekte können beispielsweise nach Maßzahlen wie Größe, Gewicht, Volumen, etc. „natürlich“ geordnet sein. Es kann nun eine Anforderung resultierend aus der Aufgabenanalyse sein, diese Ordnung im Dialog auszudrücken. Dann haben wir z.B. die Möglichkeit, die Elemente durch entsprechende Zuordnung räumlicher Koordinaten in einer bestimmten Reihenfolge anzuordnen. (Wir können die Reihenfolge der Elemente auch beispielsweise mittels Zuordnung einer bestimmten Farbe auf die durch die Frequenzen des Farbspektrums gegebene Ordnung abbilden.)
Eine andere Möglichkeit der Darstellung ist es eben auch, die gegebene Ordnung durch eine entsprechende zeitliche Reihenfolge auszudrücken.
Es handelt sich hier um eine Abbildung der gegebenen Ordnung der darzustellenden Elemente auf die durch die zeitliche Reihenfolge der Darstellung gegebene Ordnung. Aus logischer Sicht ist hier die zeitliche Anordnung nicht notwendigerweise erforderlich. Sie ist aber eine legitime Alternative der Darstellung der vorgegebenen Ordnung.

Abhängigkeiten im Sinne des Frage-Antwort-Dialogs
Eine sofort einleuchtende Abhängigkeit in der Reihenfolge von Dialogschritten liegt in einer ganzen Reihe von Dialogen begründet, die ein Charakteristikum der Frage-Antwort-Dialoge teilen. Es scheint auf den ersten Blick unmittelbar klar, dass zwischen Frage und Antwort eine zeitliche Abhängigkeit besteht, die die Frage vor die folgende Antwort stellt. Beim Frage-Antwort-Dialog verstanden als eine der bekannten Dialogarten stellt das System die Frage und der Benutzer antwortet. Wesentlich für unsere momentane Betrachtung ist, dass generell eine Abhängigkeit zwischen Frage und Antwort besteht, die im Normalfall eine zeitliche Abhängigkeit definiert. Wir finden diese Eigenschaft jedoch bei einer Reihe anderer Dialogarten, wie zum Beispiel dem Kommandodialog, bei dem der Benutzer den Auftrag erteilt, das System den Auftrag ggf. bestätigt, erfüllt und schließlich mit dem Ergebnis des Auftrags antwortet. Generell können wir vielleicht sagen, dass zunächst ein Problem von einem der Dialogpartner formuliert wird und der andere Partner mit der Problemlösung antwortet. Dies gilt z.B. insbesondere auch für Information-Retrieval-Systeme, bei denen der Benutzer eine Suchfrage stellt und das System Information zur Suchfrage ausgibt. Ähnliches gilt für ein Kalkulator-Programm, in das der Benutzer eine Rechenformel mit Parametern eingibt und das Programm mit dem errechneten Ergebnis antwortet, usw.
Wir können nun die Frage stellen, inwieweit diese Abhängigkeit zwischen Frage und Antwort eine logisch notwendige Abhängigkeit darstellt, die immer die zeitliche Reihenfolge zwischen Formulierung des Problems und Darstellung der Problemlösung erfordert: Denn wir können uns z.B. auch vorstellen, Problemstellung und Problemlösung gleichzeitig darzustellen. Gelingt uns dies, so kann sich der Benutzer dieses so gestalteten Systems die Fragestellung an das System sparen. Er kann die Problemlösung unmittelbar neben der formulierten Problemstellung ablesen. Denken wir z.B. an eine Tabelle mit den wichtigsten Hauptstädten der Erde und den dazugehörigen aktuellen Temperaturen. Rein „problemlogisch“ gesehen, im Sinne des logisch physikalischen Sachverhaltes der Zuordnung zwischen Ort und Temperatur haben wir keine Abhängigkeit zwischen dem Städtenamen und der Temperatur, die es erfordern würde, erst den Städtenamen und dann die Temperatur darzustellen. Es ist genauso denkbar, dass erst eine bestimmte Temperatur und danach die zugehörigen Städte sichtbar werden oder eben dass beides gleichzeitig in der Tabelle angeboten wird.
Die notwendige zeitliche Abhängigkeit ergibt sich allerdings doch in Abhängigkeit von den Bedürfnissen eines der beiden Dialogpartner in einer aktuellen Situation. Die so verstandene aufgabenlogische Abhängigkeit ergibt sich z.B. daraus, dass für den Benutzer der Ort als

Reiseziel festliegt, er jedoch nicht die aktuelle Temperatur kennt. Umgekehrt könnte der Benutzer sein Reiseziel erst aufgrund einer gewünschten Temperatur wählen wollen.
Die Reihenfolge der Darstellung von Information liegt in diesem Fall also darin begründet, dass ein Dialogpartner nur über einen Teil der von ihm benötigten Information zu einem bestimmten Zeitpunkt verfügt und sich über diese Teilinformation vom anderen Dialogpartner die ergänzende Information verschaffen möchte. Dies ist wiederum darauf zurückzuführen, dass ein Dialogpartner nur beschränkte Kapazität zur gleichzeitigen Aufnahme und Speicherung von Information besitzt und somit gezwungen ist, sich Information zeitlich hintereinander zu beschaffen, zu speichern und zu verarbeiten.
Fazit: Die Modellierung der Dialogeinheiten und die zeitliche Reihenfolge der Darstellung ergibt sich also aus einem bestimmten Informationsbedarf eines der Dialogpartner. Um möglichst effizient an die gewünschte Information zu kommen, ist es notwendig, dem anderen Dialogpartner sein Informationsbedürfnis mitzuteilen, da dieses wiederum dem Partner in der Regel nicht unmittelbar bekannt ist.

Logisch nebenläufige Ereignisse
In vielen Fällen ergibt eine Analyse der Aufgabenstellung, dass aus logischer Sicht eine bestimmte Reihenfolge in den Dialogen bzw. Teildialogen nicht vorgegeben ist. Die Dialoge können aus logischer Sicht in beliebiger Reihenfolge oder auch gleichzeitig abgearbeitet werden. Wir sprechen dann von nebenläufigen Dialogen bzw. Teildialogen.
Im Zusammenhang mit einer möglichen Trennung zwischen der Eingabe von Daten und der darauf folgenden Abarbeitung durch die Applikationskomponente haben wir oben bereits ein derartiges Beispiel angesprochen: die Reihenfolge der Dateneingabe kann z.B. irrelevant sein, wenn das Lesen der Daten durch die Applikationskomponente erst erfolgt, wenn alle Daten erfasst sind. Die Eingabe der Daten kann also parallel bzw. in beliebiger Reihenfolge durchgeführt werden, soweit nicht logische Abhängigkeiten dagegen sprechen.
Im obigen Beispiel der Auftragserfassung sind zunächst die beiden Teildialoge zur Erfassung der Kundendaten und zur Erfassung der Auftragspositionen voneinander unabhängig. Ebenso sind in vielen Fällen auch die einzelnen Auftragspositionen voneinander logisch unabhängig. Dasselbe gilt für die Kundendaten. Logisch gesehen könnten alle Daten gleichzeitig erfasst werden. In der Praxis bedeutet dies, dass zumindest die Daten in beliebiger Reihenfolge erfassbar sind.
Der Benutzer kann also bei der Erfassung beliebig von Feld zu Feld, von Teildialog zu Teildialog springen. Er ist dabei nicht gezwungen einen Teildialog abgeschlossen zu haben, wenn er einen anderen aufnimmt oder fortsetzt. Damit ergibt sich eine Vielzahl möglicher Übergänge von einem einzelnen atomaren Dialogereignis zu einem anderen.

In modernen Dialogsystemen ist die Realisierung einer derartigen Flexibilität des Dialogs mit entsprechender Freiheit der Dialogführung für den Benutzer selbstverständlich.
Den gegebenen Möglichkeiten muss auch im Modell entsprechend Rechnung getragen werden.
Selbst wenn logische Abhängigkeiten zwischen den einzelnen Teildialogen bestehen, kann es dennoch sinnvoll sein, dem Benutzer die maximale Freiheit in der Wahl seiner Dialogschrittfolge zu belassen. In obigem Beispiel der Auftragserfassung könnte z.B. eine automatische Einblendung des Ortes gewünscht sein, wenn der Benutzer die Postleitzahl im Kundendaten-Dialog eingibt. Dennoch sollte das System dem Benutzer erlauben, den Ort explizit vor der Postleitzahl einzugeben, für den Fall, dass dem Benutzer die Postleitzahl nicht bekannt ist.
Im Modell müssen wir nun sowohl diese logische Abhängigkeit zwischen Postleitzahl und Ort als auch die Möglichkeit des Einstiegs über die Eingabe des Ortsnamens ausdrücken können.

Logisch gleichzeitige Ereignisse
Der Vollständigkeit halber sei hier auch erwähnt, dass bei bestimmten Ereignissen gefordert sein kann, dass sie echt gleichzeitig eintreten müssen.

4.2.1.2 Technische und layoutbedingte Reihenfolgen

Wir haben oben einige Gründe für die logische Ordnung der Dialogschritte in zeitlicher Reihenfolge genannt und herausgearbeitet, dass in vielen Fällen eine Unabhängigkeit von Dialogschritten aus aufgabenlogischer Sicht vorliegt.
Häufig ergibt sich erst durch die Implementierungstechnik die notwendige Wahl einer bestimmten Reihenfolge.
Dies gilt insbesondere dann, wenn aufgrund eines Mangels an sonstigen Ausdrucksmitteln bzw. Ressourcen zur Darstellung die Ressource Zeit verwendet werden muss, um die Information zu übermitteln. Wir können dann von implementierungstechnisch bedingten und damit auch layoutbedingten Reihenfolgen sprechen.
Denken wir z.B. an den Dialog auf einem Handy-Display mit stark eingeschränkter Anzahl gleichzeitig darstellbarer Zeichen im Vergleich zu einem Dialog auf einem 20-Zoll-Bildschirm. Der Dialogdesigner (sei er ein Mensch oder ein Automat) ist zur Darstellung auf dem Handy-Display gezwungen, den inhaltlich gleichen Dialog in bestimmter zeitlicher Folge hintereinander anzuordnen, den er auf dem Großbildschirm gleichzeitig auf einmal darstellen kann. Durch die technischen Gegebenheiten sind also unterschiedliche Layouts nötig. Auf dem Handy-Display ergibt sich ein Layout, in dem z.B. Formularfeld für Formularfeld zeitlich hintereinender angeboten werden muss, während auf dem 20-Zoll-Bildschirm das Formular mit allen Feldern komplett auf einen Blick sichtbar ist.

Benutzerkonforme Reihenfolgen (Benutzergerechte Layoutgestaltung)
Wir haben oben bereits von zeitlichen Anordnungen aufgrund von Anforderungen an den Ablauf gesprochen, den wir aus der aufgabenlogischen Analyse ableiten. Dabei wurde bereits der Benutzer als Teil des Gesamtsystems angesprochen, der zur Erfüllung seiner Aufgaben rechtzeitig mit der nötigen Information versorgt werden muss. Die zeitliche Darbietung der Information für den Benutzer und die zeitliche Darstellung durch den Benutzer wird aber nicht nur durch rein aufgabenlogische optimierende Gesichtspunkte beeinflusst (Qualitätskriterium Aufgabenangemessenheit).
Hier spielen natürlich auch die Gewohnheiten und Erwartungen des Benutzers eine wichtige Rolle, die die Anordnung der Information und damit auch die zeitliche Reihenfolge beeinflussen (Qualitätskriterium Erwartungskonformität).
Ein einfaches Beispiel sei wiederum die Eingabe einer Adresse: Wenn wir auch oben festgestellt haben, dass es aus logischen Gesichtspunkten heraus gleichgültig ist, ob zunächst Postleitzahl oder Ort eingegeben werden, dürfte es in der Erwartung des Benutzers liegen, dass bei einer Adresserfassung die Postleitzahl zuerst und unmittelbar danach der Ortsname einzugeben ist. Damit ergibt sich eine entsprechende räumliche und auch zeitliche Anordnung der Erfassungsdialoge.
Auch die Fähigkeiten und Kenntnisse des Benutzers bestimmen die Aufteilung der Information in einzelnen Dialogschritten. Dies beginnt bereits z.B. mit den physiologischen Gegebenheiten des Menschen zur Wahrnehmung von Information. Die begrenzte Möglichkeit zur gleichzeitigen Aufnahme von Information durch den Menschen legt es nahe, die Information entsprechend zu portionieren und ggf. auch zeitlich hintereinander anzubieten (magical number seven etc. …).
Wir wollen hier nicht weiter auf physiologische Aspekte der Wahrnehmung und allgemein auf software-ergonomische Aspekte der Dialoggestaltung eingehen. Dazu sei auch auf Literatur zum Thema Software-Ergonomie, Erarbeitung von Benutzermodellen, etc. verwiesen [ZEIDL92][DIN88][ISO90].

Wir wollen bei obigem Beispiel der Auftragserfassung bleiben. Sicherlich ist es nicht optimal, den 20-Zoll-Bildschirm mit sämtlichen erdenklichen Informationen zu einem Auftrag zu überfrachten. Stattdessen ist es sinnvoll, die Information in logisch eingängige Einheiten aufzuteilen und z.B. in Folgebildschirmen oder Subfenstern anzubieten, die über entsprechende Verweise erreichbar sind. Wir wandeln damit wiederum Information, die auf einer Fläche gleichzeitig dargestellt wird, in Information, die zeitlich hintereinander angeboten wird, um; diesmal jedoch nicht, wie oben beim Beispiel des Handy-Displays aus technischer Notwendigkeit, sondern aus Aspekten der Ergonomie. Dazu sei vermerkt, dass wir bei der Aufteilung der Information nach logischen Gesichtspunkten vorgehen und nicht eine rein technische Aufteilung im Sinne eines Seitenumbruchs logisch unstrukturierter Information durchführen. Hier scheint eine Trennung zwischen Logik und Layoutgestaltung im Modell schwierig, bzw. unnötig, da wir eine Einheit von Form und Inhalt erreichen: die logische Informationseinheit entspricht jeweils der technischen Einheit eines Bildschirmfensters (Widgets).
Wir können festhalten, dass zeitliche Reihenfolgen in der Dialogführung, und damit also die Dialogdynamik im Allgemeinen sehr stark durch Belange des Benutzers, seine Erwartungen, seine Aufnahmefähigkeit,usw. geprägt ist und das Dialogmodell in der Lage sein sollte, entsprechende daraus resultierende Anforderungen an die Dynamik auszudrücken.

4.2.1.3 Asynchrone und synchrone Ereignisse

Asynchrone Ereignisse
Wie wir oben Echtzeitereignisse gerade als asynchrone Ereignisse eingeordnet haben, gibt es eine Reihe anderer Ereignisse, die wir als asynchron bezeichnen wollen. Generell verstehen wir unter asynchronen Ereignissen diejenigen Ereignisse, deren Zeitpunkt der Darstellung aus Sicht der Steuerung der Dialogkomponente des Systems nicht von vorneherein vorherbestimmt werden kann.
Besitzt das Ereignis ein definiertes Vorgängerereignis, so tritt es zeitlich versetzt zu einem unbestimmten Zeitpunkt und nicht unbedingt unmittelbar danach auf. Das asynchrone Ereignis kann allerdings auch eines oder mehrere Nachfolgeereignisse besitzen. (Die Existenz von Vorgänger- und Nachfolgerereignissen legt es nahe, ggf. von teilsynchronisierten Ereignissen zu sprechen, da in gewissem Sinne eine zeitliche Einordnung (Synchronisation) des Ereignisses doch erfolgt. Siehe Diskussion des Begriffs asynchron im Anhang)
Das Eintreten eines asynchronen Ereignisses und damit auch seine Darstellung an der Dialogschnittstelle wird im Allgemeinen extern, also von einer Komponente oder Instanz außerhalb der Dialogkomponente gesteuert. In der Regel können wir nur einen mehr oder weniger langen Zeitraum bzw. Zustand im Dialogsystem angeben, in dem wir ein asynchrones Ereignis erwarten. Dieser Zeitraum ist durch Vorgänger- und Nachfolgeereignisse eingegrenzt.

Asynchrone Ausgabeereignisse
Ausgabeereignisse können also, wenn das in Dialogsystemen auch nicht die überwiegende Mehrheit sein mag, asynchron gesteuert sein. Asynchrone Ausgabeereignisse gehen ohne Beschränkung der Allgemeinheit von der Applikationslogik-Komponente des Systems aus.

Asynchrones Lesen durch Applikationskomponente
Leseereignisse, durchgeführt durch die Applikationslogik, sind ebenfalls Ereignisse, die nicht von der Dialogkomponente des Systems, sondern von der Applikationslogik ausgelöst werden können. Sie können ebenfalls als asynchron betrachtet werden, wenn der Zeitpunkt des Lesens nicht unmittelbar von der Dialogkomponente bestimmt ist.

Asynchrone Benutzereingaben
Benutzereingaben sind als asynchrone Ereignisse einzustufen. Wir gehen davon aus, dass die Eingabe des Benutzers prinzipiell nicht vom Dialogsystem bestimmt werden kann. Es liegt grundsätzlich im Ermessen des Benutzers, ob und wann er eine Eingabe vornimmt. Das System kann eine Benutzereingabe nur insofern steuern, als es eine Eingabeschnittstelle zur Verfügung stellt, an der die Eingabe dann vom Benutzer getätigt werden kann. Das System kann den Benutzer nur insoweit zu einer Eingabe zwingen, als es nur eine begrenzte Möglichkeit von Eingaben vorsieht und der Benutzer diese begrenzte Anzahl von Möglichkeiten nutzen muss, wenn er den Dialog fortsetzen möchte. Eine derartige Situation liegt im so genannten modalen Dialog vor.
Eingaben des Benutzers sind also Ereignisse die nicht unbedingt und unwillkürlich einem anderen Dialogereignis nachfolgen. Es gibt keinen unausweichlichen, deterministischen Zusammenhang zwischen einem Ereignis an der Dialogschnittstelle und einer nachfolgenden Benutzereingabe.

Asynchrone Wahrnehmung durch Benutzer
Das Ereignis der Wahrnehmung eines Ereignisses an der Dialogschnittstelle ist ebenfalls ein asynchrones Ereignis. Wir können grundsätzlich nicht erwarten, dass die Ausgabe bzw. Darstellung einer Information durch das System auch synchron, also zeitlich unmittelbar danach vom Benutzer zur Kenntnis genommen wird.

Synchrone Ereignisse
Ausgabeereignisse sind häufig unmittelbar synchron aufgrund eines Vorgängerereignisses ausgelöste Ereignisse; d.h. es kann kein anderes Ereignis zwischen Vorgängerereignis und dem Ausgabeereignis erfolgen oder es ist ein Ereignis zumindest eine notwendige Folge eines anderen. Die Notwendigkeit liegt im Allgemeinen in einer Programmvorschrift begründet, die eine unmittelbare Reaktion auf ein bestimmtes Vorgängerereignis definiert.
Ebenso sind Leseereignisse durch das System häufig synchrone Ereignisse.

Im vorangehenden Abschnitt haben wir Kriterien erarbeitet, die die Reihenfolge von Dialogschritten begründen. Dabei haben wir festgestellt, dass bestimmte Übergänge von Dialogereignis zu Dialogereignis logisch aus der Aufgabenstellung begründbar sind, dass Übergänge teilweise aus einer mehr oder weniger logischen oder technischen Gliederung der Informationseinheiten resultieren und dass es Übergänge gibt, die im Sinne der vorgegebenen Logik willkürlich erscheinen. Sie spiegeln zum einen die Freiheit des Benutzers wider, von einem logischen Dialogkontext in einen anderen zu wechseln, zum anderen tragen sie der Tatsache Rechnung, dass unvorhergesehene bzw. asynchrone Ereignisse im System bzw. der Applikationslogik-Komponente im System auftreten. Dieser Wechsel des Kontexts kommt dadurch zustande, dass während des Dialogablaufs nebenläufige Dialogfolgen quasiparallel oder auch echt parallel bedient und aber wiederum auf eine einzige Sequenz von Dialogereignissen abgebildet werden.

4.2.2 Reduktion auf logische Stellenübergänge

Bisher haben wir im Modell alle diese unterschiedlich begründbaren Übergänge ohne Unterschied dargestellt. Wir erreichen nun eine drastische Reduktion in der Darstellung der Übergänge, wenn wir zwischen den Übergängen zwischen nebenläufigen Dialogen und den Übergängen innerhalb logisch begründbar zusammenhängender Dialoge unterscheiden.

Die Reduktion wird dadurch erreicht, dass wir dann nur mehr die logisch begründbaren bzw. logisch notwendigen Übergänge explizit darstellen und auf die Darstellung der Übergänge zwischen nebenläufigen Dialogen (mit Ausnahmen) grundsätzlich verzichten.
Dialoge, zwischen denen keine expliziten Übergänge definiert sind, gelten somit als nebenläufig. Als Ersatz für die nunmehr fehlende explizite Auszeichnung der Übergänge fordern wir aber, dass Dialogschritte in nebenläufigen Dialogen parallel oder quasiparallel durchführbar sind. Dies bedeutet insbesondere, dass jederzeit von einem Dialog in den anderen nebenläufigen Dialog gewechselt werden kann.

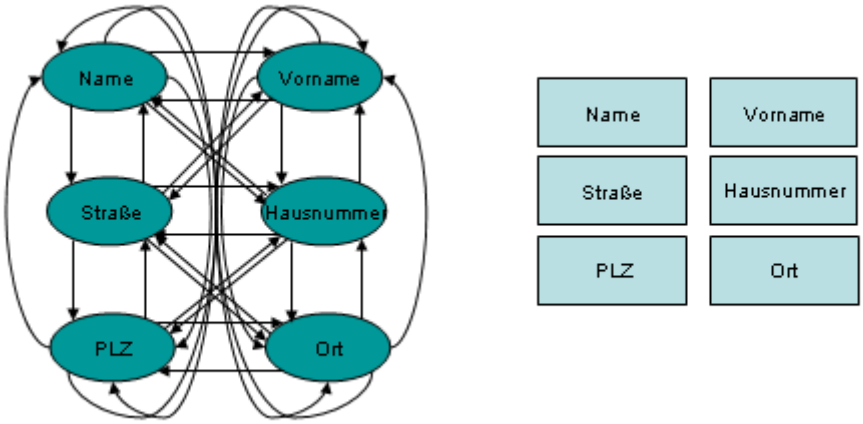


Abbildung 4-11: Dialog zur Adresserfassung in Modell mit expliziter Darstellung der Übergänge und Modelldarstellung mit nebenläufigen Dialogen.
Wir führen im Modell eine Darstellung ein, in der die vielen möglichen Übergänge zwischen nebenläufigen Dialogen nicht mehr angegeben werden.

Abbildung 4-11 zeigt am Beispiel eines Dialogs zur Adresserfassung ein Modell dieses Dialogs mit expliziter Darstellung aller möglichen Übergänge (alte Modelldarstellung) und ein Modell, in dem die Teildialoge zur Erfassung des Namens, Vornamens usw. als unabhängige nebenläufige Dialoge ohne notwendige logische Übergänge aufgefasst werden (neue Darstellung). Kein Teildialog setzt in der Reihenfolge der Abarbeitung einen anderen Teildialog voraus. Implizit wird per Definition dieser Darstellung angenommen, dass jedoch von jedem Teildialog in einen beliebigen anderen jederzeit gewechselt werden kann. Bei Start des Dialogs zur Adresserfassung stehen per Definition alle Teildialoge zur Verfügung, d.h. es kann mit jedem der Teildialoge begonnen werden. Theoretisch können alle Teildialoge echt zeitgleich bedient werden, wenn dies die tatsächliche Implementierung erlaubt. Dies setzt zumindest voraus, dass die Teildialoge in der Implementierung durch unterschiedliche parallel bedienbare Ressourcen realisiert werden. Bei einer üblichen Implementierung der Dialogschnittstelle über Bildschirm, Tastatur und Maus ist jedoch in der Regel mindestens die Eingabe über Tastatur sequenziell. Jedoch ist es auch hier implementierungstechnisch möglich, die Eingabe quasiparallel durchzuführen und jederzeit von einem Teildialog in den anderen zu wechseln. Unsere logische Darstellung im Modell entspricht also in diesem Fall auch einer üblichen Implementierungstechnik bei grafischen Oberflächen.

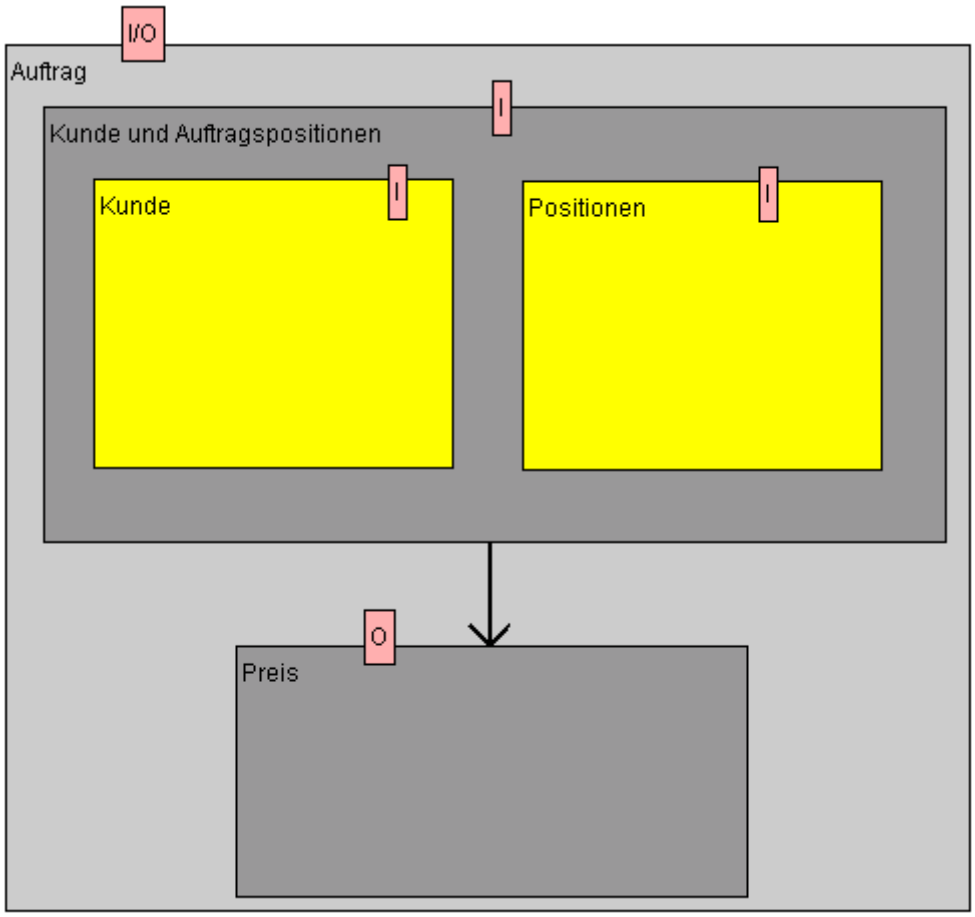


Abbildung 4-12: Beispieldialog Auftragserfassung

Abbildung 4-12 zeigt am Beispiel der Auftragserfassung einen Dialog, der im Wesentlichen den eben vorgestellten Dialog als Teildialog enthält. Insgesamt besteht der Dialog aus drei Teildialogen, nämlich der Erfassung der Kundendaten (entspricht obigem Dialog Adresserfassung), einem Dialog zur Erfassung der Auftragspositionen und einem dritten zur Darstellung des Preises. Dabei sei aus der logischen Analyse der Auftragserfassung hervorgegangen, dass die Ermittlung des Preises automatisch durch das System erfolgen soll und vor der Ermittlung des Preises die Kundendaten und die Auftragspositionen bekannt sein müssen.
Somit haben wir eine logische Abhängigkeit des Preisdialogs von den beiden anderen Dialogen, die zueinander nebenläufig gestaltet werden können. Die Erfassung der Kundendaten kann logisch unabhängig von der Erfassung der Auftragspositionen erfolgen. Dies können wir nun auf einer relativ abstrakten Hierarchie-Ebene zunächst dadurch ausdrücken, dass wir die beiden Teildialoge Erfassen Kundendaten und Erfassen Auftragspositionen als nebenläufige Dialoge darstellen und in einem übergeordneten Dialog „Erfassen Kundendaten und Leistungen“ zusammenfassen, dem der Preisdialog folgt.
Der Pfeil in Abbildung 4-12 drückt also die einzige definierte Abhängigkeit zwischen den Dialogen aus.

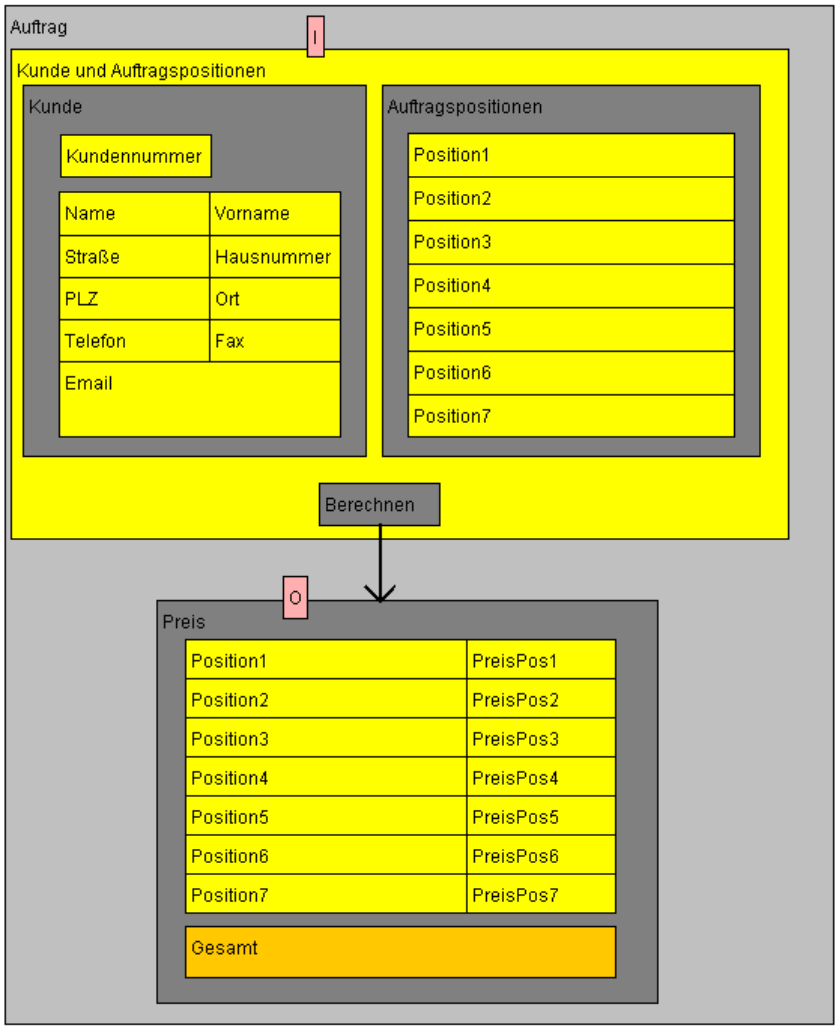


Abbildung 4-13: Auftragserfassung mit primitiven Teildialogen und Übergang zur Preisberechnung gesteuert durch Benutzereingabe.

Für eine vollständige detaillierte Definition bis zur Ebene der primitiven Teildialoge (Abbildung 4-13) ist es noch notwendig, gemäß der im Zusammenhang mit der Einführung hierarchischer Dialoge gewählten Darstellung (Abschnitt 3.0.3.0) die Anschlussereignisse zu definieren, die den Übergang zum Preisdialog ermöglichen. D.h. wir müssen im Einzelnen definieren, welche Ereignisse in den jeweiligen Teildialogen einen Übergang zum Preisdialog erlauben, welche Ereignisse bzw. Stellen also innere Anschlussstellen zum Pfeil auf der obersten Hierarchieebene darstellen.
Gemäß Modell in Abbildung 4-12 erfolgt der Übergang zum Preisdialog dann, wenn der Benutzer den Übergang wünscht, was er durch Eingabe von „Berechnen“ ausdrückt. Im konkret realisierten Dialog kann dies z.B. das Drücken eines PushButtons sein oder aber auch äquivalent dazu etwa in einem Kommandodialog die Eingabe eines entsprechenden Kommandos. Die hier gewählte Modellvariante erlaubt zu jedem Zeitpunkt nach Eröffnen des Dialogs zur Auftragserfassung die Eingabe des Kommandos „Berechnen“. Ebenso erlaubt es die völlig nebenläufige Eingabe aller anderen Daten im Kundendialog und gleichzeitig im Dialog zur Erfassung der Auftragspositionen. Es handelt sich also bei dieser Wahl des Modells um eine relativ freizügige Gestaltung des Dialogs. Aus aufgabenlogischer Sicht kann der Übergang zum Preisdialog allerdings erst tatsächlich erfolgen, wenn alle notwendigen Daten zum Kunden und mindestens eine Auftragspositionen vorliegen. Die Erfüllung der logisch geforderten Bedingung für den Übergang zum Preisdialog kann sowohl durch ein Ereignis aus dem Dialog „Erfassen Kundendaten“ als auch aus dem Dialog „Erfassen Auftragsposition“ erfolgen. Denn die Vollständigkeit der zu erfassenden Daten kann grundsätzlich durch Eingaben aus beiden nebenläufigen Teildialogen erreicht werden.
Die Prüfung der Vollständigkeit der Daten geschieht nun mittels einer entsprechend zu definierenden Bedingung (Condition), die wir prüfen, nachdem der Benutzer den Wunsch zur Preisberechnung geäußert hat. Erst wenn alle Daten vorliegen, erfolgt gemäß der formulierten Bedingung der Übergang zum Preisdialog, d.h. die Ausgabe der Preisdaten, tatsächlich (siehe Abbildung 4-15).

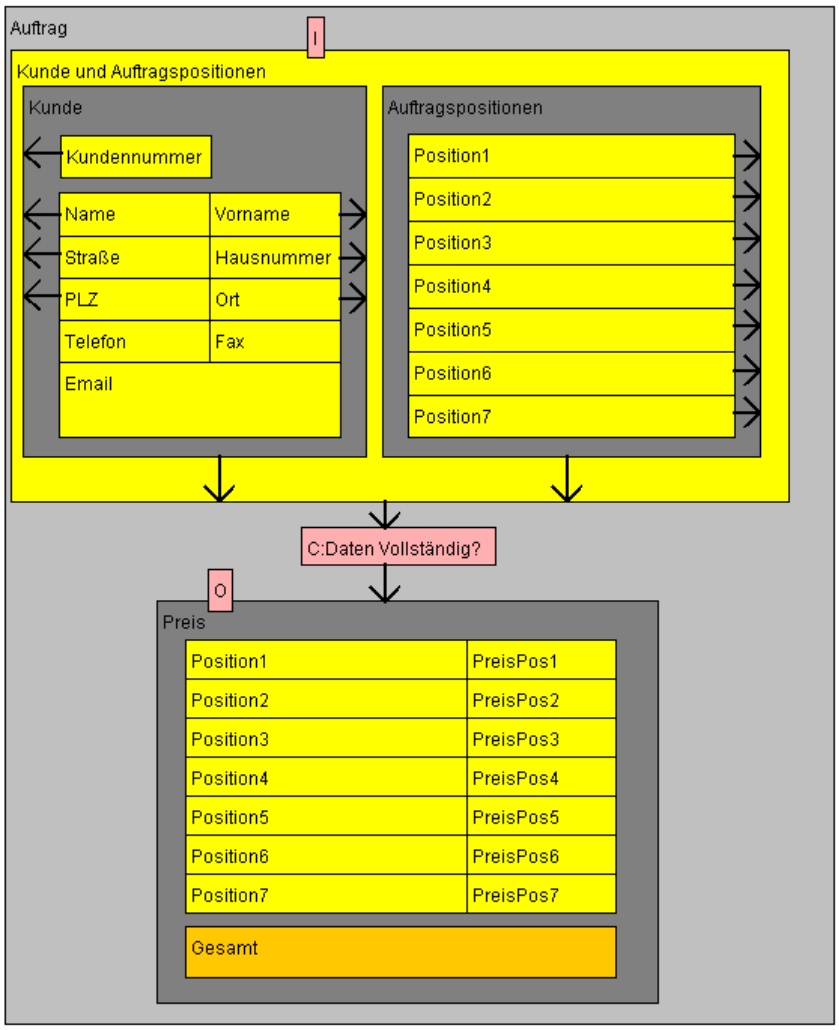


Abbildung 4-14
Auftragserfassung mit Condition und automatischem Übergang zu Preisberechnung

In einer anderen Variante des Modells (siehe Abbildung 4-14) soll die Berechnung der Preise automatisch erfolgen, ohne dass der Benutzer dazu ein Kommando erteilt. Nun kann der Übergang zur Ausgabe der Preise nach Eingabeereignissen in fast allen Teildialogen des Kundendialogs und des Dialogs zur Erfassung der Auftragspositionen erfolgen. Lediglich die Erfassung von Telefon, Fax und Email sind nicht obligatorisch. In dieser Modellvariante muss nun nach jedem Ereignis mittels Condition die Vollständigkeit der Daten überprüft werden, um dann ggf. die Preisberechnung zu triggern.
(Vorschlag: Um auch die hier etwas aufwändige Definitionsarbeit aller Anschlussereignisse zu sparen, können wir vereinbaren, dass ein Übergang aus jedem der Ereignisse in den Teildialogen erfolgen kann, dass also defaultmäßig alle Ereignisse des Dialogs den Übergang triggern, es sei denn, es werden explizit Anschlussereignisse im jeweiligen Teildialog angegeben.)

Abbildung 4-15 (nächste Seite) zeigt ein erweitertes Beispiel des Auftragsdialoges im Modell unter Verwendung einer Condition und zweier Übertragungsfunktionen. Dabei ist die Variante mit expliziter Ansteuerung der Preisberechnung durch den Benutzer gewählt.
Je nach Ergebnis der Auswertung der Condition erfolgt entweder die Preisberechnung oder ein einfacher Dialog mit Ausgabe einer Fehlermeldung. In einer komfortableren Ausgestaltung ist der Fehlerdialog wiederum ein zusammengesetzter Dialog, der präzise auf den Fehler in den erforderlichen Daten hinweist und zur Korrektur bzw. Vervollständigung der Daten auffordert.
In exemplarischer Form (siehe auch Abschnitt Exemplarische Darstellung und Variation) sind zwei Übertragungsfunktionen abgebildet, die die jeweilig auszugebenden Werte im Preisdialog für die Auftragsposition 3 bestimmen. Die eine überträgt einfach den identischen Wert der Auftragsposition 3 aus dem vorangegangenen Dialog Auftragspositionen. Die andere berechnet den Preis der Position in Abhängigkeit von den Angaben der Auftragsposition und den Kundendaten. Ausgelöst werden die Funktionen, falls die Conditionauswertung TRUE liefert.

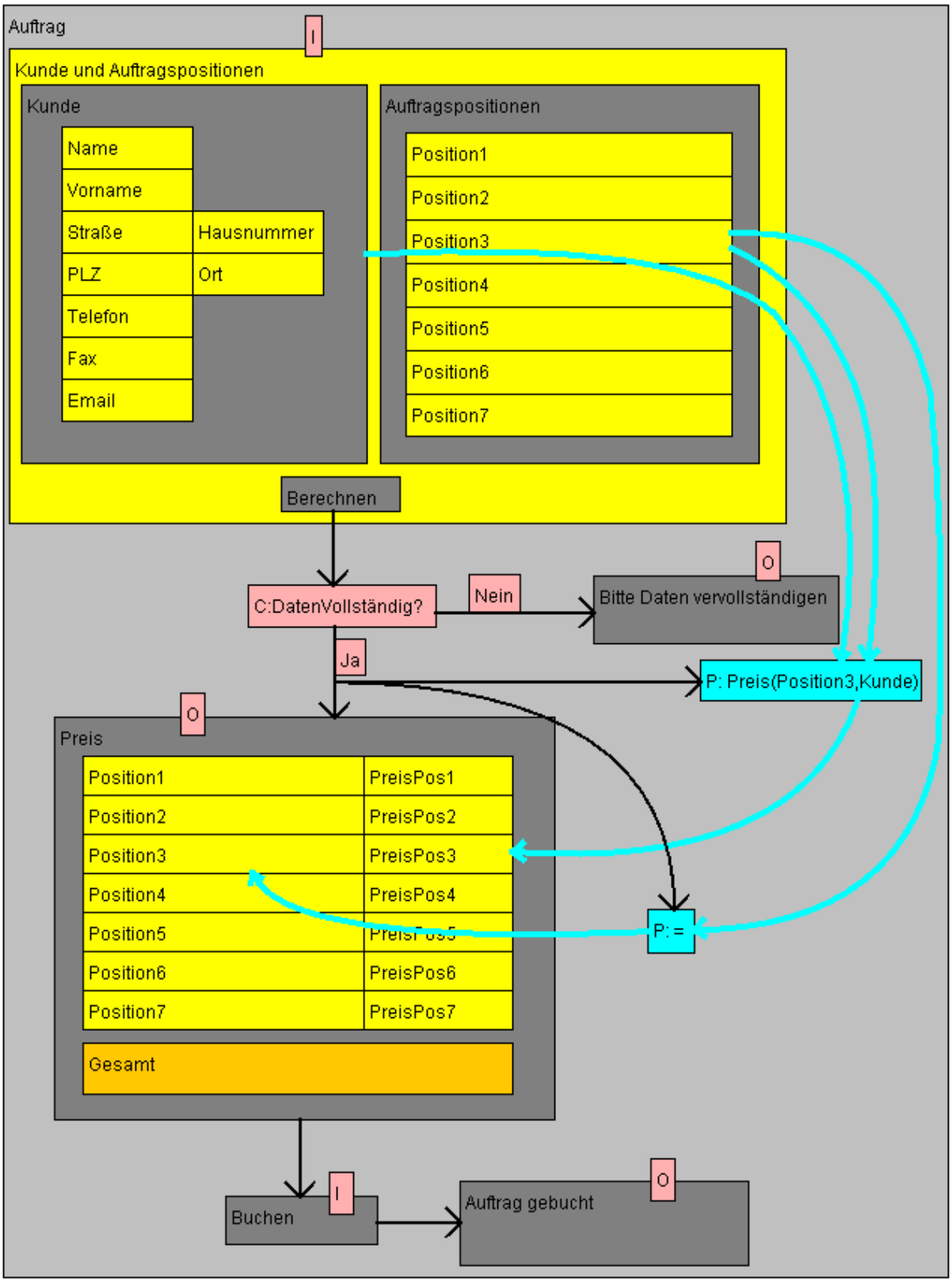


Abbildung 4-15 Auftragserfassung mit Condition und Propagationsfunktionen

Wir gehen davon aus, dass bei Umsetzung des Modells zur Laufzeit die Eingabewerte der Funktionen zur Verfügung stehen und die Ergebnisse der Funktionsaufrufe an den entsprechenden Ausgabestellen gesetzt werden.

Im obigen Beispiel haben wir nebenläufige Teildialoge, die in sich wiederum keine logischen Abhängigkeiten ihrer Subdialoge enthalten. Im Folgenden soll noch einmal dieser Fall an einem schematischen Beispiel zweier Teildialoge mit definierten inneren Abhängigkeiten ihrer Teildialoge betrachtet werden:

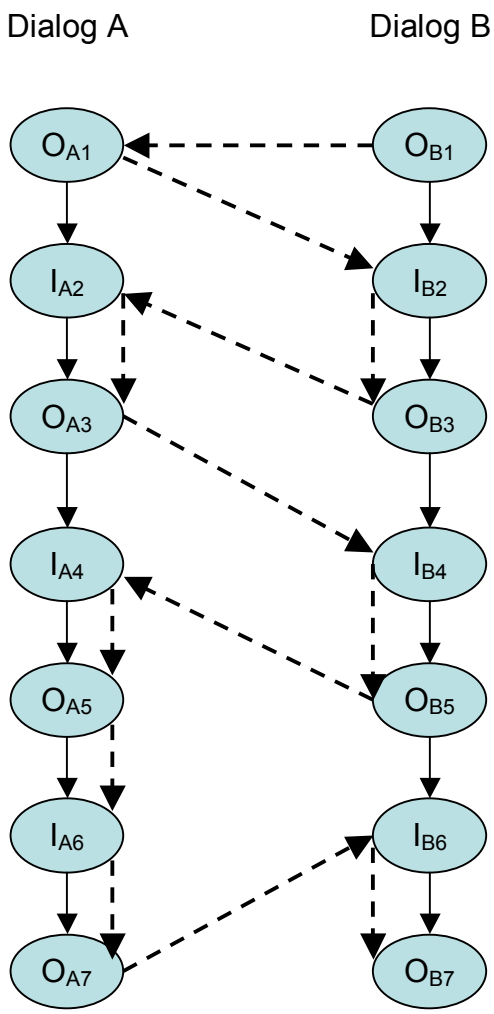


Abbildung 4-16 zeigt zwei nebenläufige Dialoge A und B, die in sich jeweils eine Folge von Ausgaben (O) und Eingaben (I) enthalten. Die Folgen drücken eine logische Abhängigkeit in dem Sinne aus, dass erst ein Ereignis aus der definierten Ereignismenge der Vorgängerstelle den Dialog, also das Eintreten eines Ereignisses an der Nachfolgerstelle ermöglicht. Diese Abhängigkeiten des Dialogablaufs sind in der Abbildung durch Pfeile mit durchgezogenen Linien zwischen den Stellen ausgedrückt.
Die Pfeile mit gestrichelter Linie zeigen eine der möglichen tatsächlichen Ablauffolgen von Ereignissen. Diese Folge ergibt sich dadurch, dass der Benutzer zwischen den beiden nebenläufigen Dialogen in seiner Bedienung der Eingabestellen abwechselt und den Dialog quasiparallel abwickelt. Beide Dialoge werden abwechselnd vorangetrieben. Die Abbildung zeigt auch, dass ein Übergang von A nach B (oder umgekehrt) nur immer an die jeweils „aktuelle“ Stelle des anderen Dialogs erfolgen kann. Mit aktueller Stelle ist die Nachfolgestelle der Stelle gemeint, an der das letzte Ereignis im jeweiligen (Teil-)Dialog eingetreten war. Diese aktuelle Stelle wollen wir auch als verfügbare, ereignisbereite oder geöffnete Stelle bezeichnen.

Abbildung 4-16: Logische und willkürliche Übergänge, schematische Darstellung zweier nebenläufiger Dialoge

Wir sehen also, dass der Pfeil mit durchgezogener Linie nicht die unmittelbare Folge der Ereignisse bezüglich aller Ereignisse im Gesamtdialog ausdrückt. D.h. unmittelbares Nachfolgeereignis ist nicht unbedingt ein Ereignis an der durch den Pfeil ausgedrückten Nachfolgerstelle.
Die durchgezogenen Pfeile zeigen jedoch genau die Folge im Teildialog an, die sich trivialerweise ergibt, wenn man alle Ereignisse des jeweiligen anderen Teildialogs aus der Gesamtfolge streicht.
Dies gilt unter der Annahme, dass der Pfeil so als gerichtete Kante eines Grafen interpretiert wird, wie wir dies auch in der bisher vorangegangenen Entwicklung unserer Modelldarstellung getan haben: durch die Verfolgung der Kanten vom Startknoten aus ermitteln wir eine Folge von Ereignissen, die einen möglichen zeitlichen Ablauf des Dialogs widerspiegelt. Insbesondere kann eine Stelle nur dann wiederholt erreicht werden, wenn ein Zyklus im Grafen existiert, in dem die Stelle als Knoten vertreten ist. Ist die Stelle in keinem Zyklus vertreten, kann sie höchstens einmal erreicht werden. Die Stelle ist dann auch nur höchstens einmal geöffnet. Handelt es sich beispielsweise um eine primitive Stelle, ereignet sich an der Stelle dann nur höchstens einmal ein Ereignis, die Stelle ist danach nie mehr geöffnet bzw. für den gesamten restlichen Dialogablauf „geschlossen“.

Diese Art der Darstellung führt dazu, dass wir von jeder Nachfolgerstelle oder von der Stelle selbst aus eine gerichtete Kante zur Stelle führen müssen, wenn wir eine Wiederholung eines Ereignisses an besagter Stelle unmittelbar nach einem Ereignis an der Nachfolgerstelle oder unmittelbar nach einem Ereignis an der Stelle selbst ermöglichen (wir sagen auch eröffnen) wollen.
Wir sparen nun wiederum eine Vielzahl sonst nötiger expliziter Definitionen von Kanten, sprich Definitionen von Übergängen, wenn wir auch annehmen, dass eine Stelle solange verfügbar oder geöffnet ist, solange sie nicht explizit geschlossen wird. D.h. das Eintreten eines Ereignisses an einer (ohne Beschränkung der Allgemeinheit primitiven) Stelle führt nicht notwendig zum Schließen dieser Stelle. Ereignisse können an der Stelle solange wiederholt werden, bis die Stelle explizit geschlossen wird.
Dies bedeutet, dass zwei Stellen A und B mit Folgebeziehung (B folgt auf A) zu nebenläufigen Stellen werden, sobald die nachfolgende Stelle B einmal z.B. durch ein Ereignis in A geöffnet wurde.

In unserem Beispiel der Auftragserfassung bedeutet diese Interpretation, dass die Teildialoge zum Erfassen der Kundendaten und der Auftragspositionen auch geöffnet bleiben, nachdem der Preis ermittelt und ausgegeben wurde.

5 Ereignisstellenmodell

Unter Einbeziehung der letzten Überlegungen gelangen wir ausgehend von unserem Grundmodell (Kapitel 3) zu einer Modelldarstellung des Dialogs, die wir als Ereignisstellenmodell oder Ereignisstellendarstellung bezeichnen wollen. Deren graphische Form nennen wir Ereignisstellendiagramm.
An die Stelle der expliziten Darstellung aller möglichen Übergänge von einem Ereignis zum anderen mit einer expliziten Darstellung aller Ereignisse an eigenen Modellstellen in Form konkreter alternativer Ereignisfolgen tritt im Wesentlichen eine Darstellung der Stellen mit ihren Ereignissen und eine Darstellung zweier Relationen, nämlich der so genannten Öffne- und der so genannten Schließerelation zwischen Stellen. Das Öffnen und Schließen einer Stelle kann dabei jeweils als Metaereignis betrachtet werden, das durch ein konkretes Ereignis ausgelöst wird und das jeweils eine Menge von Ereignissen einer Stelle eröffnet bzw. sperrt. Entsprechend der in den vorangegangenen Abschnitten gemachten Überlegungen werden außerdem in das Modell Prädikate (Conditions) und Übertragungsfunktionen mit aufgenommen.
Nach einer formalen Darstellung des Ereignisstellenmodells wird im Folgenden der Zusammenhang mit Zustandsautomaten diskutiert. Wir werden feststellen, dass jeder deterministische endliche Automat in eine entsprechende Ereignisstellendarstellung abgebildet werden kann und umgekehrt unter gewissen Bedingungen jede primitive Ereignisstellendarstellung einem deterministischen endlichen Automaten entspricht.

5.0 Formale Darstellung

Ein primitives Ereignisstellenmodell ESM ist ein Tupel

$$(E, T, W, S, S_0, RO, RC, C, P, \tau, \sigma, \varepsilon_{RO}, \varepsilon_{RC}, \omega e, \omega start, \kappa_{RO}, \kappa_{RC}, \pi_{RO})$$

mit

E: Menge von Ereignissen, wobei jedes Ereignis ein Tupel aus Wert, Stelle und Typ:

$$e = (w, s, t) \in E \subseteq W \otimes S \otimes T$$

T: Menge von Ereignistypen $T = \{ I_B, O_S, M_S \}$
Im Gegensatz zum Grundmodell sehen wir im ESM vereinfachend keine Leseereignisse explizit vor. Der Vorgang des Lesens ist implizit über eine zustandsspezifische Wertezuordnung zu Stellen (siehe unten WSHist, Menge der Wertezuordnungen) im Modell enthalten. Es wird angenommen, dass Eingabeereignisse (entspricht Typ I_B) vom System wahrgenommen, also gelesen werden können, bis die dem Ereignis zugeordnete Stelle geschlossen wird. Ebenso wird gefordert, dass Ausgabeereignisse und Schreibereignisse an Hilfsstellen (Typen O_S bzw. M_S) von Benutzer bzw. System gelesen werden können, bis die zugeordnete Stelle geschlossen wird.

W: Menge von Werten

S: Menge von Ereignisstellen

S_0: Menge von Startstellen $S_0 \subseteq S$

τ: Zuordnungsfunktion von Ereignistypen $\tau \mid E \rightarrow T$

σ : Funktion zur Zuordnung von Stellen zu Ereignissen: $\sigma \mid E \rightarrow S$

Wir fordern $\tau(e_1) = M_S \wedge (\tau(e_2) = O_S \vee \tau(e_2) = I_B) \Rightarrow \sigma(e_1) \neq \sigma(e_2)$.
d.h. Stellen sind entweder „Hilfsstellen“ oder „Dialogstellen“.

ωe: Funktion zur Zuordnung eines Wertes zu Ereignis $\omega e \mid E \rightarrow W$

ω_{Start}: partielle Funktion zur Zuordnung eines Wertes zu einer Ereignisstelle zum Startzeitpunkt:

$$\omega_{Start} \mid S \rightarrow W,$$

RO: Öffne-Relation (O für Open) $RO \subseteq (S \otimes S)$

RC: Schließe-Relation (C für Close) $RC \subseteq (S \otimes S)$

ε_{RO}: Zuordnung von Ereignissen, die Öffnen auslösen

$$\varepsilon_{RO} \mid (S \otimes S) \rightarrow \text{Potenzmenge}(E)$$

mit $\varepsilon_{RO}(s,s') \subseteq (\sigma^{-1}(s) \cup \varnothing) \; \forall \; (s,s') \in S \otimes S$,
$\varepsilon_{RO}(s,s') \neq \varnothing$, falls $(s,s') \in RO$, $\varepsilon_{RO}(s,s') = \varnothing$ sonst

ε_{RC}: Zuordnung von Ereignissen, die Schließen auslösen

$$\varepsilon_{RC} \mid S \otimes S \rightarrow \text{Potenzmenge}(E)$$

mit $\varepsilon_{RC}(s,s') \subseteq (\sigma^{-1}(s) \cup \varnothing) \; \forall \; (s,s') \in (S \otimes S)$,
$\varepsilon_{RC}(s,s') \neq \varnothing$, falls $(s,s') \in RC$, $\varepsilon_{RC}(s,s') = \varnothing$ sonst

C: Menge der Prädikate (Conditions) c, als Funktionen der Form

$$c : WS_c \rightarrow \{ \text{TRUE, FALSE} \},$$

wobei WS_c Menge aller Wertezuordnungen auf einer definierten Stellenmenge $S_c \subseteq S$, also:

$$WS_c = \{ \omega sc, \omega sc \text{ ist Funktion } \omega sc \mid S_c \rightarrow W \}.$$

κ_{RO}: Zuordnung von Condition zu Öffne-Relation

$$\kappa_{RO} \mid RO \rightarrow (C \cup \varnothing),$$

die für die Entscheidung des Öffnens von Stelle s’ bei Ereignis an Stelle s mit $(s,s') \in RO$ herangezogen wird.

κ_{RC}: Zuordnung von Condition zu Schließe-Relation

$$\kappa_{RC} \mid RC \rightarrow (C \cup \varnothing),$$

die für die Entscheidung des Schließens von Stelle s’ bei Ereignis an Stelle s mit $(s,s') \in RC$ herangezogen wird.

P: Menge von Übertragungsfunktionen (Propagationsfunktionen) p der Form:

$$p \mid WS_P \rightarrow W,$$

WS_P ist ähnlich zur Definitionsmenge von Conditions (siehe oben) eine Menge von Wertezuordnungen auf einer für die Übertragungsfunktion p definierten Stellenmenge S_P.

π_{RO}: Zuordnung von Übertragungsfunktion zu Öffne-Relation

$$\pi_{RO} \mid RO \rightarrow (P \cup \varnothing).$$

Durch obiges Modell wird eine Übergangsfunktion δ wie folgt definiert:

δ: Übergangsfunktion δ $\mid ((QE \otimes WSHist) \otimes E) \rightarrow ((QE \otimes WSHist) \otimes E^*)$

QE: Menge von Teil-Zuständen, die definiert sind durch die jeweilige Menge der Ereignisse an geöffneten Stellen („Erwartungszustände“ oder „Ereigniszustände“), also für q, q’ ∈ QE gilt: $q = q' \Leftrightarrow \varepsilon(q) = \varepsilon(q')$, wobei $\varepsilon(q), \varepsilon(q') \subseteq E$, die dem jeweiligen Gesamt-Zustand zugeordnete Menge der möglichen Ereignisse.
Zum Startzeitpunkt gilt für den Startteilzustand $q_0 \in QE$

$$\varepsilon(q_0) = \bigcup_{s_0 \in S_0} \sigma^{-1}(s_0) \text{ mit } S_0 \subseteq S.$$

Die übrigen Ereignismengen ε(q) definieren sich rekursiv.

WSHist: Menge von Teil-Zuständen, definiert durch Zuordnungen von Werten zu Stellen, die jeweils in einem Zustand festgestellt werden können („Speicherzustände“ oder „Gedächtniszustände“),

$$WSHist \subseteq \{ \omega s, \omega s \text{ Funktion } \omega s \mid S \rightarrow W \}$$

Als Wertezuordnung ωs_0 zum Startzeitpunkt haben wir ωstart. Die übrigen Wertezuordnungen aus WSHist werden wir im Folgenden rekursiv definieren.

(QE ⊗ WSHist) definiert die Menge der Zustände Q. Es handelt sich bei einem Zustand also um eine Kombination aus einer bestimmten Menge von Ereignissen ε(q) und einer bestimmten Wertebelegung ωs an den Stellen. Die Menge der Ereignisse beschreibt die aktuell möglichen Ereignisse und die Wertebelegung können wir auch als die aktuell gültige Wertezuordnung bezeichnen.
(Die nun folgende formale Darstellung ist mit zusätzlichen Kommentaren in Kursivschrift versehen.)

Für $q, q' \in QE$, $\omega s, \omega s' \in WSHist$, $e \in E$, $\sigma(e) := s$ und $enext \in E^*$ gilt:
$\delta(q, \omega s, e) \rightarrow (q', \omega s', enext) \Leftrightarrow$

(

$e \in \varepsilon(q)$

e muss in der Menge der möglichen Ereignisse von q liegen

<u>und</u> $\varepsilon(q') = (\varepsilon(q) \cup \bigcup_{s' \in S'} \sigma^{-1}(s')) \setminus (\bigcup_{s'' \in S''} \sigma^{-1}(s''))$

Die q' zugeordnete Menge möglicher Ereignisse ergibt sich aus der Menge im Ausgangszustand q erweitert um die Ereignismenge der neu zu öffnenden Stellen S' (Definition siehe unten) und vermindert um die Ereignismenge der zu schließenden Stellen S'' (Definition siehe unten)

<u>und</u> $\forall st \in S \quad \omega s'(st) = \omega e(e), \quad$ falls st = s,
$\omega s'(st) = \omega s(st) \quad$ sonst

Die neue Wertezuordnung ωs' nach Übergang ergibt sich an allen Stellen aus der alten mit Ausnahme an der Stelle s, wo der Wert mit dem Wert des Ereignisses e, also ωe(e) belegt wird (vereinfachter Fall).

<u>und</u>

Mit dem Öffnen einer Stelle s', also der Realisierung eines Stellenübergangs (s, s') werden unter Umständen auch Schreibereignisse des Systems „eröffnet". Dies ist dann der Fall, wenn der Stelle s' eine Ereignismenge zugeordnet ist, die auch Ereignisse der Typen O_S oder M_S enthält.
In diesem Falle wird über die (s, s') zugeordnete Propagationsfunktion die Entscheidung gefällt, welches der möglichen Ereignisse als nächstes an der Stelle s' stattfinden soll. Hier wird der Sicht des Modells Rechnung getragen, dass auch das Schreiben des Systems an Ausgabestellen und Hilfsstellen über Ereignisse mit entsprechenden Zustandübergängen abgehandelt werden, wobei die Abarbeitung dieser Ereignisse unmittelbar nach dem Öffnen der Stelle, also synchron erfolgt.
Wenn nun mehrere Stellenübergänge bei einem δ-Übergang erfolgen, muss über eine Ordnungsfunktion auf den Stellenpaaren in RO entschieden werden, in welcher Reihenfolge die Abarbeitung der Schreibereignisse erfolgen soll.

Diese Ordnung ergibt dann die Reihenfolge im Tupel enext der propagierten Ereignisse:
enext = $(e_0 , .. e_i, .. e_m)$, $0 <= i <= m$, mit $i,m \in N$,
falls $| S' | = m+1$
<u>mit</u> $e_i = (p_{(sROsi)}(\omega s), s_i, t_i)$, mit $s_i \in S'$, $t_i \in \{ O_S, M_S \}$
falls $p_{(sROsi)}$ definiert,
also $\pi_{RO}(sROs_i) = p_{(sROsi)} \neq$ null
$e_{(s,si)}$ = null, sonst
<u>und</u> $ord_{RO}(s,s_i) = i$
ord_{RO} sei eine geeignete Ordnungsfunktion auf den Paaren $(s_k,s_l) \in RO$, die jeweils allen Paaren mit gleicher Ausgangsstelle s_k eine eindeutige Ordnungszahl zuordnet.
und $t_i = O_S$, falls s_i Dialogstelle, $t_i = M_S$, sonst
enext = null, sonst

<u>und</u>

Bleiben noch S' und S'' zu definieren.
Ein δ-Übergang kann das Öffnen oder Schließen von Stellen bewirken und damit die Menge der möglichen Ereignisse erweitern und/oder einschränken. In der nächsten Klammerung werden die Kriterien für das Öffnen einer Stelle und in der darauf folgenden die für das Schließen festgelegt. Dies geschieht in Form der Definition zweier Stellen-Teilmengen S' bzw. S'' bzw. zweier Relationen RO' und RC', die RO bzw. RC einschränken:

(
1.Definition S'
Zu Öffnende Stellen S': Voraussetzung für das Öffnen einer Stelle s' ist, dass die Stelle s, an der das Ereignis stattfindet, über RO in Beziehung steht. Es müssen jedoch noch weitere Bedingungen erfüllt sein, die zu einer weiteren Einschränkung der Übergänge s nach s' in RO führen.
Wir definieren also S', respektive RO':

$s' \in S' \Leftrightarrow (s,s') \in RO' \Leftrightarrow$
(
$(s,s') \in RO$
und

Ereignis e muss aus der Menge der auslösenden Ereignisse stammen, die dem Übergang s,s' zum Öffnen zugeordnet ist, also:

$e \in \varepsilon_{RO}(s,s')$
und

Die dem potentiellen Übergang s,s' aus RO zugeordnete Condition muss bei der aktuell gültigen Wertebelegung ωs den Wert TRUE liefern:
$c(\omega_{akt}) = TRUE,$ mit $c = \kappa_{RO}(sROs')$
und $\omega_{akt} = \omega s_{|Sc}$,
wobei $\omega s_{|Sc}$ die aktuelle Wertezuordnung ωs, eingeschränkt auf die für c definierte Stellenmenge S_c
)
) *Ende Definition S', RO' mit aktuellen Übergängen s,s' zum Öffnen s'*
und
(
2. Definition S'':
Zu schließende Stellen S''. Ähnlich zu 1. muss nun ein potentieller Übergang s,s'' aus RC existieren, der auch noch weitere Bedingungen erfüllt. Definition S'', respektive RC':

$s'' \in S'' \Leftrightarrow (s,s'') \in RC' \Leftrightarrow$
($(s,s'') \in RC$
und
$e \in \varepsilon_{RC}(s,s'')$
und

c' (ω_{akt}) = TRUE, mit c' = κ_{RC} (sRCs')
und $\omega_{akt} = \omega s_{|Sc'}$,
wobei $\omega s_{|Sc'}$ die aktuelle Wertezuordnung
ωs, eingeschränkt auf die für c definierte
Stellenmenge $S_{c'}$

)
) *Ende Definition S'', RC' mit aktuellen Übergängen s, s'' zum Schließen s''*

) *Ende Definition Übergangsfunktion*

Zusammenfassend halten wir fest: Bei Eintreten eines Ereignisses e geht die Menge der q zugeordneten möglichen Ereignisse in die q' zugeordnete Menge ε(q´) über, falls
1. das Ereignis e ein Öffne-Ereignis an Stelle s für eine Stellenmenge S'
darstellt ($e \in \varepsilon_{RO}(s,s´)$, $s' \in S'$), die Öffne-Bedingung c erfüllt ist oder/und
2. das Ereignis e ein Schließe-Ereignis an Stelle s für eine Stellenmenge S´´ darstellt ($e \in \varepsilon_{RC}(s,s´´)$, $s'' \in S''$) und die Schließe-Bedingung c' erfüllt ist.
Der Wert an der Stelle s nimmt den Wert des Ereignisses e an. Außerdem werden ggf. Schreibereignisse des Systems enext ∈ E* an Stellen in S' propagiert, die dann unmittelbar (synchron) weitere δ-Übergänge nachfolgen lassen.
Dies bedeutet, dass Ereignisse, die von außen an das System herangetragen werden, also o.B.d.A. vom Benutzer verursachte und aktuell anstehende Ereignisse, erst abgehandelt werden, wenn alle vom System erzeugten Ereignisse enext abgearbeitet sind. Erzeugt die Abarbeitung der Ereignisse in enext wiederum neue Ereignisse, werden diese unmittelbar nach den in enext stehenden Ereignissen und vor allen von außen erzeugten und noch anstehenden Ereignissen behandelt.

Hierarchisches Modell

In einem hierarchischen Ereignisstellenmodell ist zusätzlich eine zweistellige Relation $HS \subseteq (S \otimes S)$ auf der Menge der Stellen definiert mit den üblichen Eigenschaften einer Hierarchie (siehe auch Abschnitt 3.0.3.0: Grundmodell).

Ein hierarchisches Ereignisstellenmodell ESMH ist somit ein Tupel:

$$(E, T', W, S, S_0, HS, RO, RC, C, P, \tau', \sigma, \varepsilon_{RO}, \varepsilon_{RC}, \omega e, \omega start, \kappa_{RO}, \kappa_{RC}, \pi_{RO}),$$

wobei alle Elemente des Tupels außer T', τ' und HS wie oben definiert sind und HS eine hierarchische Relation über S als Teilmenge von ($S \otimes S$) darstellt. T' und τ' sind die entsprechenden Erweiterungen von T und τ für zusammengesetzte Stellen und zusammengesetzte Ereignisse (siehe auch Abschnitt 3.1 Formale Darstellung des Grundmodells):

$T' = \{ I_B, O_S, M_S, I_BO_S, I_BM_S, I_BO_SM_S, O_SM_S\}$ und
$\tau' \mid E \rightarrow T'$.

Die oben definierte Übergangsfunktion δ ist dann so zu erweitern, dass die Übergänge von Stellen zu zusammengesetzten Stellen und ihren Teilstellen eindeutig definiert sind.

Dazu sind insbesondere die beiden Relationen RO, RC, die die Öffne- bzw. Schließerelation in einem nicht hierarchischen Modell ausdrücken, durch Relationen auszutauschen, die die Hierarchie von Stellen mit einbezieht. Wir definieren deshalb zwei Relationen ROH' und RCH', die aufsetzend auf den Relationen RO', RC' (wie oben definiert) und HS, festhalten, wann eine Stelle aufgrund eines Ereignisses an einer anderen Stelle eröffnet bzw. geschlossen wird.

Definition von ROH' $\subseteq$ (S $\otimes$ S):
$\exists$ (s,s') $\in$ ROH' $\Leftrightarrow$ ($\exists$ (s,s') $\in$ RO'
oder
($\exists$ (s,s'') $\in$ RO' und $\exists$ (s'',s') $\in$ HSStart
d.h. s' ist Startknoten von s''
)
)

HSStart, eine Stelle u ist Startstelle von t, sei dabei wie folgt definiert:

(t,u) $\in$ HSStart $\Leftrightarrow$ (
$\exists$ k-Tupel (s_0, ..s_i, .. s_k) mit s_0, s_i, $s_k \in S$, i, k $\in$ N,
und $s_0 = t$ und $s_k = u$
und (s_i, s_{i+1}) $\in$ HS für alle i = 0, .., k-1,
d.h. s_{i+1} ist Teilstelle von s_i
und $s_i \in S_0$ für alle i = 1, .., k
d.h. s_i ist aus der Menge der Startstellen S_0
)

Definition von RCH' $\subseteq$ (S $\otimes$ S):
$\exists$ (s,s') $\in$ RCH' $\Leftrightarrow$ ($\exists$ (s,s') $\in$ RC'
oder
($\exists$ (s,s'') $\in$ RC' und $\exists$ (s'',s') $\in$ HS*
d.h. s' ist Teilstelle von s''
)
)

Für die Übergangsfunktion ist im hierarchischen Fall des Ereignisstellenmodells der Anfangszustand $q_0 \in$ QE anders als für ein primitives ESM zu definieren: Am Anfang sind nämlich nicht alle Ereignisse der Startstellen S_0 möglich, sondern nur der Startstellen, die keine übergeordneten Stellen haben oder deren übergeordnete Stellen alle auch Startstellen sind:

Dafür definieren wir eine neue Menge „globaler Startstellen" $S_{0global}$
$g_0 \in S_{0global} \Leftrightarrow$ ($g_0 \in S_0$ und
($g_0 \in S_{Top}$
oder
$\exists$ $s_{Top} \in S_{Top}$ und $s_{Top} \in S_0$ und (s_{Top}, g_0) $\in$ HSStart
)
)
S_{Top} sei die Menge aller Stellen ohne übergeordnete Stelle, also
$s \in S_{Top} \Leftrightarrow ((s',s'') \in HS \Rightarrow s'' \neq s)$

Der Anfangszustand $q_0 \in QE$ ist dann wie folgt definiert:

$$q = q_0 \quad \Leftrightarrow \quad \varepsilon(q) = \bigcup_{g_0 \in S_{0global}} \sigma^{-1}(g_0) .$$

Im Anhang (Anhang B) ist die vollständige Definition der Übergangsfunktion für das hierarchische Ereignisstellenmodell zu sehen. Im Wesentlichen sind dabei die Relationen RO' und RC' durch ROH' und RCH' ersetzt.

5.1 Abbildung eines deterministischen endlichen Automaten auf die Ereignisstellendarstellung

Sei A ein deterministischer endlicher Automat mit endlicher Zustandsmenge QA, Anfangszustand $q_0 \in QA$, Endzuständen $FA \subseteq QA$, TA Menge der Token (Eingabesymbole) und δA der Übergangsfunktion $\delta A \mid QA \times TA \rightarrow QA$.
Dann lässt sich ein Ereignisstellenmodell ESA wie folgt konstruieren:

Für jeden Zustand $q \in QA$ wird genau eine Stelle $s_q \in S$ konstruiert.
Die q_0 entsprechende Stelle s_0 wird Anfangsstelle von ESA, also $S_0 = \{ s_0 \}$. Ebenso werden die den Endzuständen entsprechenden Stellen als Endstellen in ESA ausgezeichnet. Dazu müssen wir allerdings unser Ereignisstellenmodell um die Möglichkeit der Auszeichnung von Endstellen erweitern, was aber keine wesentliche Änderung des oben vorgestellten Modells bedeutet.
Für jedes Token $t \in TA$ und jeden Zustand $q \in QA$, der das Token t als Eingabe akzeptiert, definieren wir ein Ereignis $e_{qt} \in E$ mit $e_{qt} := \{ w_t, s_q, I_B \}$. Dabei sei w_t ein jedem Token zugeordneter eindeutiger Wert aus W (z.B können wir W als Teilmenge der Menge der natürlichen Zahlen wählen), s_q ist die dem Zustand q zugeordnete Stelle aus S.
Für jeden Übergang $\delta A : (q,t) \rightarrow q'$ sehen wir ein Paar (s_q , s_q') in RO vor und erweitern die Menge $\varepsilon_{RO}(s_q,s_q')$ um e_{qt}. Damit wird bei Eintreten des Ereignisses e_{qt} an der Stelle s_q die Stelle s_q' (neuer Zustand q') eröffnet.
Falls s_q ungleich s_q' fügen wir ebenso ein Paar (s_q, s_q') in RC ein und erweitern die Menge $\varepsilon_{RC}(s_q,s_q')$ um e_{qt}. Dies bedeutet, dass dann durch das Ereignis e_{qt} das Schließen der Stelle s_q (alter Zustand q) ausgelöst wird.
ωe, σ, τ sind als Projektionen von E eindeutig definiert. Die Funktionen κ_{RO}, κ_{RC} und π_{RO} bilden jeweils auf die leere Menge ab, da keine Conditions und Übertragungsfunktionen benötigt werden. Dies bedeutet insbesondere, dass bei einem Stellenübergang die Menge der erzeugten neuen Ereignisse enext leer ist, also kein weiterer automatischer Übergang propagiert wird (Alle an den Stellen akzeptierten Ereignisse wurden auch als per Definition asynchrone Ereignisse vom Typ I_B definiert.)
ωstart kann beliebig gewählt werden.

Es ist nun leicht zu sehen, dass ein Übergang im endlichen Automaten von einem Zustand q in einen Zustand q' über ein Token t genau dann erfolgt, wenn an der geöffneten Stelle s_q sich ein Ereignis e_{qt} ereignet und damit die Stelle s_q' eröffnet, die Stelle s_q jedoch geschlossen wird, falls s_q und s_q' sich unterscheiden.

Umgekehrt können wir bei bestimmten Einschränkungen des Ereignisstellenmodells dieses leicht auf einen deterministischen endlichen Automaten abbilden, wie das im folgenden Abschnitt gezeigt wird:

5.2 Abbildung einer eingeschränkten primitiven Ereignisstellendarstellung auf einen endlichen Automaten

Wenn wir von Werten und Übertragungsfunktionen, Conditions und ihren Zuordnungen zu den Stellen absehen und einige in der Praxis ohnehin naheliegende Annahmen bezüglich Endlichkeit darin vorkommender Elementmengen machen, können wir diese eingeschränkte primitive Ereignisstellendarstellung leicht in einen endlichen Automaten abbilden.
Kern der Überlegung ist dabei, dass jeweils ein Zustand dieses Automaten durch die Menge der Ereignisse aller gerade geöffneten Stellen definiert ist.

Ein Zustandsübergang von einem Zustand q zu einem Zustand q‘, mit q ungleich q‘, erfolgt genau dann, wenn eine Stelle geöffnet bzw. geschlossen wird. Dies bedeutet, dass die Menge der akzeptierten Ereignisse erweitert bzw. verringert wird und zwar jeweils um die Ereignismenge der geöffneten bzw. geschlossenen Stelle.
Die im Automaten erwarteten Token können als die Werte gesehen werden, die den Ereignissen zugeordnet sind. Dabei sind diese Ereignisse auf verschiedene Stellen der Dialogschnittstelle “verteilt”.

Eine formale Darstellung des oben grob skizzierten „Ereignisstellenautomaten“ (ESA)

Q: Endliche Menge von Zuständen
E: Endliche Menge von Token
δ: Übergangsfunktion $\delta \quad | Q \times E \rightarrow Q$
q_0: Startzustand
F: Menge der Endzustände, $F \subseteq Q$

Der deterministische, endliche Automat

$$ESA = (Q, E, \delta, q_0, F)$$

sei nun wie folgt durch ein eingeschränktes Ereignisstellenmodell definiert:

E, S, S_0, σ, σ^{-1}, RO, RC, ε_{RO}, ε_{RC} seien die wie oben definierten Mengen, Relationen und Abbildungen eines vereinfachten Ereignisstellenmodells (ohne W, C, P, τ, ωe, $\omega start$, κ_{RO}, κ_{RC}, π_{RO})
ε: Zuordnung der Ereignismenge pro Zustand $\quad \varepsilon | Q \rightarrow$ Potenzmenge(E)

q_0: sei als Startzustand definiert über $\varepsilon(q_0) = \bigcup_{s_0 \in S_0} \sigma^{-1}(s)$

E, die Menge der Ereignisse, diene nun für den Ereignisstellenautomaten als Menge der Token.

für $q \in Q$ und $e \in E$:
$\delta(q,e) = q'$, wobei $q \neq q'$ (q ungleich q‘), falls gilt:
(($\exists$ s, s‘$\in$ S mit $e \in \varepsilon_{RO}(s,s')$ und $(s,s') \in RO$
und
$\exists\, e' \in \sigma^{-1}(s')$ und $e' \notin \varepsilon(q)$, wobei $\varepsilon(q)$ die q zugeordnete Ereignismenge)
oder

($\exists$ s, s´´$\in$ S mit e $\in \varepsilon_{RC}$ (s,s´´) und (s,s´´) $\in$ RC
und
$\exists$ e´´ $\in \sigma^{-1}$(s´´) und e´´ $\in \varepsilon$(q), wobei ε(q) die q zugeordnete Ereignismenge))
und
(S' $\cap$ S'') = $\varnothing$
mit s´ $\in$ S´ $\Leftrightarrow$ (s,s´) $\in$ RO und s´´$\in$ S'' $\Leftrightarrow$ (s,s´´) $\in$ RC

d.h. bei Eintreten eines Ereignisses e erfolgt ein Übergang von Zustand q in einen anderen Zustand q´, dessen Menge akzeptierter Ereignisse aus der Menge der Ereignisse des Vorgängerzustands q besteht, erweitert um die Ereignismengen der gerade geöffneten Stellen und vermindert um die Ereignismengen der gerade geschlossenen Stellen. Dies gilt, falls
1. das Ereignis Öffne-Ereignis an Stelle s für eine Stelle s´ darstellt (e $\in \varepsilon_{RO}$(s,s´)) und die Stelle s´ nicht bereits geöffnet ist und
2. das Ereignis e Schließe-Ereignis an Stelle s für eine Stelle s´´ darstellt (e $\in \varepsilon_{RC}$(s,s´´)) und die Stelle s´´ nicht bereits geschlossen ist.

Es gilt: $\varepsilon(q') = (\varepsilon(q) \cup \bigcup_{s' \in S'} \sigma^{-1}(s')) \setminus (\bigcup_{s'' \in S''} \sigma^{-1}(s''))$

δ (q,e) = q´ wobei q $\equiv$ q´ sonst,
d.h. andernfalls wird der Zustand q beibehalten

5.3 Grafische Darstellung

Im Folgenden soll im Einzelnen auf alle grafischen Elemente näher eingegangen werden. Die Elemente sind teilweise aus vorher angeführten Beispielen bereits bekannt, werden aber hier alle noch einmal der Vollständigkeit halber aufgeführt.
Zuvor soll noch einleitend auf einige grundlegende Aspekte der gewählten grafischen Darstellung eingegangen werden.

5.3.0 Kernaspekte der grafischen Darstellung

In dem in diesem Kapitel bisher vorgestellten Ereignisstellenmodell sind Sprachelemente enthalten, die es erlauben, den Dialog zwischen Mensch und Maschine in einer relativ komprimierten Darstellung zu beschreiben. Wir haben im vorhergehenden Kapitel (Kap. 4) diskutiert, wie wir von einer extensiven konkreten Darstellung von Ereignisfolgen in mehreren Stufen zu dieser intensivierten Darstellung gelangen. Unser Augenmerk lag dabei ausgehend von einem Grundmodell mit Ereignisfolgen auf der Vermeidung zu vieler Folgenstellen und Stellenübergänge.
Dazu trugen im Wesentlichen bei

- die Zusammenfassung einwertiger Stellen zu Stellen mit Wertebereichen,
- die Einführung eines Gedächtnisses in Form von Hilfsstellen,
- Prädikate (Conditions) und Propagationsfunktionen,
- die Hierarchisierung,
- die Bildung von Schleifen und Rekursionen und
- die Einführung nebenläufiger Stellen.

Hintergrund dieser Überlegungen war es immer, mit den vorliegenden Sprachelementen eine grafische, intuitive Darstellung zu entwickeln, wie dies in den vorangehenden Abbildungen bereits skizziert wurde, ohne dass wir dort schon von formal definierten gültigen Darstellungselementen eines Ereignisstellendiagrammes gesprochen haben.
Im Folgenden soll nun zusammenfassend gezeigt werden, wie Ereignisstellenmodelle (ESM's) in eine grafische Darstellung mit Hilfe so genannter Ereignis-Stellen-Diagramme (ESD's) umgesetzt werden können. Dabei werden auch mehrere Varianten der Darstellung entwickelt.

Hierarchische Stellenstruktur
Die wesentlichen grafischen Elemente dieser Darstellung bilden Rechtecke für die Stellen, die wiederum Rechtecke als Teilstellen enthalten können. Die Rechtecke sind ggf. über Pfeile (gerichtete Kanten) miteinander verbunden.

Kontrollflussaspekt
Dabei ist ein Aspekt der Verbindung über Pfeile die Darstellung des Eröffnens von Ereignissen und des Sperrens von Ereignissen an Stellen aufgrund von Ereignissen an anderen Stellen. Wir können dies als Aspekt des Kontrollflusses bezeichnen. Dieser Aspekt entspricht in der oben entwickelten Modellsprache im Wesentlichen den Relationen RO und RC.

Datenflussaspekt
Ein weiterer Aspekt ist die Darstellung der Abhängigkeiten der Ereigniswerte an den geöffneten Stellen und damit auch der Abhängigkeit der resultierenden Stellenwerte von

Werten an anderen Stellen, was wir als Aspekt des Datenflusses auffassen können. Dieser Aspekt steckt im oben dargestellten formalen Modell im Wesentlichen in den Definitionsbereichen der Propagationsfunktionen, mit denen definiert ist, welche Stellen für die Berechnung des Wertes an der Zielstelle des zugeordneten Stellenübergangs aus RO herangezogen werden. Wir können diese Abhängigkeit der Werte auch als Wertetransformation oder Übertragung der Werte von Ausgangsstellen zu Zielstellen verstehen.
Ein ähnlicher Aspekt, für den Pfeile als grafisches Ausdrucksmittel verwendet werden, ist die Abhängigkeit der einzelnen Prädikate von Werten bestimmter Stellen zum Zeitpunkt eines potentiellen Stellenübergangs.

5.3.1 Darstellungselemente eines Ereignisstellendiagramms (ESD)

Im Folgenden werden die einzelnen Elemente der grafischen Darstellung aufgeführt und den oben vorgestellten formalen Elementen des ESM gegenübergestellt.

Darstellung der Stellen und Stellenhierarchie
Zentrales Element in einem Ereignis-Stellen-Diagramm ist die Ereignisstelle, manchmal auch Dialogstelle (oder auch nur kurz Stelle oder Dialog genannt).
Stellen können geöffnet und geschlossen werden. Ist eine Stelle geöffnet, können an ihr Ereignisse geschehen bzw. festgestellt werden.
Im obigen formalen Modell sind Stellen durch die Menge S repräsentiert.
Stellen können Teilstellen enthalten, sind also ggf. hierarchisch geordnet. Im formalen Modell entspricht dies der Relation HS.

Stellen als Rechtecke mit Stellenbezeichnern
Eine Stelle wird im ESD als umrandetes Rechteck gezeichnet. In einem Rechteck können wiederum Rechtecke enthalten sein, die jeweils eine Teilstelle der übergeordneten Stelle repräsentieren.
Wir geben der Darstellung als Rechteck den Vorzug gegenüber der Darstellung als Kreis oder Oval, weil sich zum einen für Beschriftungen innerhalb von Rechtecken für Teilstellen mehr Fläche ergibt (weniger Flächenverschnitt) und zum anderen übliche Fenster und Felder am Bildschirm suggeriert werden.
Um auf Stellen im Modell textuell verweisen zu können, führen wir Stellenbezeichner für die Modellstellen in Form von Textstrings ein. Der Stellenbezeichner steht üblicherweise links oben im Rechteck oder kann außerhalb des Rechtecks (normalerweise links oben) angebracht werden. Im letzteren Fall muss die Zuordnung der Annotation allerdings klar erkennbar sein. Dies erreichen wir etwa durch das Anbringen der Annotation mit gestrichelter Linie zum Rechtecksrahmen. Um weiter Verwechslungen zu vermeiden, kann dem Bezeichner auch ein N mit Doppelpunkt (N:) vorangestellt werden.

Stellenbezeichner tauchen in der oben beschriebenen formalen Darstellung des ESM nicht auf. Stellenbezeichner (im Folgenden auch Stellennamen genannt) werden allerdings nicht nur dazu verwendet, um es einem Benutzer zu ermöglichen, die Stellen im Diagramm anzusprechen. Mit Stellennamen kann z.B. auch Rekursivität ausgedrückt werden: Taucht eine Stelle innerhalb einer zusammengesetzten Stelle mit gleichem Bezeichner auf, handelt es sich um einen rekursiven Aufruf. Auf eine formale Darstellung der Rekursivität wird in dieser Arbeit verzichtet, da es sich um eine bekannte Technik handelt, wie sie auch in gängigen Programmiersprachen eingesetzt wird.

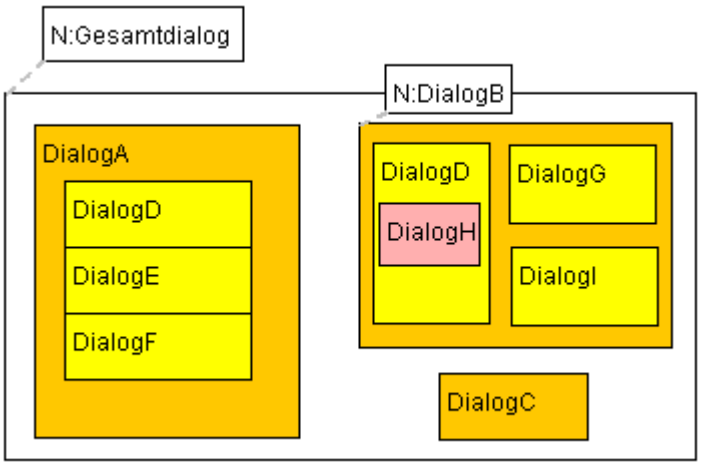


Abbildung 5-1 Hierarchie von Stellen in geschachtelten Rechtecken

Ein Ereignisstellendiagramm zeigt Dialogstellen in hierarchischer Struktur mittels in sich geschachtelter Rechtecke. Der Stellenbezeichner (Stellenname) ist entweder außen annotiert oder im Rechteck (nach Möglichkeit links oben) angebracht. Eine Stelle ist eindeutig über den kompletten Pfad der Bezeichner in der Hierarchie definiert. D.h., Teilstellen mit unterschiedlichen übergeordneten Stellen können denselben Bezeichner besitzen (Beispiel in der Abbildung: zwei Stellen mit Bezeichner DialogD unter DialogA und DialogB).

Gerichtete, beschriftete Kanten für Öffne und Schließerelationen

Bei Eintritt eines Ereignisses an einer Stelle kann eine andere Stelle geöffnet oder geschlossen werden. Im formalen Modell wird dies durch die beiden zweistelligen Relationen RO und RC ausgedrückt. Grafisch wird das Öffnen durch einen so genannten Öffne-Pfeil dargestellt. Dies ist eine gerichtete Kante, die von der Stelle, an der das Ereignis eintritt, zu der Stelle, die geöffnet wird, führt. Zur Verdeutlichung wird der Öffne-Pfeil wahlweise mit einem T (=Trigger) oder mit einem O(=Open) beschriftet. Den Öffne-Pfeil wollen wir auch als Triggerpfeil oder Triggerkante bezeichnen. Der Pfeil beginnt jeweils am Rahmen eines Stellenrechtecks und endet an einem Rahmen.
Entsprechend gibt es zur Darstellung der Schließe-Relation den so genannten Schließe-Pfeil. Dieser wird zur eindeutigen Interpretation wahlweise an der Spitze mit einem durchkreuzten Kreis (Symbol des Schließens) versehen und/oder mit einem C (=Close) beschriftet.
Im Folgenden werden zwei Schreibweisen bezüglich der Pfeildarstellung vorgeschlagen, die die Anzahl der Kanten reduziert und damit die Übersicht im Diagramm steigern kann.
1. Als Kurzschreibweise führen wir eine gerichtete Kante ein, die am Schaft, also in Nähe der Stelle, aus der die Kante weggeführt wird, mit einem durchkreuzten Kreis versehen ist. Es handelt sich dabei um eine Kombination aus Schließe und Öffne-Pfeil. Dieser „Schließe-Öffne-Pfeil“ kann zwischen den Stellen A und B gezogen werden, wenn durch ein Ereignis an der Stelle A die Stelle A selbst geschlossen und Stelle B geöffnet wird.
2. Als weitere Kurzschreibweise erlauben wir, dass Öffnepfeile (Triggerkanten) von Teilstellen an den Rechtecksrand einer übergeordneten Stelle geführt werden und dann vom Rechtecksrand der übergeordneten Stelle nur mehr eine Triggerkante weitergeführt wird. Die weitergeführte Triggerkante ist als Bündelung der auf den Rand geführten Triggerkanten der Teilstellen zu interpretieren. Dies ist nur erlaubt, wenn von der übergeordneten Stelle selbst kein Triggerereignis ausgeht, also keine Verwechslung möglich ist. Die Darstellung ist dann möglich, wenn wir fordern, dass eine übergeordnete Stelle immer geöffnet ist, wenn eine Teilstelle offen ist, und deshalb ein Öffnen einer übergeordneten Stelle von einer Teilstelle aus keinen Sinn macht.

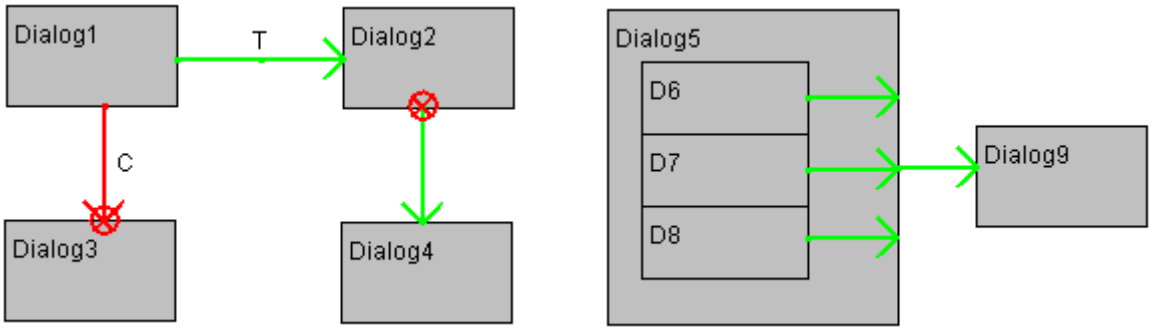


Abbildung 5-2 Darstellung der Öffne- und Schließe-Relation durch Öffne- und Schließe-Pfeil
Der mit T gekennzeichnete Pfeil (Öffnepfeil oder Triggerpfeil genannt) triggert (eröffnet) bei Eintreten eines Ereignisses an der Stelle „Dialog1“ Ereignisse an Stelle „Dialog2“.
Der mit C und dem durchkreuzten Kreis gekennzeichnete Pfeil (Schließepfeil) schließt bei Eintreten eines Ereignisses in „Dialog1“ die Stelle „Dialog3“, d.h. an dieser Stelle sind keine Ereignisse mehr möglich.
Die Annotation C könnte immer auch weggelassen werden, da der durchkreuzte Kreis den Pfeiltyp eindeutig macht. T kann ebenso fehlen, da bei einem einfachen Pfeil ohne Annotation angenommen wird, dass es sich um einen Triggerpfeil handelt
Die unterschiedlichen Farben sind nicht obligatorisch und dienen lediglich als zusätzliche visuelle Unterstützung.
Der Pfeil von Dialog2 zu Dialog4 ist eine Zusammenfassung eines Öffnepfeils und Schließepfeils. Er besagt, dass bei Ereignissen in Dialog2 dieser selbst geschlossen und Dialog4 geöffnet wird.
Ebenso ist die Darstellung mit Dialog5 und Dialog9 der Vorschlag für eine Kurzfassung: Der Pfeil von Dialog5 zu Dialog9 ist als Fortführung der Pfeile aus den Dialogen D6, D7 und D8 zu interpretieren. Diese Darstellung kann nur verwendet werden, wenn Dialog5 selbst keine Ereignisse direkt zugeordnet hat und in Dialog5 selbst dann auch keine Triggerereignisse erfolgen.

Annotation der Ereignistypen

Den Ereignissen, die sich an einer Stelle ereignen können, sind jeweils Ereignistypen zugeordnet. Im formalen Modell handelt es sich um die Typen I_B (Benutzereingabe), O_S (Ausgabe durch System) und M_S (Schreiben durch System an interner, unsichtbarer Hilfsstelle). Bei Stellen, die Ereignisse unterschiedlichen Typs zulassen, können entsprechende Typenkombinationen auftreten.
Grafisch wird dies durch eine entsprechende Annotation des die Stelle repräsentierenden Rechtecks durch die Buchstaben I, O, M bzw. Kombinationen ausgedrückt. Es wird damit gekennzeichnet, dass die Ereignisse, die sich an der Stelle ereignen, einen der in der Buchstabenkombination enthaltenen Typen besitzen.
Falls Verwechslungen möglich sind, stellen wir den Ereignistypen ein „T:“ voran.

Eintragung bzw. Annotation der Werte und Wertebereiche

Jedem Ereignis, das an einer Stelle eintreten kann, ist ein Wert zugeordnet. Zur Beschreibung der möglichen Werte wird die Bezeichnung eines Wertebereichs an der Stelle angegeben. Wir führen also in der grafischen Darstellung Bezeichner für Wertebereiche ein und fordern, dass diese dann anderswo definiert sind. Diese Bezeichner werden an der Stelle annotiert. Um Verwechslungen mit anderen Annotationen zu vermeiden, wird dem Bezeichner des Wertebereichs ein V mit nachfolgendem Doppelpunkt vorangestellt („V:“, V steht für Value) und wahlweise zur Verdeutlichung der Bezeichner in spitze Klammern gesetzt. Alternativ kann dem V: eine Aufzählung einzelner Werte, die dann zur Unterscheidung von Wertebereichsbezeichnern in Hochkommata gesetzt werden.
Es gibt Stellen, an denen sich nur Ereignisse mit genau einem Wert ereignen. Zum anderen wollen wir im Modell auch Werte an einer Stelle exemplarisch darstellen und Startwerte angeben können. Zu diesem Zweck erlauben wir zusätzlich zur Angabe eines Wertebereichs die Beschriftung mit einem Einzelwert. Der Einzelwert wird in der Regel in die Mitte des

Rechtecks gesetzt. Ist ein Einzelwert angegeben, wird dieser als Startwert und als exemplarischer Wert an der Stelle interpretiert. Ist kein Wertebereich angegeben, wird aus dem angegebenen Einzelwert der Wertebereich erschlossen (abhängig von Implementierung des Modellsystems). In einer „Primitivlösung“ wird der Wert als Wertebereich mit einem Element interpretiert. Intelligentere Lösungen ziehen weitere Angaben im Modell zur Ableitung des Wertebereichs heran. So kann z.B. die Wahl der Widgetklasse eine automatische Ableitung des Wertebereichs beeinflussen. Ein PushButton z.B. legt einen einelementigen Wertebereich nahe, während ein Textwidget auf einen Wertebereich schließen lässt, der mehr Elemente besitzt, etwa vom Typ Zahl bis hin zu beliebigem Text.
Im formalen Modell wird die Belegung der Stellen mit Startwerten durch die Funktion ωstart bezeichnet. Die Menge aller Werte ist dort mit W bezeichnet.

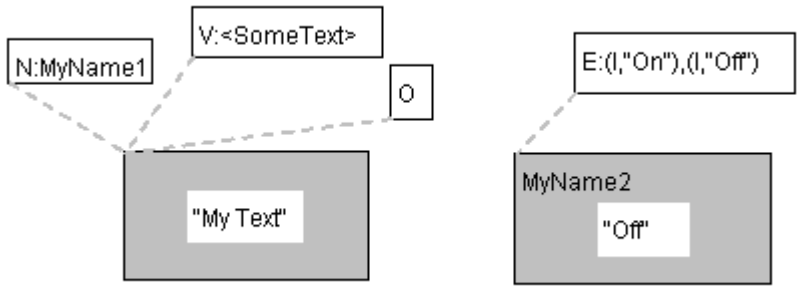


Abbildung 5-3 Beispiel-Annotationen an Stellen
Die Stelle mit Stellenbezeichner MyName1 (N:MyName1) ist eine Ausgabestelle (Typ O) mit Wertebereich SomeText. Dabei wird vorausgesetzt, dass SomeText bekannt ist. Als exemplarischer Wert und als Startwert der Stelle ist „My Text“ eingetragen. Die Stelle MyName2 hat als Wertebereich die Werte „On“ und „Off“, ist vom Typ Eingabe (I) und hat als exemplarischen Wert der Eingabe „Off“. Für die Implementierung kann dieser Wert als Startwert (Defaultwert) der Eingabe benutzt werden. In einem Textwidget würde dann z.B. „Off“ bereits angeboten werden. Bei Realisierung mit einem RadioButton würde dann konsequenterweise der Button so initialisiert, dass die erste Eingabe das Ausschalten ermöglicht.

Annotation der Ereignisse

Durch die Annotation von Ereignistypen und Wertebereichen werden implizit in der grafischen Darstellung Ereignismengen definiert. Die Ereignismenge entsteht dadurch, dass die im formalen Modell vorgestellten ein Ereignis repräsentierenden Tripel, bestehend aus Stelle, Wert und Typ aus der Kombination der Stelle mit jedem Wert aus dem annotierten Wertebereich und jedem einzelnen annotierten Typ (I, O, oder M) entsteht.
Einer Stelle können alternativ oder zusätzlich zur Annotation der Ereignistypen und Werte/Wertebereiche die an der Stelle möglichen Ereignisse als Annotation mitgegeben werden.
Gekennzeichnet wird die Annotation wie bei der Zuordnung der Ereignisse zu Stellen durch ein vorangestelltes E mit Doppelpunkt („E:“) und einem Bezeichner für die Ereignismenge oder mit einer expliziten Auflistung von Typ-Wert- oder Typ-Wertebereichs-Paaren.

Darstellung der Conditions

Das Öffnen und Schließen von Stellen kann an Bedingungen geknüpft werden. Im formalen Modell wird dies über eine Menge von Conditions C ausgedrückt. Eine Condition kann einem Stellenpaar aus den Öffne- und Schließe-Relationen RO und RC zugeordnet werden. Dies geschieht im formalen Modell über die Zuordnungsfunktionen κRO bzw. κRC.
In der grafischen Darstellung wird dies wahlweise in zwei Arten dargestellt:

1. Annotation der Conditions an Triggerkanten
   Die Condition kann am Öffne-Pfeil oder Schließe-Pfeil annotiert werden. Dies geschieht dadurch, dass ein Bezeichner der Condition gefolgt von einer Klammer mit den Parametern, die in die Condition eingehen, annotiert wird. Alternativ dazu kann ein

boolescher Ausdruck annotiert werden. Sowohl dem Bezeichner wie auch dem booleschen Ausdruck wird ein C mit Doppelpunkt vorangestellt.
Die mit Komma getrennten Parameter in der Klammer nach dem Conditionbezeichner sind Stellenbezeichner der Stellen, deren aktuelle Werte in die Auswertung der Condition eingehen.

2. Darstellung der Condition als "Pseudostelle" mit „Ausgängen" TRUE und FALSE
Eine Alternative zur Annotation an eine Kante ist die Darstellung einer Condition in einem Rechteck. Im Rechteck sind zwei Rechtecke mit Beschriftung „TRUE" bzw. „FALSE" eingebettet. Die Beschriftung des Rechtecks entspricht der oben beschriebenen Annotation, also „C:" gefolgt vom Bezeichner und der Parameterklammer bzw. gefolgt vom booleschen Ausdruck. Zum Rechteck führt eine Triggerkante. Von den Teilrechtecken mit Beschriftung TRUE bzw. FALSE gehen jeweils Öffne- oder Schließe-Pfeile aus.
Mit dieser Darstellung können mehrere mit einer Condition oder deren Negation annotierte Kanten zusammengefasst werden, die von der selben Stelle ausgehen und denen die identische Menge an auslösenden Ereignissen zugeordnet ist.

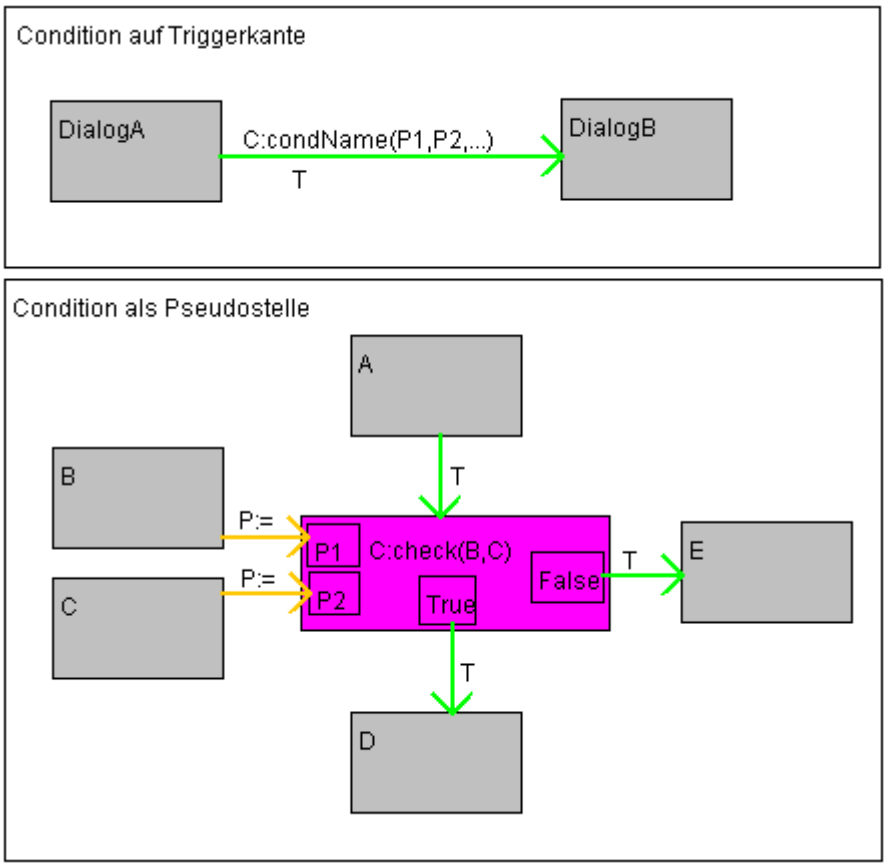


Abbildung 5-4 Darstellungen von Conditions

Darstellung der Propagationsfunktionen

Propagationsfunktionen haben wir eingeführt, um dynamisch bestimmen zu können, welcher Wert an einer Ausgabestelle oder an einer Hilfsstelle gesetzt werden soll, wenn sich dort ein Ausgabeereignis (Typ O_S) bzw. Schreibereignis (Typ M_S) ereignet. Es handelt sich dabei um Stellen, an denen Ereignisse mit unterschiedlichen Werten auftreten können. Dabei gehen in Propagationsfunktionen u.U. aktuelle Werte bestimmter definierter Stellen ein, die als Parameter übergeben werden.
Im formalen Modell sind die Propagationsfunktionen in der Menge P dargestellt. Deren Zuordnung zu Stellenpaaren in der Öffne-Relation RO wurde mit π_{RO} bezeichnet.

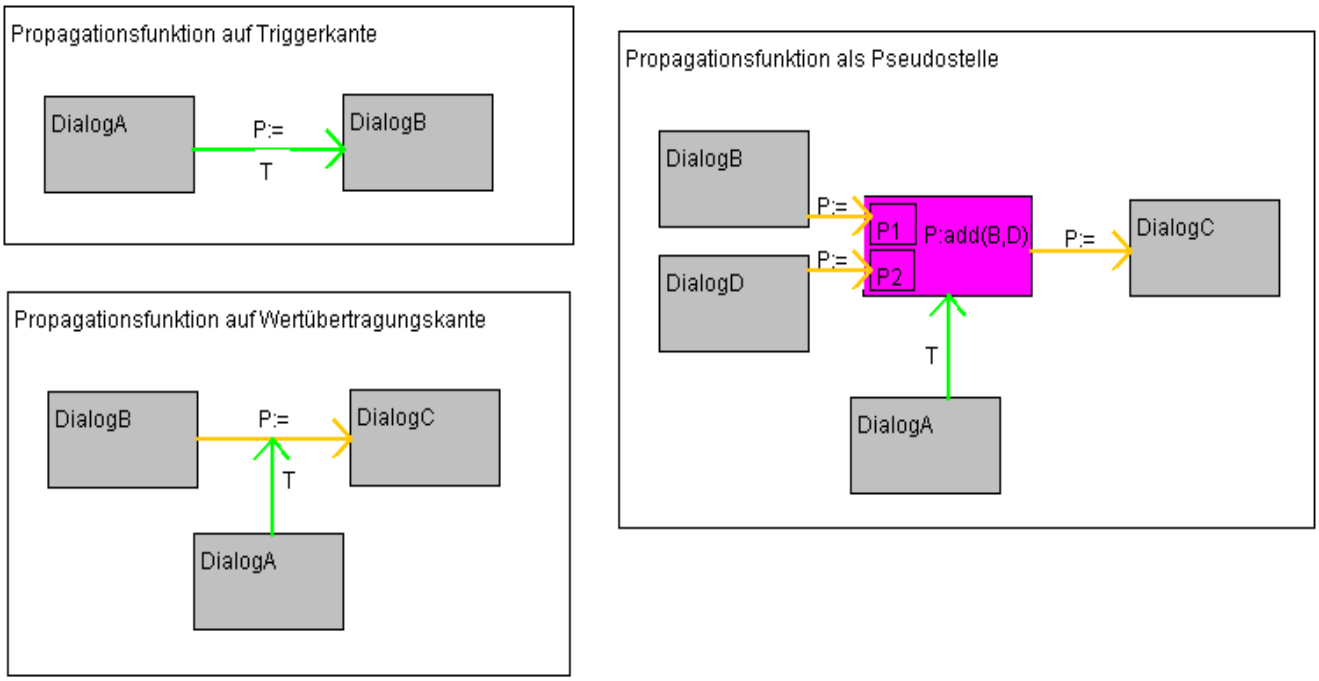


Abbildung 5-5 Darstellungen der Propagationsfunktion

Annotation der Propagationsfunktionen an Triggerkanten

Im einfachen Fall werden Propagationsfunktionen ähnlich Conditions an Triggerkanten annotiert. Dies geschieht, indem mit vorangestelltem „P:“ der Name der Propagationsfunktion gefolgt von den kommagetrennten, geklammerten Eingangsparametern an die Triggerkante geschrieben wird. Die Eingangsparameter sind wiederum Bezeichner von Stellen, deren aktuelle Werte zur Bestimmung des Funktionswertes benötigt werden. Anstelle eines Funktionsnamens mit geklammerten Parametern kann auch ein Gleicheitszeichen („=“) gefolgt von einem Ausdruck zur Berechnung eines Wertes annotiert werden (siehe Implementierung Modellsystem). Als Kurzform sei erlaubt: Steht nur ein Gleichheitszeichen nach „P:“, also „P:=“ so bedeutet dies, dass der Wert der Stelle, aus der die Kante führt, identisch an die Zielstelle übertragen wird.

Annotation der Propagationsfunktionen an Wertübertragungskanten

Um grafisch sichtbar zu machen, von welchen Stellen Werte in die Berechnung eingehen, und an welche Stelle der berechnete Wert „übertragen“ wird, führen wir so genannte Wertübertragungskanten ein (siehe oben Datenflussaspekt der grafischen Darstellung).
Wenn grafisch dargestellt werden soll, dass bei Eintritt eines Ereignisses an einer Stelle A der Wert einer Stelle B an eine Stelle C „übertragen“ werden soll, zeichnen wir eine gerichtete Kante zwischen B und C (Wertübertragungskante) und führen eine Triggerkante von A auf die Wertübertragungskante (siehe Abbildung 5-5 Darstellungen der Propagationsfunktion).
An die Wertübertragungskante annotieren wir die Propagationsfunktion, die den Wert von B als Eingangsparameter erhält und den berechneten Wert an C „überträgt“.
Im einfachen Fall handelt es sich bei der Propagationsfunktion wiederum um die Identitätsfunktion, d.h. der Wert von B wird identisch an C übergeben.
In Sprechweise des Ereignisstellenmodells handelt es sich bei der „Wertübertragung“ um die Berechnung des Wertes eines Ereignisses das an C eintritt, nachdem ein bestimmtes Ereignis an Stelle A eingetreten ist.

Propagationsfunktion als „Pseudostelle“ mit Eingangs- und Ausgangssubstellen
Um grafisch zeigen zu können, dass eine Propagationsfunktion von mehreren Stellen aus mit Parameterwerten versorgt wird und außerdem mehrere Ausgabewerte besitzt, die an verschiedene Stellen übertragen werden, führen wir die Darstellung einer Propagationsfunktion in einem Rechteck ein, das eingebettet Rechtecke („Teilrechtecke“) für die Eingangsparameter und die Ausgangsparameter besitzt.

Im obigen formalen Modell fehlt der Fall, dass mehrere Ausgabewerte durch eine Propagationsfunktion geliefert werden. Es handelt sich formal gesehen jedoch hier einfach um das gleichzeitige Ausführen mehrerer Propagationsfunktionen in einer zusammenfassenden Funktion. Im formalen Modell wird davon ausgegangen, dass alle relevanten Propagationsfunktionen, die von einem Ereignis „ausgelöst“ werden, quasi gleichzeitig ausgeführt werden und sich nicht gegenseitig beeinflussen.

In die Rechtecke für die Eingangsparameter führen Wertübertragungskanten von den jeweiligen Stellen, aus denen die Parameterwerte übertragen werden. Von den die Ausgangsparameter repräsentierenden Rechtecken aus führen wiederum Wertübertragungskanten zu den jeweiligen Zielstellen, an denen ein Ereignis propagiert wird. Alle diese Stellen werden dem formalen Modell gemäß auch geöffnet, bevor das Ereignis propagiert wird. Von der Stelle, an der ein Ereignis eintritt, das die Propagation auslöst, wird eine Triggerkante zum Rechteck der Propagationsfunktion geführt.
Die Rechtecke, die Eingangsparameter darstellen, können wahlweise mit Bezeichnern für die formalen Parameter der Propagationsfunktion ausgestattet werden. In diesem Fall ist es nicht nötig, hinter dem Namen der Propagationsfunktion die Parameter anzugeben. Falls jedoch andererseits die Parameter in Klammern angegeben sind, ist es nicht notwendig Teilrechtecke für Eingabeparameter zu zeichnen. Gibt es mehrere Ausgangswerte, müssen diese bezeichnet werden. Gibt es nur einen Ausgabewert, kann das Teilrechteck dafür entfallen. Die Wertübertragungspfeile zu den Zielstellen gehen dann vom Rand des Rechtecks der Propagationsfunktion aus. Entfallen die Teilrechtecke für die Eingabeparameter, führen die Wertübertragungskanten der Eingabewerte zum Rand des Rechtecks der Propagationsfunktion.

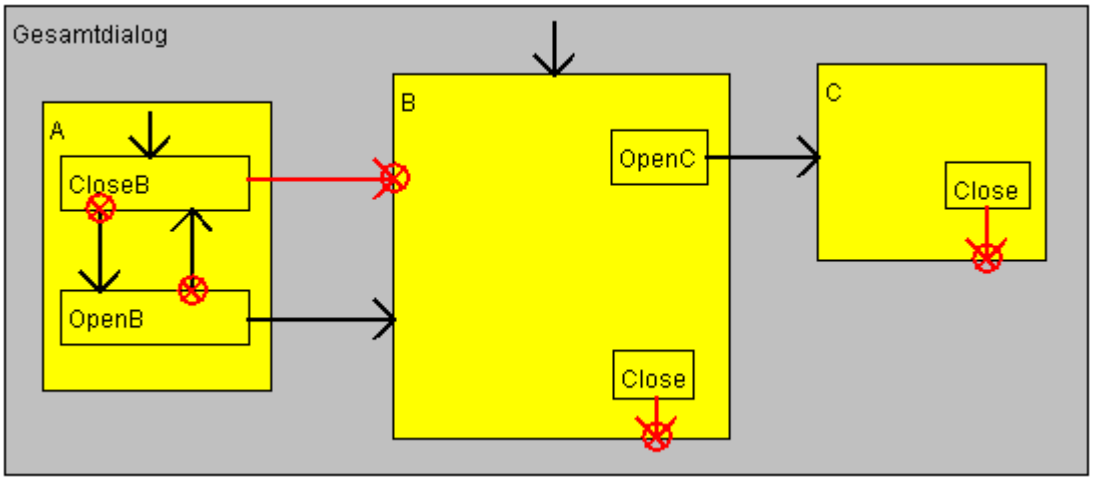


Abbildung 5-6 Kennzeichnung von Startstellen
Nach Definition wird eine Stelle nur geöffnet, wenn mindestens ein Öffnepfeil von einer anderen Stelle zur Stelle führt oder wenn es sich um eine Startstelle handelt. Startstellen werden dadurch erkannt, dass entweder kein Öffnepfeil in die Stelle mündet oder dass ein Pfeil ohne Ausgangsstelle („aus der leeren Fläche“) auf die Stelle zeigt. Im Beispiel gilt letzteres für die Stellen B und CloseB. B ist eine globale, CloseB eine lokale Startstelle. Weitere Startstellen sind A, Close unter B, Close unter C und OpenC, wobei A wiederum global ist. Globale Startstellen werden sofort mit Start des Gesamtdialogs eröffnet. Dies gilt in unserem Fall also für die Dialogstellen A und B. lokale Startstellen werden mit ihrer jeweils direkt übergeordneten Stelle geöffnet.

Darstellung der Startstellen
Startstellen sind Ereignisstellen, die zum Startzeitpunkt der Applikation geöffnet sind (globale Startstellen) oder Teilstellen übergeordneter Stellen der Stellenhierarchie, die sofort mit dem Öffnen der übergeordneten Stelle geöffnet sind (lokale Startstelle).
Wir erkennen Startstellen in der grafischen Darstellung implizit oder durch explizite Kennzeichnung:
Eine implizite Erkennung ist dadurch möglich, dass keine Triggerkante, also kein Öffnepfeil in die Stelle führt. Ist die Stelle keine Teilstelle einer anderen, handelt es sich dann um eine globale Startstelle, andernfalls um eine lokale Startstelle.
Falls eine Triggerkante in eine Stelle, die als Startstelle fungieren soll, führt, müssen wir diese explizit kennzeichnen. Dies geschieht durch eine Triggerkante, die „aus dem Leeren kommt", die also von keinem Rechteck ausgeht.
Im formalen Modell sind Startstellen in der Menge S_0 zusammengefasst.

Annotation der Triggerereignisse
Den Stellenübergängen zum Öffnen oder Schließen einer Stelle (grafisch: Öffne- und Schließe-Pfeile) sind jeweils Ereignismengen zugeordnet, die den Übergang bzw. das Öffnen und Schließen der Zielstelle auslösen. In manchen Fällen stimmt die Menge dieser Ereignisse mit der Menge der Ereignisse überein, die der Ausgangsstelle als Ereignismenge zugeordnet ist. D.h. alle Ereignisse an der Stelle bewirken alle Übergänge. In vielen Fällen muss allerdings differenziert werden. Wir benötigen deshalb dann eine explizite Zuordnung der Menge der auslösenden Ereignisse für den jeweiligen Stellenübergang.
Im formalen Modell wird dies durch die Zuordnungen ε_{RO} bzw. ε_{RC} ausgedrückt.
Im ESD bewerkstelligen wir dies durch Annotation der Ereignismenge am Öffne- bzw. Schließepfeil und zwar um Verwechslungen mit Zielereignismengen (siehe unten) zu vermeiden in der Nähe der Stelle, von der die Kante wegführt. Gekennzeichnet wird die Annotation wie bei der Zuordnung der Ereignisse zu Stellen durch ein vorangestelltes E mit Doppelpunkt („E:") und einem Bezeichner für die Ereignismenge oder mit einer expliziten Auflistung von Typ-Wert- oder Typ-Wertebereichs-Paaren (siehe Implementierung Modellsystem).

Annotation zum Öffnen und Schließen von Ereignisteilmengen (Zielereignisse)
Bei Stellen, die nicht ein einzelnes Ereignis sondern eine Ereignismenge repräsentieren, ist es manchmal sinnvoll, das Eröffnen oder das Schließen (Sperren) nur für eine Teilmenge durchzuführen. (Im obigen formalen Modell ist dieser Fall nicht ausgeführt). Der Fall tritt zum Beispiel auf, wenn wir zusammengesetzte Stellen im Modell nicht mehr weiter detaillieren wollen und der zusammengesetzten Stelle zur Implementierung direkt ein Widget zugeordnet wird, das mehrere Ereignisse repräsentiert (siehe unten im Anschluss Kap.6).
Wir bringen dazu am Öffne- oder Schließepfeil eine Annotation einer Ereignismenge (siehe oben Annotation der Triggerereignisse) an, der wir, im Gegensatz zur Annotation von Triggerereignissen ein „ZE:" (=Zielereignismenge) voranstellen. Außerdem wird die Annotation in Nähe der Zielstelle positioniert.

5.3.2 Modellsichten

5.3.2.0 Zielsetzung

Im vorangegangenen Abschnitt haben wir gezeigt, wie ein Ereignisstellenmodell (ESM) in einem Ereignisstellen-Diagramm (ESD) mit grafischen Elementen spezifiziert werden kann. Wir haben dabei die einzelnen Sprachelemente der formalen Darstellung des ESM den Sprachelementen eines ESD gegenübergestellt. In vielen Fällen in der Praxis ergibt sich, dass trotz der bisher vorgestellten Möglichkeiten einer komprimierten Darstellung eine vollständige Spezifikation in einem einzigen ESD zu komplex und damit unübersichtlich würde.
Beispielsweise gehen wir bisher noch davon aus, dass eine zusammengesetzte Stelle alle ihre Teilstellen beinhaltet.
Wir wollen im Folgenden ein Konzept diskutieren, das die Aufteilung der Modellinformation in so genannte Modellsichten ermöglicht. Dies ist ein Ansatz, der nicht nur die hierarchische Struktur der Stellen für eine Aufteilung der Information in verschiedene Sichten nutzt, sondern generell davon ausgeht, dass „beliebige“ Teilmengen der Modellinformation in Teilsichten darstellbar sind. Durch die Einführung dieser Modellsichten wird auch der Tatsache Rechnung getragen, dass im Entwurfsprozess häufig zunächst bottom-up Teilsichten des Dialogablaufs entstehen, die dann sukzessive zu einer Gesamtsicht verschmolzen werden.

Wir gehen von zwei verschiedenen Ausgangspunkten im Entwurf aus.

1. Ausblenden aus Gesamtdiagramm
Zum einen benötigen wir Modellsichten, um ein einmal komplett erstelltes Modell in übersichtliche Teile aufzuteilen. Dies erreichen wir relativ einfach, wenn wir von einem Ereignisstellendiagramm ausgehen können, das das gesamte Modell beschreibt. Wir können dann prinzipiell beliebige Teilsichten erzeugen, indem wir jeweils eine bestimmte Menge von Darstellungselementen für die spezifische Teilsicht als unsichtbar definieren und ausblenden. In einer Darstellung des Modells am Bildschirm bedeutet dann die Auswahl einer bestimmten Sicht im primitiven Fall einfach das Ausblenden der als unsichtbar definierten Elemente. Gegebenfalls können wir für jede Sicht noch das Layout, wie etwa die Größe und Position der dargestellten Stellen variieren, etc. Für den auf dem Rechner realisierten Modelleditor gibt es dabei keine Probleme, was die logische Zuordnung der einzelnen dargestellten Elemente in den Teilsichten im Hinblick auf das Gesamtmodell betrifft, da die Information über das Gesamtmodell durch die Bildung der Teilsichten nicht beeinflusst wird.
Für den Benutzer erfordert die Darstellung von Teilsichten allerdings den intellektuellen Aufwand, die Einordnung der dargestellten Elemente in den Gesamtzusammenhang zu bringen. Deshalb ist es sinnvoll, den Benutzer bei der Wahl vernünftiger Teilsichten zu unterstützen und z.B. zumindest grafisch zu visualisieren, dass Information in den Teilsichten ausgespart wurde.

2. Zusammenführung von Teilsichten
Ein anderer Ausgangspunkt für Teilsichten ist der, dass üblicherweise der Entwurf des Systems nicht in einem Guss erfolgt, sondern sukzessive Teilinformation festgehalten und zusammengefasst wird. Um diesen Vorgang zu unterstützen, macht es Sinn, in einem Modelleditor die Definition von Information in Teilen zu ermöglichen, die dann durch die Maschine unterstützt zu einem Modell zusammengeführt wird. In diesem Fall benötigen wir ein Konzept, das es nicht nur dem Benutzer intellektuell, sondern auch dem Modelleditor ermöglicht, die Darstellungselemente aus den einzelnen Teilsichten in ein Gesamtmodell einzuordnen.

Im Folgenden wollen wir einige sinnvolle Möglichkeiten des Ausblendens von Modellinformation besprechen und dazu einfache grafische Elemente einführen, die dem Benutzer und dem System einen Hinweis auf die Unvollständigkeit der Information in einer Teilsicht geben. Danach besprechen wir speziell Möglichkeiten der redundanzfreien Darstellung hierarchischer Stellen und darauf folgend ein einfaches Konzept, das eine Zusammenführung von Teilsichten zu einem Gesamtmodell ermöglichen soll.

5.3.2.1 Ausblenden von Teilstellen

In einem ESD, das dann eine Teilsicht repräsentiert, können alle oder einige direkte Teilstellen einer Dialogstelle weggelassen werden. Grafisch wird dies durch drei waagrecht angeordnete Punkte im Rechteck der Stelle visualisiert. Dies führt dazu, dass eventuell von diesen Teilstellen ausgehende oder hinführende Kanten ebenfalls unsichtbar sind.
Mit dem Weglassen einer direkten Teilstelle sind im Normalfall auch alle deren Teilstellen im Sinne der transitiven Hülle nicht dargestellt. Allerdings ist es auch möglich, indirekte Teilstellen dennoch in der Sicht darzustellen, also nur bestimmte im Sinne der Hierarchie zwischengelagerte Teilstellen auszulassen. Eine Teilstelle deren übergeordnete Stelle(n) in der Darstellung fehlt (fehlen), wird gesondert gekennzeichnet. Dies kann dadurch geschehen, dass eine gestrichelte Linie von links oben an das Rechteck herangeführt wird (siehe Abbildung 5-7) oder/und dass dem Bezeichner der Stelle drei Punkte vorangestellt werden. Wir nennen derartige Stellen in einem Diagramm **Jokerstellen**. Jokerstellen haben keine eindeutige hierarchische Einordnung. Sie dienen als Vorlage für andere Stellen und führen bei Implementierung des Modells zu keiner eigenen Instanz.

Ausblenden von Kanten
Zur besseren Übersicht ist es häufig sinnvoll, nur Kanten zwischen den Teilstellen einer Stelle wegzulassen. In diesem Fall zeichnen wir als Hinweis der Weglassung in das Rechteck der Stelle einen Pfeil mit gestrichelter Linie.

Ausblenden von Annotationen
Mit dem Ausblenden von Teilstellen oder Kanten ist automatisch das Ausblenden aller zugehörigen Annotationen verbunden. Darüberhinaus sei es der Implementierung des Modelleditors überlassen, einzelne Annotationen oder Annotationstypen an sichtbaren Teilstellen oder Kanten auszublenden.

5.3.2.2 Darstellung der Hierarchie in Modellsichten

Die Hierarchieinformation einer zusammengesetzten Stelle kann in ein eigenes Diagramm ausgelagert werden. D.h., zusammengesetzte Stellen können auch ohne ihre Teilstellen oder nur mit einer Teilmenge der Teilstellen dargestellt werden. Hierbei nutzen wir obigen allgemeinen Ansatz, dass wir in einer Teilsicht einerseits Detailinformation ausblenden können zum anderen in einer anderen Teilsicht die komplette Umgebung einer Stelle. In der Teilsicht mit ausgeblendeten Teilstellen zeichnen wir zur Kennzeichnung der Weglassung von Information drei Punkte. Umgekehrt zeichnen wir in der anderen Sicht, in der lediglich die zusammengesetzte Stelle mit ihren Teilstellen dargestellt ist, außerhalb der Stelle drei waagrechte Punkte und kennzeichnen die Stelle selbst mit einer gepunkteten schrägen Linie (siehe Abbildung 5-7) zum Hinweis, dass im Diagramm übergeordnete Stellen weggelassen wurden. Wie oben bereits erwähnt, handelt es sich um eine so genannte Jokerstelle, die mit

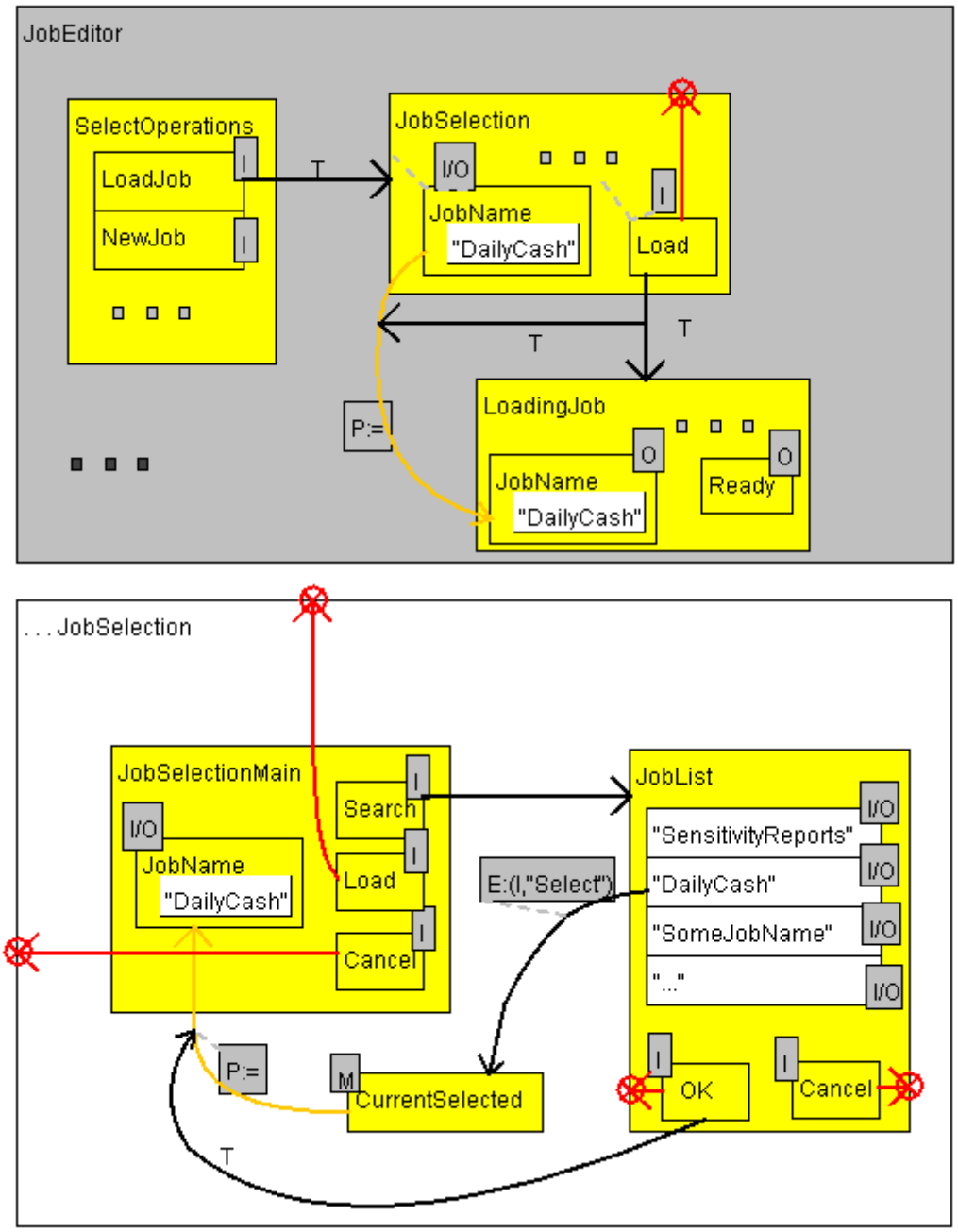


Abbildung 5-7 Auslagerung und gegenseitige Ergänzung in Teilsichten
Die Abbildung nimmt zwei mögliche Teilsichten des Modellbeispiels von Kapitel 8 vorweg. Wichtig zum Verständnis ist hier nur, dass das obere Rechteck den Gesamtdialog „JobEditor“ als Teilsicht zeigt, was an den drei Punkten in der Fläche des JobEditor-Rechtecks zu erkennen ist. Die drei gezeigten Teilstellen stellen wiederum nur jeweils eine Teilsicht dar. Die Teilstelle „JobSelection“ wird nun durch die im unteren Rechteck gezeigte Teilsicht „ . . . JobSelection“ ergänzt. „ . . . JobSelection“ selbst ist durch die vorangestellten drei Punkte im Namen eine Jokerstelle, die mit der obigen Aufrufstelle oder Fixstelle „JobSelection“ zusammengeführt wird. Die Jokerstellen „JobName“ und „Load“ oben in „JobSelection“, erkennbar an den schräg zum Rechteck geführten Linien fallen mit den entsprechenden Stellen „JobName“ und „Load“ unten zusammen.

ihrer Information über die Teilstellen die andere Stelle, die wir als Aufrufstelle bezeichnen können, ergänzt.
Generell gilt, dass eine Jokerstelle alle ihre Information an eine andere Stelle überträgt, die den gleichen Bezeichner besitzt und deren hierarchische Einordnung mit der hierarchischen

Einordnung der Jokerstelle vereinbar ist. Als weitere Voraussetzung gilt, dass die Aufrufstelle mit drei Punkten gekennzeichnet ist, sie also unvollständig definiert ist.
Die hierarchische Einordnung ist genau dann zwischen der Stelle und der Jokerstelle vereinbar, wenn die hierarchisch übergeordneten Stellen der Jokerstelle im Hierarchiepfad der anderen Stelle enthalten und gleich geordnet sind.
Auf diese Weise kann eine Jokerstelle ihre Information an mehrere Stellen übertragen. Besitzt die Jokerstelle wie gerade oben angeführt in der Teilsicht, in der sie aufgeführt ist, keine übergeordnete Stelle (außer dem Gesamtdialog), überträgt sie Information an alle gleichnamigen unvollständigen Aufrufstellen ungeachtet deren hierarchischer Einordnung.
Gerade wenn im Modell mehrere Stellen auftreten, die sich in ihrer „inneren Struktur", d.h. in den Teilstellen und deren Relationen zueinander gleichen, besteht das Interesse, diese innere Struktur nicht mehrfach redundant definieren zu müssen.

Subdiagramme als spezielle Teilsichten
Wird eine zusammengesetzte Stelle in einem Diagramm ohne „innere" Teilstellen dargestellt und werden in zusätzlichen eigenen Diagrammen wiederum rekursiv jeweils lediglich die unmittelbaren Teilstellen beschrieben, handelt es sich im Wesentlichen um die bekannte Technik der Darstellung einzelner Hierarchieebenen mittels Subdiagrammen (siehe z.B. komplexe Stellen bei Dialognetzen[JANSS96]).

5.3.2.3 Gegenseitige Ergänzung von Stellen

Erscheint dieselbe Teilstelle in unterschiedlichen Sichten, kann die Darstellung dieser Teilstelle in einigen oder in allen Sichten unvollständig sein. Die verschiedenen Darstellungen der Teilstelle ergänzen sich dann gegenseitig. Dies wird wiederum durch das Drei-Punkte-Symbol gekennzeichnet. Wir legen fest, dass die unvollständigen Darstellungen im Rechteck drei Punkte enthalten. Außerdem gehen wir davon aus, dass die Teilstellen ohne drei Punkte die vollständige Darstellung der Teilstelle zeigen.
Dass es sich um dieselbe Teilstelle in den unterschiedlichen Sichten handelt, erkennen wir dabei am identischen Stellenbezeichner und derselben hierarchischen Einordnung.

Die Ergänzung einer unvollständigen Stelle durch eine Jokerstelle ist nur ein Spezialfall mit einer Erweiterung des Konzeptes, dass sich Stellen in verschiedenen Sichten wechselseitig oder in einer Richtung ergänzen können. Die Erweiterung besteht lediglich darin, dass eine Jokerstelle mehrere Stellen fester hierarchischer Einordnung vertritt bzw. ergänzt.

5.3.2.4 Zusammenführung der Teilsichten

Wir wollen dem Benutzer des Modellsystems erlauben, in verschiedenen zunächst voneinander unabhängigen Teilen das Modell zu entwerfen. Diese unabhängigen Teilmodelle sollen dann zu einem konsistenten Gesamtmodell zusammengeführt werden, so dass wir die Teile als Teilsichten des Gesamtmodells betrachten können. Soll diese Zusammenführung vom System durchgeführt werden, benötigen wir ein Konzept bzw. Verfahren, das es der Maschine ermöglicht, die einzelnen Elemente der Teilsichten im Gesamtmodell richtig einzuordnen.
Prinzipieller Ansatz der Zusammenführung ist, dass die Darstellungselemente des Gesamtmodells eine Vereinigung aller Darstellungselemente der Teilsichten sind.
Hier ist nun zu klären, welche der Elemente der Teilsichten im Sinne des Gesamtmodells als identisch erkannt werden.

Dazu ist es notwendig, dass vom System erkannt werden kann, welche Stellen der Teilsichten mit welchen Stellen in anderen Teilsichten im Sinne des Gesamtmodells identisch sind und wie deren hierarchische Beziehung definiert ist.
Ausgehend davon können dann die übrigen Modellaussagen, wie etwa die Relationen der Stellen zum Öffnen und Schließen, zugeordnete Ereignismengen, etc. eingeordnet werden.

Wir wollen im Folgenden informell skizzieren, wie aus den Teilsichten die Menge der Stellen des Gesamtmodells und deren hierarchische Struktur ermittelt werden kann:

Wie oben bereits erwähnt, wollen wir zwei Stellen aus zwei Sichten als identisch definieren, wenn ihre Stellenbezeichner gleich sind und ihre in der Hierarchie der jeweiligen Teilsicht übergeordneten Stellen (falls vorhanden) identisch sind.
Für Stellen, die keine Jokerstellen oder deren Teilstellen sind, lässt sich dies leicht feststellen. Z.B. können wir rekursiv in jeder Teilsicht von oben beginnend nach unten alle Pfade für Nicht-Jokerstellen bestehend aus den Stellenbezeichnern aufbauen. Die so erhaltene Menge von Bezeichner-Pfaden kann elementweise verglichen werden. Alle jeweils übereinstimmenden Pfade definieren eine Stelle des Gesamtmodells. Wir gehen davon aus, dass die Menge der so erhaltenen Stellen („Fixstellenmenge“) nicht leer ist, sonst würden alle Teilsichten nur mit Jokerstellen in der obersten Hierarchieebene beginnen.
Für jedes Element der Fixstellenmenge können wir nun nach Jokerstellen suchen, die zur Ergänzung jeweils einer Fixstelle herangezogen werden können. Dies geschieht, indem wir die Pfade der Fixstellen mit den Pfaden der Jokerstellen vergleichen. Der Pfad einer Jokerstelle entsteht dadurch, dass alle Stellenbezeichner der übergeordneten Rechtecke von oben beginnend aneinander gereiht werden und vor dem Stellenbezeichner einer Jokerstelle im Pfad statt eines Stellenbezeichners drei Punkte eingetragen werden. Der Pfad einer Fixstelle stimmt dann mit dem Pfad einer Jokerstelle überein, wenn alle Teilsequenzen des Jokerstellenpfades, die hintereinander aus echten Stellenbezeichnern bestehen, in gleicher Weise direkt hintereinander als Teilsequenz im Pfad der Fixstelle vorkommen und diese Teilsequenzen ebenfalls in gleicher Reihenfolge in den Pfaden auftreten.
Die letzte Teilsequenz besteht dann aus dem gemeinsamen Stellenbezeichner von Fixstelle und Jokerstelle.
Die Teilehierarchie der so gefundenen Jokerstelle kann nun verwendet werden, um die Teilehierarchie der Fixstelle entsprechend zu erweitern. Dies geschieht, indem alle in der Jokerstelle enthaltenen Stellen mit fixer hierarchischer Einordnung (relativ zur Jokerstelle) in die Teilehierarchie der Fixstelle mit aufgenommen werden.
Für die so neu gewonnenen Fixstellen kann nun wiederum nach Jokerstellen gesucht werden. Dies geschieht so lange, bis für alle gefundenen Fixstellen alle Jokerstellen untersucht wurden und keine neuen Fixstellen mehr produziert werden können.
Um ein endloses Einhängen von Jokerstellen mit rekursivem Vorkommen von Stellenbezeichnern zu verhindern, stellen wir sicher, dass die gefundene Jokerstelle selbst nicht mehr bei der Suche nach Jokerstellen für die eigenen Teilstellen verwendet wird.

Sind die Stellen des Gesamtmodells erkannt, kann die Zuordnung der mit den Stellen verbundenen Information erfolgen. In einem einfachen Fall lösen wir diese Aufgabe so, dass die Information der Stelle im Gesamtmodell die Vereinigung der Information aller Stellenvorkommen an den Teilstellen bedeutet. Dabei fordern wir, dass diese Informationen nicht widersprüchlich sind.

Die Auslagerung bzw. Verteilung von Information auf mehrere Diagramme ist ein allgemeiner Ansatz, der nicht nur die Auslagerung einzelner Hierarchieebenen auf verschiedene Diagramme ermöglicht. Es können generell damit mehrere Teilsichten auf das

Modell dargestellt werden. Dies erhöht die Überschaubarkeit und ist unabdingbar für komplexe Anwendungsfälle. Insbesondere kann damit auch in vielen Fällen erreicht werden, dass die Stellen relativ ortsgetreu dargestellt werden können: Stellen, die sich in einer koordinatentreuen WYSIWYG-Darstellung überlagern, werden in mehrere Diagramme mit Teilsichten aufgeteilt.
Generell gesehen haben wir dann mehrere Diagramme, die jeweils nur eine Teilsicht des gesamten Modells bieten. Erst in ihrer Zusammenführung ergibt sich wieder ein Gesamtbild des Modells. Durch die Vergabe von eindeutigen Stellenbezeichnern kann definiert werden, wie die Zusammenführung erfolgen kann.
Spätestens dann, wenn das Modell vollständig spezifiziert sein soll, muss die unvollständig dargestellte Stelle an anderer Diagrammstelle im selben Ereignisstellendiagramm oder in einem anderen Diagramm vervollständigt werden.

6 Abstraktion vom konkreten Widget

In Abbildung 4-1 haben wir ein einfaches Modellierungsbeispiel gesehen, das konkrete Widgets aus einer realen Laufzeitumgebung im Dialogablauf zeigt. In den dann nachfolgenden Abbildungen wurde auf die konkrete Darstellung von Widgets verzichtet. An die Stelle von Widgets traten in der Darstellung der Modell-Grafen Knoten, die wir auch als Dialogstellen oder auch nur als Stellen bezeichnet haben, Stellen, die letztlich Ereignisse eines bestimmten Typs mit bestimmten Werten repräsentieren. Diese Stellen sind, wie bereits in Abschnitt 3.0.1 erwähnt, eine Abstraktion einer konkreten Ressource, mittels derer zur Laufzeit die zugeordneten Ereignisse realisiert werden.
Allerdings ist in den Abbildungen mancher vorangehender Beispiele, etwa dem Beispiel des Modells zur Auftragserfassung (Abbildung 4-13, Abbildung 4-14, Abbildung 4-15) eine Darstellung zu sehen, die einer möglichen konkreten Darstellung zur Laufzeit sehr nahe kommt.
Dies liegt im Wesentlichen daran, dass das vorgestellte Modell der Ereignisstellen (ESM) mit einer Darstellung konkreter Dialoge durch bestimmte Widgets stark übereinstimmt.
Die Gründe für die Übereinstimmung werden im folgenden Abschnitt diskutiert. Danach zeigen wir anhand des Beispiels RadioButton, wie ein Ereignisstellenmodell durch verschiedene Widgets zur Laufzeit realisiert werden kann und untersuchen im darauf folgenden Abschnitt die Möglichkeit der Mischung konkreter Widgetdarstellungen mit abstrakten Ereignisstellen in einem Ereignisstellendiagramm.

6.0 Ähnlichkeit von Stellendarstellung und Widgets

Besondere Übereinstimmung zwischen der Darstellung abstrakter Ereignisstellen im ESD mit konkreten Widgets liegt bei bestimmten häufig verwendeten Widget-Typen vor.
Textwidgets repräsentieren auf einer rechteckigen Fläche einen Eingabewert oder Ausgabewert in textueller Form. Die Bedeutung bzw. logische Einordnung in den Kontext wird ebenfalls durch ein Labelwidget mit entsprechendem Führungstext dargestellt. Ein Labelwidget kombiniert mit einem leeren Textwidget entspricht im Wesentlichen der Darstellung einer Stelle in unserem Modellgrafen, wenn die Stelle ohne konkreten Wert aber mit ihrer Stellenbezeichnung dargestellt ist.
Bei einer Implementierung einer Modellstelle durch eben diese Kombination von Label und Textwidget ist die Darstellung der Modellstelle also nahezu identisch mit der Darstellung zur Laufzeit.

Abbildung 6-1 Eingabestelle im Modell (links) und Implementierung durch Label- und Textwidget mit Java (rechts)

Eine ähnliche Übereinstimmung mit der Darstellung im Modell gilt z.B. für PushButtons. Auch hier ist das Wesentliche in der Darstellung des Pushbuttons die grafisch abgehobene Schaltfläche, die mit dem Text versehen ist, der die Bedeutung der Betätigung des Buttons beschreibt.
Bei Text-, Label- und Pushbutton-Widgets kommt die abstrakte Modelldarstellung der konkreten Darstellung nicht zuletzt deshalb so nahe, weil die Widgets selbst bereits einen hohen Grad an Abstraktion der Darstellung per se bieten: Die Beschreibung einer Aussage in

Form eines prägnanten Begriffes einer natürlichen Sprache in textueller Form stellt bereits eine Abstraktion dar, die im Modell kaum übertroffen werden muss.

Überhaupt sind Widgets in der Regel über ein ggf. umrandetes Flächenstück am Bildschirm repräsentiert. Es gibt bei Widgets wie bei Ereignisstellen eine hierarchische Beziehung (Parent-Child-Beziehung), die meist durch die Einbettung des untergeordneten Widgets (Child-Widgets) innerhalb der Fläche des übergeordneten Widgets (Container-Widgets) erkennbar ist, wie das auch bei der gewählten Darstellung von Teilstellen im ESD der Fall ist. Ebenso sind den Widgets ähnlich wie den Ereignisstellen bestimmte Ereignisse zugeordnet, die Aktionen bzw. Ereignisse an anderen Widgets bzw. Stellen auslösen können.

Die heute in gängigen Baukastensystemen (Toolkits, Frameworks) gebotenen Widgets können als Beispiel einer Konkretisierung von Stellen in unserem Modellsystem betrachtet werden. Im Zuge der Realisierung eines konkreten Dialogs zur Laufzeit können wir also Modellstellen auf Widgets eines Baukastensystems leicht abbilden.
Wesentlich für das Verständnis des Stellenbegriffs, wie er im ESM verwendet wird, ist allerdings, dass diese Abbildung nicht immer eine 1-1-Abbildung einer Modellstelle auf eine (räumlich verstandene) Laufzeitstelle, bzw. auf ein Widget, bedeutet. Mehrere Stellen im Modell können mitunter einem Widget entsprechen und umgekehrt kann eine Modellstelle auf mehrere Widgets zur Laufzeit umgesetzt werden. Dies liegt daran, dass ein Widget oft mehrfach für die Darstellung von Ereignissen verwendet wird, für deren Beschreibung wir jeweils eine eigene Modellstelle verwenden und dass umgekehrt in einer Modellstelle eine Menge von Ereignissen zusammengefasst sein kann, die ihre Darstellung zur Laufzeit in verschiedenen Instanzen von Widgets finden.
Zum anderen ist wesentlich, dass ein und dieselbe Stelle durch unterschiedliche Typen von Widgets konkretisiert werden kann.
Wesentlich ist letztlich auch, dass eine Stelle tatsächlich eine Abstraktion darstellt, die nicht unbedingt auf ein Widget im engeren Sinne − verstanden als Element eines bestimmten Widget-Toolkits − abzubilden ist. Dialogstellen sind auch in nicht grafischen, zeichen- und zeilenorientierten Dialogarten wie z.B. Kommandodialog oder Menüdialog zu erkennen. Die Implementierung letzterer Dialogarten muss nicht notwendigerweise mit grafischen Toolkits unter Verwendung von Widgets erfolgen.

6.1 Abbildung der Ereignisstellen auf konkrete Widgets

Wir wollen die eben erwähnte Tatsache, dass eine Stelle durch unterschiedliche Typen von Widgets konkretisiert werden kann, in diesem Abschnitt anhand des Beispiels der Abstraktion eines RadioButtons in einem ESM erläutern.
Ein RadioButton unterstützt die Eingabe zweier Werte, nämlich „An“ und „Aus“ oder gedrückt“ und „nicht gedrückt“, bzw. „true“ und „false“. Die Eingabe wird für den Benutzer visualisiert und steht für das System zur Abfrage zur Verfügung. Außerdem kann der Button auch vom Programm aus gesetzt werden. Eine Besonderheit des RadioButtons ist, dass die gleiche Eingabeaktion des Benutzers im Wechsel eine unterschiedliche Bedeutung signalisiert, nämlich abwechselnd „An“ oder „Aus“, respektive „true“ oder „false“.
Eine vom konkreten Widget abstrahierende Darstellung im ESM kann wie folgt gewählt werden:
Wir nehmen vier Modellstellen, also S = { S_0, S_1, S_2, S_3 }
Die Menge der Ereignisse unterteilen wir in vier Klassen, die wir den vier unterschiedlichen Stellen zuordnen.
1. e0: Eingabe „An“ durch Benutzer (S_0)

2. e1: Ausgabe „Aus“ durch System (S_1)
3. e2: Ausgabe „An“ durch System (S_2)
4. e3: Eingabe „Aus“ durch Benutzer (S_3)

Also E = { e0 = („An“ , S_0, I_B),
e1 = („Aus“, S_1, O_S),
e2 = („An“ , S_2, O_S),
e3 = („Aus“, S_3, I_B)
}

Es gelten folgende Bezüge zwischen den Stellen:
Nach dem Start des Dialogs werden S_0 und S_1 geöffnet.
Ereignis in S_0 schließt S_0 und S_1, öffnet S_2 und S_3.
Ereignis in S_3 schließt S_3 und S_2, öffnet S_0 und S_1.

Also RO = { (S_0, S_2), (S_0, S_3), (S_3, S_0), (S_3, S_1) }
und RC = { (S_0, S_0), (S_0, S_1), (S_3, S_3), (S_3, S_2) }

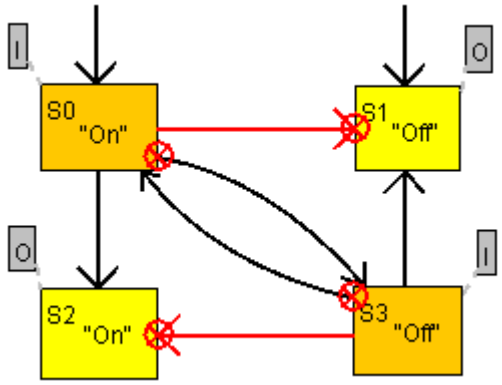


Abbildung 6-2 Abstraktion eines Radio-Buttons mit 4 ESD-Stellen

Das Öffnen der Stellen zur Eingabe für den Benutzer bewirkt das Warten des Systems auf die asynchron erfolgende Eingabe. Das Öffnen der Ausgabestellen bewirkt die unmittelbare (synchrone) Ausgabe der Information durch das System. Das wechselseitige Öffnen und Schließen der Eingabestellen gewährleistet, dass nur jeweils eine der Eingaben zu einem Zeitpunkt erfolgen kann. Das gleiche gilt für die Ausgabe. Die Visualisierung des aktuellen Zustands für den Benutzer erfolgt über die Ausgabestellen.
Eine Abbildung der abstrakten Stellen auf konkrete Stellen zur Laufzeit kann nun in unterschiedlichsten Varianten erfolgen. Wir wollen im Folgenden zwei Varianten vorstellen. Dabei gehen wir davon aus, dass uns zur Implementierung ein Baukastensystem mit Widgets, z.B. ohne Beschränkung der Allgemeinheit Widgets der Swing-Klassen in Java [FISCHE00], zur Verfügung steht.

6.1.0 Konkretisierung als Instanz der Klasse JRadioButton:

Alle vier Modellstellen werden auf eine Instanz der Klasse JRadioButton abgebildet. Zum Startzeitpunkt wird der RadioButton instantiiert und in den Zustand selected = false versetzt. Der im Modell dargestellte Dialog benötigt bei einer Implementierung mithilfe der Klasse JRadioButton lediglich dieser Instantiierung. Alle durch das Modell ausgedrückten

wesentlichen Merkmale des Dialogs werden allein durch die Implementierung der Klasse selbst repräsentiert und bedürfen keines zusätzlichen Implementierungsaufwands.
D.h. die Ausgabe, also die Visualisierung des Wertes „Aus“ (Stelle S_1) wird mit dem Zustand des Buttons „selected = false“ erreicht, entsprechend die Visualisierung von „An“ (Stelle S_2) über den Zustand „selected = true“.
Die beiden Eingabeereignisse an den Stellen S_0 und S_3 werden automatisch durch das Eingabeereignis vom Typ „Mouse-click“ auf dem RadioButton realisiert. Wir haben also eine Abbildung der beiden logischen Eingaben „An“ und „Aus“ auf ein konkretes Ereignis „Mouse-click“, das standardmäßig mit der Konstruktion der Instanz aktiviert ist.
Das Öffnen der einzelnen Modellstellen wird auf die leere Operation abgebildet, falls die Instanz bereits existiert. (Leere Operation sei hier so verstanden, dass wir bei der Implementierung mit Hilfe der Klasse JRadioButton keinen zusätzlichen Aufwand benötigen, da die Operation bereits im Code der Klasse implizit als Realisierung des Zustandswechsels enthalten ist.) Ebenso wird das Schließen der Stellen auf die leere Operation abgebildet, solange nur eine der anderen Modellstellen geöffnet ist und diese im Zuge der Interpretation des Ereignisses auch nicht geschlossen werden soll. Letztere Voraussetzungen sind im dargestellten Modell immer gegeben: Das Schließen einer Stelle geschieht im Modell immer zusammen mit dem Öffnen einer anderen Stelle oder es bleiben beim Schließen einer Stelle andere Stellen geöffnet.
Das Öffnen und Schließen der Modellstellen aufgrund von Ereignissen an einer der Stellen bewirkt also kein Öffnen oder Schließen der Widget-Instanz in der hier gewählten Implementierung. Stattdessen erfolgt ein Zustandswechsel, der sich über die Wertänderung des Attributs selected bzw. eine Visualisierung des Zustands an der Oberfläche ausdrückt. Beispielsweise bedeutet das Schließen der Stelle S_1 ein Beenden der Anzeige des Zustandes „Aus“ an der Oberfläche und ein Löschen des Wertes false im Attribut selected. Dies geht im Modell einher mit dem Öffnen der Stelle S_2, was in der Implementierung der Visualisierung des Zustandes „An“ und dem Setzen des Wertes true im Attribut selected entspricht.
Damit ist das im Modell beschriebene Verhalten vollständig auf ein entsprechendes Verhalten in der Implementierung abgebildet.

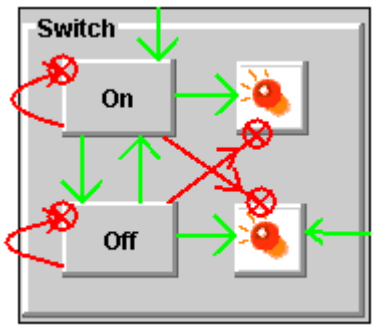


Abbildung 6-3 Alternative Realisierung eines RadioButtons mit vier Widgets
Das abstrakte Modell eines RadioButtons (Abbildung 6-2 Abstraktion eines Radio-Buttons mit 4 ESD-Stellen) wird hier alternativ auf vier verschiedene Widgetinstanzen abgebildet. Die vier abstrakten Ereignisstellen sind durch zwei Widgets der Klasse JButton und zwei Widgets der Klasse JLabel realisiert. Eingebettet sind die vier Widgets hier in eine Instanz der Klasse JPanel. Zur Darstellung der Stellenereignisse werden dabei bestimmte Merkmale dieser Widgets verwendet.
Die Abbildung zeigt ein Ereignisstellendiagramm mit WYSIWYG-Darstellung (siehe auch unten Abschnitt 6.2). In diesem Diagramm sehen wir pro Widget nur exemplarisch die Darstellung des geöffneten Zustands der jeweiligen abstrakten Stelle, also die Buttons links im sensitiven Zustand.und die Labels mit leuchtenden Lampen.

6.1.1 Konkretisierung mit vier Instanzen

Das Modell des RadioButtons läßt sich auch mit zwei Instanzen der Klasse JButton und zwei Instanzen der Klasse JLabel realisieren:
Jede der vier Modellstellen wird dabei auf jeweils eine eigene Widget-Instanz abgebildet. Für die Ausgabe der Information „Aus“ bzw. „An“ verwenden wir die beiden Widgets der Klasse JLabel. Dabei werden in unserem Beispiel in jedem Label die Werte „Aus“ bzw. „An“ auf zwei Icons, eine leuchtende und eine nicht leuchtende Lampe abgebildet. Das Öffnen und Schließen der Modellstellen S_1 und S_2 würde bei einer Standardimplementierung durch sichtbar machen bzw. unsichtbar machen der beiden entsprechenden Widgets erfolgen. Alternativ dazu wählen wir hier jedoch die Änderung der beiden Icons des jeweiligen Labels. Die Eingabeereignisse „An“ und „Aus“ realisieren wir diesmal mit zwei Buttons der Klasse JButton und den zugeordneten Ereignissen vom Typ Mouse-click. Dabei werden gemäß Standardimplementierung die logischen Eingabewerte über den Text der Buttons, im Beispiel der Abbildung in der englischsprachigen Variante „On“ und „Off“, visualisiert.

Stelle abstrakt	*Ereignis*		*Widget konkret*		*Ereignis*	*Attribut*		*Operationen*	
	Typ	***Wert***	***Klasse***	***Instanz-ID***	***Typ***	***Typ/Name***	***Wert***	***Öffnen***	***Schließen***
S_0	**I**	**„An“**	JButton	„OnButton“	Mausklick			Widget kreieren, sichtbar setzen, sensitiv setzen	Widget insensitiv setzen, oder unsichtbar setzen, oder löschen
S_1	**O**	**„Aus“**	JLabel	„OutLabel“	Ausführen Operation zum Setzen Attribut-wert	LabelIcon	„LampeAn“	Widget kreieren, sichtbar setzen, “LampeAn”-Icon setzen	„LampeAus“-Icon setzen, oder Widget unsichtbar setzen, oder löschen
S_2	**O**	**„An“**	JLabel	„OnLabel“	Ausführen Operation zum Setzen Attribut-wert	LabelIcon	„LampeAn“	Widget kreieren, sichtbar setzen, „LampeAn“-Icon setzen	“LampeAus”-Icon setzen, oder Widget unsichtbar setzen, oder löschen
S_3	**I**	**„Aus“**	JButton	„OutButton“	Mausklick			Widget kreieren, sichtbar setzen, sensitiv setzen	Widget insensitiv setzen, oder unsichtbar setzen, oder löschen

Abbildung 6-4 Tabelle zur Implementierung des RadioButton-Modells mit vier Widgets
Die Tabelle zeigt schematisch die Umsetzung der vier abstrakten Ereignisstellen des RadioButton-Modells auf vier Java-Widgets mit konkret verwendeten Ereignissen, Attributen (Ressourcen) und Operationen. Da die zum Triggern der Eingabeereignisse benutzte Buttonklasse JButton kein Attribut zum Speichern des Eingabewertes besitzt, entfallen hier die Angaben zu Attributtyp und -Wert. Die Operationen zum Ermöglichen (Öffnen) des Ereignisses sind alle erforderlich und werden, soweit sie nicht von vorher wirksam sind, beim Übergang zur Stelle ausgeführt. Die Operationen zum Schließen können alternativ für die Implementierung gewählt werden.

6.1.2 Einbettung des Modells RadioButton in anderen Dialog

In der Regel ist ein RadioButton natürlich in einen umfangreicheren Dialog eingebettet und es gibt dann häufig Abhängigkeiten zu anderen Teildialogen.
Im ESM können sich Abhängigkeiten eines Dialogs, wie er z.B. durch das Modell „RadioButton“ (im Folgenden auch einfach RadioButton oder rad genannt) beschrieben ist, zu anderen Dialogen aufgrund der in der formalen Modelldarstellung gegebenen Relationen wie folgt ergeben:

1. Der RadioButton ist hierarchisch in einen Dialog als Teildialog eingeordnet.
   Zur Erinnerung an die formale Definition: Es gibt dann ein Element $(s,rad) \in HS$, mit $s \in S$.
2. Aufgrund eines Ereignisses an einer anderen Dialogstelle wird der durch den RadioButton ausgedrückte Dialog eröffnet oder geschlossen.
   Zur Erinnerung an die formale Definition in Kap.5: Es gibt dann ein Element $(s,rad) \in ROH'$ oder $(s,rad) \in RCH'$
3. Aufgrund eines Ereignisses an einer anderen Dialogstelle werden Ereignisse an einer der Teilstellen eröffnet bzw. gesperrt (Teilstelle wird geöffnet bzw. geschlossen). Gegebenenfalls wird ein der Teilstelle zugeordnetes Ausgabeereignis propagiert.
   Zur Erinnerung an die formale Definition: Es gibt dann ein Element $(s,S_i) \in ROH'$ oder $(s,S_i) \in RCH'$ für ein S_i, mit S_i Teilstelle von rad, i = 0,1,2 oder 3.
   Ggf. existiert dann eine Propagationsfunktion p mit $\pi_{RO}((s_{RO}S_i)) = p$, so dass das propagierte Ereignis $e = (p(\omega s), S_i, O_S)$;
   ωs ist dabei die im entsprechenden Zustand gültige Wertezuordnung zu Stellen in einem definierten Stellentupel s^k.

Wie schlagen sich nun die oben geschilderten möglichen Beziehungen anderer Modellstellen zu den Modellstellen des RadioButton in der Implementierung nieder? Wenn insbesondere das Modell RadioButton durch die Klasse JRadioButton realisiert wird, kann die Konkretisierung folgendermaßen aussehen:

Hierarchische Einordnung als Teildialog
Die hierarchische Einordnung als Teildialog eines anderen Dialogs erfolgt entweder über die übliche im Widgetkonzept gegebene hierarchische Zuordnung: Die Instanz der Klasse JRadioButton wird als Kind einem Container-Widget zugeordnet. Dies setzt voraus, dass der übergeordnete Dialog als Container-Widget realisiert werden kann, wovon wir in der Regel ausgehen können. Andernfalls muss die hierarchische Beziehung der Dialogstellen des Modells gesondert implementiert werden.

Öffnen oder Schließen über Ereignis an anderer Dialogstelle
Wenn wir wiederum davon ausgehen können, dass die andere Dialogstelle als Instanz einer Widgetklasse instantiiert werden kann, bedeutet das Öffnen der Modellstelle RadioButton im Wesentlichen das Kreieren einer Instanz als Kind eines passenden Container-Widgets, falls nicht bereits erfolgt, und das Sichtbar-Machen des Widgets (setVisible(true)). Die beiden Operationen werden üblicherweise in einer Ereignisroutine aufgerufen, die dem entsprechenden Ereignis des Widgets zugeordnet ist. Dies setzt wiederum voraus, dass das im Modell definierte auslösende Ereignis auf ein entsprechendes Ereignis des Widgets abgebildet werden kann.

Öffnen und Schließen von Teilstellen
Gehen wir davon aus, dass die Einwirkung von anderen Dialogstellen auf das Modell des RadioButtons nur so erfolgen soll, dass die charakteristische Funktionsweise des Modells dadurch nicht zerstört wird. Als sinnvolle Einflussnahme auf Teilstellen verstehen wir dann

a) eine Änderung des Zustands von „An“ nach „Aus“ oder umgekehrt,
b) die Eingabe für den Benutzer zu sperren.

Im Folgenden sollen beide Fälle bezüglich Darstellung im Modell und bezüglich Implementierung behandelt werden:

Änderung des Zustands „An“ bzw. „Aus“
1. Darstellung im Modell
Im Modell bedeutet dies, Übergänge von der angenommenen anderen Dialogstelle zum Öffnen und Schließen aller vier Teilstellen des RadioButtons zu definieren, die mit entsprechenden Bedingungen (Conditions) versehen sind. Die Conditions sind dabei ggf. unter Einsatz von Hilfsstellen so formuliert, dass sie den aktuellen Wert des RadioButtons prüfen und abhängig davon die entsprechenden Stellen geöffnet bzw. geschlossen werden.

2. Implementierung
In der Implementierung mit der Instanz der Widgetklasse JRadioButton bedeutet nun wiederum das Öffnen und Schließen der Stellen lediglich ein Umsetzen des Zustandes des RadioButtons (setSelected(true/false)) nach Prüfung des Attributes selected (mittels isSelected()).

Sperren der Benutzereingabe
In der Sprache des Modells bedeutet das Sperren der Eingabe für den Benutzer beide Eingabestellen S_0 und S_3 zu schließen. In der Implementierung bedeutet dies ein „Insensitiv-Setzen“ des RadioButtons mittels Methodenaufruf setEnabled(false).

6.2 WYSIWYG-Darstellung der Ereignisstellen

Im vorangegangenen Abschnitt haben wir anhand eines Beispiels diskutiert, wie abstrakte Ereignisstellen auf konkrete Widgets in der Implementierung zur Laufzeit abgebildet werden können.
Im Folgenden wollen wir nun erarbeiten, wie bereits in einer Darstellung im Modell eine WYSIWYG-Darstellung der Ereignisstellen unter Verwendung konkreter Widgets verwirklicht werden kann. Wir werden zeigen, dass damit eine gemischt abstrakt-konkrete Darstellung im Modell möglich wird, die einen relativ „nahtlosen“ Übergang von abstrakter zu konkreter Modellierung und umgekehrt innerhalb einer Modelldarstellung eröffnet.
Die WYSIWYG-Darstellung in Abbildung 4-1 Beispiel für WYSIWYG-Modellierung zeigt konkrete Widgets in einzelnen Bildern. Dabei repräsentiert jedes Bild ein Ereignis des Typs Eingabe oder Ausgabe. Soweit notwendig wird das Ereignis, das „auf dem Widget“ bzw. mit der Darstellung des Widgets erfolgt, noch besonders gekennzeichnet. Damit wird sowohl dem Betrachter als auch für das System klar, welche Art von Ereignis, d.h. welcher Typ, welcher Wert und ggf. welche Stelle genau (falls das dargestellte Widget ein Ereignis an einer Teilkomponente aufweist), gemeint ist.
Wir müssen nun im Folgenden untersuchen, inwieweit eine WYSIWYG-Darstellung nach wie vor möglich ist, nachdem im Ereignisstellendiagramm von einer Darstellung von Folgeschritten einzelner Ereignisse abgewichen werden kann und das Ereignisstellenmodell die in Abschnitt 4 und 5 besprochenen Schritte der Reduktion der extensiven Darstellung zu einer komprimierten intensiven Darstellung erlaubt.

1 zu 1 Darstellung Modellstelle mit einem Ereignis zu Widget mit einem Ereignis
Das Konzept der Ereignisstellen in einem ESM unterstützt neben allen Möglichkeiten der komprimierten Darstellung nach wie vor eine extensive Darstellung, bei der eine Ereignisstelle genau ein Ereignis repräsentiert. Hier kann die WYSIWYG-Darstellung so erfolgen, wie dies in Abbildung 4-1 angedeutet ist: Das zur Laufzeit für die Realisierung verwendete Widget wird visualisiert zusammen mit einer geeigneten Darstellung des dort erfolgenden Ereignisses der Eingabe oder Ausgabe.

Zusammenfassung mehrerer abstrakter Modellstellen zu einer WYSIWYG-Stelle
In manchen Fällen ist es sinnvoll, in der WYSIWYG-Darstellung mehrere abstrakte Ereignisstellen zu einer WYSIWYG-Stelle zusammenzufassen. Dies gilt z.B. dann, wenn ein Widget sowohl eine Ausgabe von Information als auch die Eingabe von Information vorsieht und die Ausgabeinformation bereits durch die Darstellung des Widgets am Bildschirm realisiert wird. Als Beispiel dient die oben geschilderte Modelldarstellung eines RadioButtons. In abstrakter Darstellung haben wir dort vier Modellstellen angegeben, die in WYSIWYG-Darstellung beispielsweise auf zwei WYSIWYG-Stellen zurückgeführt werden kann: Die WYSIWYG-Stelle des RadioButtons repräsentiert dann sowohl die Ausgabe des aktuellen Zustandes als auch die mögliche Eingabe, also im Sinne der abstrakten Darstellung zwei Ereignisse. Bei den von der Stelle ausgehenden Kanten ist es dann notwendig, zu kennzeichnen, auf welches der beiden repräsentierten Ereignisse sich die Kante bezieht. Wir haben hier also allgemein gesprochen den Fall, dass mehrere Modellstellen mit einzelnen Ereignissen durch eine WYSIWYG-Stelle mit mehreren Ereignissen repräsentiert wird.

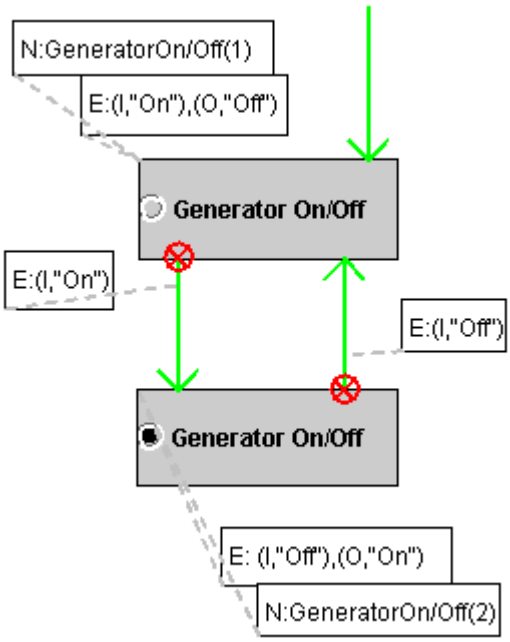


Abbildung 6-5 Darstellung des RadioButton-Modells in zwei WYSIWYG-Stellen
Die Abbildung zeigt einen RadioButton in WYSIWYG-Darstellung seiner beiden Zustände (Klasse JRadioButton in Java). Es handelt sich um ein Ereignisstellendiagramm mit zwei Stellen namens „GeneratorOn/Off(1)“ und „GeneratorOn/Off(2)“. Die eine Stelle eröffnet die Eingabe von „On“ bei gleichzeitiger Ausgabe-Information „Off“. An der anderen ist umgekehrt die Eingabe von „Off“ möglich und es wird „On“ ausgegeben. Dies ist durch Annotation an der jeweiligen Stelle sichtbar.
GeneratorOn/Off(1) ist Startstelle, was durch den Triggerpfeil ohne Ausgangsstelle („aus dem Leeren“) angedeutet ist.

Die Pfeile mit dem durchkreuzten Kreis legen fest, dass bei der durch Annotation angegebenen Ereignismenge die Ereignismenge der Ausgangsstelle gesperrt und die Ereignismenge an der Zielstelle geöffnet wird (Zusammenfassung von Öffne- und Schließepfeil in jeweils einem).
Beide Stellen werden auf dasselbe Widget zur Laufzeit abgebildet. Im vorliegenden Prototypen des Modellerditors geschieht das in diesem Fall dadurch, dass die Namen der beiden Stellen sich nur im so genannten Indexteil „(1)“ und „(2)“ unterscheiden. Generell wird bei der Umsetzung des Diagramms in die Implementierung Information für die Abbildung der Stellen auf Widgetinstanzen benötigt. Diese kann durch zusätzliche Annotation den Stellen mitgegeben werden oder wird aus der Namensgebung der Stellen automatisch ermittelt. Desweiteren ist für die Implementierung eine Zuordnung der hier gezeigten logischen Ereignisse auf die möglichen Ereignisse der Widgetinstanz nötig. Ein intelligenter Abbildungsmechanismus erkennt in diesem Fall, dass nur ein logisches Eingabeereignis pro Stelle existiert und bildet standardmäßig dieses Ereignis auf „Mausklick“ ab. Ebenso kann hier standardmäßig die Ausgabe von „Off“ auf isSelected=false erfolgen.

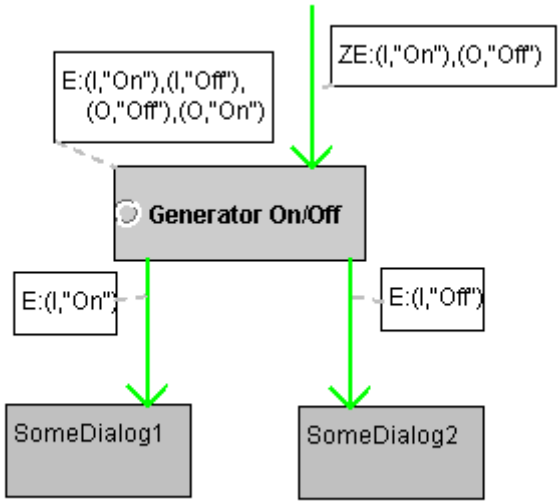


Abbildung 6-6 WYSIWYG-Darstellung eines RadioButton mit nur einer Stelle
Hier wird das RadioButton-Modell mit nur einer Stelle gezeigt. Der WYSIWYG dargestellte RadioButton kann nur exemplarisch einen Zustand darstellen. Die Stelle des Ereignisstellendiagramms fasst alle vier abstrakten Stellen (Abbildung 6-2 Abstraktion eines Radio-Buttons mit 4 ESD-Stellen) zu einer Stelle zusammen, der alle möglichen Ereignisse annotiert sind.
Die Zustandswechsel des RadioButtons sind hier nicht mehr durch Kanten gezeigt. Dies ist allerdings für eine korrekte Implementierung nicht nötig, da die Zustandswechsel in der Implementierung der Klasse JRadioButton bereits enthalten sind. Allerdings muss jetzt explizit eine Zuordnung der logischen Ereignisse zu den möglichen konkreten Ereignissen des RadioButtons erfolgen. Außerdem muss durch Annotation gekennzeichnet werden, welche z.B. der möglichen Eingabe-Ereignisse einen Stellenübergang bewirken, wenn nur bestimmte Ereignisse an der Stelle als Triggerereignis in Betracht kommen.

WYSIWYG-Darstellung einer abstrakten Ereignisstelle mit mehrelementiger Ereignismenge

Wir haben eben alternative WYSIWYG-Darstellungen eines Modells diskutiert, in dem primitive abstrakte Stellen an WYSIWYG-Stellen zusammengefasst wurden. Umgekehrt können im ESM bekanntermaßen abstrakte Ereignisstellen mehrere Ereignisse zusammenfassen, die unterschiedliche Werte und unterschiedliche Typen besitzen. Die Ereignisse mit ihren unterschiedlichen Werten können dabei hintereinander eintreffen und/oder stellen Alternativen für ein zeitlich gesehen einmaliges Ereignis dar.
Wenn wir die abstrakte Stelle 1 zu 1 mit einer WYSIWYG-Stelle darstellen wollen, haben wir prinzipiell dieselbe Problemstellung wie bei der oben geschilderten Zusammenfassung mehrerer abstrakter Stellen an einer WYSIWYG-Stelle: Die WYSIWYG-Darstellung ist nur eine Momentaufnahme, also ein bestimmter Zustand bzw. eine bestimmte alternative Ausprägung der Stelle mit einer ganz konkreten Belegung von Werten.

Hier gibt es wie oben verschiedene Möglichkeiten der Darstellung, die einer gewünschten WYSIWYG-Charakteristik unterschiedlich nahe kommen.

1. Exemplarische Darstellung mit Annotationen
Wir können das Problem so lösen, wie wir es oben am RadioButton-Beispiel auch für den Fall der Zusammenfassung mehrerer abstrakter Stellen mit einelementigen Ereignismengen geschildert haben: Die WYSIWYG-Darstellung wird nur als exemplarische Darstellung aufgefasst und soweit nötig werden Annotationen angefügt. So können Wertebereich und Ereignistypen und/oder Ereignismengen annotiert werden. Wertübertragungen an die Stelle können mithilfe von Propagationsfunktionen als dynamischer Vorgang spezifiziert werden. Triggerereignisse unterschiedlicher Ausprägung können als Annotationen an die entsprechenden Triggerkanten angebracht werden (siehe dazu auch Abbildung 6-6 WYSIWYG-Darstellung eines RadioButton mit nur einer Stelle).

2. Auflösung der Ereignisstelle in mehrere „WYSIWYG-Stellen“ pro Einzelwert.
Zur ursprünglichen extensiven WYSIWYG-Darstellung nach Beispiel in Abbildung 4-1 gelangen wir wiederum, wenn wir die abstrakte Ereignisstelle mit ihren mehreren möglichen Werten in alternative WYSIWYG-Stellen auflösen, die jeweils eines der einzelnen Ereignisse konkret darstellen. Dies macht nur da Sinn, wo die Anzahl der möglichen Werte nicht zu groß ist. Dies kann formal auch betrachtet werden als die Auflösung einer abstrakten Modellstelle in mehrere zunächst abstrakte Stellen, die dann jeweils auf eine WYSIWYG-Stelle abgebildet werden.

3. Exemplarische Darstellung und Annotation der Variation
Eine besondere Art, Varianten von Ereignissen an einer WYSIWYG-Stelle zu beschreiben, ist die, dass wir exemplarisch eines der Ereignisse im WYSIWYG-Modus zeigen und das Exemplar wiederum mit einer speziellen Annotation versehen, die beschreibt, in welcher Weise die Darstellung in den alternativen Fällen abweichen kann. Dabei wird die Information, die in der WYSIWYG-Darstellung enthalten ist, nicht nur als exemplarische Visualisierung für den Benutzer des Modelleditors, sondern weitgehend als Spezifikation zur Umsetzung in die Implementierung genutzt. Dazu sei insbesondere auf das folgende Kapitel verwiesen.

7 Exemplarische Darstellung und Variation

Im folgenden Kapitel wird eine informelle Einführung in den durch diese Arbeit neu eingeführten Begriff der exemplarischen Darstellung und Variation (kurz **Exemplarvariation**) gegeben. Nach einer Skizzierung der Problemstellung und der daraus resultierenden Zielsetzung werden die wichtigen Aspekte zur Definition einer Exemplarvariation festgehalten und an zwei Beispielen erläutert.

7.0 Problemstellung

Der Bottom-up-Ansatz mit dem Ziel des konkreten Darstellens der Dialogelemente und des Ablaufs kommt bisher u. a. an seine Grenzen,

- wenn prinzipiell sehr viele alternative Elemente aufzuzeichnen sind und die Darstellung schlicht an der Menge der darzustellenden Elemente scheitert,
- wenn insbesondere einzelne Elemente in alternativen Situationen häufig ihre Eigenschaften ändern. Dies ist z.B. dann der Fall, wenn ein Dialogelement durch direkte Manipulation z.B. durch Ziehen der Maus in Position oder Größe verändert wird (dynamische grafische Elemente).
- wenn der konkrete Ablauf und die Ausgestaltung des Dialogs von Kriterien abhängt, die erst im konkreten Einsatzfall, der konkreten Anwendung in einem spezifischen Kontext dynamisch bestimmt werden können.

Ein Beispiel für letzteren Fall ist die Darstellung eines Suchergebnisses einer Datenbankrecherche: im einfachsten Fall ist die Ausgabe in den Ergebniswerten variabel und die Struktur der Ausgabe noch fix (z.B. fixes Ausgabeformular). Häufig ist aber auch die Struktur der Ausgabe flexibel zu gestalten, etwa variabel in der Anzahl der Felder des Formulars oder variabel in Spalten und Zeilen etwa bei tabellarischer Ausgabe. Tabellen- oder Formularfelder können selbst wieder rekursiv Tabellen oder Formulare beinhalten. Ist das Ergebnis der Suche z.B. eine Baumstruktur, ist häufig neben der Anzahl der jeweiligen Unterknoten auch z.B. die Anzahl der Hierarchieebenen variabel.

Üblicher genereller Weg ist zur Lösung oben genannter Probleme die Verallgemeinerung, Zusammenfassung, Abstraktion. Im Ereignisstellenmodell sind wir diesen Weg teilweise bereits gegangen, indem wir z.B. Dialogstellen mit verschiedenen Werten zusammenfassen zu Stellen mit Wertebereichen, Hierarchien bilden und indem wir Schleifen und Rekursivität erlauben. Die bisher gebotenen Sprachelemente sind aber in vielen der oben angeschnittenen Fälle noch nicht ausreichend.
So haben wir beispielsweise noch kein komfortables Sprachelement im ESM dafür, dass eine variable Anzahl von Dialogelementen bei Eintritt eines Ereignisses geöffnet werden soll. Wir können dies zwar ausdrücken, indem wir z.B. die Maximalanzahl der Elemente explizit aufführen und über jeweils ein Prädikat abprüfen, ob der jeweilige Stellenübergang zum Öffnen des Dialogelements durchzuführen ist. Da die Maximalanzahl aber sehr groß bzw. unendlich sein kann, ist diese Lösung ungenügend.

Wird während des Entwurfs eines Systems festgestellt, dass der gemachte Ansatz nur einige konkrete Fälle von einer noch nicht vollständig absehbaren Menge von Fällen abdeckt, ist in gängigen Entwicklungssystemen ein übliches notwendiges Vorgehen, dass der zunächst gemachte konkrete Ansatz verworfen und ein neuer Ansatz mit anderen Sprachmitteln begonnen wird. Denken wir z.B. an die Darstellung von Icons in einer Toolbar. In einem

frühen Stadium des Systementwurfs seien die Icons der Toolbar bekannt und ihre Anzahl auf einige wenige beschränkt. Der Entwickler modelliert die Toolbar deshalb, indem er beispielsweise in direkter Manipulation die konkreten bekannten Icons als fixe Teilelemente in die Toolbar legt. Zu einem späteren Stadium des Systementwurfs oder vielleicht erst nachdem das System bereits im Einsatz ist, wird erkannt, dass die Anzahl und damit auch die Ausprägung der Icons in der Toolbar variabel zu gestalten ist und unter Umständen sogar vom Benutzer zur Laufzeit verändert werden kann.
Gängige Vorgehensweise ist dann, dass die fix entworfene Toolbar aus der Anwendung beseitigt wird und stattdessen auf der Ebene einer Programmiersprache ein konstruktiver Ansatz zum dynamischen Aufbau der Toolbar gewählt wird.

7.1 Zielsetzung der Exemplarvariation

Ziel ist nun, zu einer allgemein gültigen dynamischen Lösung zu kommen, ohne dabei die konkreten Ansätze in Entwurf und Realisierung zu verlieren und die Architektur des Modells umwerfen zu müssen. Aus einer Skizze mit konkreter exemplarischer Darstellung oder aus einer konkreten, aber nicht umfassenden Darstellung, die bereits implementiert ist, soll die variable allgemeine Lösung unter maximaler Nutzung der zunächst durchgeführten fixen konkreten Entwurfs- und Entwicklungsschritte formuliert werden.
Ein weiterer Vorteil dieses Vorgehens ist, dass nach wie vor auch dem Entwickler und dem zukünftigen Benutzer des zu entwerfenden bzw. zu entwickelnden Systems eine unmittelbare konkrete Vorstellung vom System durch die exemplarische Darstellung erhalten bleibt.

Aus dieser Zielsetzung leitet sich nun der Ansatz der „exemplarischen Darstellung mit Variation“ ab, der die Darstellung eines oder mehrerer Exemplare im Modell vorsieht, dem oder denen Information zugeordnet wird, die die verallgemeinernde Lösung beschreibt.
Die dem Exemplar zugefügte Variationsbeschreibung ist grafischer oder textueller Natur und definiert die allgemeine Lösung, die unterschiedlichen Varianten des Exemplars.

Mit der Darstellung des Exemplars bleibt das Modell so konkret wie möglich. Ausgehend vom konkreten Beispiel werden die verschiedenen Abwandlungen beschrieben. Dies geschieht in einer Sprache, die sich weitgehend am Ergebnis orientiert und weitgehend als deskriptiv verstanden werden kann.
Zentrale Elemente der Sprache sind die Exemplare als zu variierende Objekte sowie Aspekte, Eigenschaften, Attribute dieser Objekte in ihren möglichen Abwandlungen oder Alternativen. Dabei ist zu berücksichtigen, dass diese Abwandlungen bei Eintritt bestimmter Ereignisse oder Bedingungen, in einem bestimmten Kontext erfolgen, bzw. wahrgenommen werden können.
Wenn wir von unserem Modell ausgehen, sind die Objekte der Variation grundsätzlich die Elemente unseres Modells, in erster Linie die Stellen. Als Aspekte der Variation einer Stelle kommt alles in Frage, was sich an Aussagen zu einer Stelle im Modell befindet, also die Struktur der Stelle, d.h. die Relation X im Bezug zu s aus S, die Aussage, wann die Stelle geöffnet bzw. geschlossen wird, was durch Elemente in RO bzw. RC festgehalten ist, sowie die Werte und Ereignisse, die mit einer Stelle in Verbindung stehen (im Modell ausgedrückt durch $\sigma^{-1}(s)$), sowie Wertebereiche und Propagationsfunktionen.

7.2 Merkmale der Exemplarvariation

Als wesentliche Merkmale der Variation ergeben sich:

1. der Gegenstand der Variation, bestehend aus Objekten mit ihren Aspekten bzw. den die Aspekte eines Objektes beschreibenden Modellelementen.
2. eine Beschreibung, unter welchen Bedingungen und Abhängigkeiten welche Varianten zutreffen bzw. eine Beschreibung der Vorschrift zur Erzeugung der jeweiligen Variante (Bildungsgesetz).
3. der Zeitpunkt, zu dem die Variation wirksam wird. Hierbei ist zu verstehen, dass es bei einem Dialogsystem um ein zeitlich sich veränderndes System handelt und die Darstellung eines Exemplares die Darstellung eines Objektes im System zu einem bestimmten Zeitpunkt bedeuten kann. D.h., das Exemplar repräsentiert eine Momentaufnahme im System. Die Variation besteht dann in der Veränderung dieses Exemplars bzw. bestimmter Aspekte des Exemplares zu verschiedenen Zeitpunkten während des Ablaufes der Anwendung. Das Exemplar variiert innerhalb der Laufzeit einer Anwendung („fortlaufende dynamische Variation“). In einem anderen Fall kann die Variation verstanden werden als die einmalige Abwandlung und der einmalige Austausch eines Exemplares im Modell gegen eine Variante des Exemplares für eine aktuelle Anwendung, die über die gesamte Dauer der Anwendung anhält. Es handelt sich dann um eine Variantenbildung pro Anwendung, in gewissem Sinne um eine „statische“ Variation des Modellexemplares.
4. Letztere Überlegung führt zu einem weiteren Kriterium zur Beschreibung der Variation, das wir gesondert hervorheben können, nämlich den Zeitpunkt bzw. das Ereignis, zu dem die jeweilige Variation entschieden werden kann, d.h. zu dem die konkreten Variationsergebnisse oder Varianten berechnet werden können. Dabei können zwei Extremfälle unterschieden werden: in einem Fall wird die Variation unmittelbar zu dem Ereignis entschieden, zu dem dann auch die Variation stattfindet, und zwar dynamisch zur Laufzeit. Im anderen Extremfall steht die Variation bereits zum Modellierungszeitpunkt fest. In letzterem Fall dient die exemplarische Darstellung dann nur der Übersichtlichkeit und der Ersparnis von Darstellungsaufwand. Ein Fall zwischen diesen beiden Extremen ist, wenn z.B. die Variante zum Startzeitpunkt der Anwendung entschieden wird (in einer Initialisierungsphase des Systems aufgrund von Startparametern).
5. Als letztes Kriterium soll hier ein Variationsmerkmal eingeführt werden (in folgenden Beispielen als Variationstyp bezeichnet), das zwischen einer sogenannten vollständigen Variation und einer inkrementellen Variation unterscheidet. Bei einer vollständigen Variation werden alle durch eine Exemplarmenge repräsentierten Varianten vollständig gegen die neu bestimmten Varianten ausgetauscht. Bei einer inkrementellen werden die neuen Varianten der alten Variantenmenge hinzugefügt.

7.3 Einfaches Variationsbeispiel

Nehmen wir an, wir arbeiten mit einem Baukastensystem, in dem u.a. Dialog-Objekte vom Typ Toolbar und Dialog-Objekte vom Typ Icon zur Verfügung stehen.
Ein Icon sei durch folgende spezifischen Attribute beschrieben:
Symbol (Bitmap), beschreibender Text, bei Selektion auszulösender Dialog.
Daneben besitze ein Icon wie jedes andere grafische Objekt des Baukastens die Attribute Größe, Position, Hintergrundfarbe, Vordergrundfarbe und sei in einem anderen Objekt

(Parent) als Kind enthalten. Wie alle Objekte des Baukastensystems besitze ein Icon einen eindeutigen Bezeichner zur Identifikation.

Im Modell werden in eine Toolbar im ersten Entwurf drei Icons gelegt. Nachdem erkannt wird, dass die Anzahl der Icons variabel zu halten ist, wird eine Variation wie folgt definiert:

Exemplare und damit auch Gegenstand der Variation seien die drei Icons.

Zur näheren Bestimmung der Variation wird auf bestimmte den Variationsgegenständen zuzuordnende Aspekte eingeschränkt: Als ein Aspekt der Variation sei angegeben die Anzahl der Exemplare, etwas formaler: die Anzahl der Instanzen, die durch die Modellexemplare repräsentiert werden. Mit der Variation der Anzahl der Instanzen ist notwendigerweise eine Variation der eindeutigen Bezeichner für eine Iconinstanz verbunden. Außerdem variieren die den Icons zugeordneten Symbole (Bitmaps), die Position der Icons, deren beschreibender Text, sowie die Dialoge, die bei Selektion eines der Icons getriggert werden sollen.
Alle anderen Merkmale der Variationsexemplare werden beibehalten: Größe, Hintergrundfarbe, Vordergrundfarbe und Parent.

Die Festlegung, welche Attribute (Aspekte) der Exemplare variieren und welche nicht, kann optional vom System automatisch getroffen werden, indem es die gewählten Exemplare in ihren Attributen und Attributwerten vergleicht.
Die automatische Bestimmung der variierenden Attribute ist in unserem Beispiel möglich, weil sich jeweils Position, auszulösender Dialog, beschreibender Text, sowie Bezeichner unterscheiden. Dies wird vom System als Hinweis dafür genutzt, dass diese Attribute zusammen mit der Anzahl der Instanzen sich ebenfalls ändern.
In gleicher Weise sei die automatische Erkennung der nicht variierenden Attribute durch das System deshalb erfolgt, weil die drei Icons in Größe, Hintergrund- und Vordergrundfarbe übereinstimmen.
In jedem Fall aber kann eine automatische Festlegung der variierenden Merkmale vom Benutzer überschrieben werden: wenn beispielsweise erwartet wird, dass sich zukünftige Icons auch in Größe und Farben unterscheiden könnten, kann der Benutzer des Entwurfssystems Größe, Hinter- und Vordergrundfarbe ebenfalls als zu variierende Merkmale festlegen. Ebenso ist denkbar, dass der Benutzer bei der Position eine Differenzierung machen möchte und definiert, dass die Position nur in der horizontalen Koordinate variieren darf.

Als Zeitpunkt der Berechnung (Bestimmung) der Variation wird das Öffnen der Toolbar gewählt. Der Zeitpunkt, zu dem die Variation stattfinden (wirken) soll, sei ebenfalls das Öffnen der Toolbar.

Als Variationstyp geben wir an, dass es sich um eine vollständige Variation handelt. Dies bedeutet, dass bei jedem Öffnen der Toolbar die Menge der Iconvarianten komplett neu bestimmt und ausgetauscht wird. Dies hätte etwa zur Konsequenz, dass eine Änderung der Daten in der Benutzerdatenbank durch erneutes Öffnen der Toolbar eine komplett neue Toolbar-Füllung nach sich ziehen könnte. Hätten wir als Variationstyp „inkrementell“ angegeben, erwarten wir von der Variationsfunktion (siehe nachfolgend), dass sie bei wiederholtem Aufruf nur neu hinzuzufügende Icons liefert und die bereits in der Toolbar befindlichen Icons erhalten bleiben.

Variationsfunktion zur Berechnung der Varianten
Es fehlt nun noch festzulegen, wie die zu variierenden Attribute bzw. Attributwerte bestimmt werden.

Eine nahe liegende, einfache Art ist, dass wir eine Funktion angeben, die uns die Varianten ermittelt. Dazu legen wir die Parameter fest, deren Werte zum definierten Zeitpunkt der Berechnung in die Funktion eingehen. Die Funktion kann in einer im Entwicklungssystem verfügbaren Programmiersprache realisiert werden.
In unserem Beispiel wird die Funktion beim Öffnen der Toolbar mit den entsprechenden Parameterwerten versorgt und aufgerufen. Als Parameter könnte in unserem Beispiel etwa die Identifikation des aktuellen Benutzers der Anwendung eingehen. Wir nehmen an, dass die Id des Benutzers im Modell als Wert einer Dialogstelle vorliegt. (Mit der Benutzeridentifikation sind in der Praxis etwa entsprechende Datenbankeinträge als weitere Parameter verbunden, auf die die Implementierung der Funktion - dem Modell verborgen - zugreift). Die Funktion liefert in einer definierten Struktur die Anzahl der zu öffnenden Icons, deren Bezeichner, deren Position, Bitmaps, Texte sowie die Bezeichner der zugeordneten Dialoge zurück. Die von der Funktion gelieferten Daten werden nun vom System automatisch dazu verwendet, um die Icons in der Toolbar anzuzeigen. Die nicht variierenden Werte werden aus den Exemplaren gewonnen.
Wenn wir davon ausgehen, dass eine mächtige Programmiersprache im Entwicklungssystem vorhanden ist, ist dies ein im Allgemeinen gangbarer Weg zur Bestimmung der Varianten. Damit wird zwar ein Teil der Lösung unserer Aufgabe auf die Ebene einer (anderen) Programmiersprache ausgelagert und bleibt, was die Darstellung im Modell betrifft, im Verborgenen. In der Praxis erscheint dies jedoch als ein durchaus gangbarer und auch sinnvoller Weg, wenn die Ermittlung der Varianten komplex ist.

Für einfache Fälle der Bestimmung der Varianten können wir in unserer Modellsprache Sprachelemente zulassen, die eine Auslagerung in eine gesonderte Programmiersprache unnötig macht. So können wir z.B. alternativ zur Definition einer Funktion die Angabe von Ausdrücken zur Zuweisung einfacher Stellenwerte erlauben.

7.4 Im ESM implizit vorhandene Variationsarten (Built-in Variations)

Die Variation von Stellenwerten ist eine spezielle Art von Variation, die wir bereits im ESM vorfinden. Im Abschnitt grafische Darstellung (Abschnitt 5.3) wurde bereits erwähnt, dass in einer Stelle exemplarisch ein Wert eingetragen werden kann. Eine Variation dieses Wertes im gerade oben erläuterten Sinn findet statt, wenn bei Eintritt eines Ereignisses mithilfe einer Propagationsfunktion ein neuer Wert ermittelt wird und dieser über ein Ausgabeereignis an der Stelle eingetragen wird. Die Stelle kann als ein Exemplar betrachtet werden, das stellvertretend für verschiedene Varianten mit unterschiedlichen Stellenwerten im Modell dargestellt ist. Die Propagationsfunktion übernimmt in diesem Fall die Aufgabe der Variationsfunktion.
Ebenso kann im erweiterten Sinne als eine Variation betrachtet werden, wenn eine Stelle vom Zustand geschlossen in den Zustand geöffnet wechselt oder umgekehrt. Dabei betrachten wir den Zustand des „Geöffnet-Seins“ als ein Attribut der Stelle mit booleschem Wert TRUE oder FALSE. Eine Condition, die bei Eintritt eines bestimmten Ereignisses ausgewertet wird, kann dann als Variationsfunktion betrachtet werden, die den neuen Wert des „Geöffnet-Zustandes“ berechnet.
In diesem Sinne kann der Wechsel des Geöffnet-Zustands von TRUE nach FALSE und umgekehrt auch als ein Dialog-Ereignis betrachtet werden. Im oben geschilderten Beispiel wurde dies bereits stillschweigend angenommen, als wir die Durchführung der Variation an das „Ereignis“ des Öffnens der Toolbar gekoppelt haben. (Eine exakte Definition bzw. Interpretation dieses Ereignistyps sei allerdings der formalen Darstellung vorbehalten.)

Die Variation des Geöffnet-Zustands wollen wir im Folgenden ebenfalls etwas salopp als Existenzvariation einer Stelle bezeichnen. Ist eine Stelle geöffnet, ist sie für den Benutzer des Dialogsystems bzw. das System existent. Im anderen Fall steht sie für die Wahrnehmung von Eingabe- oder Ausgabeereignissen bzw. für Lese- und Schreiboperationen nicht zur Verfügung und ist damit aus Sicht von Benutzer und/oder System nicht existent.

7.5 Variation einer Existenzvariation

Im obigen Beispiel der Toolbar mit Icons haben wir als Exemplare der Variation die Icons gewählt. Im Sinne eines ESM handelt es sich dabei um Konkretisierungen von Dialogstellen s aus S. Diese wurden als Bezugspunkte oder Ausgangspunkte für die Variation gewählt. In der Variation wurde die Menge der Exemplarstellen durch eine andere Menge von Dialogstellen ersetzt und gleichzeitig damit wurden Aussagen über die Exemplarstellen, wie z.B. die zugeordnete Bitmap verändert. In der abstrakten Sicht eines ESM handelt es sich um Teilstellen der Exemplarstellen bzw. der die Exemplarstellen ersetzenden Stellen, die durch die Variation in ihren Werten verändert wurden.
In gleicher Weise können nun auch die im Modell vorgesehenen Elemente der Relationen, insbesondere die Stellenpaare aus den Relationen RO und RC als Ausgangspunkte einer Variation dienen. Dies bedeutet, dass wir dann sozusagen nicht die Dialogstellen selbst, sondern Aussagen über Beziehungen zwischen Dialogstellen als Bezugspunkt der Variation heranziehen.

7.5.0 Beispiel-Variation zur Auslösung eines Druckdialogs

Dies soll durch folgendes Beispiel erläutert werden:
Ein Druckdialog soll in einem System von jedem Dialogelement aus durch eine bestimmte Tastenkombination geöffnet werden können. Mit exemplarischer Darstellung und Variation lösen wir diese Aufgabe wie folgt:
Wir zeichnen im Modell von einer einzigen Dialogstelle aus einen Öffnepfeil zur Dialogstelle, die den Druckdialog repräsentiert und definieren folgende Variation:

Gegenstand der Variation ist der Öffne-Pfeil zwischen den beiden Dialogstellen, formal gesprochen also ein Stellenpaar aus der Öffne-Relation RO. Als Aspekt der Variation wird wiederum die Anzahl (der durch den Pfeil als Exemplar repräsentierten Menge) angegeben sowie als weiterer Aspekt die Dialogstelle, von der der Pfeil ausgeht. Um exakt zu sein, meinen wir hier nicht die Stelle selbst, sondern eine Referenz auf die Stelle.
D.h. wir wollen in diesem Fall nicht die Eigenschaften der Dialogstelle selbst verändern, sondern nur den Bezug des Pfeils auf die Dialogstelle, deren spezifisches Ereignis das Öffnen des Druckdialogs auslöst.
Zu diesem Zweck sehen wir als Möglichkeiten der Angabe eines Variationsaspektes die Unterscheidung zwischen Referenz auf die Stelle und Stelle selbst vor.
Der Unterschied zeigt sich darin, dass dann nicht die Stelle, von der exemplarisch im Modell der Pfeil ausgeht, zur Veränderung bzw. Belegung der Eigenschaften der referenzierten Stelle verwendet wird. Die referenzierte Stelle wird als im Modell bereits bekannt (in S enthalten) angenommen und deren Eigenschaften bleiben unverändert, sieht man von dem Hinzukommen der Rolle als auslösende Stelle für den Druckdialog ab.
Zum Unterschied waren im obigen Beispiel der Toolbaricons die Stellen (Icons) selbst Gegenstand der Variation und haben von der zu variierenden Modellstelle Eigenschaften übernommen.

Wir wollen, dass von jeder Dialogstelle aus der Druckdialog geöffnet werden kann. Deshalb geben wir als Zeitpunkt der Variationsberechnung jeweils das Öffnen jeder Dialogstelle an. Wir fordern zu diesem Zweck von unserem System, dass es in der Lage ist, nicht nur konkrete voll qualifizierte Ereignisse als Zeitpunkte einer Variation zu akzeptieren, sondern auch die Spezifikation von Ereignispatterns, die implizit eine Menge von Ereignissen beschreiben. In diesem Fall beschreibt das Ereignispattern Ereignisse mit beliebiger Stelle (*) und konkret vorgegebenem Wert (Tastenkombination „Shift+Print“), und konkretem Typ (I_B), e=(*,“Shift+Print“, I_B).
Als Variationsfunktion geben wir hier eine im System standardmäßig vorhandene Funktion an, die den Bezeichner der Stelle zurückliefert, an der das die Variation auslösende Ereignis eingetreten ist. Damit kann das System dann eine Implementierung vornehmen, die bewirkt, dass bei Eintreten der Tastenkombination „Shift+Print“ mit Focus auf dem gerade geöffneten Dialogelement der Druckdialog geöffnet wird.
Wir fordern dabei von unserem System, dass es diese Implementierung nicht vornimmt, wenn sie bereits erfolgt ist oder wenn das geöffnete Dialogelement der Druckdialog selbst oder einer seiner Teilstellen ist. Letzteres wäre allerdings insofern nicht problematisch, falls das Öffnen einer bereits geöffneten Stelle nichts bewirkt.

Als Variationstyp geben wir in diesem Beispiel „inkrementelle Variation“ an. Denn wir erhalten durch die Variationsfunktion pro Variation jeweils nur die Referenz einer Dialogstelle, die mit dem Druckdialog in Beziehung gesetzt werden soll. Alle anderen Referenzen auf bereits geöffnete Stellen bleiben bestehen und die referenzierten Stellen behalten somit die Möglichkeit des Druckdialogstarts bei.

Für dieses Beispiel können wir zusammenfassen: Die Information, dass der Druckdialog per Tastenkombination an einer Dialogstelle geöffnet werden kann, wird im Modell exemplarisch grafisch festgehalten (Öffne-Pfeil zwischen den beiden Dialogstellen). Durch Hinzufügen von Variationsinformation wird diese Funktionalität auf alle Dialogobjekte durch das System automatisch erweitert.
Die im Beispiel gezeigte Art der Variation wollen wir als **Anwendungsvariation** bezeichnen: die Anwendung bzw. der „Aufruf“ des Druckdialogs wird im Modell beispielhaft an einem Dialog gezeigt und variiert dann über alle Dialogstellen. Etwas formaler betrachtet wird ein Stellenpaar der Relation RO durch eine Menge von Stellenpaaren ersetzt, bei denen sich jeweils die erste Stelle unterscheidet.

7.5.1 Variation der Zielstelle

Wenn wir die Stelle, in die der Pfeil einmündet, also im Beispiel den Druckdialog, als weiteren Variationsgegenstand mit in unser Variations-Beispiel aufnehmen und als weiteren Aspekt der Variation die Identifikation dieser Stelle angeben, können wir damit auch die Instanz des Druckdialogs variieren. In die Ausgabestruktur der Variationsfunktion ist dann noch der Bezeichner für die jeweilige Instanz des Druckdialogs mit aufzunehmen.
Da wir die Stelle selbst und nicht die Referenz auf die Stelle als Gegenstand der Variation definieren, bedeutet dies, dass vom System eine Instanz mit dem von der Variationsfunktion gelieferten Bezeichner, falls nötig, erzeugt und mit den Merkmalen des Druckdialogexemplares belegt wird.

Wir haben bisher den Gedanken der Exemplarvariation informell und möglichst allgemeingültig ausgeführt. Zu einer **formalen Darstellung** der **Exemplarvariation** für den Aspekt Kardinalität mit inkrementeller Variation sei auf **Anhang E**) verwiesen. Dazu wird eine Erweiterung des Ereignisstellenmodells aus Kap.5 vorgenommen und es werden sogenannte Modellkonfigurationen, die eine dynamische Änderung des Modells beschreiben, eingeführt. Die Anregung zur formalen Darstellung der Systemzustände als Folge von Konfigurationen stammt dabei aus [EICKEL01].

8 Modell-Beispiel

In diesem Kapitel soll das bisher vorgestellte Ereignisstellenmodell an einem Anwendungsbeispiel aus der unmittelbaren Praxis des Autors beispielhaft erläutert und diskutiert werden. Es handelt sich bei der zu modellierenden Anwendung um ein Dialogsystem zur Darstellung und Edition von Batchabläufen im Umfeld von zeitintensiven Berechnungen von Massendaten für das Risikomanagement einer Bank.

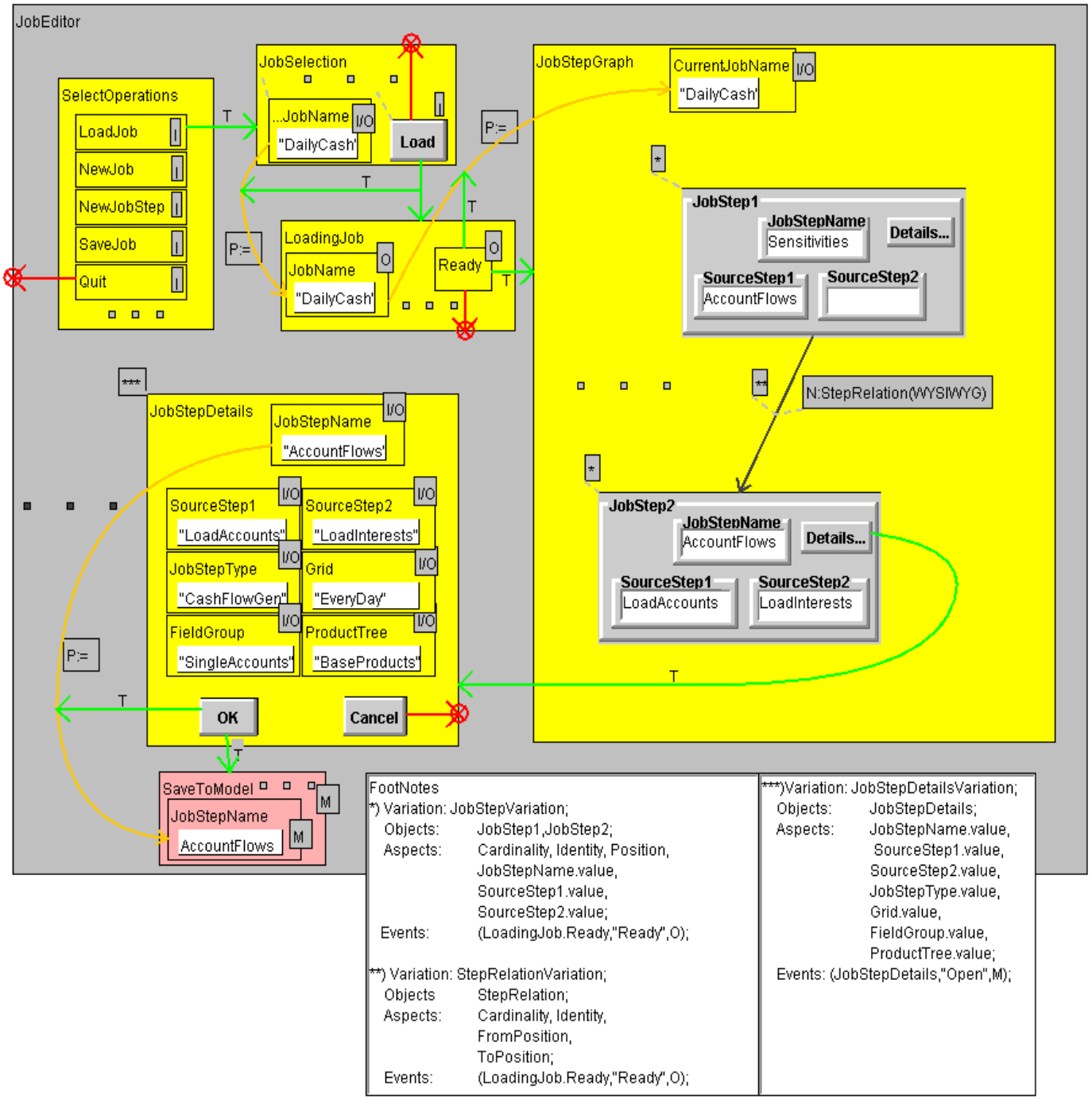


Abbildung 8-1: JobEditor als Anwendungsbeispiel eines Ereignisstellendiagramms

In Abbildung 8-1 handelt es sich um eine Teilsicht auf das Modell in einem Ereignisstellendiagramm, die hier näher besprochen werden soll. Das Gesamtmodell ergibt sich aus einer Reihe von Teilsichten, die im Wesentlichen jeweils den Dialog darstellen, der sich aus der Selektion einer der Hauptfunktionen (Operationen) des Systems ergeben.

8.0 Aufgabenstellung des zu modellierenden Systems

Zunächst wollen wir noch kurz die Aufgabenstellung des zu erstellenden Dialogsystems erläutern. Ein Batchablauf sei im Folgenden auch als Job bezeichnet. Jeder Job besteht aus einer Menge von Teilschritten (JobSteps), wobei ein Teilschritt von jeweils zwei Vorgänger-Teilschritten (SourceStep1 und SourceStep2) abhängig sein kann. Dass nur zwei Vorgängerschritte erlaubt sind, ist historisch aus der Anwendung begründet und soll hier keine weitere Rolle in der Diskussion spielen.
Jeder Teilschritt repräsentiert im Wesentlichen einen Berechnungsschritt, der zu einem Berechnungsergebnis führt, das ggf. in Folgeschritten zur Weiterverarbeitung benötigt wird. Zur Beschreibung der Art der Berechnung und zur Beschreibung des Ergebnisses des Rechenschrittes sind jedem JobStep neben den beiden Vorgängerschritten eine Reihe weiterer Attribute zugeordnet.
Als Beispiel seien hier nur JobStepType, FieldGroup, ProductTree und Grid aufgezeigt. Hinter dem JobStepType verbirgt sich letztlich das Programm, das zur Berechnung des Rechenschrittes aufgerufen werden soll.
FieldGroup ist der Name einer Gruppe von Datenfeldern, die zu berechnen sind, ProductTree und Grid sind zwei weitere Parameter, die die Berechnung beeinflussen. Auf sie soll hier nicht näher eingegangen werden, da sie für die Diskussion unseres Modells nur beispielhaften Charakter als bestimmte Attribute eines zu erfassenden bzw. zu pflegenden Datenbank-Records haben.
Die einzelnen Teilschritte eines Jobs sollen nun im JobEditor in ihren Attributen (JobStepName, SourceStep1, ..., FieldGroup, ...) definiert werden. Insbesondere sollen die Abhängigkeiten der JobSteps voneinander grafisch visualisert und ebenfalls grafisch editiert werden können.

8.1 Elemente eines Ereignisstellen-Diagramms für das JobEditor-Modell

Abb. 8-1 zeigt die Dialogmodell-Teilsicht des JobEditors in einem Ereignisstellendiagramm, wie es derzeit in einem Prototypen dargestellt werden kann.
Die verschiedenen Farben der Diagrammelemente haben formal keine Bedeutung für die Interpretation. Die Elemente sind auch ohne Farbe eindeutig definiert. Sie werden hier aber als nützliches Hilfsmittel zur leichteren Unterscheidung auch in der folgenden Diskussion verwendet.

Ereignisstellen als Rechtecke
Die Rechtecke zeigen Ereignisstellen bzw. die dadurch repräsentierten Dialoge in ihrer hierarchischen Struktur.
Das graue umfassende Rechteck repräsentiert den kompletten JobEditor-Dialog (die alle Stellen zusammenfassende Stelle). Darin enthalten sind Ereignisstellen

- zur Abwicklung des Dialogs zur Auswahl der Operationen (SelectOperations),
- zur Auswahl eines aus der Datenbank zu ladenden Jobs (JobSelection),
- zur Anzeige des Ladevorgangs (LoadingJob),
- zur grafischen Darstellung und Edition der JobSteps (JobStepGraph)

- sowie zur Darstellung und Edition der Detail-Information zu einem JobStep (JobStepDetails).

Eine besondere Rolle kommt der Ereignisstelle SaveToModel als reiner Hilfsstelle (wir sagen auch Applikationsstelle) zu (siehe unten). In den genannten Ereignisstellen sind weitere Teilstellen enthalten (LoadJob...,NewJob..., usw.).
Ein Teil des Diagramms ist in WYSIWYG-Darstellung gezeigt. Dies gilt für die primitiven Dialoge Load, OK und Cancel und für den Inhalt des Dialogs JobStepGraph. Dies gilt für die beiden Dialoge JobStep1 und JobStep2 sowie für den Pfeil StepRelation, der die Abhängigkeit des Rechenschrittes „Sensitivities“ vom Rechenschritt „AccountFlows“ definiert. (Zur Berechnung von Risikokennziffern einer Bankposition müssen erst alle Cashflows dieser Position ermittelt werden).
Das Diagramm zeigt also eine Momentaufnahme des Entwurfsvorganges, in der bestimmte Dialogelemente konkret bereits feststehen, andere erst abstrakt vorliegen. Es könnte jedoch genauso gut der Entwurfsvorgang bereits abgeschlossen sein und nur für bestimmte Dialogelemente gerade statt der konkreten Sicht die abstrakte Sicht gewählt worden sein.

Pfeile zum Triggern von Ereignissen mit Wertübertragung
Mit Ausnahme des schwarzen Pfeils sind alle übrigen Pfeile Ausprägungen von bereits vorgestellten Pfeilen eines Ereignisstellendiagramms: die grünen Pfeile stellen Öffne-Pfeile oder Pfeile zum Triggern von Wertübertragungen dar, die Pfeile in Rot sind Schließe-Pfeile. Die orangefarbenen Pfeile repräsentieren die Wertübertragung von einer Ereignisstelle zur anderen (getriggert durch einen grünen Pfeil).

Variationsannotation
Einigen Elementen im Modell ist Variationsinformation zugeordnet, die hier als Fußnoten im Diagramm angefügt sind (siehe auch unten). Die Zuordnung der Variationsinformation zu den Dialogelementen ist durch die annotierten Fußnotensterne zu erkennen.

8.2 Erläuterung und Diskussion des Beispiel-Diagramms

Wie ist das Diagramm nun zu interpretieren:

8.2.0 Dynamik

Start mit SelectOperations-Dialog
Mit Start der Applikation wird der Dialog (die Ereignisstelle) JobEditor und in ihm der Teildialog (Teilereignisstelle) SelectOperations eröffnet. Dies geschieht per Definition deshalb, weil weder in diese Ereignisstellen noch in hierarchisch übergeordnete Ereignisstellen Öffne-Pfeile münden (Startstellen).
Das gleiche gilt für alle Teildialoge von SelectOperations. Es handelt sich hier um nebenläufige Ereignisstellen, die sofort mit dem Eröffnen des übergeordneten Dialogs verfügbar sind.

Anstoß JobSelection
Die Durchführung eines durch die Stelle LoadJob repräsentierten Ereignisses eröffnet den Dialog zur Selektion der aus der Datenbank zu ladenden Jobbeschreibung. Die Selektion des Jobs ist hier als zusammengesetzte Ereignisstelle modelliert. Sie besteht aus einer Ereignisstelle zur Darstellung des Jobidentifikators und einer Ereignisstelle, über die der Dialog zum Laden des Jobs (der Jobbeschreibung) angestoßen wird.

LoadingJob
Das Anstoßen wird durch den Öffnepfeil von der Ereignisstelle Load zur Ereignisstelle LoadingJob angedeutet.
Ein weiterer Triggerpfeil von der Ereignisstelle Load triggert die Wertübernahme von der Ereignisstelle JobName der JobSelection-Stelle an die Teilstelle JobName in der Ereignisstelle LoadingJob. Wertübernahme bedeutet in Diktion des Ereignisstellenmodells: Bei Eintritt eines Ereignisses an der Stelle JobSelection.Load wird ein Ausgabeereignis an der Stelle LoadingJob.JobName propagiert, dessen Wert durch eine Propagationsfunktion bestimmt wird. In diesem Falle handelt es sich um die Identitätsfunktion, die ihren Wert aus dem Wert der Stelle JobSelection.JobName herleitet. Ein Wertübernahmepfeil „wirkt“ somit nur dann, wenn durch einen zugeordneten Triggerpfeil ein Ereignis ausgelöst wird.

Öffnen JobStepGraph
Die Ereignisstelle Ready markiert den Abschluss des LoadingJob-Ereignisses und eröffnet über einen Öffnepfeil die Ereignisstelle zur Darstellung und Edition der JobSteps eines selektierten Jobs, nämlich die zusammengesetzte Ereignisstelle JobStepGraph. Die Stellen JobStep1, JobStep2 und StepRelation sind dabei als Exemplare zu betrachten. Der nach dem Öffnen tatsächlich zur Laufzeit im JobStepGraph dargestellte Inhalt wird zum Teil explizit durch die Exemplare selbst und zum Teil durch die den Exemplaren zugeordnete Variationsinformation bestimmt (siehe unten).

Ein Triggerpfeil ausgehend von der Ereignisstelle Ready triggert die Wertübertragung von LoadingJob.JobName zu CurrentJobName.
Das Eintreffen eines Ereignisses an der Stelle Details in der zusammengesetzten Stelle JobStep2 eröffnet den JobStepDetails-Dialog, der wiederum ein zusammengesetztes Ereignis repräsentiert.
Ein Eintreffen eines Ereignisses an der Teilstelle OK triggert die Ereignisse an der Ereignisstelle SaveToModel und ein Triggerpfeil ebenfalls ausgehend von OK triggert die Wertübertragung zwischen den beiden Stellen JobStepName in JobStepDetails und SaveToModel.

Die Pfeile in Rot definieren ein Schließen der Ereignisstelle. Der Pfeil aus der Ereignisstelle Quit schließt somit den gesamten Dialog und damit alle Teilstellen. Der Schließepfeil von Ready den Teildialog LoadingJob, entsprechend bedeuten die Pfeile von OK und Cancel das Schließen der Teilstelle JobStepDetails. Schließen bedeutet, dass kein Ereignis an der Stelle mehr möglich ist, bis nicht explizit (per Öffnepfeil) wieder eröffnet wird.

8.2.1 Ereignistypen

Von wem, ob von Benutzer oder System die Dialoge nun wirklich durchgeführt werden, d.h. durch wen an den Stellen entsprechende Ereignisse erzeugt werden, hängt vom Typ der zugeordneten Ereignisse ab. Die explizite Typzuordnung ist in obigem Diagramm an den entsprechenden Annotationen (I für Eingabe, O für Ausgabe, M für Schreibereignis an Hilfsstelle, I/O für Stelle mit Eingabe- und Ausgabeereignissen) zu erkennen.
Der Prototyp, in dem oben dargestelltes Diagramm editiert wurde, realisiert auch eine implizite Zuordnung im Rahmen der Codegenerierung. So werden primitive Ereignisstellen prinzipiell mit dem Typ Eingabe belegt, bei Wertzuweisung zusätzlich mit dem Typ Ausgabe.

Der durch die Selektion im Dialog JobSelection ermittelte Wert wird in der Ereignisstelle JobName repräsentiert. Um einen sinnvollen Wert an der Stelle ablesen zu können, setzen wir voraus, dass entweder eine Eingabe an der Stelle durch den Benutzer erfolgt ist oder das

System ein Ausgabeereignis durchgeführt hat. Wir können die Stelle also mit dem Ereignistyp I/O versehen.

In unserem Modell wird es sicher Sinn machen, Ready als reine Ausgabestelle zu kennzeichnen. Denn das Ereignis Ready signalisiert den Abschluß des Ladevorgangs und die Bereitstellung der Daten eines Jobs durch die Applikationslogik zur nachfolgenden Darstellung an der Dialogschnittstelle.
Ebenso wird die Teilstelle CurrentJobName mit dem Namen des aktuell geladenen Jobs belegt. CurrentJobName ist also notwendigerweise mindestens eine Ausgabestelle. Ob es gewünscht ist, dass CurrentJobName auch eine Eingabestelle repräsentieren soll, liegt in der Entscheidung, ob die Änderung des Jobnamens durch den Benutzer erlaubt sein soll oder nicht.
Die Teilstellen JobStepName, SourceStep1 und SourceStep2 der jeweiligen Stellen JobStep1 und JobStep2 sind Stellen, die sowohl die Ausgabe als auch die Eingabe von Werten erlauben.

8.2.2 Hilfsstelle SaveToModel

Durch die Kennzeichnung mit Ereignistyp M ist die Ereignisstelle SaveToModel als Hilfsstelle definiert, die für den Benutzer unsichtbar ist. Die Stelle repräsentiert letztlich den im System verborgenen Ablauf einer Ereigniskette. In der Implementierung realisiert die Stelle das Abspeichern der im Dialog JobStepDetails neu eingegebenen oder geänderten Werte. Dabei ist es nicht die Intention des Dialogmodells, den genauen inneren Ablauf der Ereignisstelle SaveToModel darzustellen. Es soll hier lediglich der Anstoß dazu definiert werden. Dazu wird mit dem „Eröffnen“ (dem Start) der Stelle der Wert der Stelle JobStepDetails.JobStepName an die Teilstelle JobStepName von SaveToModel übergeben.

8.2.3 Variationen

Dynamische Belegung der Werte über Variation
Woher kommen nun die Werte an den Ereignisstellen? Prinzipiell kann ein Wert an einer Stelle entweder als Konstante im Modell bereits sozusagen a priori festgelegt werden oder die Bestimmung des Wertes kann erst zur Laufzeit erfolgen.
In letzterem Fall kann die Darstellung im Diagramm höchstens ein Exemplar aus dem der Stelle zugeordneten Wertebereich darstellen. Die Werte der Teilstellen JobStepName etc. bestimmen sich erst frühestens nachdem der Benutzer sich für das Laden eines bestimmten Jobs aus der Datenbank entschieden hat.
Hier kommt die Einführung einer „Variationsannotation“ zum Tragen: Der mit dem Ausgabeereignis darzustellende Wert ist variabel und wird erst dynamisch zu einem bestimmten Zeitpunkt definiert.

Variationen von Teilstellen in JobStepGraph und von JobStepDetails
Genauso wie die Werte der primitiven Ereignisstellen JobStepName, SourceStep1 und SourcStep2 variieren, ist auch die Anzahl der diese Stellen enthaltenden zusammengesetzten Stellen JobStep1 und Jobstep2 variabel. Denn wir wissen erst nach dem Laden aus der Datenbank, wie viele JobSteps ein Job enthält. Das gleiche gilt für die Abhängigkeiten der JobSteps, die durch jeweils einen schwarzen Pfeil StepRelation repräsentiert werden.
Wir betrachten deshalb JobStep1 und JobStep2 sowie StepRelation als Exemplare oder Repräsentanten einer Menge von Ereignisstellen, deren Kardinalität erst bei Eintreten des Ereignisses Ready bestimmt wird. Die zu definierende Variation wird wiederum mittels einer bestimmten Funktion durchgeführt. Dabei ist zu bedenken, dass mit der Anzahl der zu

öffnenden Teilstellen auch andere Attribute von Teilstellen variieren. In unserem Beispiel variieren mit der Anzahl auch die Werte der primitiven Teilstellen.
Weitere Variationen ergeben sich bei Änderung der Anzahl allerdings auf der Ebene der zugeordneten konkreten Laufzeitinstanzen. Wenn wir etwa als Laufzeitinstanz ein Widget zuordnen, müssen wir auch die Position variieren, zumindest falls die Kardinalität größer 1 ist.
Im Folgenden sollen die im Beispiel definierten drei Variationen besprochen werden. Sie sind im Modell in Fußnotendarstellung definiert und über die entsprechenden Sternsymbole mit den zugehörigen Objekten oder Exemplaren der Variation verknüpft.

Variation JobStepVariation
Die Variation mit Namen JobStepVariation hat als Exemplare oder Objekte der Variation die beiden Ereignisstellen JobStep1 und Jobstep2. Als Aspekte der Variation werden genannt Cardinality, Identity, Position und die Werte bestimmter Teilstellen, definiert durch JobStepName.value, SourceStep1.value und SourceStep2.value. Als Zeitpunkt der Variation wird das Ausgabeereignis mit Wert „Ready“ an der Stelle LoadingJob.Ready genannt.
Dies bedeutet, dass anstelle der beiden genannten Objekte nach Eintreffen dieses Ereignisses eine Menge von Objekten bestimmt wird, die an die Stelle der beiden Exemplare im Modell treten und geöffnet werden. Dass die Anzahl der Objektmenge dynamisch bestimmt wird, ist durch Angabe des Aspekts Cardinality erreicht. Andernfalls bliebe die Anzahl der Exemplare fix bei zwei.
Mit der Anzahl der Objekte variiert notwendigerweise auch die Identität der Objekte und für jedes Objekt wird ein eindeutiger Bezeichner bestimmt. Dass hier Identity angegeben ist, verdeutlicht dies, ist aber nicht notwendig. Da wir uns bei den zu variierenden Exemplaren auf der Ebene konkreter Widgets befinden, ist es bereits möglich, auch Attribute der Widgets als Variable anzugeben. In der sinnvollen Annahme, dass die Widgets in einem Container positioniert werden müssen, geben wir als zu variierendes Attribut die Position mit. Die Angaben zur Variation der Teilstellenwerte setzen hier voraus, dass eine Standardabbildung des abstrakten Attributes „value“ auf ein konkretes Attribut des für die Teilstelle verwendeten Widgets existiert.
In unserem Beispiel können wir annehmen, dass value auf das Attribut „Text“ der hier benutzten Java-Widgetklasse JTextArea unter Verwendung der Schnittstellenfunktion setText abgebildet wird.
Da bei der Variationsinformation die Angaben zum Aufruf einer Variationsfunktion fehlen, können wir wiederum annehmen, dass ein Default-Protokoll zum Aufruf einer Variationsfunktion verwendet wird.
Im realisierten Prototypen ist die Konvention, dass zu jedem Dialogobjekt, dem ein Variationsereignis zugeordnet wird, ein Model-Objekt existiert, das für die einzelnen zu variierenden Aspekte eine entsprechende Methode besitzt, die die gewünschten Daten zur Variation zurückliefert (siehe dazu auch Prototyp Kap.10).

Variation StepRelationVariation
Die Variation StepRelationVariation ist ähnlich zur vorherig beschriebenen Variation zu verstehen: Anstelle des einen Pfeils wird zum Variationszeitpunkt von einer den Konventionen entsprechenden Variationsmethode Information über die Anzahl der tatsächlich anzuzeigenden Pfeile geliefert. Ebenso werden von entsprechenden Methoden die der Anzahl entsprechenden Identifikatoren der Instanzen der Pfeilklasse sowie die Werte zu den beiden Attributen FromPosition und ToPosition geliefert, die vom System benötigt werden, um die Pfeile entsprechend richtig zu positionieren. Bei der Pfeilklasse handelt es sich um eine spezielle dem Standard-Widgetrepertoire hinzugefügte Klasse, die im angestrebten Entwicklungssystem zusätzlich angeboten wird.

Variation JobStepDetails
Bei der dritten Variation in unserem Modellbeispiel handelt es sich um eine Variation, die im Gegensatz zu den anderen beiden Beispielen nicht die Anzahl der Exemplare verändert. Als Variationsaspekte sind lediglich die Werte von Teilstellen angegeben. Die Variation findet statt, wenn das Dialogobjekt JobStepDetails geöffnet wird. Dieses Beispiel zeigt, dass wir in die Ereignismenge bestimmte „Metaereignisse“, wie z.B. das Öffnen einer Stelle oder das Schließen mit aufnehmen.

Randbemerkung: Im derzeitigen Prototyp sind die oben beschriebenen Variationen etwas anders zu definieren, da dort die Variation von Teilstellenattributen (z.B. SrcName1.value) sowie eine automatische Umsetzung der logischen Ereignisse ((LoadingJob.Ready, “Ready“,O)) auf konkrete Ereignisse der Implementierung noch nicht realisiert ist. Stattdessen müssen im Prototyp pro Teilstelle eigene Variationen definiert und als Ereignis ein konkretes Ereignis des zur Implementierung verwendeten Widgets angegeben werden. Z.B. kann dort als Ereignis das Metaereignis „Open“ für die Stelle Ready angegeben werden.

8.2.4 Zuordnung konkreter Laufzeitstellen

Das obige Diagramm macht abgesehen von den WYSIWYG dargestellten Dialogen keine explizite Aussage, wie die Modellstellen in konkrete Laufzeitstellen umzusetzen sind. Ob es sich also z.B. bei der Auswahl der Operationen im SelectOperations-Dialog etwa um einen Kommandodialog handelt, der auf einer Kommandozeile die Eingabe eines Kommandos durch den Benutzer ermöglicht, oder ob es sich um ein grafisches Panel mit einer Gruppe von Buttons oder gar einen Baum zur Selektion per Mausklick handelt, ist offen.
Eine Konkretisierung kann nun erfolgen, indem Information zur Abbildung auf eine konkrete Laufzeitstelle der Modellstelle zugeordnet wird. Im realisierten Prototypen geschieht dies zum einen durch die Annotation einer in Java implementierten Klasse. Dies bedeutet zunächst, dass zur Laufzeit eine oder mehrere Instanzen der Klasse erzeugt werden. Zum anderen geschieht dies durch Zuordnung von Variationsinformation über die Anzahl der zu erzeugenden Instanzen (siehe oben Kardinalitätsvariation). Mithilfe einer Variationsannotation kann u.a. auch die Klasse der Laufzeitinstanz dynamisch verändert werden.
Wie für Ereignistypen ist im Prototypen auch eine implizite automatische Zuordnung konkreter Laufzeitstellen implementiert. D.h. das Diagramm kann per Codegenerierung unmittelbar in ein sinnvolles ablauffähiges Testsystem umgewandelt werden (siehe Kap.10).

8.2.5 Anbindung der Applikationslogik

Im derzeitigen Prototypen erfolgt die Anbindung der Applikationslogik entweder durch den Aufruf einer entsprechenden Methode eines einer Laufzeitinstanz zugeordneten Modelobjekts (MVC-Konzept), das in einer Variationsannotation angegeben wird (siehe oben z.B. Variationsfunktionen zur Ermittlung der variablen Ausgabewerte und der Teilstellenstruktur in JobStepGraph) oder durch die Zuordnung der konkreten Laufzeitklasse, die die Applikationslogik oder zumindest eine Schnittstelle zu ihr beinhaltet.
Ein Beispiel ist im obigen Diagramm der LoadingJob-Dialog, dessen konkrete Laufzeitklasse dann bei Instantiierung auch das Laden aus der Datenbank selbst vornimmt.
Ein weiteres Beispiel ist die pinkfarbene Ereignisstelle SaveToModel. Sie stellt eine für den späteren Benutzer des Dialogsystems „unsichtbare“ Komponente dar, eine komplexe Hilfsstelle, die letztlich die Instanz einer Klasse repräsentiert, die für das Ablegen der Daten der Oberfläche in ein Modelobjekt zuständig ist.

8.2.6 Darstellung von Teilsichten

Obiges Ereignisstellendiagramm ist nicht nur, was die Ereignistypen betrifft, unvollständig spezifiziert. Es spiegelt einen Zwischenstand der Modellierungsphase wieder und/oder eine bestimmte Teilsicht. So sind z.B. keine Folgeereignisse für die Ereignisstellen NewJob, NewJobStep und SaveJob modelliert bzw. dargestellt.

Ergänzung unvollständiger Ereignisstellen durch weitere Teilsichten
Die im Beispieldiagramm mit drei Punkten dargestellte Stelle JobSelection ist eine Stelle, die wie die Stellen LoadingJob, SelectOperations, JobStepGraph und SaveToModel entweder in diesem Diagramm noch voll ausspezifiziert werden muss oder die in einer anderen Teilsicht des Modells, repräsentiert durch ein anderes Ereignisstellendiagramm ausmodelliert ist. Von der Stelle sind im Diagramm nur die nötigen Teilstellen JobName und Load angezeigt. Abbildung 5-7 zeigt die voll spezifizierte Teilsicht von JobSelection.
(Der Prototyp sieht beispielsweise die Definition von Teilsichten auf das Ereignisstellendiagramm vor, um ein übersichtliches Arbeiten zu gewährleisten.)

9 Vergleich mit bekannten Ansätzen

In Kapitel 1 haben wir bereits einen kurzen Überblick über bestehende und für die Arbeit relevante Ansätze der Modellierung gegeben. In den nun folgenden Abschnitten wollen wir auf einige der Arbeit nahe liegende Konzepte etwas mehr eingehen.
Es handelt sich dabei zum einen um klassische Ansätze, wie das Koroutinen-Konzept von Jacob [JACOB86], das Ereignismodell nach Green [GREEN86], Janssens Dialognetze [JANSS96], die in die UML eingegangenen State-Charts von Harel [HAREL87] [RUMBA91], Den genannten im Umfeld der Dialogmodellierung bekannten Modellen, folgen Ausführungen zum Pi-Kalkül von Milner [MILNER99], einem allgemeinen Ansatz zur Modellierung von Interprozesskommunikation sowie modernere Ansätze zur Dialogmodellierung, wie das HIT-Konzept von Schreiber [SCHREI97] und so genannte Sketchingsysteme (z.B. [LANDAY00]).

9.0 Koroutinen-Konzept von Jacob

Rekursive Zustandsübergangsdiagramme mit Aktionen und Bedingungen
Ausgangspunkt bei Jacob sind Zustandsübergangsdiagramme. Jacob sieht allerdings keine zusammenhängende Darstellung von Ereignissen und Zuständen aller Dialogobjekte in einem Zustandsübergangsdiagramm. Zustandsübergangsdiagramme sind nur jeweils einzelnen Dialogobjekten, den Interactive Objects, zugeordnet. Ein Zustandsübergangsdiagramm kann Subdiagramme mit rekursiven Aufrufen beinhalten. Übergänge können mit Aktionen (**Actions**) und Bedingungen (**Conditions**) belegt werden.

Tokens für Eingabe und Ausgabe
Interessant ist, dass auch bei ihm Ausgabeereignisse als Übergangsereignisse dargestellt werden. Es gibt also Zustände, die durch Ausgaben erreicht werden.
Jacob sieht den Dialog als Folge abstrakter Eingabe- und Ausgabeereignisse, wobei Layout und grafische Details davon getrennt zu beschreiben sind. Er sieht die Dialogspezifikation zunächst als die Darstellung der möglichen Abläufe der Ein-/Ausgabeereignisse in Form von Tokens. Details von Layout, grafischer Repräsentation und Geräteverwaltung sind Internas der Tokendefinition.

Interactive Objects mit Objekthierarchie und Vererbung
Die Objekte im Modell von Jacob können andere Objekte im Sinne der Bestandteilshierarchie (**Component Objects**) beinhalten. Außerdem können Objekte von anderen Objekten erben (**Inheritance**).

Executive
Für die Beschreibung des Gesamtzusammenhangs führt er ein „Executive“ ein, das die Ereignisse an die Objekte mittels Aufruf einer dem jeweiligen Objekt zugeordneten Koroutine verteilt. Die Objekte erhalten nach Aufruf der Koroutine die Kontrolle und arbeiten die ankommenden Ereignisse ab, bis sie ein Ereignis empfangen, das sie gemäß ihrem aktuellen Zustand im Zustandsübergangsdiagramm nicht akzeptieren. Die Objekte behalten den bis dahin erreichten Zustand und setzen ihre Arbeit in diesem Zustand wieder fort, sobald sie vom Executive wieder ein Ereignis zugeordnet bekommen, das sie in diesem Zustand akzeptieren.
Mit dem Konzept trägt Jacob der Nebenläufigkeit von Dialogen Rechnung und der Tatsache, dass die Darstellung aller Übergänge der Kontrolle von einem Objekt zum anderen mittels

Zustandsübergang in einem alle Objekte umfassenden Übergangs-Diagramm zu einer Unmenge von Übergängen führen würde.
Dabei liegt der Ehrgeiz bei Jacob noch darin, alle diese Übergänge zu modellieren, bzw. das Prinzip der externen Kontrolle, wie es bei grafischen Oberflächen zugrunde liegt, zu beschreiben.

Kommunikation mit anderen Objekten über Funktionen
Das Öffnen von anderen Objekten aufgrund von Ereignissen in einem Objekt wird bei Jacob nicht als Ausgabeereignis gesehen, sondern wird über eine Applikationsfunktion erreicht, während wie bereits oben erwähnt, durchaus Ausgabeereignisse innerhalb eines Objektes als Ereignisse mit ggf. Zustandswechsel dargestellt werden.

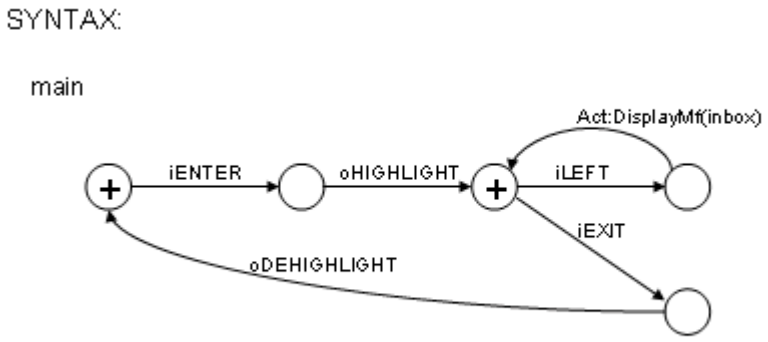


Abbildung 9-1 Beispieldiagramm nach Jacob aus [JACOB86] mit Anzeige eines Oberflächenobjektes über Funktionsaufruf
Das Diagramm zeigt die Zustandsübergänge eines Buttons. Die mit dem Pluszeichen gekennzeichneten Zustände sind Zustände mit Eingabeerwartung. iENTER, iEXIT definieren Tokens für die Positionierung des Mauscursors. oHIGHLIGHT und oDEHIGHILIGHT stehen für Ausgabetokens. Drücken der linken Maustaste (Eingabetoken iLEFT) löst die Aktion DisplayMf(inbox) aus, die das Öffnen eines anderen Dialogobjektes bewirkt.

Kommunikation über Synthetic Tokens
Übergänge in ZÜD´s können mit so genannten Synthetic Tokens, also zusammengesetzten Tokens, versehen werden. Das Erkennen eines derartigen Tokens bestehend aus einer Sequenz elementarer Tokens, wird dann auf ein anderes in einem definierten Scope liegendes Objekt (i.d.Regel hierarchisch untergeordnetes Objekt) verlagert: Nach Erkennen der elementaren Tokensequenz durch das Unterobjekt erfolgt dort ein Zustandsübergang, in dem das erkannte Synthetic Token versendet wird, wodurch dann der Zustandsübergang des auf das Token wartenden (übergeordneten) Objektes durchgeführt wird.

Vergleich mit ESD
Zustandsübergänge können in einem ESD mittels Stellenübergängen realisiert werden, bei denen alle Stellen, die nicht den jeweiligen Zustand repräsentieren, geschlossen werden und umgekehrt alle Stellen, die den neuen Zustand darstellen, geöffnet werden Im einfachen Fall, bei dem eine Stelle genau einen Zustand darstellt, bedeutet dies, mit jeder öffnenden Kante von einer Stelle zur nächsten auch eine schließende Kante von der Ausgangsstelle auf sich selbst zu führen.

In einem ESD ist ebenso eine Hierarchie der Diagramme mit rekursivem Aufruf vorgesehen. Die Conditions im ESD entsprechen denen bei Jacob. Aktionen können über Propagations-funktionen ausgeführt werden. Ein entscheidender Unterschied ist jedoch, dass Propagations-

funktionen in der Regel auf andere Stellen wirken. Im ESD werden damit Werte an anderen Stellen belegt und damit gleichzeitig ein Ereignis ausgelöst, das ggf. wiederum andere Ereignisse nach sich zieht. Die Wirkung auf die Stelle durch die Propagationsfunktion wird grafisch mit der Darstellung des Stellenübergangs, dem die Propagationsfunktion zugeordnet ist, angedeutet.
In gleicher Weise kann im ESD ein Öffnen und Schließen einer Stelle grafisch durch den Stellenübergang angezeigt werden.

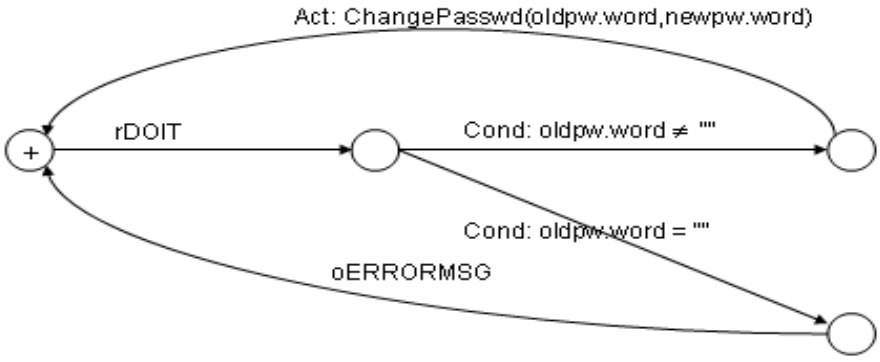
```
INTERACTION_OBJECT FillinForm is

IVARS:
position;
but                 := new INTERACTION_OBJECT FillinButton(
                            position=>position);
oldpw               := new INTERACTION_OBJECT FillinField(
                            position=>position+Height(FillinButton),
                            legend=>"Old password",
                            word=>"");
newpw                := new INTERACTION_OBJECT FillinField(
                            position=>position+Height(FillinButton)+Height(FillinField),
                            legend=>New password");
METHODS:
Draw ()             { DrawBorder(position); }

TOKENS:
oERRORMSG           { ShowErrorMsg("Please enter old password first"); }

SYNTAX:

Main
```




```
end INTERACTION_OBJECT;
```

Abbildung 9-2 Spezifikation eines Dialogs zur Passwortänderung nach Jacob [JACOB86], 1.Teil
Definition des Formulars zur Passwortänderung FillinForm:
Der Abschnitt zur Variablendefinition IVARS enthält neben der Variablen position die drei Component Objects but, oldpw und newpw (Definition siehe nächste Abbildung).
Im Methoden-Abschnitt gibt es eine Methode Draw() zum Zeichen des Formularrahmens.
Für das einzige Token des Formulars oERRORMSG ist im TOKEN-Abschnitt die Implementierung angegeben, die die Ausgabe einer Fehlermeldung bestimmt..
Im Syntaxdiagramm (Abschnitt SYNTAX) erfolgt aus dem Startzustand, der als Eingabezustand mit „+“ gekennzeichnet ist, der Übergang rDOIT, wenn das Synthetic Token DOIT von einem anderen Interaktionsobjekt, im Beispiel dem FillinButton but (siehe nächste Abbildung) erkannt wurde und mittels sDOIT versendet wurde. Danach folgt die Prüfung auf Eingabe eines Passworts (Conditions eingeleitet mit Cond:) und in Abhängigkeit ggf. die Ausgabe der Fehlermeldung oder der Aufruf der Aktion (eingeleitet mit Act:) zum Durchführen der Passwortänderung.
Abbildung 9-3 zeigt die Spezifikation für die Component Objects FillinButton und FillinField.
Abbildung 9-4 zeigt zum Vergleich eine Spezifikation mit einem Ereignisstellendiagramm.

Hinter einer Stelle steckt o.B.d.A. zum Zeitpunkt der Implementierung ein Objekt oder ein Objektattribut. Entsprechend wird erwartet, dass das Öffnen und Schließen sowie die Propagationsfunktion entsprechend implementiert wird. Wird der Stelle z.B. eine Widgetinstanz als Implementierungsobjekt zugeordnet, bedeutet das Öffnen im Default-Fall auch tatsächlich das Öffnen des Widgets am Bildschirm (falls nicht bereits in Zusammenhang mit einer früheren Stelle geschehen). Der von der Propagationsfunktion gelieferte Wert wird standardmäßig dem Attribut des Widgets zugeordnet, das als Default-Wertattribut festgelegt ist (z.B. das Value-Attribut im Widget JText).

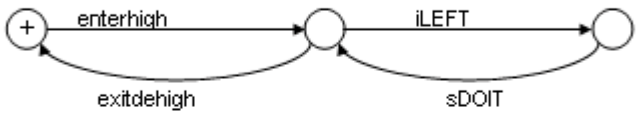
```
INTERACTION_OBJECT FillinButton is
FROM:                GenericButton

IVARS:
position;
legend               := „Change Password";

TOKENS:
sDOIT                { SendTo(parent); }

—This syntax section overloads the standard one inherited from GenericButton
SYNTAX:

main
```



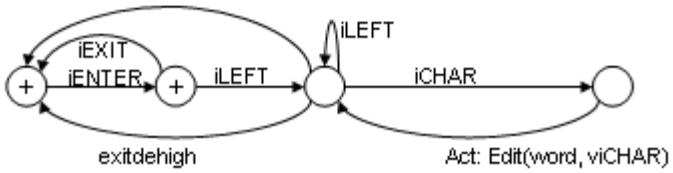

```
end INTERACTION_OBJECT;


INTERACTION_OBJECT FillinField is
FROM:                GenericItem

IVARS:
position;
legend;
word;

METHODS:
Draw ()              { null; } –does not display word

SYNTAX:
main
```




```
end INTERACTION_OBJECT;
```

Abbildung 9-3 Spezifikation Passwortdialog Teil 2 (Fortsetzung von Abbildung 9-2 **) nach Jacob**
Spezifikation der Unterkomponenten des Formulars FillinButton und FillinField: Im TOKENS-Abschnitt des FillinButtons ist zu sehen, dass bei Erkennen des TOKENS sDOIT an das übergeordnete Objekt, in diesem Fall das Formular, die Nachricht des Erkennens des Tokens per Funktionsaufruf SendTo(parent) verschickt wird. Im Syntaxdiagramm ist der entsprechende Übergang sDOIT aufgeführt, der nach einem linken Mausklick (repräsentiert durch das Token iLEFT) nach vorheriger Positionierung der Maus im Button erfolgt. Die Positionierung im Button ist in einem Subdiagramm namens enterhigh enthalten, welches den Übergang nach dem Start im Detail spezifiziert.

INTERACTION_OBJECT FillinField ist die Spezifikation der beiden Felder zur Eingabe des alten und neuen Passworts in FillinForm. Es sei hier noch erwähnt, dass im FROM-Abschnitt Vererbung von anderen Objektspezifikationen, nämlich von GenericButton und GenericItem, angegeben ist.

Im ESD wird das Öffnen eines anderen Objektes bzw. jede Auswirkung auf ein anderes Objekt aufgrund eines Ereignisses in einem Objekt (in einer Stelle) wieder als ein Ereignis (Metaereignis) an einer anderen Stelle mittels „Öffnepfeil“ und/oder „Wertübertragungspfeil“ grafisch dargestellt. Damit kann der Zusammenhang des Dialogs über alle Stellen (in der Implementierung ggf. Objekte) in einem Diagramm dargestellt werden. Der Grundansatz ist im ESD, dass Ereignisse prinzipiell Ereignisse an anderen Stellen oder an den gleichen Stellen nach sich ziehen, wobei es sich dann in letzteren Fall um eine mehrere Ereignisse zusammenfassende Stelle handelt.

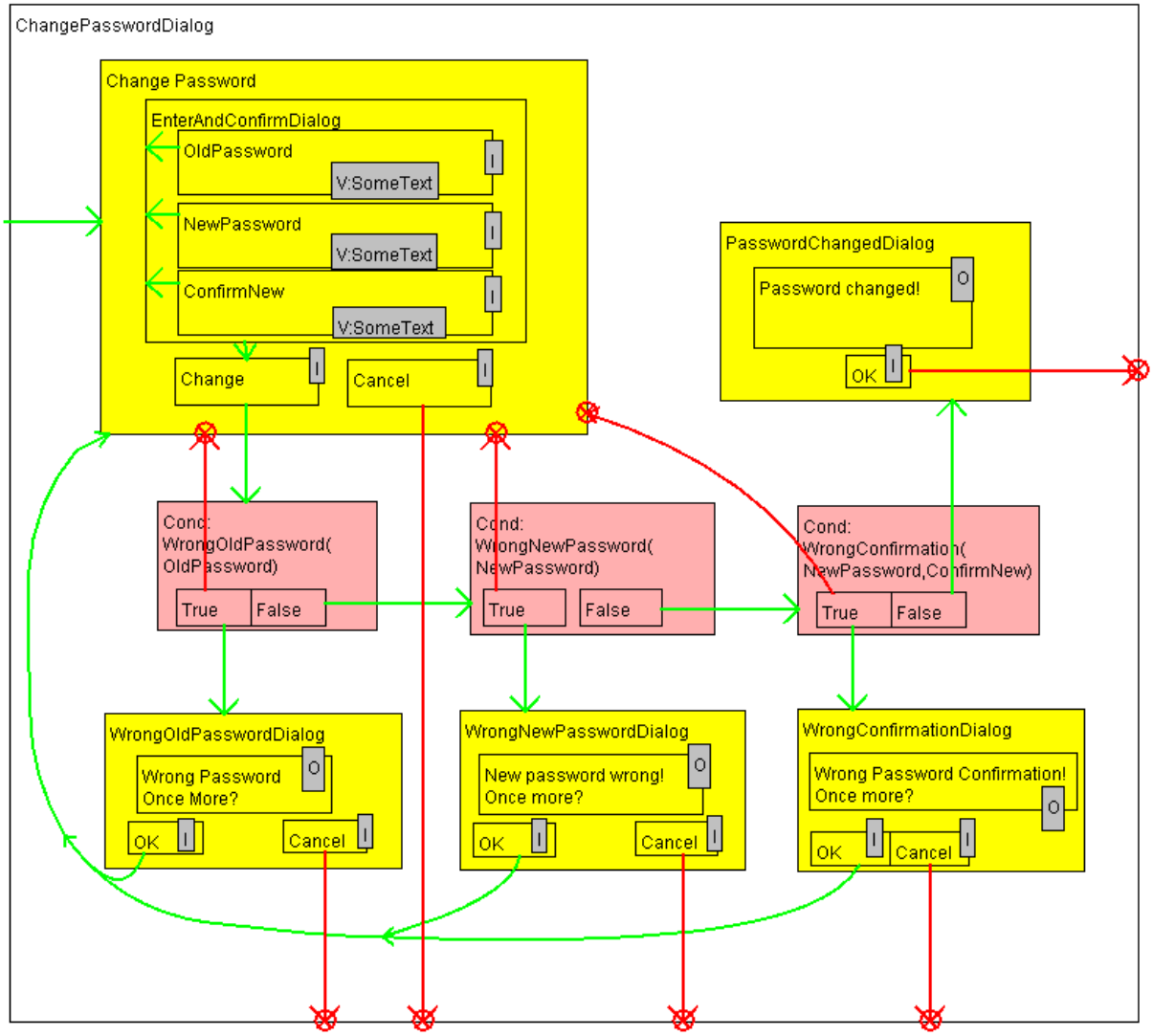


Abbildung 9-4 Ereignisstellendiagramm (ESD) eines Dialogs zur Passwortänderung
Verkürzte Schreibweise für Kanten mit Conditions: Statt mehrere gerichtete Kanten von der Ausgangsstelle zur Zielstelle mit Konditionen darzustellen, wird bei identischen oder sich logisch negierenden Konditionen die Kondition mit ihrer Ausgangskante nur einmal gezeichnet. Die Kanten zu den Zielstellen verzweigen dann von der Kondition aus den entsprechenden Ausgängen „true“ oder „false“. Die drei Konditionen sind außerdem „hintereinandergeschaltet“. Dies stellt ebenfalls eine Alternative zu einer Darstellung mit drei Konditionen dar, deren Kanten direkt nebeneinander aus der Stelle Change geführt würden. Die Konditionen wären dann entsprechend so zu formulieren, dass sich die Übergänge gegenseitig ausschließen.
Die gerichteten Kanten in Grün, die das Eröffnen eines Dialogs bedeuten, sind hier aus den Ausgängen der OK-Dialoge in verkürzender Schreibweise auf eine Zielstelle zusammengeführt.

Im ES-Modell ist auch neben dem Objektbegriff kein Zustandsbegriff explizit vorgesehen, sondern nur Stellen. Erst mit der Zusammenfassung von Ereignissen an ein und derselben Stelle kann es Sinn machen, in der Implementierung mit jedem dieser Ereignisse an der Stelle einen neuen Zustand der zusammenfassenden Stelle zu verbinden, statt der Stelle für die einzelnen Ereignisse Substellen zuzuordnen.

Das Koroutinenkonzept von Jacob wird im Grunde beim ESD als gegeben vorausgesetzt. Nebenläufige Stellen werden einfach nebeneinander dargestellt. Ein Verteilungsmechanismus der Ereignisse an die akzeptierenden Stellen wird als gegeben gefordert. So stellt das Konzept von Jacob in gewisser Weise eine Basis oder Ergänzung zum ESD dar, das aber nicht eigens modelliert werden muss.

Mit der Einführung des Objektbegriffs ist Jacob viel näher an der Implementierung als das ESD mit dem Stellenbegriff. Denn Stellen können sich in der Implementierung in Objekte, aber auch in Objekt-Zustände oder in andere Phänomene, Ressourcen, wie z.B. Objektattribute etc. verwandeln. Stellen stehen grundsätzlich für irgendwelche Ressourcen, an denen bzw. über die (Informations-)Ereignisse wahrgenommen werden können. Mit Stellen können Stellen in einem örtlichen wie in einem zeitlichen Raum verstanden werden. Allerdings sieht das dem ESD standardmäßig zugeordnete Implementierungskonzept vor, dass, wie oben erwähnt, Stellen auf Objektinstanzen bzw. Attribute von Objektinstanzen abgebildet werden. Dieser Default-Abbildungsmechanismus ermöglicht dann eine rasche direkte Umsetzung in die Implementierung. Wird die WYSIWYG-Darstellung im Modell gewählt, ist die Abbildung der Stelle auf ein konkretes Widget bereits unmittelbar vorgegeben.

9.1 Ereignismodell von Green

Die modelltheoretische Basis gängiger, auf dem Markt befindlicher Systeme zur Realisierung von grafischen Dialogoberflächen ist in der Regel ein Ereignismodell. Green stellt ein abstraktes Ereignismodell vor. Er betont dabei, dass in der Praxis das Modell in eine Programmiersprache eingebettet ist, die dem Modell entsprechende Mächtigkeit verleiht.

Merkmale des Ereignismodells
Im Vordergrund stehen Ereignisse und damit verbunden so genannte Event-Handler. Ein Ereignis ist ein Tripel aus Name des Event-Handlers an dem das Ereignis entgegengenommen wird, Name des Ereignisses und Wert des Ereignisses. In der Praxis stellen für Green Event-Handler Routinen einer Programmiersprache dar. Im abstrakten Modell sind jedem Event-Handler eine Reihe von Registern zugeordnet, die Ereigniswerte oder Namen von Event-Handlern speichern können. Ein Event-Handler kann mehrere Ereignistypen verarbeiten. Die an den Event-Handler gesendeten noch nicht verarbeiteten Ereignisse werden in einer Ereignisschlange gehalten. Jedem Ereignistyp wird eine so genannte Event-Handler-Prozedur zugeordnet, die in ihrem Rumpf eine Folge von Statements enthält. Die vorgesehenen sechs Typen von Statements sind

- Expressions zur Berechnung neuer Registerwerte, bestehend aus Registern, Konstanten und Operatoren
- Senden von Ereignissen an einen Event-Handler
- Kreieren eines neuen Event-Handlers
- Löschen eines Event-Handlers
- „If condition THEN statement ELSE statement“, das auch geschachtelt auftreten darf, wobei conditions aus Registern, Konstanten, Operatoren und logischen Konnektoren besteht und

- Dem Aufruf von sogenannten Applikationsprozeduren, denen wiederum Expressions als Parameter übergeben werden

Ein Ereignissystem besteht nun aus einer endlichen Menge von Event-Handlern, wobei die Anzahl der Event-Handler sich dynamisch verändern kann.
Eingaben einer Präsentationskomponente werden in Ereignisse abgebildet, die den daran interessierten Event-Handlern in ihre Ereignisschlange gestellt werden. Das nächste anstehende Ereignis in der Schlange wird von der entsprechenden Event-Handler-Prozedur abgearbeitet, dabei werden die Registerwerte des Event-Handlers verändert und ggf. Ereignisse in die eigene Ereignisschlange oder die anderer Event-Handler gestellt. Die Event-Handler arbeiten unabhängig voneinander die Ereignisse ab. Es wird auch keine Aussage über die Reihenfolge des Sendens von Ereignissen an verschiedene Event-Handler gemacht.

Abgrenzung zum ESM

Ereignisse, die durch Typ und Wert charakterisiert werden, sind sowohl im Ereignismodell wie auch im Ereignisstellenmodell ein zentraler Begriff. Dabei ist im ESM der Ereignistyp, der nach Eingabe, Ausgabe und internem Schreibereignis unterscheidet, abstrakter als der im Ereignismodell, wo zunächst an physikalische Ereignisse, die über Eingabegeräte erzeugt werden, gedacht ist. Aber auch im Ereignismodell werden Ereignistypen, erlaubt, die frei definiert und den Bedürfnissen der Applikation angepasst werden können [GREEN86]. An die Stelle der Event-Handler tritt im ESM der Stellenbegriff.
Interessant ist, dass im Ereignismodell Event-Handler kreiert und gelöscht werden können, wie dies in ähnlicher Form für Stellen im ESM geschehen kann, die eröffnet und geschlossen werden können.
Der Begriff Event-Handler drückt die Vorstellung der prozeduralen Abarbeitung der zugeordneten Ereignisse aus. Eine Stelle im Ereignismodell ist im Unterschied dazu ein abstrakter Begriff, der die Ressource in Ort und Zeit repräsentiert, an der ein Ereignis festgestellt werden kann und an der u.U. ein Wert als Resultat des Ereignisses abgelesen werden kann.
Eine Ereignisstelle repräsentiert also mehr die Quelle des Ereignisses oder das Ereignis selbst, während der Event-Handler die darauf folgende prozedural verstandene Abarbeitung beschreibt. Im Ereignisstellenmodell steckt die prozedurale Sicht der Abarbeitung in den Propagationsfunktionen. Der Grundansatz im ESM, der zum Stellenbegriff führt, liegt darin, dass logischer Dialog aus einer Struktur voneinander abhängiger oder unabhängiger Ereignisse besteht.
Ein wesentlicher Unterschied zwischen Ereignismodell und ESM ist, dass in Form der Öffne- und Schließe-Relationen unmittelbar Abhängigkeiten zwischen Ereignissen ausgedrückt werden. Die Struktur des Dialogs wird damit explizit sichtbar. Ein Ereignis an einer Stelle hat Ereignisse an anderen Stellen zur Folge oder ermöglicht zumindest diese Ereignisse. Dabei handelt es sich sowohl um Ausgabe- wie um Eingabeereignisse. Es ist eine konsequent symmetrische Sicht des Dialogs zwischen Ausgabe- und Eingabe im Wechsel ausgedrückt. Ebenso können Ereignisse unmittelbar Ereignisse an anderen Stellen oder an derselben Stelle unmöglich machen.
Das Ereignismodell ist zunächst für eine grafische Darstellung, wie Green selbst betont [GREEN86], nicht geeignet. Dies liegt wohl in erster Linie an der stark prozeduralen Sicht des Event-Handlers mit Event-Handler-Prozeduren.
Im Gegensatz dazu ist der Ansatz im ESM gerade der, mit Ereigisstellen-Diagrammen eine grafische Darstellung zu ermöglichen und außerdem WYSIWYG-Komponenten als konkrete Ergebnisse bzw. Darstellungsmittel von Ausgabe und Eingabe in die Darstellung mit aufnehmen zu können.
Deshalb wurde hier der Stellenbegriff und die Relationen zwischen Stellen zum Öffnen und Schließen eingeführt. Außerdem wird über Wertübertragungskanten in der grafischen Darstellung die Übertragung von Werten zwischen Stellen visualisierbar.
Im Ereignismodell fehlt zudem ein hierarchischer Ansatz.

9.2 Dialognetze

Janssen [JANSS96] verwendet für die Modellierung der Dialogdynamik so genannte Dialognetze. Er setzt dabei auf einer Variante der Petrinetze, so genannten beschrifteten Bedingungs-/Ereignisnetzen (B/E-Netzen) [JESSEN87] auf, die er um einige Sprachelemente erweitert. Er führt so genannte optionale Flüsse, modale Stellen, komplexe Stellen, den dynamischen Aufruf von Unterdialognetzen und Makros ein.

Interpretation der Marken in Dialognetzen
Ein Kernkonzept der (einfachen) Petrinetze ist, dass eine Transition nur feuert, wenn alle Eingangsstellen der Transition eine so genannte Marke aufweisen und alle reinen Ausgangsstellen keine Marke besitzen. Bei Dialognetzen stehen Stellen für Fenster und der Besitz einer Marke wird interpretiert als „Fenster geöffnet". Nach dem „Feuern" der Transition wandern die Marken von den Eingangsstellen zu den Ausgangsstellen. Dies bedeutet bei Dialognetzen, dass die Fenster der Eingangsstellen geschlossen, die der Ausgangsstellen geöffnet werden.

Nebenläufigkeit
Die Nebenläufigkeit von Dialogen wird in Dialognetzen nun dadurch ausgedrückt, dass ein Fenster als Eingangsstelle und gleichzeitig als Ausgangsstelle einer Transition fungiert. D.h. der Zustand „geöffnet" bleibt beim Öffnen des Fensters eines anderen nebenläufigen Dialogs erhalten, indem eine zusätzliche gerichtete Kante von besagter Transition wieder zur Eingangsstelle zurückführt.

Optionale Flüsse
Um nun ausdrücken zu können, dass von einem ersten Fenster aus ein Übergang zu einem zweiten Fenster mehrfach erfolgen kann, obwohl das zweite Fenster unter Umständen schon geöffnet ist, wird eine optionale Flussrelation eingeführt (gestrichelte Kante zur Ausgangsstelle). Umgekehrt bedeutet eine gestrichelte Kante von einer Eingangsstelle zur Transition, dass für das Feuern einer Transition nicht unbedingt eine Marke an der Eingangsstelle existieren muss, falls an einer anderen Stelle eine Marke existiert. Damit wird der Erfordernis Rechnung getragen, dass eine Transition schalten (feuern) kann, auch wenn nicht alle Fenster der Eingangsstellen geöffnet sind.

Modale Stellen
Werden als modal gekennzeichnete Stellen (fette Umrandung) geöffnet, können Transitionen, die von anderen Stellen ausgehen, nicht mehr schalten.

Komplexe Stellen
Komplexe Stellen (gekennzeichnet mit Doppelrahmen) repräsentieren Dialoge, die in Subnetzen näher spezifiziert werden. Damit wird eine Hierarchisierung der Dialognetze erreicht. Komplexe Stellen können mehrfach in Netzen und Subnetzen auftreten, definieren aber genau eine Instanz eines Fensters. Ein Subnetz besitzt mitunter mehrere Start-Transitionen, so dass ein Subnetz an den entsprechenden komplexen Aufrufstellen unterschiedlich „angesprungen" werden kann.

Dynamischer Aufruf von Unterdialognetzen
Um bei mehrmaligem Übergang zu einer komplexen Stelle jeweils das Kreieren einer neuen Fensterinstanz zu bewirken, werden komplexe Stellen mit einem Stern versehen.

Makros
Ein Dialognetz kann als so genanntes Makro als Muster für mehrere verschiedene Unterdialognetze dienen. Die unterschiedliche Verwendung ergibt sich aus dem Ersetzen von Platzhaltern des Makros in den Beschriftungen (Bezeichnung in spitzen Klammern), die durch die aktuellen Werte an den Aufrufstellen (Werte in runden Klammern) belegt werden.

Voll spezifiziertes Dialognetz
Ein voll spezifiziertes Dialognetz zeichnet sich durch Beschriftungen an den Stellen und Kanten aus. Es sind dies im Wesentlichen in den Stellen Aktionen, die beim Öffnen der Stelle, bzw. beim Schließen der Stelle ausgeführt werden, sowie in den Transitionen ein Tripel, das aus Ereignis, Bedingung und Aktion besteht. Die Transition wird über Ereignis und Bedingung zusätzlich zu den übrigen Schaltregeln gesteuert. Die Aktion der Transition wird ggf. nach der Schließe-Aktion der Eingangsstellen und vor den Öffne-Aktionen der Ausgangsstellen durchgeführt.

Spezifikation des Makrodialogs
Ziel der Darstellung mit Dialognetzen ist die Dialogstruktur auf der Ebene der Fenster bzw. so genannter Sichten [ZIEGLE93], die jeweils einen zusammenhängenden Aufgabenkomplex für den Benutzer im Dialog umfassen. Er unterscheidet die Fensterebene von der Objektebene, die primitive Dialogobjekte in den Fenstern (z.B. Buttons) umfassen. Die Dynamik auf Objektebene wird nicht mit Netzen, sondern mithilfe so genannter Constraints [HUDS89] beschrieben, wobei ein Teil der Dynamikbeschreibung letztlich einer Programmiersprache überlassen bleibt, die in einem Ziel-System (UIMS) verfügbar ist.

Constraints für Mikrodialog
So genannte One-way-Constraints legen prinzipiell die Abhängigkeit des Wertes einer variablen Größe (o.B.d.A. linke Seite) von anderen variablen Größen (in Ausdruck auf rechter Seite) fest und sorgen für die permanente Gültigkeit der definierten Abhängigkeit. Ändert sich der Wert einer Größe auf der rechten Seite, sorgt der Constraint-Mechanismus automatisch für die Neuberechnung des Wertes der Variablen auf der linken Seite.
Janssen verwendet Constraints für die Belegung von Variablen bzw. Werten der Attribute auf der Objektebene, beispielsweise für die Steuerung der Sensitivität von Eingabewidgets (Buttons etc.) und die Werte von Ausgabewidgets (z.B. Labels) in Abhängigkeit von Ausdrücken ggf. mit Bedingungen, die wiederum auf Variablen oder Attributwerte von Objekten zurückgreifen.

Vergleich mit ESD
Ähnlichkeiten finden sich bei Ereignisstellen-Diagrammen und Dialognetzen insbesondere im grundsätzlichen Ansatz, dass Dialoge über Stellen repräsentiert werden und Stellen über Ereignisse aktiviert bzw. geöffnet werden, was im Wesentlichen durch Stellenübergänge beschrieben wird.
Allerdings interpretiert Janssen in Dialognetzen (auch in der Hierarchie komplexer Stellen) Stellen nur als Fenster. Ereignisstellen-Diagramme eröffnen in der Interpretation ihrer hierarchischen Struktur die Implementierung von Dialogobjekten auf der Ebene primitiver Dialogobjekte. In diesem Zusammenhang ist in Dialognetzen auch nicht an die grafische Darstellung von Wertübertragungen zwischen Stellen gedacht. Ebenso gibt es bei Dialognetzen keine Differenzierung in Eingabe- und Ausgabe. Das gilt für Stellen wie für Ereignisse.

Die Vorstellung des Markenflusses, wie er in Petrinetzen eingeführt ist, erfordert vom Entwickler und potentiellen Systembenutzer einen Paradigmenwechsel und behindert manchmal eher eine intuitive Vorstellung vom Dialogablauf.

Zur Darstellung der Dynamik auf der Ebene primitiver Dialogobjekte schlägt Janssen die Anwendung von Constraints vor. In Ereignisstellen-Diagrammen können dagegen Wertzuweisungen über Propagationsfunktionen definiert und mittels Anwendung von Wertübertragungspfeilen grafisch visualisiert werden.
Anstelle der Constraints tritt als deskriptives Sprachelement die Exemplarvariation, die die Möglichkeiten der Beschreibung von Wertabhängigkeiten in Constraints umfasst.

9.3 State-Charts (UML,Harel)

State-Charts haben ihren Einzug in die Unified-Modelling-Language (UML) zur Modellierung dynamischer Vorgänge gefunden. Sie gehen ursprünglich auf Harel [HAREL87] zurück. Varianten existieren etwa von Wellner [WELLN89] für die Modellierung von Dialog. Eine Integration in die Objektorientierte Modellierung stammt von Rumbaugh [RUMBA91].

Wesentliche Merkmale
State-Charts zeigen im Wesentlichen Zustände und Zustandsübergänge, die über Ereignisse getriggert werden. Sowohl Zustandsübergänge wie auch Zustände können mit Aktionen versehen werden. Die den Zuständen zugeordneten Aktionen können für den Eintritt und für den Austritt aus dem Zustand bestimmt sein [WELLN89]. Ebenso können Aktivitäten in einem Zustand permanent ausgeführt werden, bis eine Endebedingung zutrifft (Guards). Übergänge können an Bedingungen (Conditions) geknüpft sein. Ereignisse können neben den Zustandsübergängen auch den Zuständen zugeordnet werden und dort bestimmte Aktionen auslösen. Den Zuständen zugeordnete Ereignisse bewirken keinen Zustandsübergang. Mithilfe des so genannten Broadcasting-Mechanismus können Ereignisse an Zustände propagiert werden, die dann dort die zugeordneten Aktionen auslösen. Nach Definition der UML können Zuständen Zustandsvariablen mitgegeben werden, die eine Untermenge der Attribute einer Objektklasse darstellen.

Gruppierung alternativer und nebenläufiger Zustände
Zustände lassen sich in State-Charts hierarchisch beliebig tief schachteln, d.h. in Unterzustände aufteilen bzw. lassen sich, umgekehrt gesehen (Bottom-up), in Gruppierungszuständen zusammenfassen.
Unterzustände (im Folgenden auch Teilzustände genannt) können dabei nebenläufig aktiv sein. Dies wird durch die Zusammenfassung in einem so genannten **AND-Grouper-Zustand** manifestiert. Stellen die Zustände dagegen sich gegenseitig ausschließende Alternativen dar, werden sie in einem so genannten **XOR-Grouper-Zustand** gebündelt.
In State-Charts werden Zustände in abgerundeten Rechtecken dargestellt, wobei alternative Teilzustände in Form abgerundeter Rechtecke innerhalb des übergeordneten Zustandsrechtecks liegen. Zur Darstellung paralleler Teil-Zustände wird das Rechteck des übergeordneten Zustands (AND-Grouper) durch gestrichelte Linien unterteilt, wobei die so sich ergebenden Teilflächen jeweils einen nebenläufigen Unterzustand beschreiben.
Der Vorteil der Darstellung liegt darin, dass durch die hierarchische Unterteilung in Teilzustände die Anzahl der Zustände bzw. Zustandsübergänge stark reduziert wird. Es müssen nicht mehr alle möglichen Zuständsübergänge von einem Zustand einer Gruppe zu einem Zustand der anderen Gruppe gezeichnet werden. Zustandsübergänge von den

Unterzuständen nach außen oder von äußeren Zuständen nach innen in die XOR-Gruppe bzw. AND-Gruppe können (teilweise) durch jeweils einen Übergang zum übergeordneten Gruppenzustand ersetzt werden: Ein Übergang zum übergeordneten Zustand (XOR-Grouper oder AND-Grouper) bedeutet implizit auch einen Übergang zu einem dann aktiven Unterzustand im XOR-Grouper bzw. zu allen Unterzuständen eines AND-Grouper-Zustands.

Vergleich mit Ereignisstellendiagrammen (ESD´s)
Der Gedanke der hierarchischen Aufteilung und der Darstellung nebenläufiger Zustände ist aus den State-Charts in die Darstellung in Ereignisstellendiagrammen eingegangen. Allerdings sind in Ereignisstellendiagrammen Stellen, die nicht auf einem Pfad über Triggerkanten direkt oder indirekt verbunden sind, grundsätzlich als nebenläufige Stellen aufzufassen, d.h. als Stellen an denen prinzipiell gleichzeitig Ereignisse akzeptiert werden. Die Aufteilung eines Zustandes respektive einer Stelle durch gestrichelte Linien entfällt. Hinzu kommt dagegen die Notwendigkeit bzw. Möglichkeit, eine Stelle explizit zu schließen, um zu kennzeichnen, dass die Stelle keine Ereignisse mehr annimmt.

Während bei State-Charts der Übergang von einem Zustand A zu einem anderen Zustand B automatisch den Zustand A schließt, bleibt die Stelle A in einem ESD nach wie vor geöffnet, d.h. akzeptiert Ereignisse, wenn ein Übergang zu einer Stelle B erfolgt. Der Übergang zu Stelle B bedeutet nur ein zusätzliches Eröffnen einer weiteren Menge von Ereignissen oder ein Schließen der Stelle B. Die damit mögliche Darstellung der inkrementellen Erweiterung und der schrittweisen Reduktion von akzeptierten Ereignismengen (an unterschiedlichen Stellen) ist in State-Charts so nicht gegeben. Was zunächst als Nachteil eines ESD angesehen werden könnte, nämlich dass die Stellen in einem ESD explizit geschlossen werden müssen, damit sie keine Ereignisse mehr akzeptieren, erweist sich in vielen Fällen als Vorteil:

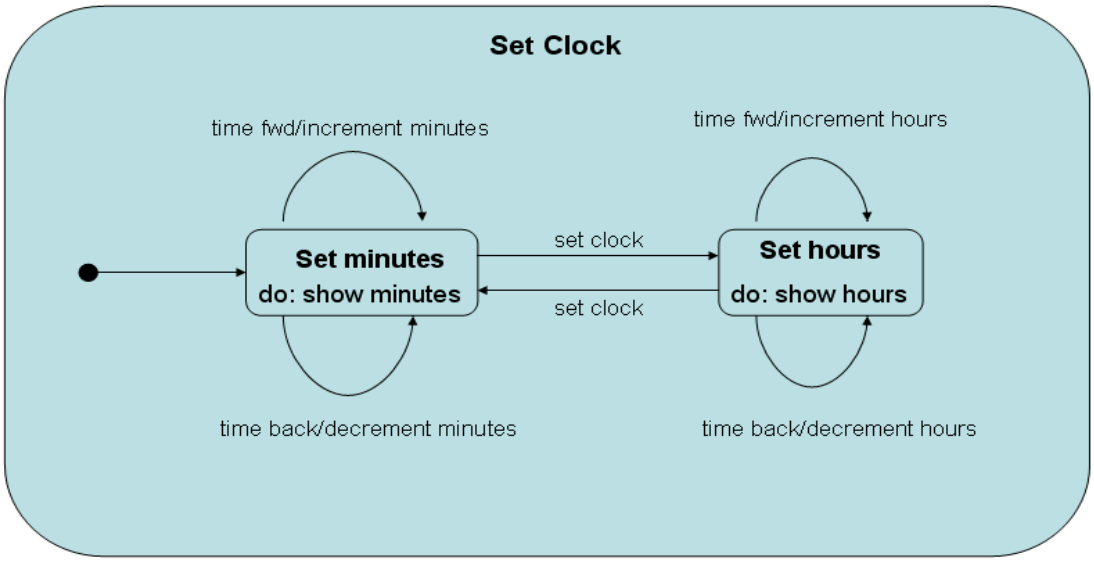


Abbildung 9-5 State Chart für Set Clock Dialog nach Rumbaugh [RUMBA91]
Die Abbildung zeigt ein State Chart mit einem Xor-Gruppierungs-Zustand „Set Clock“, der zwei sich gegenseitig ausschließende Unterzustände „Set minutes“ und „Set hours“ zum Einstellen der Uhrzeit enthält. „Set minutes“ ist Startzustand. Das Ereignis „set clock“ bewirkt einen Wechsel in Zustand „Set hours“ und umgekehrt. Innerhalb eines Zustands laufen permanent Aktivitäten, die die Ausgabe „show minutes“ bzw. „show hours“ realisieren sollen. Die Ereignisse „time fwd“ bzw. „time back“ bewirken jeweils die dahinter stehende Aktion. Das State Chart zeigt eine typische Modellierung mit dem Zustandsparadigma, wobei Ausgaben in Aktionen versteckt auftreten und Eingaben als Ereignisse an Kanten zu finden sind.

Sie müssen nicht mehr explizit geöffnet werden, wenn an anderen Stellen Ereignisse stattgefunden haben. Abgesehen von den nicht mehr nötigen gestrichelten Linien sind also auch keine expliziten Übergänge mehr in die Stellen zum erneuten Öffnen notwendig.

In State-Charts fehlt im Gegensatz zu ESD´s die direkte Visualisierung von Wertübertragungen. Die Kanten im State-Chart zeigen den durch Ereignisse gesteuerten Kontrollfluss von Zustand zu Zustand. Der Datenflussaspekt, der sich in den Kanten zur Wertübertragung im ESD widerspiegelt, ist in State-Charts nicht grafisch unterstützt. Dies manifestiert sich auch darin, dass Ausgaben am Bildschirm in Aktionen „verborgen“ sind und bei Ereignissen zunächst an Eingabeereignisse gedacht wird [WELLN89].
In ESD´s hingegen sind Ausgaben wie Eingaben Stellen zugewiesen, an denen als Resultat der dort stattfindenden Ereignisse Werte vom Benutzer bzw. System gelesen werden können. Dies ist Voraussetzung für eine WYSIWYG-ähnliche Darstellung von grafischem Dialog.

Insgesamt gesehen sind Ereignisstellendiagramme mehr am Ereignismodell orientiert, das gängigen Systemen zur Realisierung grafischer Oberflächen zu Grunde liegt. Der Modellierer denkt bei ESD´s in erster Linie an Stellen, an denen Ereignisse passieren und nicht an Zustände. Dabei ist aber sehr wohl eine Darstellung von Zuständen möglich. Im Gegensatz zu State-Charts sind ESD´s für eine WYSIWYG-Darstellung von Oberflächenobjekten im Modelldiagramm geschaffen.

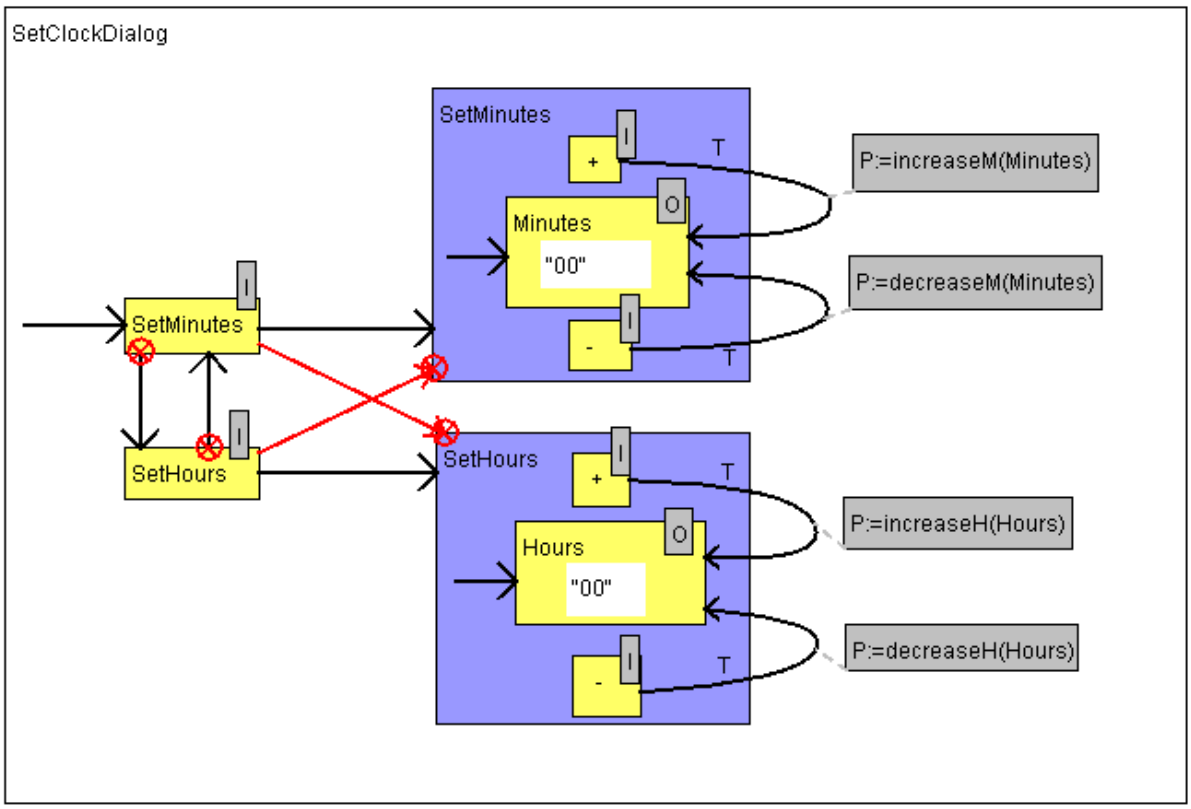


Abbildung 9-6 Set Clock Dialog in ESD-Darstellung
Die Abbildung zeigt die Modellierung des Rumbaugh-Beispiels (Abbildung 9-5) mit einem Ereignisstellen-Diagramm. Im Gegensatz zum State Chart sind hier den Ausgabe- und Eingabeereignissen jeweils Stellen zugeordnet, deren Typ durch die Annotationen (I bzw. O) erkennbar ist. Aktionen werden lediglich in Form von Propagationsfunktionen zum Hochzählen und Erniedrigen der Stellenwerte eingesetzt. Die angegebenen exemplarischen Werte „00“ dienen als Voreinstellung.

9.4 Pi-Kalkül

Robin Milner stellt mit dem so genannten Pi-Kalkül [MILNER99] eine Theorie vor, mit der der Ablauf und das Zusammenwirken konkurrierender Prozesse beschrieben werden kann. Ein besonderes Augenmerk des Kalküls liegt auf der Beschreibung des dynamischen Aufbaus und Abbaus von Verbindungen (process links) zwischen existierenden Prozessen, was mit dem Begriff Mobilität von Prozessen bzw. besser Mobilität von Prozessverbindungen charakterisiert wird. Als Anwendungsbeispiele werden von Milner etwa der Mobilfunk, das Internet aber auch Prozessabläufe innerhalb eines Rechners, z.B. die Zuteilung von Ressourcen an Prozesse etc. genannt.
Der Ansatz ist jedoch so allgemein, dass er auch für die Beschreibung von Dialogsystemen und die Modellierung von Dialog zwischen Mensch und Maschine in Betracht gezogen werden kann und näher untersucht werden muss. Im Rahmen dieser Arbeit sollen nur einige wesentliche Charakteristika im Vergleich mit dem vorgestellten Ereignisstellenmodell angesprochen werden. Eine ausführliche Beschreibung und Untersuchung kann und soll hier nicht vorgenommen werden.

Der Pi-Kalkül baut auf einem vorangehend entwickelten Prozess-Kalkül (Calculus of Communicating Systems CCS, [MILNER80]) auf. Andere Beiträge zum Thema Kommunikation konkurrierender Prozesse stammen z.B. von Hoare (Communicating Sequential Processes [HOARE85]).
Der Pi-Kalkül umfasst einige wenige Sprachkonstrukte, mit denen so genannte Prozessausdrücke P formuliert werden können:

$$P \;::=\; \sum \pi_i \,.\, P_i \;\mid\; P_1 \mid P_2 \;\mid\; \text{new a } P \;\mid\; !P$$

Ein Prozessausdruck setzt sich also aus Summation, Komposition, Restriktion oder Replikation zusammen.

Summation $\sum \pi_i \,.\, P_i$:
Eine Summation, z.B.

in x(y) . A + **out** w(z) .B

stellt im Wesentlichen alternative Prozessabläufe, im Beispiel also die beiden Alternativen **in** x(y) . A und **out** w(z) .B dar. (Anstelle von **in** und **out** werden in der Darstellung bei Milner auch alternativ bei Eingaben **in** und bei Ausgaben **out** weggelassen und stattdessen Namen für Ausgabeaktionen mit obenliegendem Querbalken versehen.)
Dabei sind **in** x(y) bzw. **out** w(z) sogenannte Aktions-Präfixe, die den Prozesskomponenten A bzw. B vorangestellt sind. **in** x(y) bzw. **out** w(z) sind Aktionen, die durchgeführt werden müssen, bevor die durch A bzw. B repräsentierten Prozessteile ablaufen können.

Als Präfixe kommen im Pi-Kalkül drei Arten in Frage, die das Senden einer Nachricht, das Empfangen einer Nachricht oder die Durchführung einer „nicht beobachtbaren Transition“ (siehe unten Transitionssystem) bedeuten, also:

π_i ::=	**in** x(y)	steht für: Empfangen einer Nachricht y mittels x
	out w(z)	steht für: Senden einer Nachricht z mittels w
	τ	steht für: nicht beobachtbare Transition

x,y,w,z sind Namen, die zum einen für so genannte Ports stehen, zum anderen aber für Nachrichten, bzw. Objekte, die mittels der Ports von einem Prozess bzw. einer

Prozesskomponente zur anderen gesendet werden. Im obigen Beispiel sind mit x und w Ports, mit y und z Nachrichten gemeint.
Entscheidend im Pi-Kalkül ist aber, dass derselbe Name sowohl die Rolle eines Ports als auch die Rolle einer Nachricht übernehmen kann. Hinter den Namen verbergen sich letztlich Objekte, die ggf. im Kalkül abhängig von der Betrachtung unterschiedliche Rollen spielen. So kann eben der Name eines Ports, also einer Verbindung zwischen Prozessen, als Nachricht wiederum über einen anderen Port übermittelt werden und an einer bestimmten anderen Stelle dann selbst als Port dienen (siehe unten Namensersetzung bei Reaktion).

Komposition $P_1 \mid P_2$:
Das Zeichen „|“ steht für die Nebenläufigkeit der beiden Prozesskomponenten P1 und P2.

Restriktion new a P: mit dem Restriktionsoperator new wird der Name a innerhalb P als lokal gültiger Name gekennzeichnet. Der Scope von a ist auf P eingeschränkt.

Replikation !P:
Das Zeichen „!“ kennzeichnet, dass der nachfolgende Ausdruck beliebig oft auftreten kann. Da mittels Replikation auch Rekursivität ausgedrückt werden kann, wird auf letztere als Sprachmittel zugunsten einer minimierten Darstellung im Pi-Kalkül verzichtet.

Transitionssystem (Labelled Transition System LTS)
Die oben skizzierten Prozessausdrücke stellen jeweils Zustände eines Systems konkurrierender Prozesse dar. Ein Prozessausduck definiert letztlich, welche Zustandsübergänge im System möglich sind. Dabei wird zwischen den Übergängen innerhalb eines Prozesses und den Übergängen unterschieden, die durch Interaktion der Prozesse miteinander erfolgen.
Für die möglichen Übergänge von einem Prozessausdruck zum anderen werden fünf sogenannte Reaktionsregeln (TAU, REACT, PAR, RES und STRUCT) definiert, die die Basis für die Transitionen eines Transitionssystems bilden.
Als wohl charakteristische Regel für das Kalkül sei hier die Regel REACT näher erläutert:

REACT: (**in** x(y).P + M) | (**out** x(z).Q + N) $\rightarrow$ { z/y } P | Q

Die Regel beschreibt den Übergang des Systems, der dann erfolgt, wenn zwei nebenläufige Prozesskomponenten über einen gemeinsamen Port interagieren.
In obiger Formel sind dies die Komponenten (**in** x(y).P + M) und (**out** x(z).Q + N).
In beiden Komponenten stehen als nächste mögliche Aktionen die Aktionen **in** x(y) bzw. **out** x(z) über den gemeinsamen Port x an. Die rechte Komponente sendet dabei über Port x das Objekt z (mittels **out** x(z)), das von der linken Komponente empfangen wird (**in** x(y)).

Namensersetzung bei Reaktion
Nun erfolgt eine Ersetzung von y durch z in dem nach **in** x(y). folgenden Ausdruck P, was in der obigen Regel durch das Voranstellen von { z/y } auf der rechten Seite der Ersetzungsregel ausgedrückt wird. Taucht nun innerhalb von P vor der Namensersetzung y als Port auf, etwa in der Form **out** y(w), ist nach der Ersetzung an der Stelle von y nun z und damit z als Port zugeordnet.
Als Ergebnis der Reaktion bzw. Interaktion beider Prozesskomponenten oder Prozesse bleiben nun die beiden Prozesskomponenten oder Prozesse P und Q. Die Alternativen Prozessabläufe M bzw. N fallen weg.

Grafische Darstellung

Die Entwicklung des Kalküls geschieht nicht im Hinblick auf eine intuitive, grafische Darstellung. Das Ziel ist vielmehr, eine Theorie zu entwickeln, um Aussagen etwa über die Äquivalenz von Prozessstrukturen im Sinne eines äquivalenten Verhaltens der Prozesse machen zu können (Einführung des Begriffs der Bisimulation etc.).
Ein Vorschlag zur grafischen Visualisierung stammt jedoch z.B. von Joachim Parrow [PARRO95].

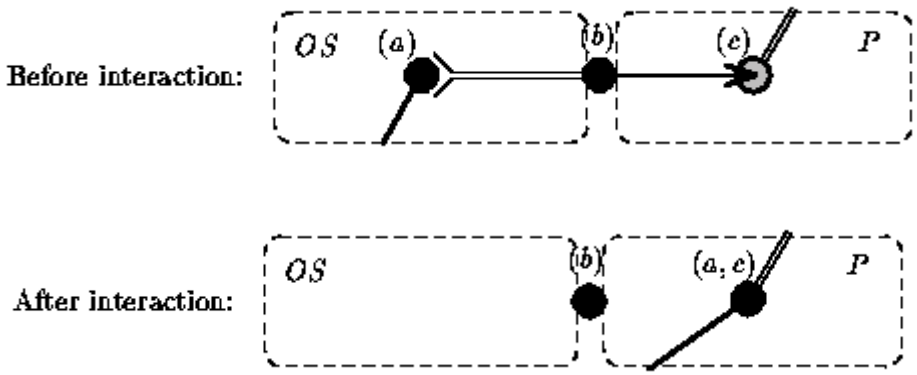


Abbildung 9-7 Grafische Darstellung des Pi-Kalküls mit Interaction Diagrams nach Parrow aus [PARRO95] (Diagrammausschnitte).

Dargestellt sind in **Abbildung 9-7** zwei Diagrammausschnitte, die zwei Prozesse vor und nach der Durchführung einer Interaktion zeigen. Der Prozess OS (Operating System) stellt dem Prozess P über den Port b ein Objekt a an der Stelle c zur Verfügung. Bei a handelt es sich dabei um eine Verbindung (Link) zu einem anderen Prozess oder zu einer Ressource (im Beispiel Parrows einen Drucker). Über die Verbindung können nach erfolgter Interaktion Daten von P oder einer anderen Komponente über P an den Drucker gesendet werden. Die mit doppelter Linie gezogene Kante zwischen a und b repräsentiert eine Ausgabe, nach obiger Darstellung im Kalkül **out** b(a) bzw. das Senden von a nach b. Der mit einfach gezogener Line geführte Pfeil repräsentiert das Gegenstück für eine Interaktion, nämlich **in** b(c) bzw. das Empfangen eines Objektes c über Port b. Dabei ist c nur Platzhalter, was in Interaktionsdiagrammen mittels schwach schattierter oder leerer Kreise dargestellt wird. „Konkrete“ Objekte werden als gefüllte Kreise dargestellt.

Neben den oben dargestellten Diagramm-Elementen gibt es eine Reihe weiterer, z.B. doppelt und einfach umrandete Rechtecke zur Darstellung von Replikation und Rekursion, polyadische Pfeile zur Darstellung der Übertragung von Datenvektoren, etc.

Mithilfe von Interaktionsdiagrammen können letztlich mit relativ wenigen und einfachen grafischen Elementen auch mehr oder weniger komplexe Funktionen und Speicherstrukturen in der Art von Schaltplänen dargestellt werden.

Vergleich Interaction Diagram mit ESD

Mit den Darstellungselementen eines Interaction Diagrams sind keine einfachen WYSIWYG-orientierten Darstellungen einer grafischen Dialogoberfläche möglich. Dies liegt z.B. neben anderen Eigenschaften eines Interaction Diagrams daran, dass eine detaillierte Darstellung der Interaktion, nämlich eine Darstellung sowohl des Sendens als auch des Empfangens eines Datums über einen Port, vorgenommen wird. Dies führt bereits zu drei Stellen im Diagramm, die mit Pfeilen verbunden sind. In einem ESD wird dagegen für die Darstellung der Interaktion zwischen Mensch und Maschine in der Regel eine Stelle benötigt, die mit den

entsprechenden Typen für Eingabe, Ausgabe oder beides versehen wird. Was die Interaktion zwischen Komponenten auf der Maschine betrifft, resultieren die Bezüge zwischen diesen in einem ESD in den Pfeilen, die ein Öffnen, Schließen oder die Übertragung eines Wertes kennzeichnen. Auch hier werden nur zwei Stellen, nämlich Quelle und Ziel benötigt. Auf die Darstellung eines Ports wird verzichtet. Sind die beiden Stellen Dialogstellen der Oberfläche, entsprechen diese in der Regel den Stellen, die auch in der späteren Konkretisierung als grafische Elemente am Bildschirm erscheinen (natürlich vorausgesetzt, wir bilden auf eine grafische Oberfläche und nicht auf andere Dialogarten ab).

Informeller Modellvergleich zum ESM
Der Modellansatz des Pi-Kalküls wie auch des Vorgängers CCS weist eine Reihe interessanter Gemeinsamkeiten mit dem Ansatz auf, der zum Ereignisstellenmodell (ESM) führte.

Wie im ESM werden Aktionen und Ereignisse gleichgesetzt. Milner geht sogar soweit, dass er Aktionen auch als Beobachtungen (Observations) bezeichnet, womit der Vorgang des Lesens (im ESM Ereignistyp L für Lesen) als eine Seite der Interaktion zwischen zwei kommunizierenden Komponenten zum Ausdruck gebracht wird. Im Gegensatz dazu sind die internen Ereignisse innerhalb eines Prozesses nicht beobachtbar (Transition τ im Pi-Kalkül). Im ESM dürfte diese Sicht in etwa der synchronen Abarbeitung von Ereignissen in der Maschine an unsichtbaren Stellen (Hilfsstellen) entsprechen.

Wie im ESM ist die Nebenläufigkeit durch „einfaches Nebeneinanderstellen“ (Komposition) der nebenläufigen Komponenten ausgedrückt. Die Darstellung von Abhängigkeiten im sequentiellen Ablauf werden im Pi-Kalkül durch Präfixe ausgedrückt, im ESM entspricht dies im Wesentlichen den Relationen zum Öffnen und Schließen der Stellen.
Alternative Übergänge, wie sie im Pi-Kalkül mittels der Summation beschrieben werden, werden im ESM in einer Kombination von Öffne-, Schließe-Relation und Conditions sowie Propagationsfunktionen gesteuert. Welche der Alternativen einer Summation zum Übergang jeweils gewählt wird, hängt im Pi-Kalkül von der Synchronisation mit anderen Prozesskomponenten ab.
Hier ist der Ansatz im Pi-Kalkül rigoroser, durchgängiger als im ESM, wo auf Conditions und Propagationsfunktionen, die in einer gängigen Programmiersprache zu schreiben sind, die Entscheidung für die Fortsetzung „ausgelagert“ wird. Allerdings trägt dieser konsequente Ansatz im Pi-Kalkül nicht zu einer einfachen Realisierung und übersichtlichen Darstellung bei. Im ESM ist bewusst die Auslagerung komplexer Vorgänge auf Funktionen einer gängigen mächtigen Programmiersprache gewählt, um die Übersicht in einer Diagrammdarstellung zu vereinfachen und um die Programmierung in einer gewohnten, mächtigen Sprache zu ermöglichen.
Ähnliche Überlegungen gelten, wenn wir das Sprachelement der Replikation im Pi-Kalkül mit dem Variationskonzept im ESM bzw. in der Ergänzung zum ESM vergleichen: Die Flexibilität, die durch die Replikation ausgedrückt werden kann, ist in einem ESD weitgehend in die Annotation von Variationsinformation ausgelagert. Für die Replikation gibt es im ESM ohne Variation, was dem grafischen Anteil der Darstellung im ESD entspricht, kein eigenes Sprachelement.
Der new Operator des Pi-Kalküls entspricht weitgehend der Erzeugung einer neuen Instanz. Auch hierfür gibt es im ESM ohne Variation selbst kein entsprechendes Sprachelement. Welche Menge von Instanzen einer Stelle in der Implementierung zugeordnet werden, wird ebenfalls über textuelle Variationsinformation bestimmt.

9.5 Hierarchical Interactive Graph Templates (HITs)

Im Rahmen der Gesamtarchitektur von FUSE (Formal User Interface Specification Environment [LOSCH96b][LOSCH96a]) stellt Schreiber im System BOSS (Benutzer-Oberflächen-Spezifikations-System [SCHREI94] [SCHREI95]) die so genannte HIT-Spezifikationstechnik vor [SCHREI97].
Hierarchical Interactive Graph Templates (HITs) stellen Bausteine dar, mit denen hierarchisch strukturiert das Modell einer Dialoganwendung formuliert werden kann. Basis der Technik sind so genannte dynamische Attributgrammatiken [GANZ78][PRJD96], die um ein Ereigniskonzept erweitert wurden.
Es werden im Folgenden die wesentlichen Sprachelemente der Hitspezifikation angesprochen, soweit dies zum Vergleich mit den Elementen eines ESM notwendig ist. Zu einer exakten, vollständigen Beschreibung sei auf [SCHREI97] verwiesen.

Bausteine der HIT-Spezifikation
Die wesentlichen zusammenfassenden Bausteine der Modellierung sind so genannte Tupel-Hits und Alternativen-Hits. Eine Hitspezifikation setzt sich aus einer Hierarchie dieser Bausteine zusammen.

Alternativen-Hits
Ein Alternativen-Hit, der als Teilkomponenten wiederum Hits beinhaltet, dient zur alternativen Auswahl und Aktivierung einer seiner Teilkomponenten. Ob ein Hit aktiv wird, hängt von seiner zugeordneten **Anwendbarkeitsbedingung** ab.
Ein Alternativen-Hit besitzt so genannte Slots zur Darstellung von Werten, die in **Argumentslots** und **Resultatslots** unterschieden werden. Die Hit-Bausteine, die die alternativen Teilkomponenten eines Alternativenhits bilden, besitzen die gleichen Argumentslots und Resultatslots wie der sie zusammenfassende Alternativen-Hit, so dass die Werte der Argumentslots des Alternativen-Hits an die ausgewählte Teilkomponente weitergegeben werden können bzw. aus den Resultatsslots der ausgewählten Teilkomponente an den Alternativen-Hit zurück übergeben werden können.
Neben den Slots besitzt ein Alternativen-Hit auch ggf. so genannte **Messageports** als Argumente oder Resultate. Im Wesentlichen unterscheiden sich Messageports von Slots dadurch, dass sie so genannte Nachrichten aufnehmen können und diese auch an andere Stellen weitergeben (propagieren) können. Messageports besitzen, etwas informell gesagt, einen mehr aktiven und Slots einen eher passiven Charakter.
Die Werte, die beispielsweise in Slots oder Messageports gespeichert werden, werden durch die Zuordnung ganz bestimmter so genannter **Sorten** zu den Slots bzw. Messageports typisiert. Eine entsprechende Sortierung gilt auch bezüglich Werten anderer Hitelemente (siehe unten Regeln, Vorbedingungen etc.).

Tupel-Hits
Ein Tupelhit besitzt eine u.U. auch leere Menge von Hits als Teilkomponenten sowie eine Menge von Regeln. Neben Argument- und Resultatslots hat ein Tupelhit außerdem so genannte lokale Slots. Lokale Slots können so genannte Eingabe- oder Ausgabeslots sein. Diese dienen der Speicherung von Werten der Eingabe des Benutzers bzw. der Ausgabe zur Benutzeroberfläche. Neben den Slots kann ein Tupel-Hit wiederum entsprechende Messageports mit einer den Slots entsprechenden Typisierung besitzen.

Regeln definieren letztlich Aktionen, in denen Argumentwerte in Resultatwerte umgewandelt werden. Bei der Ausführung von Regeln werden Werte aus so genannten Slots oder Messageports, die den Hits zugeordnet sind, ausgelesen und gehen in eine Berechnung ein, deren Ergebnisse wiederum anderen Slots oder Messageports zugeführt werden.
Regeln besitzen so genannte **Vorbedingungen**, die darüber entscheiden, ob eine Regel in einer bestimmten Situation ausgelöst werden kann.
Es gibt verschiedene Regeltypen, die sich durch die Art der Aktivierung (Auslösung), in ihrer Versorgung mit Argumentwerten, in der Weiterleitung ihrer Resultatwerte bzw. in ihrer Wirkung unterscheiden:

Gleichungsregeln
Gleichungsregeln werden berechnet (ausgelöst), wenn beispielsweise die entsprechende HIT-Instanz instantiiert wird oder wenn bei einem durch ein Ereignis ausgelösten Zustands-übergang der Instanz eines der Argumente der Regel betroffen ist. Gleichungsregeln haben als Argumente Slots (genauer Slotvorkommen in den Instanzen) und keine Messageports. Weitere erlaubte Argumente sind jeweils Referenzen auf Eingabeslots, Eingabe-Messageports, Benutzertransaktionsregeln und Instantiierungsregeln.

Transaktionsregeln
Im Gegensatz zu Gleichungsregeln besitzen Transaktionsregeln neben Slots mindestens einen Messageport als Argument und können als Ausgänge sowohl Slots als auch wieder Messageports besitzen, an die Nachrichten weitergegeben werden können. Transaktionsregeln werden ausgelöst, wenn an all ihren Messageporteingängen Nachrichten anliegen.

Benutzertransaktionsregeln
Benutzertransaktionsregeln sind eine besondere Form von Transaktionsregeln. Sie werden durch ein Benutzerereignis ausgelöst und können zusätzlich so genannte interaktive Argumente besitzen, die bei Eingabe vom Benutzer mitgegeben werden.
Benutzertransaktionsregeln gleichen bezüglich ihrer Eingänge sonst den Gleichungsregeln und bezüglich ihrer Ausgänge den anderen Transaktionsregeln.

Instantiierungsregeln
Statt der Erzeugung von Resultatwerten haben Instantiierungsregeln die Wirkung, dass eine Hitinstanz angelegt wird. Sie werden wie Benutzertransaktionsregeln ausgelöst und können die gleichen Argumentarten besitzen.

Informelle Gegenüberstellung Hit-Spezifikation und ESM
Im ESM tritt als zentraler Begriff die Ereignisstelle an die Stelle der Begriffe Hit, Slot und Messageport einer Hitspezifikation. Dies bedeutet, dass im ESM kein begrifflicher Unterschied gemacht wird zwischen einer zusammenfassenden Einheit, wie dem Hit einserseits, der zudem noch in Alternativen- und Tupelhit differenziert wird, und einer Einheit andererseits, der unmittelbar ein Wert zugeordnet werden kann, nämlich einem Slot bzw. einem Messageport.
Stattdessen existiert im ESM in der Regel eine Hierarchie von Stellen. Es gibt keine explizite Unterscheidung zwischen Alternativ- und Tupelstellen. Ob eine Stelle alternativ zu einer anderen Stelle geöffnet (instantiiert) wird, wird über entsprechende alternative Conditions an Stellenübergängen zu diesen Stellen gesteuert.
Es bedeutet weiter, dass der Begriff Stelle im ESM auch die Aufgabe der Begriffe Slot und Messageport übernimmt. Dies liegt im Wesentlichen daran, dass eine Stelle einerseits wie ein Slot oder Messageport einen Wert besitzt und andererseits eine Stelle Ereignisse erwarten kann, die wiederum Ereignisse mit Werten an anderen Stellen propagieren können, was in der

Hitsprechweise dem Verbreiten einer Nachricht via Messageport entspricht. Ähnlich wie bei einem Messageport kann eine Stelle einen Übergang, eine Transaktion, auslösen, wenn ein Ereignis an der Stelle sich ereignet, was beim Messageport in diesem Sinne dem Eintreffen einer Nachricht und Propagieren der Nachricht gleichgesetzt werden kann.
Stellen im ESM werden nach Typ Eingabe, Ausgabe und Hilfsstelle (=verborgene Stelle für Schreibereignisse des Systems) unterschieden. Eine differenziertere Typisierung wie die der Slots und Messageports mit Argument- und Resultatslots sowie lokalen Slots, wobei nur diese Eingabe oder Ausgabeslots sein können, ist im ESM nicht gemacht.
An die Stelle der Anwendbarkeitsbedingung und der Vorbedingungen tritt im Wesentlichen im ESM die Condition. Dabei ist allerdings zu unterscheiden, dass eine Condition einem Stellenübergang zugeordnet wird, während Anwendbarkeitsbedingungen den Hitbausteinen unmittelbar zugeordnet und Vorbedingungen den Regeln, Eingabe-Messageports und Eingabeslots zugewiesen sind. Letztlich kann aber über beide Mechanismen vergleichbar gesteuert werden, ob Objekte instantiiert werden, ob Ereignisse stattfinden und bestimmte Werte gesetzt werden können.
Die Funktion der verschiedenen Regeln der Hitspezifikation kann in gewissem Sinne im Kern mit der von Propagationsfunktionen, die Stellenübergängen im ESM zugeordnet sind, gleichgesetzt werden. Regeln werden wie Propagationsfunktionen bei Eintreten bestimmter Ereignisse aktiviert und berechnen Werte, die an Slots und Messageports respektive Ereignisstellen weitergegeben werden. Instantiierungsregeln übernehmen dabei zusätzlich die Aufgabe der Instantiierung eines Hits, was beim ESM bereits alleine durch den Stellenübergang einer Öffnerelation bewirkt wird.
Insgesamt kann festgestellt werden, dass die verwendeten Begriffe im ESM weniger differenziert und bewusst auf einer einfachen klar strukturierten abstrakten Ebene gehalten sind. Im Vergleich dazu werden in der Hitspezifikation die Elemente bereits stärker differenziert. Die Sprache ist komplexer. Der Benutzer ist dadurch eher gezwungen, im Entwurf bereits bestimmte spezifische Elemente einzusetzen. Die Wahl der Begriffe der Hitspezifikation ist eher technisch abstrakt und weniger an die Vorstellungen und Bedürfnisse auch laienhafter Benutzer angepasst.
Für die frühe Entwurfsphase scheint daher die ESM-Sprache eher geeignet. Ebenso ist die gewählte Struktur im ESM-Modell eher geeignet eine grafische WYSIWYG-Darstellung der Dialog-Oberfläche zu erzielen, da im Zentrum eines ESM die Eingabe- und Ausgabestellen stehen, an denen Ereignisse passieren, sowie die dynamischen Relationen zwischen den Stellen.
Hit-Bausteine ermöglichen ebenfalls eine grafische Darstellung. Allerdings ist hier mehr eine hierarchische, abstrakt logische Darstellung der Implementierungseinheiten eines Dialogsystems gemeint.
In diesem Sinne wird in der Hitspezifikation auch streng zwischen logischer Spezifikation und der Umsetzung in ein konkretes Layout unterschieden. Die Anbindung des Layouts geschieht dabei über Gleichungsregeln, die den logischen Hitbausteinen zusätzlich zugeordnet werden können. Diese Gleichungsregeln beinhalten eine Beschreibung der Layoutstruktur der für den logischen Hitbaustein zu verwendenden Dialogobjekte. Dies sind Objekte eines virtuellen Baukastensystems, das dann auf ein jeweils konkretes Baukastensystem (Motif, Java,…) umgesetzt wird.
In den grafischen Darstellungen des ESM, den Ereignisstellen.Diagrammen (ESD's), ist die konkrete Darstellung von Widgets anstelle abstrakter Ereignisstellen vorgesehen. Ansonsten geschieht die Zuordnung konkreter Widgets zu Ereignisstellen über entsprechende Annotation von Abbildungsinformation, die z.B. die zu verwendende Widgetklasse, zu verwendende Widgetattribute und weitere für die Umsetzung jeweils nötige Information beinhaltet.

9.6 Sketchingsysteme

Die in den letzten Jahren entwickelten Sketchingsysteme, z.B. SILK oder DENIM [LANDAY00] haben zum Ziel, den Designer bereits in einer frühen Entwurfsphase bei der Entwicklung von Dialogsystemen zu unterstützen. Sie nutzen eine visuelle Sprache, in der der Designer oder Entwickler mehr oder weniger naiv den Verlauf des Dialogs skizziert. Das System soll, dabei ist der Anspruch von System zu System graduell unterschiedlich, aus teilweise handgefertigten Skizzen nach Möglichkeit automatisch die statischen Dialogelemente, die Aktionen des Benutzers und die daraus resultierenden Aktionen des Systems erkennen und in eine Implementierung umsetzen. Der Designer malt beispielsweise auf einem so genannten Storyboard die Dialogoberfläche vor und nach einer Aktion des Benutzers auf. Das System stellt die Unterschiede zwischen beiden Darstellungen fest und leitet daraus die notwendigen Reaktionen des Systems auf eine Benutzeraktion ab (z.B. System Anecdote [HARADA96]). Die jüngsten Arbeiten stellen u.a. eine Fortsetzung der Arbeiten dar, die mit den Stichworten „Programming by Example“ (System EAGER [CYPHER91]) und „Programming by Demonstration“ (z.B. System Pursuit [MODUG93]) umschrieben werden können.

Das System DENIM ist ein Beispiel eines Sketchingsystems zum Entwurf von Webseiten mit dem Kernmodell von Seiten und Übergängen, die durch Pfeile dargestellt werden.
In der Arbeit von Lin, Thomson und Landay [LINTH02] werden am Beispiel DENIM einige Nachteile heutiger Sketchingsysteme diskutiert, wie die Explosion von Übergängen bei Darstellung aller möglichen WYSIWYG-Kombinationen und redundanter Darstellung immer wiederkehrender Seitenelemente. Um diesen Nachteilen zu begegnen werden in DENIM zusätzliche Sprachelemente eingeführt, nämlich so genannte Components, Conditionals und Enhanced Arrows.

Components in DENIM
Components in DENIM sind (vermutlich in Anlehnung an den gängigen Component-Begriff) Dialogelemente, die der Designer einmalig in Storyboarding-Technik definiert (Custom Components), z.B. eine Check Box mit zwei Zuständen „On“ und „Off“, und die er in anderen Dialogelementen (Internetseiten) wieder verwenden kann. Daneben gibt es entsprechend so genannte Intrinsic Components, die bereits im System vordefiniert vorliegen.

Conditionals in DENIM
Conditionals ermöglichen in einem benutzerfreundlichen Dialog die Definition von Bedingungen. Z.B. können Zustandskombinationen mehrerer Checkboxen auf einer Seite jeweils einer Bedingung in einem so genannten Condition Stack zugeordnet werden. In einem weiteren Dialog können die einzelnen Bedingungen im Stack aktiviert werden. Während eine Bedingung aktiv ist, können dann Aktionen festgelegt werden, die bei Gültigkeit der Bedingung ablaufen sollen. So kann z.B. durch Malen eines Pfeiles von einem Button zu einer Seite das Öffnen der Seite für den Fall definiert werden, dass der Benutzer auf den Button klickt, während die Bedingung erfüllt ist. Sukzessive kann so der Bedingungsstack abgearbeitet werden.

Enhanced Arrows
Den Pfeilen (Arrows), die normalerweise Übergänge von einer Seite zu einer anderen definieren, die per Mausklick ausgelöst werden, können bei Enhanced Arrows andere Ereignisse, wie z.B. das Ablaufen eines Timers, zugeordnet werden.

Vergleich mit ESD´s
Der in der vorliegenden Arbeit ausgeführte Bottom-up-Ansatz bietet ähnlich wie bei Sketchingsystemen die Möglichkeit, Dialogabläufe auf einer konkreten (naiven) Ebene darzustellen. In einem Ereignisstellen-Diagramm kann schrittweise Ereignis für Ereignis an jeweils eigenen Ereignisstellen hintereinander festgehalten werden. Danach erfolgt eine Zuordnung zu konkreten Widgets und damit eine Zusammenführung der Ereignisstellen auf ein einziges Dialogelement und die damit verbundene Programmlogik. Insofern bieten Ereigisstellen die Voraussetzung für die Anwendung der Storyboarding-Technik. Dies wird durch die Darstellung in Modellsichten weiter unterstützt.
Darüberhinaus bieten ESD´s allerdings die Möglichkeiten einer komprimierten, intensivierten, d.h. redundanzfreien Darstellung auch komplexer Anwendungen mithilfe der zusätzlich vorhandenen Sprachmittel.

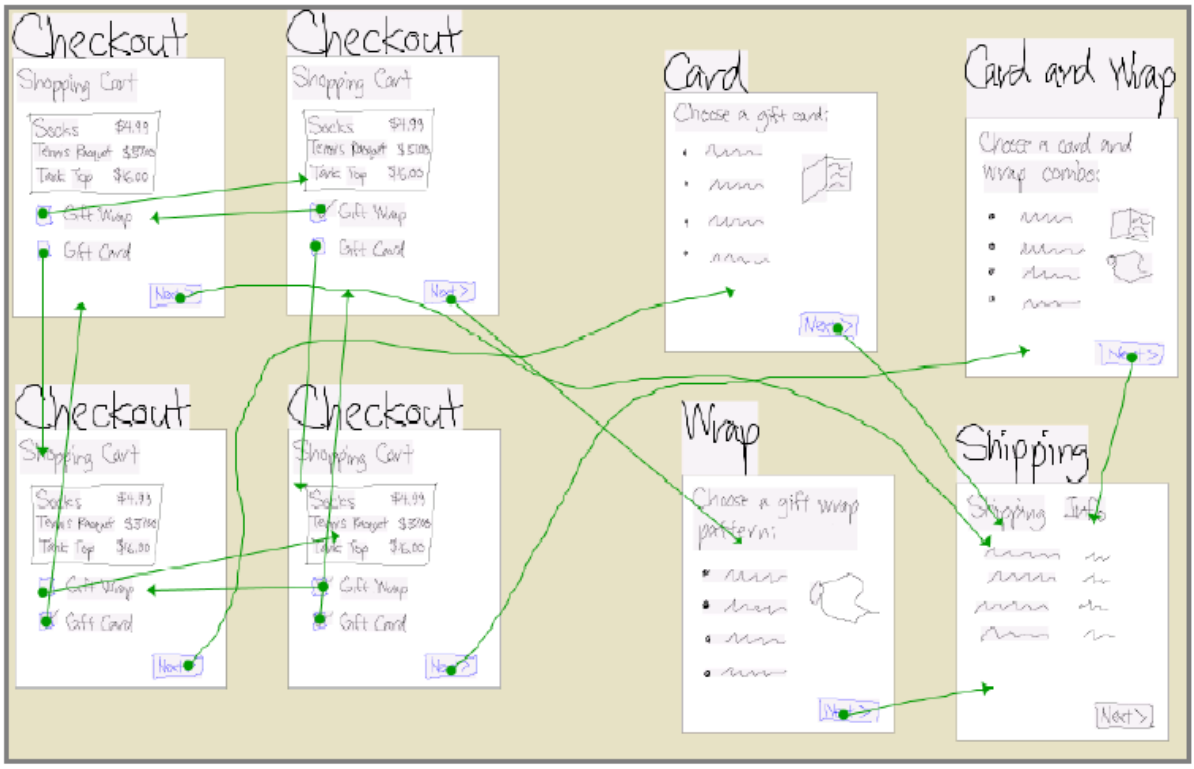


Abbildung 9-8 Storyboarding-Beispiel: Entwurf eines Shoppingsystems (aus [LINTH02])

Gezeigt wird ein System-Ausschnitt mit den Webseiten „Checkout“, „Card“, „Wrap“, „Card and Wrap“ und „Shipping“ im System DENIM. Checkout ist in 4 Zuständen zu sehen, die sich durch die 4 Kombinationsmöglichkeiten der Eingabe in den Checkboxen „Gift Wrap“ und „Gift Card“ ergeben. Je nach Zustand wird bei Mausklick auf „Next“ die entsprechende Folgeseite geöffnet. Die Pfeile repräsentieren den Übergang von einer Seite zur anderen oder zwischen „Momentaufnahmen“ derselben Seite nach Mausklick auf das jeweilige Dialogelement. Ist z.B. weder GiftCard noch GiftWrap angekreuzt, erfolgt bei Mausklick auf „Next“ das Öffnen der Shipping-Seite zur Angabe der Versandbedingungen.

Beispiel Shopping-System
Abbildung 9-9 (auf folgender Seite) zeigt, wie mithilfe eines Ereignisstellendiagramms der Entwurf in Art der in Abbildung 9-8 dargestellten Storyboarding-Technik durchgeführt werden kann. Dabei repräsentieren die vier Stellen des Diagramms mit Bezeichner „Checkout“ vier Zustände der gleichnamigen Webseite. (Wir dürfen dabei annehmen, dass eine geeignete Implementierung für das Ereignisstellendiagramm existiert, die eine umfassende Dialogstelle des ESD in eine Webseite umsetzt.) Wie die grünen Pfeile in

Abbildung 9-8 definieren die grünen Pfeile im ESD das Öffnen der jeweiligen Zielstellen aufgrund eines Ereignisses an der Ausgangsstelle.
Wenn wir annehmen, dass bei der Wahl von Checkboxen oder Buttons als konkrete Stellenrealisierung standardmäßig ein Mausklick das zugehörige auslösende Ereignis darstellt, stimmt auch das auslösende Ereignis bei Interpretation des ESD mit der Standard-Interpretation des Pfeils in DENIM überein. Im vorliegenden Prototyp ist eine derartige Standardabbildung vorhanden (siehe dazu Kap. 10).

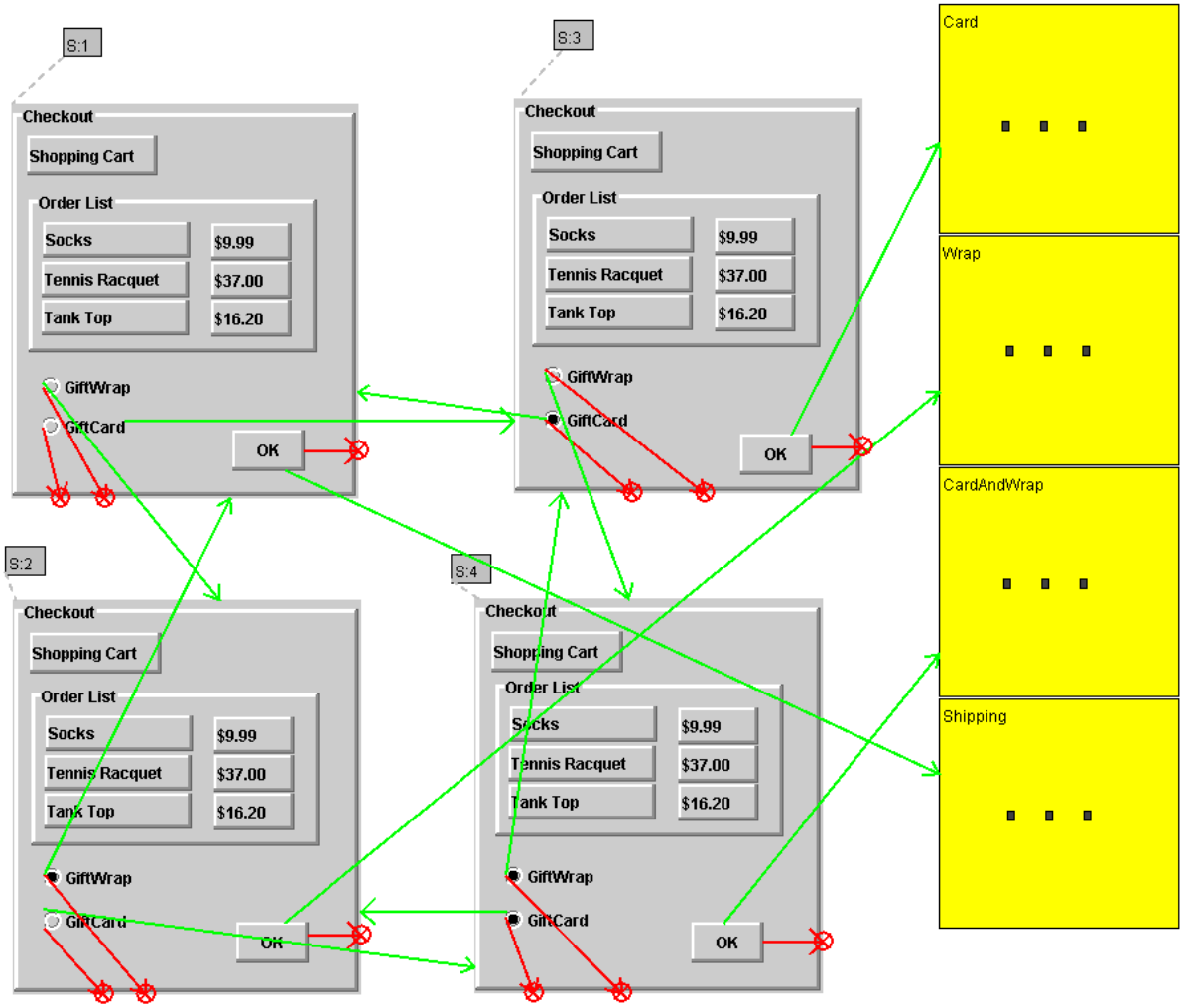


Abbildung 9-9 Ereignisstellendiagramm für Zustände der Checkout-Seite im Shoppingsystem

Im ESD in Abbildung 9-9 sind gemäß der formalen Definition des Ereignisstellen-Modells Pfeile gezeichnet, die das Schließen der jeweiligen Stellen bestimmt. Es wird eine einen Zustand der Seite Checkout repräsentierende Stelle geschlossen, wenn eine andere Zustandsstelle geöffnet wird. Ebenso wird die Stelle geschlossen, wenn ein Übergang auf eine Folgeseite (Card, Wrap, CardAndWrap, Shipping) bei Mausklick auf den OK-Button erfolgt.
Wenn wir nun annehmen, dass die Implementierung der vier Zustandsstellen der Seite Checkout über dieselbe Objekt-Instanz erfolgt, muss in der Implementierung sichergestellt sein, dass z.B. das Schließen einer Zustandsstelle nicht das Öffnen einer anderen Zustandsstelle ungewollt beeinträchtigt. Wir können die Implementierung nicht einfach von vorneherein so wählen, dass das Öffnen einer Stelle etwa das Erzeugen einer Instanz und das Schließen das Löschen der Instanz bedeutet, wenn nicht sichergestellt ist, dass prinzipiell bei Abarbeitung eines Ereignisses das Schließen immer vor dem Öffnen durchgeführt wird. Bei

Abbildung zweier Stellen auf dieselbe Instanz könnte das andernfalls zu dem ungewollten Effekt führen, dass die Instanz geöffnet und sofort wieder geschlossen wird.

Im Prototyp ist eine Annotation vorgesehen, die es erlaubt Zustandsstellen mit S:<Zustandsbezeichnung>, also z.B. S:1, S:2, … , zu kennzeichnen. Stellen werden genau dann als Zustand einer einzigen Widgetinstanz implementiert, wenn die Stellen den gleichen Bezeichner besitzen, dieselbe übergeordnete Stelle (auch die leere Stelle) und eine geeignete Zustandsannotation. Bei derartiger Darstellung von Zustandsstellen kann dann auf die Schließepfeile beim Übergang zu einer Zustandsstelle verzichtet werden. In Abbildung 9-9 sind redundant zu den Schließepfeilen diese Zustandsbezeichner verwendet.

Wie in der vorliegenden Arbeit bereits hinreichend erwähnt und wie auch durch das obige Beispiel und die Ausführungen von Lin, Thomsen und Landay belegt, ist die Darstellung der Dynamik in Form einfacher Übergänge von Bild zu Bild bzw. von Zustand zu Zustand in vielen Fällen zu aufwendig und unübersichtlich, weshalb für Sketchingsysteme nun nach Mitteln zu einer intensiveren Darstellung gesucht wird.
Abbildung 9-10 zeigt nun in Fortführung des Beispiels eine alternative intensivere Darstellung in einem Ereignisstellendiagramm.

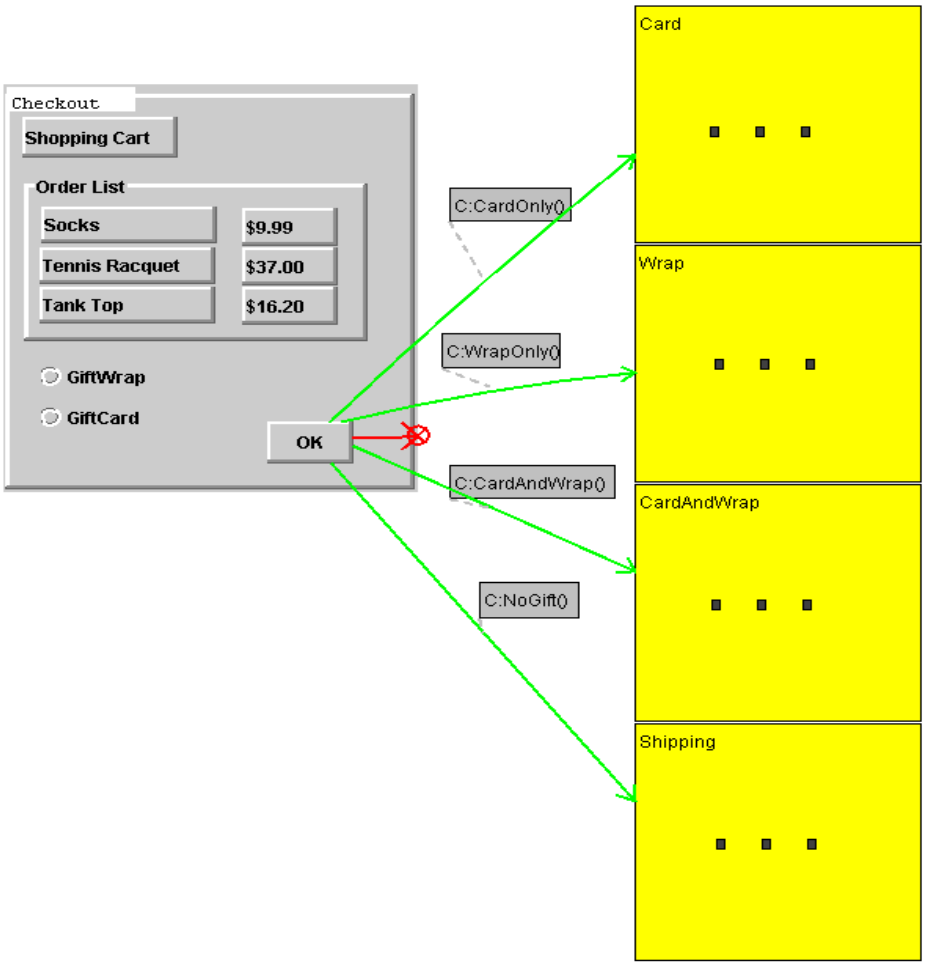


Abbildung 9-10 Ereignisstellendiagramm mit intensivierter Darstellung der Checkout-Seite im Shoppingsystem

Anstelle der extensiven, expliziten Darstellung der Zustände, die durch die Mausklicks auf die Buttons zur Auswahl von Geschenkverpackung und Begleitkarte erreicht werden können, wird jetzt nur eine Stelle gezeigt und die Übergänge mittels geeigneter Conditions gesteuert.

10 Realisierung im Prototyp

Begleitend zu den in den vorangegangenen Kapiteln gemachten Ausführungen wurde ein Prototyp in Form eines Programmsystems erstellt, der in diesem Kapitel in seinen wesentlichen Merkmalen beschrieben wird.

10.0 Zielsetzung und Rahmenbedingungen

Der Prototyp ist die beispielhafte Implementierung eines Werkzeuges für die Modellierung von Dialogsystemen mit Hilfe des Ereignisstellenmodells und der Darstellung in Ereignisstellendiagrammen. Er umfasst im Wesentlichen eine Komponente zum Editieren von Ereignisstellendiagrammen (Modelleditor) und eine damit gekoppelte Komponente zur Generierung des aus einem Modell resultierenden Programmcodes (Codegenerator).
Die wichtigsten Ziele des Prototyps sind und waren:

- der Nachweis des praktischen Einsatzes von ESD´s in der Modellierung bis hin zur automatischen Codegenerierung. Der Prototyp erlaubt damit die Darstellung aller wesentlichen Sprachelemente von Ereignisstellendiagrammen und ihre automatische Umsetzung in ablauffähigen Programmcode. Um den Nachweis für die Realisierung auch relativ komplexer realistischer Anwendungen zu erbringen, wurde und wird der Prototyp im konkreten Projektumfeld des Autors eingesetzt.
- die Gewinnung von Erfahrung im konkreten praktischen Einsatz hinsichtlich der Gestaltung und Erfordernis der Sprachelemente des Modells. Insbesondere sollten damit auch gegebenenfalls Widersprüche und etwaige Defizite in der Modellsprache aufgedeckt werden. Die Prioritäten im praktischen Einsatz bezüglich erforderlicher Leistungsmerkmale an Modellierungssysteme sollten in Ergänzung zu bereits vorliegenden Erfahrungen erarbeitet bzw. überprüft werden. Experimentell sollten verschiedene Varianten der Umsetzung des Modells verifiziert werden.

10.0.0 Kernfunktionalität für Integration in eine Entwicklungsumgebung

Ziel des Prototyps war nicht die Realisierung einer vollständigen integrierten Entwicklungsumgebung, eines Integrated Development Environments (IDE), für die Realisierung kompletter Anwendungen mit den damit verbundenen Komponenten zur Projektverwaltung, zum Debugging, zum Configuration Management, Deployment, etc. Der Prototyp realisiert vielmehr die Kernfunktionalität zur Modellierung und Realisierung des Dialogs mit Ereignisstellendiagrammen. Zur Anwendung innerhalb eines IDE ist die Integration dieser Funktionalität innerhalb entsprechender Komponenten des IDE erforderlich. Ebenso wurde im Prototyp auf eine Reihe von Funktionalität verzichtet, wie sie in einem kommerziellen System als nützlicher Bestandteil angeboten wird, Funktionalität, die aber für den Nachweis der Modellierung und die Realisierung nicht unbedingt erforderlich ist. Z.B. ist auf eine komfortable Möglichkeit zur Edition sämtlicher Widgetattribute in Form eines Attributeditors oder die Integration eines speziellen Editors für die Definition geometrischer Layout-Constraints verzichtet. Es darf jedoch angenommen werden, dass eine Ergänzung um derartige heute bereits in marktüblichen Systemen angebotenen Funktionen nicht im Widerspruch zu den im Prototyp realisierten Konzepten und deren Implementierung steht.

10.0.1 Realisierung in Java für Swing-Klassen

Die Realisierung des Prototyp selbst erfolgte in seinen beiden Komponenten Modelleditor und Codegenerator in Java. Der vom Codegenerator erzeugte Code ist ebenfalls Java-Code.

Unterstützt werden in einer exemplarischen Auswahl die wichtigsten Klassen zur Gestaltung von Dialogoberflächen aus Java-Swing. Diese werden im Modelleditor neben den abstrakten Ereignisstellen als konkrete Widgets zur Modellierung angeboten und dargestellt. Der durch den Codegenerator erzeugte Code verwendet die im Modell gewählten Swingklassen zur Realisierung der modellierten Anwendung.

10.0.2 Zusätzliche Adapter- und Komfortklassen

Alternativ und in Ergänzung zur direkten Umsetzung von Ereignisstellen in Instanzen von Java-Swingklassen wird im Modelleditor eine Reihe von Klassen angeboten, die zum einen als Zwischenschicht im Sinne eines virtuellen Baukastensystems für die leichtere Abbildung auf auch andere konkrete Toolkits dienen können und zum anderen Komfortklassen darstellen, die in Java-Swing nicht enthaltene Funktionalität bieten, z.B.mit Labeln versehene Textfelder, Widgets mit automatischer Plausibilisierung von Eingabewerten, etc. Außerdem werden diese Adapterklassen für den Anschluß der Applikationslogik z.B. bei der Realisierung der Variation verwendet (siehe unten).
Die zusätzlichen Klassen stehen dem Anwender mit Sourcecode zur Verfügung.

10.1 Editor zur Modellierung von Ereignisstellendiagrammen

Die in der Arbeit in den Abbildungen dargestellten Beispiele zur Modellierung mit Ereignisstellendiagrammen wurden fast ausnahmslos mit dem Modelleditor des Prototyps erstellt.
Die Diagrammelemente, wie z.B. die Rechtecke der abstrakten Ereignisstellen, ihre Annotationen, die Darstellung der Pfeile zum Öffnen, Schließen, zur Wertübertragung etc. werden in einfacher direkter Manipulation mittels Maus kreiert und bearbeitet. In gleicher Weise erfolgt die Erzeugung konkreter Widgets alternativ zu abstrakten Ereignisstellen sowie die nachträgliche Zuordnung konkreter Widgets zu abstrakten Stellen mittels einfacher Selektionsmechanismen und z.B. direkter Zuordnung mittels Mausklick.
Zusätzlich zur Manipulation mit der Maus wurden einige wichtige Funktionen, wie z.B. das Kopieren von Modellelementen als sogenannte Shortcuts auf bestimmte Funktionstasten der Tastatur gelegt.

Edition abstrakter Ereignisstellen
Der Benutzer hat die Freiheit, zunächst auf Ebene der abstrakten Ereignisstellen zu modellieren, ohne sich dabei auf bestimmte konkrete Dialogobjekte, etwa die Klassen des zu Grunde liegenden Toolkits, festlegen zu müssen. Er legt damit die hierarchische Bestandteilsstruktur der Ereignisstellen und ihre dynamischen Bezüge fest. Über entsprechende Annotationen können den Stellen Name, Typ, Wertebereich, Defaultwert, etc. entsprechend Ausführungen in Abschnitt.5.2 Grafische Darstellung zugeordnet werden.

Edition auf Ebene konkreter Widgets
Alternativ hat der Benutzer die Möglichkeit, statt abstrakter Ereignisstellen mit konkreten Widgets zu arbeiten, die dann weitgehend in WYSIWYG-Qualität, das heißt in der Erscheinungsform zur Laufzeit, dargestellt werden (siehe dazu auch Ausführungen in Kap.6 Abstraktion vom konkreten Widget). Wie die abstrakten Ereignisstellen können die Widgets mit denselben dynamischen Relationen in Bezug gesetzt und mit entsprechenden Annotationen versehen werden.

Mischung abstrakter und konkreter Elemente
Der Benutzer hat die Wahl, sich stellenspezifisch zwischen einer abstrakten Darstellung einer Stelle und der Darstellung eines konkreten Widgets zu entscheiden. Er hat außerdem die Wahl, in einem Modus mit ausschließlich abstrakter Darstellung aller Ereignisstellen zu arbeiten.

Robustheit bei Änderungen
Besonderes Augenmerk bei der Implementierung des Editors wurde auf die leichte Änderbarkeit von Modellelementen gelegt mit dem Ziel, bei Änderungen möglichst viel an bereits vorhandenen Informationen über die Änderung hinaus beizubehalten. So ist es z.B. jederzeit möglich, die Hierarchie der Ereignisstellen durch wenige Mausklicks zu ändern, z.B. den Parent zu wechseln oder zusätzliche Hierarchieebenen an beliebiger Stelle in der Hierarchie einzuziehen, ohne dabei Elemente löschen zu müssen oder bereits mitgegebene andere Information (Widgetklasse, Bezüge zu anderen Objekten, etc.) der Elemente zu verlieren. Ebenso ist es möglich, beispielsweise zugeordenete Widgetklassen ohne Verlust der Bezüge zu anderen abstrakten Ereignisstellen oder konkreten Widgets zu ändern.

Realisierung von Modellsichten
Das in Abschnitt 5.3.2 eingeführte Konzept der Modellsichten ist im Editor in der Form umgesetzt, dass der Benutzer zwischen verschiedenen Sichten auf das Modell wählen kann, in denen er jeweils beliebige Ereignisstellen ausblenden kann. Mit dem Ausblenden der Stelle werden automatisch die Bezüge zu oder von der Stelle mit ausgeblendet. Wird eine Stelle unsichtbar, bleiben zunächst ihre Teilstellen sichtbar. Umgekehrt können wahlweise beliebige Teilstellen aus der Sicht entfernt werden. Damit wird es möglich, in einer Sicht beispielsweise die Details einer Stelle unter Weglassung ihrer umfassenden Stellen zu bearbeiten, in einer anderen eine Grobsicht des Modells z.B. auf Ebene der Fenster bzw. der für die Dynamik der Fenster ausschlaggebenden Stellen ohne Darstellung der übrigen Details innerhalb der Fenster.

10.2 Schritte zur Codegenerierung

10.2.0 Abbildung abstrakter Ereignisse auf Widgets, Widgetattribute und Attributwerte

Wie in Kapitel 6.2 am Beispiel des Radio-Button-Modells ausgeführt, erfolgt die Realisierung eines Ereignisses einer abstrakten Ereignisstelle ohne Beschränkung der Allgemeinheit etwa im Falle der Umsetzung auf eine Java-Anwendung durch die Abbildung auf eine Widgetinstanz einer bestimmten Klasse und letztlich dort auf die Realisierung eines dieser Instanz zugeordneten Ereignisses. Um das Ereignis über einen bestimmten Zeitraum ablesen zu können, wird es in Form eines Wertes eines zugeordneten Widgetattributes gespeichert. In bestimmten Fällen handelt es sich dabei nicht um Widgetattribute im strengen Sinn, sondern um allgemein gesprochen, bestimmte der Widgetinstanz zugeordnete Ressourcen (Beispiel List-Widget mit List-Items). Ebenso kann das Ereignis auch in manchen Fällen einer bestimmten Ressource (z.B. Selektion eines List-Items) zugeordnet werden.
Um also gemäß unserem Modell ein Ereignis zu realisieren, führen wir zunächst diese Abbildung auf Widgetinstanz, Widgetklasse, Widgetereignis (Ressourceereignis), Widgetattribut (Widgetressource, in Java: Property) und Attributwert (Ressourcewert) durch. Basierend auf dieser Information kann der Codegenerator dann den notwendigen Code erzeugen, um diese Zuordnung zum richtigen Zeitpunkt im Programmablauf zu realisieren.

Gemäß Modell benötigen wir im Wesentlichen Code, der zum einen ein Ereignis (oder mehrere) an einem bestimmten Widget ermöglicht, der also das Widget samt zugedachtem Ereignis verfügbar macht und die Darstellung des Wertes für das bestimmte Widgetattribut bewerkstelligt. Zum anderen benötigen wir Code, der umgekehrt die Darstellung eines Ereignisses (oder auch mehrerer Ereignisse) verhindert und letztlich die dem Ereignis zugedachten Ressourcen freigibt. Konkret bedeutet dies, dass ein Widget bzw. eine Widgetressource, für die Darstellung eines Ereignisses gesperrt, geschlossen oder gänzlich gelöscht wird. In Java bedeutet letzteres die explizite oder implizite Freigabe zur Garbage Collection.

10.2.0.0 Defaultmechanismus zur Abbildung auf Widgetressourcen

Annotation von	**Primitive Stelle**				**Zusammengesetzte Stelle/Topstelle**
Typ ***Wert***	*Ohne Typangabe*	*I*	*O*	*I/O*	*-, I, O, I/O*
Kein Wert annotiert	JPushButton Text = Name MouseReleased	JPushButton Text=Name MouseReleased	JLabel Text=Name “ValueChanged”	JPushButton Text=Name MouseReleased	JPanel/FramedJPanel Title = Name Mouse Pressed
Wert annotiert	JPushButton Text = Wert MouseReleased	JPushButton Text = Wert MouseReleased	JLabel Text = Wert “ValueChanged”	JPushButton Text = Wert MouseReleased	JPanel/FramedJPanel Title = Wert Mouse Pressed
Werteliste	JList Item pro Wert MouseReleased	JList Item pro Wert MouseReleased	JList Item pro Wert “ItemChanged”	JList Item pro Wert MouseReleased	JPanel/FramedJPanel Title = Name Mouse Pressed
Werteliste +Defaultwert	JList Item pro Wert setSelected = “Default-Wert” MouseReleased	JList Item pro Wert setSelected = “Default-Wert” MouseReleased	JList Item pro Wert setSelected = “Default-Wert” “ItemChanged”	JList Item pro Wert setSelected = “Default-Wert” MouseReleased	JPanel/FramedJPanel Title = Name Mouse Pressed
Wertebereich	JTextField Text = “” KeyTyped	JTextField Text = “” KeyTyped	JTextField Text = “” “ValueChanged”	JTextField Text = “” KeyTyped	JPanel/FramedJPanel Title = Name Mouse Pressed

Abbildung 10-1 Defaultabbildung abstrakter Ereignisstellen mit Triggerfunktion auf Java-Swing-Klassen oder abgeleitete Komfortklassen

Pro vorliegender Typ- und Wertannotation an einer primitiven oder zusammengesetzten Stelle werden jeweils Widgetklasse, Belegung des konkreten „Value“-Attributs (z.B. Text) und konkretes Triggerereignis für die Auslösung von Ereignissen an anderen Stellen angegeben.

Eine häufig gegebene Situation des Entwurfs in der frühen Phase ist die, dass die Designer der Dialogoberfläche eines Systems sich noch nicht auf konkrete Widgetklassen festlegen wollen. In vielen Fällen möchte man zunächst lediglich die Struktur des Dialogs in seinen Dialogeinheiten festhalten, ohne auch noch konkrete Wertebereiche an einer Dialogstelle zu bestimmen. Manchmal ist auch selbst der Ereignistyp (Eingabe, Ausgabe oder Eingabe und Ausgabe) noch nicht klar.

Nicht zuletzt für dieses Anwendungsszenario sieht der Prototyp die automatische Abbildung abstrakter Ereignisstellen auf konkrete Widgets und Widgetattribute vor. Der Codegenerator erzeugt aus dieser schwach bestückten Spezifikation bereits automatisch ein ablauffähiges Programm, das zumindest zum Zweck des Tests der Dialogstruktur genutzt werden kann.

In den Tabellen in Abbildung 10-1 und Abbildung 10-2 wird die defaultmäßige Abbildung der abstrakten Ereignisstellen auf konkrete Widgets für die Fälle gezeigt, in denen keinerlei Annotation an der Stelle vom Designer gemacht wurde, bis hin zu den Fällen, in denen bereits Angaben zu den Stellenwerten und zu Ereignistypen durchgeführt wurden. Der Codegenerator unterscheidet für die Wahl des geeigneten Widgets, ob von der Stelle aus ein Ereignis an einer anderen Stelle getriggert wird, ob also ein Pfeil zum Öffnen, Schließen oder zum Auslösen einer Wertübertragung von der Stelle ausgeht (Abb. 10-1) oder keine Triggerfunktion der Stelle vorliegt (Abb. 10-2).

	Stelle ohne Trigger-Pfeil				
Annotation von	**Primitive Stelle**				**Zusammengesetzte Stelle/Topstelle**
Typ ***Wert***	*Ohne Typangabe*	*I*	*O*	*I/O*	*-, I, O, I/O*
Ohne Wert-annotation	LabeledTextfield LabelText=Name	LabeledText LabelText=Name	JLabel LabelText=Name	LabeledText LabelText=Name	JPanel/FramedJPanel Title = Name
Wert annotiert	LabeledTextfield LabelText=Name Text = Wert	LabeledTextfield LabelText=Name Text = Wert	LabeledTextfield LabelText=Name Text = Wert	LabeledTextfield LabelText=Name Text = Wert	JPanel/FramedJPanel Title = Wert
Werteliste	Liste mitCheckButtons	Liste mitCheckButtons	Liste mitCheckButtons	Liste mitCheckButtons	JPanel/FramedJPanel Title = Name
Werteliste +Defaultwert	Liste mitCheckButtons DefaultWert gesetzt	Liste mitCheckButtons DefaultWert gesetzt	Liste mitCheckButtons DefaultWert gesetzt	Liste mitCheckButtons DefaultWert gesetzt	JPanel/FramedJPanel Title = Name
Wertebereich	LabeledTextfield LabelText=Name Text = “”	LabeledTextfield LabelText=Name Text = “”	LabeledTextfield LabelText=Name Text = “”	LabeledTextfield LabelText=Name Text = “”	JPanel/FramedJPanel Title = Name

Abbildung 10-2 Defaultabbildung abstrakter Ereignisstellen ohne Triggerfunktion auf Java-Swing-Klassen oder abgeleitete Komfortklassen

10.2.0.1 Individuelle Abbildung auf Widgetklassen und Attribute

Alternativ zur automatischen Abbildung besteht die Möglicheit, den abstrakten Ereignisstellen explizit konkrete Widgetklassen, Widgetattribute und Attributwerte zuzuordnen oder von vorneherein anstelle einer abstrakten Dialogstelle ein konkretes Widget zu verwenden. Im derzeitigen Prototyp sind dabei die vom System in einem Auswahldialog angebotenen Klassen und Attribute, wie oben bereits erwähnt, auf die wichtigsten Java-Swing-Klassen und einige Adapter- und Komfortklassen beschränkt. Allerdings kann der Benutzer selbst durch Annotation des Klassenbezeichners prinzipiell eine beliebige Klasse zur Realisierung einer Ereignisstelle zuordnen. Voraussetzung ist allerdings, dass die Klasse eine vom Codegenerator geforderte Schnittstelle besitzt, die das Erzeugen einer Instanz, die

Einordnung in die Widgethierarchie, das Öffnen und Schließen sowie das Setzen und Abfragen von Werten ermöglicht.
Damit ist im Prototyp grundsätzlich die Anbindung von Klassen möglich, die über die Funktionalität von reinen Oberflächenelementen hinausgeht. Z.B. könnte im Konstruktor einer Instanz nicht nur das Erzeugen eines Widgets für die Darstellung an der Oberfläche sondern auch der Zugriff auf eine Datenbank implementiert sein.

10.2.1 Zuordnung von Widgetinstanzen

Wie bereits in Kapitel 6 erwähnt, ist es möglich, mehreren Ereignisstellen des Modelldiagramms dieselbe Widgetinstanz zuzuordnen. Dies ist im Prototyp über die Stellenbezeichner gesteuert. Haben Ereignisstellen denselben Bezeichner und sind die Stellen hierarchisch gleich eingeordnet, d.h. sind sie Teilstelle derselben übergeordneten Stelle, sorgt der Codegenerator für die Erzeugung einer einzigen Instanz für diese Stellen. In allen anderen Fällen wird jede Stelle eindeutig auf eine eigene Instanz abgebildet. (Über das Sprachmittel der Exemplarvariation (siehe Kap.7 und unten) können außerdem einer Stelle mehrere Instanzen zugedacht werden.)
Im Folgenden werden zwei verschiedene Arten der Interpretation vorgestellt, wie sie auch im Prototyp realisiert werden. Die eine Art zielt auf Anwendungsfälle, in denen z.B. im ersten Entwurf ein Dialog an mehreren Stellen in seinen Eigenschaften unvollständig im Sinne der Darstellung verschiedener Szenarios oder Teilaspekte modelliert wird, die in einer Widgetinstanz gleichsam verschmolzen werden. Wir können dies als „Merge von Szenarien“ bezeichnen. Die andere Art der Interpretation ist die, dass wir mit den verschiedenen Stellen, die auf eine Instanz abgebildet werden, verschiedene Zustände der Instanz meinen, die prinzipiell voneinander zu trennende Aussagen beschreiben.

10.2.1.0 Merge von Szenarienstellen

Die Vereinigung mehrerer Stellen im Sinne von beispielhaften Szenarien setzt voraus, dass die Aussagen an beiden Ereignisstellen nicht widersprüchlich sind bzw. im Konfliktfall eine Auflösung des Konflikts durch den Codegenerator vorgenommen werden kann. So macht es z.B. keinen Sinn, die Stellen mit unterschiedlichen Widgetklassen auszustatten. In diesem Fall ist undefiniert, welche der Klassen vom Codegenerator für die Instanz gewählt wird.
Die generelle Strategie ist, dass alle Aussagen, die an den einzelnen Stellen getroffen sind, in der Instanz übernommen werden, wenn diese nicht widersprüchlich sind. Allerdings zeigt sich, dass es mehrere Möglichkeiten der Zusammenfassung der Aussagen mehrerer Stellen zu widerspruchsfreien Aussagen gibt.
Wie soll z.B. die Zusammenführung erfolgen, wenn eine Stelle als Ausgabestelle für einen Wert „A“, eine andere gleichnamige Stelle als Eingabestelle für Wert „B“ definiert ist? Hier haben wir die Möglichkeit, durch die Zusammenfassung eine Stelle für sowohl Eingabe als auch Ausgabe beider Werte „A“ und „B“ zu bilden, oder aber eine Stelle für Ausgabe von „A“ und nur von „A“ und Eingabe von „B“ und nur von „B“.
Stellen wir uns allerdings auf den Standpunkt, nach dem in Kap.5 vorgestellten Modell vorzugehen, so haben wir zwei Ereignismengen {(s1, O, “A“)} und {(s2, I, “B“)} zu implementieren, was der letzteren Variante entspricht.
Betrachten wir nun die Zusammenführung der Übergänge zu und von den Stellen, die auf eine Instanz abgebildet werden sollen. Auch hier gibt es mehrere denkbar sinnvolle Arten der Vereinigung. Sollen z.B. an der Instanz nur die Ereignisse, die ihren Ursprung aus Stelle s1 haben, eröffnet werden oder alle Ereignisse, wenn laut Modelldiagramm ein Öffnepfeil in Stelle s1 endet? Nach strenger Interpretation des Modells ist hier wiederum die erste Variante vorzuziehen.
Ebenso ergibt sich die Frage für die Zusammenführung auslösender Ereignisse: Sollen z.B. alle Ereignisse der Instanz das Öffnen von Ereignissen an einer anderen Instanz bzw. Stelle

auslösen, wenn nach Modelldiagramm ein Öffnepfeil von s1 ausgeht, der keine explizite Annotation auslösender Ereignisse besitzt? Nach strenger Modellinterpretation gilt, dass dann die gesamte Ereignismenge der Stelle s1 die Menge der auslösenden Ereignisse darstellt, nicht aber die gesamte Menge der Ereignisse, die der Instanz durch Zusammenfassung von s1 und s2 zugeordnet ist.
Als widersprüchlich erweist sich im Normalfall die Geometrie, zumindest die Position der Stellen mit gleichem Bezeichner im Diagramm. Wird also die Position der Instanz zur Laufzeit aus der Position der Stellen im Diagramm abgeleitet, muss hier letztlich eine Entscheidung für eine der Stellen getroffen werden. Die Entscheidung ist allerdings irrelevant für die Codegenerierung, wenn die Geometrie mittels Layoutattributen des Widgets dynamisch zur Laufzeit entschieden wird oder die Layoutgestaltung generell in einem anderen Editor durchgeführt wird.
Die Vorgehensweise der Zusammenführung der Aussagen an den Stellen ist derzeit im Prototyp festgelegt und kann nicht beeinflusst werden. Hier wäre es ggf. sinnvoll, weitere Sprachelemente einzuführen, die die Art der Zusammenführung steuerbar machen.
Die Zusammenführung wird im Prototyp wie folgt implementiert: (O.B.d.A. beschreiben wir die Zusammenführung nur zweier Stellen in einer Instanz)

Zwei Diagrammstellen s1 und s2 mit gleichem Bezeichner und gleicher hierarchisch übergeordneter Stelle werden so interpretiert, als wäre es eine Stelle s, die alle Sprachelemente im Diagramm von den beiden zusammenzuführenden Stellen übernimmt.
Dies bedeutet insbesondere für:
- Eingehende Pfeile: in s enden alle Pfeile, die ursprünglich in s1 oder s2 enden. Die Annotationen der Pfeile bleiben erhalten.

- Ausgehende Pfeile: von s gehen alle Pfeile aus, die ursprünglich von s1 und s2 ausgehen. Die Annotationen der ausgehenden Pfeile bleiben ebenso erhalten.

- Annotationen der Stelle: Alle Annotationen werden von s1 und s2 nach s übernommen. Redundanzen, die durch identische Bezeichner erkannt werden, werden ggf. gestrichen.
Dies bedeutet insbesondere, dass alle einzelnen Annotationen zu Ereignistypen, zu einzelnen Werten, Wertelisten, Wertebereichen sowie Annotationen mit Typ-Wertpaaren von s1 und s2 nach s übernommen werden.
Damit wird erreicht, dass die Menge der Ereignisse, die s zuzuschreiben sind, in jedem Falle die Ereignismengen von s1 und s2 enthält, bei geeigneter Wahl der Annotationsform aber eine echt umfassende Menge der beiden Ausgangsmengen darstellt. War z.B. an s1 der Ereignistyp I annotiert und in einer gesonderten Annotation der Wertebereich <AnyText>, an s2 der Ereignistyp O und in einer gesonderten Annotation der Wertebereich <AnyNumber>, so ergibt sich als neue Ereignismenge die Menge der Ereignisse, die die Eingabe wie auch die Ausgabe von Werten aus „AnyText“ und „AnyNumber“ darstellt. Waren dagegen Annotationen als spezifische Paare (I, <AnyText>) und (O, <AnyNumber>) gegeben, bedeutet dies, dass an s die Eingabe von „AnyText“, jedoch nur die Ausgabe von „AnyNumber“ möglich ist.
Die sich ergebende Ereignismenge ist derart, dass keine Unterscheidung mehr bezüglich ihrer Herkunft von den ursprünglichen Stellen gemacht werden kann. D.h. Ereignisse beider Stellen, die sich in Wert und Typ nicht unterscheiden, werden als identisch betrachtet.

- Teilstellen: Alle Teilstellen von s1 und s2 werden als Teilstellen von s übernommen. Bei Namensgleichheit gelten wiederum die gerade beschriebenen Regeln der Zusammenführung.

10.2.1.1 Stellen als Zustände einer Instanz

Um die Interpretation der Stellen als Zustände (Zustandsstellen) einer Widgetinstanz zu interpretieren ist im Prototyp eine weitere Annotation eingeführt. Wir kennzeichnen eine Zustandsstelle mit der Annotation S:n wobei n einen Index für den entsprechenden Zustand darstellt. Alternativ dazu kann der Index in Klammern hinter dem Stellenbezeichner stehen..
Für die Interpretation der Zustandsstellen einer Widgetinstanz gelten im Sinne der Stellen unseres Ereignisstellenmodells in Kap.5 folgende Besonderheiten:

1. Wird eine Zustandsstelle geöffnet, werden alle übrigen Zustandsstellen derselben Instanz geschlossen.
2. Es muss sichergestellt sein, dass nicht zwei verschiedene Zustandsstellen derselben Instanz gleichzeitig geöffnet werden, da dies zu einem nicht definierten Verhalten führt.

Die Interpretation der Zustandsstellen entspricht sonst der Interpretation der Stellen des Ereignisstellenmodells in Kap.5 mit eben der Ausnahme, dass die Notwendigkeit entfällt, explizit die anderen Zustandsstellen derselben Instanz mittels Schließepfeil zu schließen, wenn eine Zustandsstelle geöffnet wird.

Im Gegensatz zum oben beschriebenen „Merge“ ist nun die Strategie bei der Codegenerierung, bei einem öffnenden Stellenübergang zu einer Zustandsstelle die Aussagen der übrigen Zustandsstellen der Instanz ungültig zu machen und dafür alle Aussagen der aktuellen Zustandsstelle zu aktivieren.
Dies bedeutet insbesondere, dass alle Widgets, die Ausgabestellen realisieren, geschlossen und Widgets, die ausschließlich Eingabeereignisse realisieren, insensitiv gesetzt werden, falls sie nicht als Startstellen der zu öffnenden Zustandsstelle erkannt werden.

Nachdem, wie oben beschrieben, festliegt, welcher Widgetklasse bzw. welchen einzelnen Widgetressourcen die Ereignisstellen zugeordnet sind und welche Instanzen erzeugt werden sollen, kann mithilfe dieser Information Code für die Realisierung der Ereignisstellen bzw. das Öffnen und Schließen der Stellen generiert werden.

10.2.2 Interpretation des formalen Öffnens und Schließens

Im Prototyp werden die im formalen Modell eingeführten Begriffe des Öffnens und Schließens von Ereignisstellen wie folgt umgesetzt:
Widgetinstanzen, die primitive Eingabestellen repräsentieren, d.h. Stellen, denen Eingabeereignisse zugeordnet sind, werden mit ihren Container-Widgets geöffnet, d.h. erzeugt und sichtbar gesetzt. Die Eingabe wird aber erst dann ermöglicht, wenn die Stelle nach Definition im formalen Modell zu öffnen ist. Ist die Stelle eine Startstelle, ist dies sofort mit dem Öffnen der übergeordneten Stelle, also o.B.d.A. dem Öffnen des Container-Widgets der Fall. Im anderen Fall muss erst ein entsprechendes Ereignis eintreten, das den Übergang zum Öffnen auslöst. Erst dann wird das Widget sensitiv gesetzt bzw. für die Eingabe aktiviert.
Bei reinen Ausgabestellen und allen nicht primitiven Stellen bedeutet das formale Öffnen einer Stelle auch genau das Öffnen der Widgetinstanz im üblichen Sprachgebrauch im Sinne von kreieren, falls dies nicht schon geschehen, und sichtbar machen.
Entsprechend wird im Prototyp das Schließen interpretiert:
Ist eine primitive Eingabestelle formal zu schließen, wird das Widget für die Eingabe gesperrt, bleibt aber sichtbar. Es wird erst mit dem Container unsichtbar.
In allen anderen Fällen ist wiederum das formale Schließen mit dem unsichtbar machen eines Widgets gleichzusetzen.

10.2.3 Codegenerierung für stellenspezifische Klassen

Dabei wird prinzipiell für jede Ereignisstelle eine entsprechende Klassendefinition codiert, wobei wiederum für Stellen gleichen Namens mit gleicher übergeordneter Stelle dieselbe Klassendefinition gilt, die dann entsprechend nur einmal erzeugt werden muss (siehe oben „Merge“ von Stellen und „Zustandsstellen“.
Die generierte Klasse erhält als Namen den Namen der Ereignisstelle, dem ein Prefix zur eindeutigen Unterscheidung von etwaigen generierten Instanznamen oder Klassen anderer Stellen, vorangestellt wird.
In die Klassendefinition werden im Wesentlichen Konstruktoren zur Erzeugung der Instanzen sowie Listener-Klassen zur Verarbeitung der Ereignisse an der Ereignisstelle eingefügt.

Code in Konstruktoren
Grundsätzlich beinhalten Konstruktoren den Code, der unmittelbar mit dem Kreieren einer Stelle verbunden werden kann. Z.B. kann ein Konstruktor Code zum Öffnen einer Teilstelle (Öffnen verstanden im Sinne des formalen Ereignisstellenmodells) nur beinhalten, wenn diese als Startstelle definiert ist. Im anderen Fall kann die Teilstelle allenfalls als Instanz erzeugt, aber noch nicht sichtbar oder sensitiv gesetzt werden.
Die Konstruktoren beinhalten gegebenenfalls
- Aufrufe zur Erzeugung der Instanzen der Teilstellen,
- das Setzen von Startwerten für das der Stelle zugeordnete Widgetattribut,
- entsprechenden Code zur Realisierung einer Widgetressource (z.B. ListItem)
- Code zum Sichtbarmachen der Teilstellen
- Code zum sensitiv Setzen der Teilstellen
- und die Aktivierung von Listenerinstanzen.

Listenerklassen zur Realisierung der Stellenübergänge
Grob gesehen implementiert in der Regel eine Listener-Klasse den Code, der einem Pfeil zum Öffnen, Schließen oder zur Wertübertragung (ggf. kombiniert mit Öffnepfeil) in einem Ereignisstellendiagramm entspricht.
Die Listener-Klassen beinhalten Methoden zur Reaktion auf bestimmte Ereignisse. In diesen Methoden können ähnliche Codesequenzen generiert sein, wie dies in Konstruktoren der Fall ist. D.h. es werden dort ebenfalls z.B. Instanzen anderer Stellen erzeugt und geöffnet, die dann allerdings in der Regel keine Teilstellen der Stelle sein müssen, der das aktuelle Ereignis der Listenermethode zuzuordnen ist. Ebenso können dort andere Listener-Instanzen aktiviert werden, etc. Es können aber auch andere Stellen geschlossen werden. Insbesondere erfolgt dort auch die Einbettung der Aufrufe von Conditions und Propagationsfunktionen (siehe unten).

10.2.4 Realisierung von Conditions und Propagationsfunktionen

Die im Ereignisstellendiagramm angegebenen Conditions oder Propagationsfunktionen werden als Methoden der Klasse realisiert, die der Codegenerator für die auslösende Stelle generiert. Mit auslösender Stelle ist die Stelle gemeint, an der ein Ereignis stattfindet, das den Stellenübergang mit der zugeordneten Condition oder Propagationsfunktion bewirkt.
Der Code für die Condition oder Propagationsfunktion wird im Allgemeinen vom Anwender des Prototypen zur Verfügung gestellt und vor der Kompilation in den generierten Code der Klasse integriert. Im derzeitigen Prototyp wird dabei der Aufruf der Condition oder Propagationsfunktion aus der Annotation direkt übernommen und im Code an eine Variable (evob), die die auslösende Stelle repräsentiert angehängt:

evob.<String aus Annotation zur Bezeichnung der Condition>
evob.<String aus Annotation zur Bezeichnung der Propagationsfunktion>

10.2.5 Prototypische Realisierung der Exemplarvariation

Die in Kap. 7 vorgestellte exemplarische Darstellung mit Variationsannotation ist im Prototyp soweit implementiert, dass Anwendungen der Art des in Kap. 8 beschriebenen JobStepEditors realisiert werden können. Gegenüber der in Kap.8 beschriebenen Notation gelten im derzeitigen Entwicklungsstand noch einige Einschränkungen bzw. Abweichungen, die aber die notwendige Funktionalität für die Implementierung des Beispiels von Kap.8 nicht beeinträchtigen. Die Abweichungen stellen sinnvolle und diskussionswürdige Alternativen zur Notation in Kap.8 dar.

10.2.5.0 Aspekte der Variation im Prototyp

Dies bedeutet, dass insbesondere als Aspekte der Variation die Anzahl eines Exemplars (Cardinality), die Identität eines Exemplars (Identity), der Wert (Value) sowie die Klasse des Widgets (Class), auf die eine Ereignisstelle abgebildet wird, variieren können. Sind dies Attribute, die auch einer abstrakten Stelle zugeschrieben werden können, sind als weitere Aspekte der Variation Attribute konkreter Widgets wie Position und Größe zugelassen.

10.2.5.1 Variationsereignisse

Als mögliche Ereignisse, die eine Variation auslösen, sind im Prototyp vorgesehen:
- Der Start der Applikation (EventType: ApplicationsStart),
- Das Kreieren des Parentobjektes (EventType: ParentCreation),
- Das Kreieren oder Öffnen des Exemplars selbst (EventType: Creation oder Open)
- Das Kreieren einer anderen Widget-Instanz (EventType: Creation oder Open, EventObject: <ObjectName>)
- KeyEvents einer Widget-Instanz (EventType: KeyEvent, EventSubType: keyTyped oder keyReleased, etc.)
- MouseEvents einer Widget-Instanz (EventType: MouseEvent, EventSubType: mousePressed oder mouseMoved oder mouseReleased, etc.)
- ActionEvents einer Widget-Instanz (EventType: ActionEvent)

Die Angabe logischer Ereignisse mit einer automatischen oder benutzerbestimmten Umsetzung auf obige konkrete Ereignisse an Widgets fehlt bisher.

10.2.5.2 Realisierung der Variationsfunktion

Zur Realisierung der im Modell in Kap.7 vorgestellten Variationsfunktion ist im Prototyp eine Schnittstelle aus mehreren Methoden vorgegeben, die im Folgenden erläutert werden soll.
Vom Prototyp wird zunächst erwartet, dass eine Methode getModel existiert, die ein sogenanntes Modelobjekt einer von der Klasse „DiaModel“ abgeleiteten Klasse liefert. (Der Begriff Model sei hier im Sinne des Model-View-Controller-Konzepts verstanden und soll nicht mit dem Begriff Modell im Sinne von Ereignis-Modell verwechselt werden.) Die Klasse DiaModel ist im Prototyp bereits vorgegeben.
Die Methode getModel muss der Klasse gehören, die der Codegenerator für die die Variation auslösende Stelle erzeugt. Vor dem Kompilieren wird die vom Anwender zur Verfügung gestellte Methode wie bei Conditions und Propagationsfunktionen in den Code der Klasse eingebettet.
Alternativ dazu kann auch die bereits vorhandene Methode einer Adapterklasse verwendet werden, die als übergeordnete Klasse für die Implementierung der Stelle gewählt wurde.

Die Adapterklasse ist in der Regel eine Ableitung einer Java-Swing-Klasse. Adapterklassen der wichtigsten Swingklassen werden, wie oben bereits erwähnt, vom Prototyp dem Anwender zur Verfügung gestellt und können im Editor des Prototypen per Selektion aus einer Liste einer Ereignisstelle zugeordnet werden.
Die Source der Adapterklassen, also auch die Methode getModel kann vom Anwender den Bedürfnissen der Applikation angepasst werden. Generell können jedoch auch vom Anwender eigene Adapterklassen eingeführt werden, die dann mittels Annotation der Ereignisstelle mitgegeben werden müssen.

Während die Adapterklasse in der Regel eine einfache Ableitung einer Java-Swing-Klasse oder einer Komfortklasse des Prototypen darstellt und Objekte der Oberfläche repräsentiert, verbirgt sich hinter der Modelklasse im Allgemeinen die Applikationslogik.
Die Signatur der Methode getModel zur Zuordnung eines Modelobjektes lautet:

```
DiaModel getModel(String variationName, Event e)
```

Der Methode wird beim Aufruf im generierten Code ein Bezeichner (variationName) mitgegeben, der eine Zuordnung des gesuchten Models zur gewünschten Variation ermöglicht. Außerdem wird die Methode mit dem konkreten Ereignis, das die Variation auslöst, versorgt.

Variationsfunktionen der Aspekte Cardinality und Identity
Als Methoden des Modelobjektes werden bestimmte Funktionen erwartet, die je nach Aspekt der Variation in ihrer Signatur unterschiedlich festgelegt sind. Für die Variation der Anzahl eines Exemplares (Aspekt Cardinality) beispielsweise sind zwei Funktionen vorgesehen, die im generierten Code zum Zeitpunkt der Variation zur Bestimmung der Anzahl und zur damit implizit verbundenen Bestimmung der Identität der einzelnen Instanzen aufgerufen werden:

```
int getCardinality(String variationName, String locationName, Event e)
String getObjectId(int index, String variationName, String locationName, Event e)
```

getCardinality liefert die Anzahl der Instanzen und getObjectId liefert zum jeweiligen Index einen String, der als eindeutiger Identifikator einer Instanz dient. Dabei wird angenommen, dass die Instanzen mittels Index durchnumeriert sind. Der Parameter locationName ist der Name einer Exemplarstelle, die variiert wird.

Der vom Codegenerator erzeugte Code sorgt für einen Aufruf der oben beschriebenen Methoden immer dann, wenn Information über die Anzahl und die Identität der variierten Exemplarstellen benötigt wird. D.h., dass die Methoden nicht nur beim Kreieren der Instanzen, sondern auch zu späteren Zeitpunkten, z.B. bei Wertübertragungen oder Variation anderer Aspekte als die der Kardinalität oder Identität genutzt werden, um die Menge der angesprochenen Instanzen zu bestimmen.
Was die Variation der Aspekte Kardinalität und Identität betrifft, weicht hier die Implementierung im Prototypen von der Beschreibung in Kap.7 und der formalen Darstellung insofern ab, als die Variationsfunktion zur Bestimmung der Anzahl und Identität nicht nur zum angegebenen Variationszeitpunkt sondern auch zu anderen Zeitpunkten aufgerufen wird. Letztlich liegt es dann bei der Implementierung der Methoden getCardinality und getObjectId, ob sich die Menge der Instanzen nur zum definierten Zeitpunkt der Variation oder auch zu anderen Zeitpunkten ändert. Der Zeitpunkt kann u.a. durch das mitgegebene Ereignis erkannt werden.

Für eine Variation des Aspektes Identität ohne Variation der Kardinalität erübrigt sich der Aufruf der Methode getCardinality. Zur Bestimmung der ObjectId wird eine Methode getObjectId ohne Index als Parameter verwendet.

Variationsfunktionen anderer Aspekte
Für die Variation weiterer Aspekte existieren ebenfalls entsprechende Schnittstellenmethoden, die wiederum in der Modelklasse des Modelobjektes erwartet werden, das per Aufruf von getModel() (siehe oben) zum Variationszeitpunkt referenziert wird. Als Beispiel seien hier die Methoden

Size getSize(String variationName, String locationName, Event e)

und

Size getSize(int index, String variationName, String locationName, Event e)

genannt, die bei Variation des Aspektes Size in den Fällen ohne Variation der Kardinalität bzw. mit Variation der Kardinalität verwendet werden.

11 Zusammenfassung und Ausblick

Ziel der Arbeit war, eine Modell-Darstellung eines Dialogsystems zu finden, die es erlaubt, eine logische Darstellung der Dialogoberfläche in möglichst intutitiver Form zu finden. Die Darstellung sollte den Designer wie auch den späteren Systembenutzer bereits in einer frühen Entwurfsphase unterstützen und es sollte andererseits eine rasche Umsetzung des Entwurfs in ein ablauffähiges System ermöglicht werden. Dabei sollte insbesondere der dynamische Aspekt der Oberfläche berücksichtigt und ein Überblick über die statische und dynamische Struktur des Dialogs gewährleistet werden.
Nachteile und Mängel bekannter Ansätze sollten dabei beseitigt werden.
Nach einer Diskussion dieser Ansätze wurde zunächst ein Grundmodell zum allgemeinen Verständnis des Dialogs entwickelt, das als Basis der dann folgenden Überlegungen diente.
Um eine möglichst einfache, intuitive Darstellung zu ermitteln, wurde im Bottom-up-Ansatz ausgehend von einer rigorosen extensiven Darstellung der Dialogereignisse zu intensiveren Darstellungsformen übergegangen.
Ergebnis ist ein Ereignisstellenmodell mit zugehörigen grafischen Darstellungselementen des Ereignisstellendiagrammes.
Um die Variabilität komplexer Anwendungen ausdrücken zu können wurde das ESM um das Konzept der Exemplarvariation erweitert, die es erlaubt, die einfache grafische Darstellung mit der Mächtigkeit gängiger Programmiersprachen zu verbinden, wobei die Anbindung in einer weitgehend deskriptiven Form erfolgt.
Nach der Erläuterung anhand eines konkreten Beispiels wurde das Ergebnis der Arbeit mit einigen bekannten Konzepten verglichen und eine Umsetzung der Überlegungen in einen Prototypen beschrieben.

Eine nahe liegende Fortführung der Arbeit liegt nun im Ausbau des Prototypen, der bisher nur beispielhaft für eine bestimmte Menge von Java-Widgetklassen die Funktionalität der Modellsprache umsetzt. Es ist außerdem ein Konzept zu entwickeln, das die Integration der Funktionalität in bestehende Entwicklungsumgebungen in softwaretechnisch eleganter Form erlaubt. Ein mögliches Zielsystem könnte z.B. das Eclipse-Framework darstellen.
Der Prototyp ist derzeit auf Basis von Java für Java-Swing-Oberflächen realisiert. Das Ereignisstellenmodell ist allerdings nicht auf eine Implementierung mit Java festgelegt. Was die Ebene der abstrakten Ereignisstellen betrifft, ist es von einer bestimmten Implementierungssprache unabhängig und kann auf verschiedene Dialogarten abgebildet werden. Was die Darstellung der konkreten Dialogelemente in den Ereignisstellendiagrammen betrifft, ist es zunächst nahe liegend, als Implementierungssystem ein Sytem zu wählen, das das Konzept von hierarchisch organisierten Oberflächenobjekten mit zugehörigen Attributen und eine Ereignisbehandlung im Sinne des Ereignismodells unterstützt.
Ein unmittelbar anstehendes Ziel ist die erfolgversprechende Umsetzung auf HTML-basierte Systeme.
Ein weiteres Ziel ist, den Gedanken der Exemplarvariation weiter auszubauen. Er wurde in der vorliegenden Arbeit zunächst einerseits nur allgemein und informell und andererseits formal nur für bestimmte, allerdings wichtige und nützliche Aspekte der Variation, wie Kardinalität, Identität und einige Widgetattribute entwickelt.

12 Glossar

Dialog: Dialog ist der Austausch von Informationseinheiten an einer Dialogschnittstelle zwischen Benutzer und System. Dialog kann definiert werden als eine Menge von strukturierten Dialogereignissen an einer strukturierten Dialogschnittstelle, durch die Information zwischen zwei oder mehreren Partnern ausgetauscht werden. Wir wollen im Folgenden von zwei Partnern, dem Benutzer und dem System, ausgehen. Im System kann (je nach Betrachtung) zwischen zwei Komponenten unterschieden werden, nämlich der Komponente zur Realisierung der Applikationslogik und der Komponente, die zur Realisierung der Dialogschnittstelle beiträgt.

Dialogart: Eine Dialogart beschreibt die Art und Weise in der die Kommunikation zwischen den Dialogpartnern geführt wird. Klassische Dialogarten sind z.B. Frage-Antwort-Dialog, Kommandodialog, Menüdialog, etc. Als moderne Dialogarten können wir den grafischen oder den multimedialen Dialog sehen. Dialogarten hängen zum Teil von den technischen Möglichkeiten**,** den verfügbaren Eingabe- Ausgabegeräten ab (siehe auch [FAEHN87]).

Dialogpartner: Als Partner fungieren ein oder mehrere **Benutzer** auf der einen Seite, auf der anderen Seite das System oder mehrere Systemkomponenten. Dabei lässt sich das System in zwei Bereiche aufteilen, nämlich in einen Bereich, der unmittelbar für die Realisierung des Dialogs verantwortlich ist, also die **Dialogschnittstelle** realisiert und unmittelbar den Kontakt zum Benutzer herstellt (Dialogkomponente), und einen Bereich, der mittelbar über diese Dialogkomponente als Dialogpartner kommuniziert, den wir häufig auch als **Applikationslogik**-Komponente oder Systemlogik-Komponente bezeichnen, ein Teil, der die eigentlichen und/oder nicht dialogspezifischen dem EDV-System zugedachten Aufgaben im Sinne der Gesamtapplikation oder des Gesamtsystems Mensch-Maschine realisiert.

Dialogstil: Dialogstil wird manchmal im Sinne von Dialogart (siehe oben) gesehen. Im Zusammenhang mit grafischem Dialog kann der Begriff Dialogstil aber auch verstanden werden als eine Sammlung von Darstellungsattributen, die die grafische Ausgestaltung bestimmen. So kann beispielsweise die Wahl eines bestimmten Zeichensatzes für bestimmte Abschnitte oder die Wahl einer Bestimmten Hintergrundfarbe einem bestimmten Dialogstil zugeordnet werden.

Formulardialog: Dialogart, in der Eingaben und Ausgaben überwiegend in Formularen mit Formularfeldern durchgeführt werden. Ein Formulardialog setzt eine Bildschirmfläche (Gesamtbildschirm oder Fenster) voraus, auf der ein Formular mit mehreren Feldern gleichzeitig dargestellt werden kann, sowie einen Mechanismus, der den Eingabefokus in vorgegebener Reihenfolge oder beliebig auf Felder positionieren lässt. Formulardialoge bieten durch das parallele Angebot der Felder am Bildschirm gegenüber dem Frage-Antwort-Dialog und dem Kommandodialog einen größeren Überblick für den Benutzer sowie ein bestimmtes Maß an Dialogführung.

Grafischer Dialog: Dialogart, die auf Basis hochauflösender Bildschirme mit Pixeltechnik die Darstellung grafisch gestalteter Dialogobjekte bietet. Für die Eingabe stehen dem Benutzer neben der Tastatur im Allgemeinen Geräte (Maus, Griffel, Pen, …) zur Selektion und direkten Manipulation dieser grafischen Objekte zur Verfügung. Grundlage des grafischen Dialogs ist die so genannte Fenstertechnik, die es erlaubt den Bildschirm in verschiedene unter Umständen auch überlappende Teilflächen zu unterteilen. Im grafischen

Dialog kann die Vorstellungswelt des Benutzers durch die Nachbildung realer Umgebungen und Objekte auf den Bildschirm übertragen werden (Desktop-Metapher, etc.).

Kommandodialog: Dialogart, in der der Benutzer Aufträge an das System in Form von Kommandos übergibt und gegebenenfalls das System in einer Rückmeldung das Ergebnis des Auftrags bekannt gibt. Ein Kommando wird dabei in der Regel in textueller Form, bestehend aus einem Kommando-Schlüsselwort und danach folgenden Kommandooptionen in einer Kommandozeile formuliert. Ein Charakteristikum des Kommandodialogs ist, dass der Benutzer weitgehend die Initiative im Dialog besitzt und das System reagiert. Der Kommandodialog erfordert relativ geringe technische Voraussetzungen (Kommandozeile und Tastatur) und ist einer der klassischen Dialogarten.

Layout: Unter Layout wird im weiteren Sinne die konkrete Ausgestaltung des Dialogs in einer bestimmten Dialogart und/oder in einem bestimmten Dialogstil verstanden. Das Layout des Dialogs ist in diesem Verständnis in Abgrenzung zum logischen Dialog zu sehen, der die eher inhaltlichen Aspekte des Dialogs umschreibt. Im engeren Sinne wird im grafischen Dialog unter Layout die Anordnung der verschiedenen Dialogobjekte, die Aufteilung der Bildschirmfläche auf die Objekte, die Zuordnung von Abständen der Objekte zueinander etc. verstanden.

Menüdialog: Dialogart, in der vom System Operationen und Werte dem Benutzer zur Auswahl einzeln hintereinander oder in Gruppen angeboten werden. Der Menüdialog ist in der Regel hierarchisch organisiert, so dass der Benutzer durch sukzessive Auswahl von Menüpunkten einen Auftrag absetzt bzw. einen Wert bestimmt. Im Menüdialog liegt die Initiative beim System. Der Benutzer wird durch das System geführt. Der Menüdialog benötigt relativ geringe technische Ressourcen und wird häufig in Systemen mit kleinen Ausgabedisplays eingesetzt.

Multimedialer Dialog: Der multimediale Dialog kann als eine Erweiterung des grafischen Dialogs gesehen werden. Er nutzt verschiedene Medien für Eingabe und Ausgabe, insbesondere neben den visuellen Möglichkeiten der Bildschirmdarstellung die akkustische Ausgabe über Lautsprecher und Eingabe über Mikrophon. Neben diskreter Ausgabe einzelner grafischer Objekte tritt der Einsatz von Bildanimation und die filmische Darstellung.

Widget: Unter Widget wird in dieser Arbeit im allgemeinen Sinne jegliches Objekt einer grafischen Dialogoberfläche verstanden, wie es in mehr oder weniger unterschiedlichen Ausprägungen in Baukastensystemen (Toolkits) zur grafischen Dialoggestaltung zur Verfügung steht. Im engeren Sinne kann ein Widget als eine Instanz einer Widgetklasse bekannt aus auf X-Windows aufsetzenden Systemen, wie dem Xt-Toolkit-System, verstanden werden. In dieser Arbeit sind damit beispielsweise auch die Objekte anderer Baukastensysteme, wie z.B. Java-Swing, etc. gemeint.

WYSIWYG-Darstellung: WYSIWYG steht für "what you see is what you get" und wurde ursprünglich für die exakte Wiedergabe der Ausgabe des Drucker auf dem Bildschirm geprägt. Ein erweitertes Verständnis des Begriffes, wie er in dieser Arbeit verwendet wird, bedeutet eine möglichst laufzeitgetreue Abbildung von Dialogobjekten und Dialogabläufen in einem Editor in der Phase des Dialogentwurfs.

13 Literaturverzeichnis

[BALZE94] Balzert, H.: "Das JANUS-System: Automatisierte, wissensbasierte Generierung von Mensch-Computer-Schnittstellen.", Informatik – Forschung und Entwicklung, 9(1), 3 1994

[BASTI90] Bastide, R., Palanque, P.: "Petri Net Objects for the Design, Validation and Prototyping of User-Driven Interfaces", Diaper et al. (editors), Human Computer Interaction – INTERACT90 proceedings, Elsevier Science Publishers, 1990

[BOOCH01] Booch, G., Rumbaugh, J., Jacobsen, I.: "The Unified Modeling Language User Guide", 9th printing, Object Technology Series, Addison Wesley, 2001

[CYPHE91] Cypher, A.: "EAGER: Programming Repetitive Tasks by Example", Human Factors in Computing Systems, Proceedings of SIGCHI '91, ACM, p.33-39, 1991

[DIN88] DIN Deutsches Institut für Normung: "DIN 66 234 Teil 8: Grundsätze ergonomischer Dialoggestaltung", Beuth, Berlin, Köln, 1988

[EICKEL90] Eickel, J.: „Logical and Layout Structures of Documents", Computer Physics Communication, 61:201-208, 1990

[EICKEL91] Eickel, J.: "Architektur von generativen Oberflächenwerkzeugen", Personal Communications, 1991

[EICKEL01] Eickel, J.: "Generierung von Bedienoberflächen", Vorlesungen an der TU München, Sommersemester 2001

[EISEN00] Eisenstein, J., Puerta, A.: „Adaptation in User-Interface Design", IUI 2000 New Orleans, ACM, 2000

[EISEN01] Eisenstein, J., Vanderdonckt, J., Puerta, A.: „Applying Model-Based Techniques to the Development of UIs for Mobile Computers", IUI'01 Santa Fe, New Mexico, USA, ACM, 2001

[FAEHN85] Fähnrich, K.-P., Ziegler, J.: „Direkte Manipulation als Interaktionsform an Arbeitsplatzrechnern", Software-Ergonomie'85 Mensch-Computer-Interaktion, B.G.Teubner, 1985

[FAEHN87] Fähnrich, K.-P.,(Hrsg.): Software-Ergonomie, Oldenbourg, 1987

[FISCHE00] Fischer, P.: „Grafik-Programmierung mit Java-Swing", Addison Wesley, 2000

[FOLEY74] Foley, J.D., Wallace, V.L.: "The Art of Natural Graphic Man-Machine-Conversation", Proceedings IEEE, Vol.62, No.4, p.462-470, 1974

[FOLEY90] Foley, J.D., et al.: "Computer Graphics Principles and Practice", Addison Wesley, 1990

[FOWLER97] Fowler, M., Scott, K.: "UML Distilled, Applying the Standard Object Modeling Language", 2nd printing, Object Technology Series, Addison Wesley, 1997

[GANZ78] Ganzinger, H.: "Optimierende Erzeugung von Übersetzerteilen aus implementierunngsorientierten Sprachbeschreibungen", Dissertation, Technische Universität München, 1978

[GREEN86] Green, M.: "A Survey of Three Dialogue Models", ACM Transcations on Graphics, Vol. 5, No. 3, p. 244-275, 1986

[HAREL87] Harel, D.: "Statecharts: A Visual Formalism For Complex Systems", Science of Computer Programming, North Holland, p.231-274, 1987

[HAREL88] Harel, D.: „On Visual Formalisms", Communications of the ACM, Volume 31, 1988

[HENTEN96] Van Hentenryck, P., Saraswat, V., et al.: "Strategic Directions in Constraint Programming", ACM Computing Surveys, Vol. 28, No. 4, 1996

[HERRT89] Herrtwich, F.G., Hommel, G.: „Kooperation und Konkurrenz: nebenläufige, verteilte und echtzeitabhängige Programmsysteme“, Springer-Verlag (Studienreihe Informatik), p.145-161, 1989

[HOARE85] Hoare C.: „Communicating Sequential Processes“, Prentice Hall, 1985

[HUDS89] Hudson, S.E.: „Graphical Specification of Flexible User Interface Displays”, Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology, ACM press, p.105-114, 1989

[ISO90] ISO: “ISO 9241, Ergonomic requirements for office work with visual display terminals (VDT)”, Draft, 12 1990

[JACOB86] Jacob, R.J.K.: “A Specification Language for Direct-Manipulation User Interfaces”, ACM Transactions on Graphics, Vol. 5, No. 4, p. 283-317, 1986

[JANSS96] Janssen, C.: “Dialogentwicklung für objektorientierte, graphische Benutzungsschnittstellen“, IPA-IAO Forschung und Praxis, Springer, 1996

[JESSEN87] Jessen, E., Valk, R.: „Modelle für Rechensysteme“, Springer, 1987

[LINTH02] Lin, J., Thomson, M., Landay, J.A.: “A Visual Language for Sketching Large and Complex Interactive Designs”, CHI 2002, Minneapolis, Minnesota, USA, DiavACM, 2002

[LONCZ97] Lonczewski, F.: „Providing User Support for Interactive Applications with FUSE“, IUI 97, ACM, 1997

[LOSCH96a] Lonczeswski, F., Schreiber, S.: „Generating User Interfaces with the FUSE System“, Technischer Bericht, Technische Universität München, Technical Report TUM-I9612, 1996

[LOSCH96b] Lonczeswski, F., Schreiber, S.: „The FUSE System: An Integrated User Interface Design Environment“, in Vanderdonckt, Hrsg., Computer-Aided Design of User Interfaces, Proceedings of the 2nd International Workshop on Computer-Aided Design of User Interfaces CADUI’96(Namur, 5-7 June 1996, p.37 – 56, Presses Universitaires Namur, ISBN 2-87037-232-9, 1996

[MICRO01] Reinke Solutions Team, “Microsoft Excel 2002, Das Handbuch”, Microsoft Press, ISBN 3860631608, 2001

[MILNER80] Milner, R.: “A Calculus of Communicating Systems”, Lecture Notes in Computer Science, Vol.92, Springer-Verlag, 1980

[MILNER99] Milner, R.: “Communicating and Mobile Systems: the π-Calculus”, Cambridge University Press, 1999

[MODUG93] Modugno, F., Myers, B.A.: “Graphical Representation and Feedback in a PBD-System”, in Watch What I Do: Programming by Demonstration, A.Cypher, Editor, MIT Press: Cambridge, MA, pp. 415-422, 1993

[MYERS88] Myers, B.A.: „Creating User Interfaces by Demonstration”, Academic Press, 1988

[OBERQ87] Oberquelle, H.: “Benutzerorientierte Beschreibung von interaktiven Systemen mit RFA-Netzen“, Software-Ergonomie’87, Tagung II/1987 des German Chapter of the ACM, p.271-284

[OESTER97] Oestereich, B.: „Objektorientierte Softwareentwicklung mit der Unified Modeling Language“, R.Oldenbourg, 1997

[PARRO95] Parrow, J.: „Interaction Diagrams“, Nordic Journal of Computing, p.407-443, 1995

[PETRI62] Petri, C.A.: „Kommunikation mit Automaten“, Schriften des Rhein. Westfael. Institutes für Instrumentelle Mathematik an der Universität Bonn, H.2, 1962

[PRJD96] Parigot, D., Roussel, G., Jourdan M., Duris, E., « Dynamic Attribute Grammars », Programming Languages, Implementations, Logics and Programs, PLIP’96 Proceedings, LNCS Springer, 1996

[PUERTA99] Puerta, A., Eisenstein, J.: „Towards a General Computational Framework for Model-Based Interface Development Systems", IUI 99 Redondo Beach CA USA, ACM,1999

[RUMBA91] Rumbaugh, M., et al.: „Object Oriented Modeling and Design", Prentice Hall, 1991

[SCHLU95] Schlungbaum, E., Elwert, T.: „Modellierung von graphischen Benutzungsschnittstellen im Rahmen des TADEUS-Ansatzes", Software-Ergonomie'95 Proceedings, Teubner, 1995

[SCHREI94] Schreiber, S.: "Specification and Generation of User Interfaces with the BOSS-System", in Cypher, A., Gornostaev, Hrsg., Proceedings East-West International Conference on Human Computer Interaction EWCHI'94. ICSTI Moscow 1994, also in Human Computer Interaction, Selected Papers EWCHI'94 Conference, Springer LNCS 876

[SCHREI95] Schreiber, S.: „The BOSS-System: Coupling Visual Programming with Model-Based Interface Design", in Paterno, F., Hrsg., Proceedings Eurographics Workshop, Design, Specification, Verification of Interactive Systems 1994, Carrara, Italy, June 8-10, 1994, Springer Focus on Computer Graphics Series, ISBN 3-540-59480-9, 1995

[SCHREI97] Schreiber, S.: "Spezifikationstechniken und Generierungswerkzeuge für graphische Benutzungsoberflächen", Herbert Utz Verlag Wissenschaft, 1997

[SHEBA93] Shebanek, M.: „The Complete Guide to the NeXTSTEP User Environment", Springer-Verlag Telos, 1993

[SIEMEN94] Siemens-Nixdorf Informationssysteme AG: "Dialog Builder/SQL V2.1 Datenbankerweiterung für Dialog Builder", Benutzerhandbuch, 1994

[SZWILL97] Szwillus, G.: "Objektorientierte Dialogspezifikation mit ODSN", Softwaretechnik, 1997

[VANDER88] Vander Zanden, B. T., „Constraint Grammars in User Interface Management Systems", Proceedings of Graphics Interface '88, Canadian Inf. Process. Soc., p. 176-184, 1988

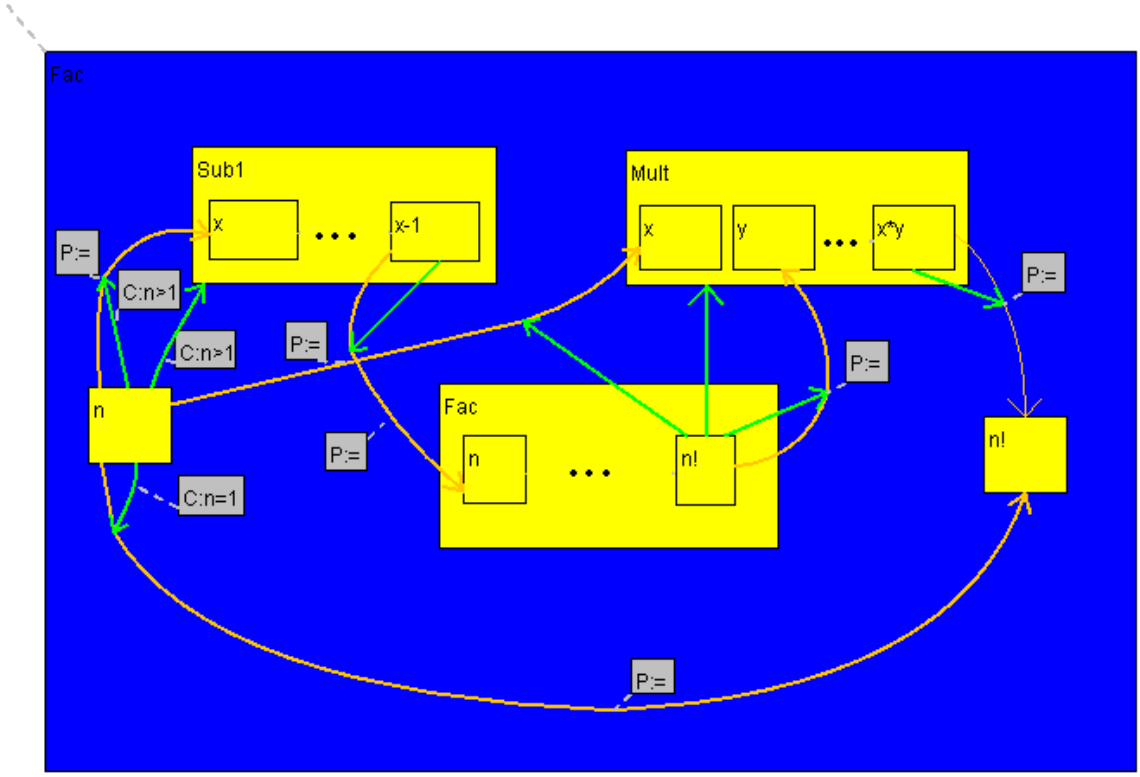
[VANDER95] Vander Zanden, B., Myers, B.A.: „Demonstrational and Constraint-Based Techniques for Pictorially Specifying Application Objects and Behaviours", ACM Transactions on Computer-Human Interaction, Vol. 2, No. 4, p. 308-356, 1995

[WELLN89] Wellner, P.,D.: "Statemaster: A UIMS based on Statecharts for Prototyping and Target Implementation", CHI'89 proceedings, ACM, 1989

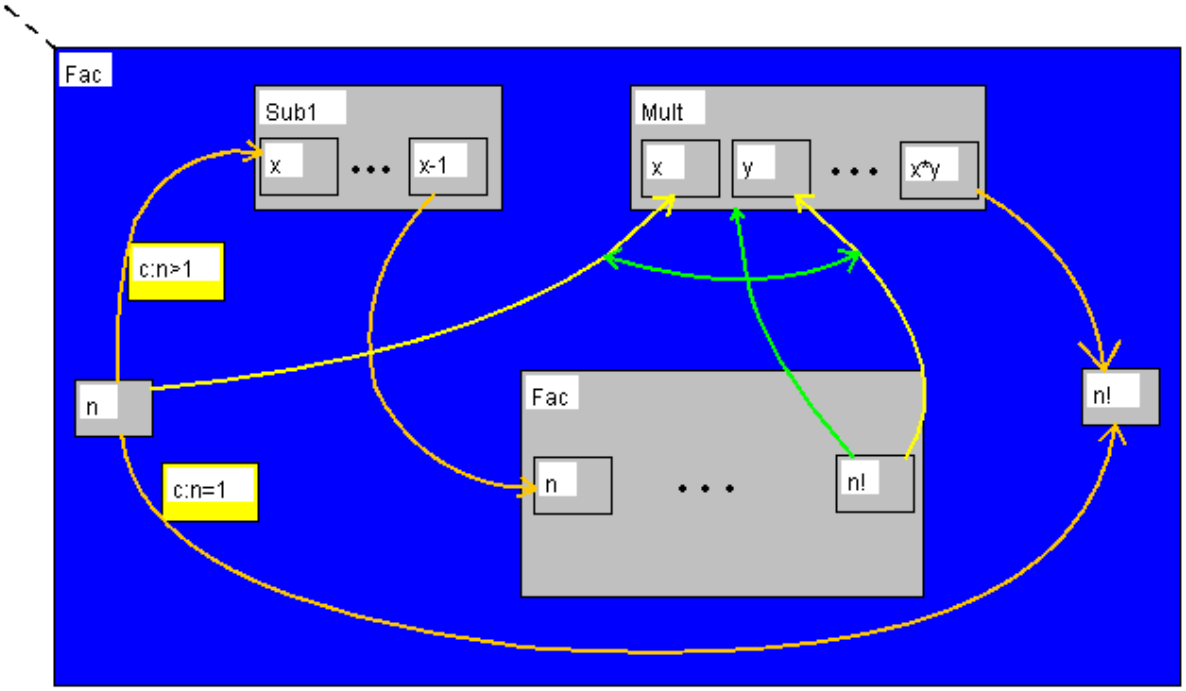
[ZEIDL92] Zeidler, A., Zellner, R.: „Software-Ergonomie, Techniken der Dialoggestaltung", Oldenbourg, 1992

[ZIEGLE93] Ziegler, J.: „Benutzergerechte Software-Gestaltung – Standards, Methoden und Werkzeuge", Oldenbourg, 1993

Anhang A) Rekursive Modellierung der Fakultätsfunktion



Fakultätsfunktion in einem Ereignisstellendiagramm. In obiger Abbildung entspricht die Darstellung der in Kap.5 beschriebenen Form. Dabei wurde auf die Annotation von Wertebereichen und Ereignistypen verzichtet. Es kann für alle Stellen der Ereignistyp M (Schreibereignis an unsichtbarer Speicherstelle) angenommen werden. Als Wertebereich gilt für alle Stellen die Menge der natürlichen Zahlen > 0.



Die Kantenannotationen „c:n>1“ und „c:n=1“ definieren die Bedingung des jeweiligen Übergangs. Die grau dargestellten Stellen, nämlich die innere Stelle Fac sowie die Stellen Mult und Sub1 sind Teilsichten, deren vollständige Definition anderswo ausgearbeitet ist (Fac

als übergeordnete Stelle im selben Diagramm). Die Substellen dieser Stellen werden nur dargestellt, soweit sie als Anschlüsse von und nach außen dienen.
In der zweiten Abbildung ist die grafische Form weiter vereinfacht, um die Übersichtlichkeit zu steigern. Dabei wurde auf die Angabe der Propagationsfunktion, die hier standardgemäß als Gleichheitsfunktion angenommen wird, verzichtet.
Die gelben Kanten beschreiben eine Wertübertragung. Die grünen Kanten triggern ein Ereignis an der Eingangsstelle aufgrund eines Ereignisses an der Ausgangsstelle bzw. triggern eine Wertübertragung. Die von der Stelle n! der inneren Stelle Fac ausgehenden drei grünen Triggerkanten sind hier zu einer sich verzweigenden Kante zusammengefasst. Die Kanten in Orange stellen eine Kombination aus einer grünen Triggerkante und einer gelben Kante zur Wertübertragung dar und symbolisieren das Auslösen einer Wertübertragung, bei der die Ausgangsstelle des zu übertragenden Wertes mit der Stelle des auslösenden Ereignisses übereinstimmt.

Anhang B) Übergangsfunktion für das hierarchische Ereignisstellenmodell

δ: Übergangsfunktion δ | $(QE \otimes WSHist \otimes E) \rightarrow (QE \otimes WSHist \otimes E^*)$
QE: Menge der Ereigniszustände,
wobei ein Ereigniszustand jeweils implizit definiert ist durch die Menge der möglichen Ereignisse
WSHist: Menge der Wertezuordnungen, die in einem Zustand an den Stellen S gelesen werden können, $WSHist \subseteq \{ \omega, \omega \text{ Funktion } \omega \mid S \rightarrow W \}$

Für $q, q' \in QE$, $\omega s, \omega s' \in WSHist$, $e \in E$, $\sigma(e) := s$ und $enext \in E^*$ gilt:
$\delta (q, \omega s, e) \rightarrow (q', \omega s', enext) \Leftrightarrow$
(

$e \in \varepsilon(q)$

e muss in der Menge der möglichen Ereignisse von q liegen

<u>und</u> $\varepsilon(q') = (\varepsilon(q) \cup \underset{s' \in S'}{U} \sigma^{-1}(s')) \setminus (\underset{s'' \in S''}{U} \sigma^{-1}(s''))$

Die q' zugeordnete Menge möglicher Ereignisse ergibt sich aus der Menge im Ausgangszustand q erweitert um die Menge der neu zu öffnenden Stellen S' (Definition siehe unten) und vermindert um die Menge der zu schließenden Stellen S''(Definition siehe unten)

<u>und</u> $\forall st \in S$ $\omega s'(st) = \omega e(e)$, falls $st = s$,
$\omega s'(st) = \omega s(st)$ sonst

Die neue Wertezuordnung ωs' nach Übergang ergibt sich an allen Stellen aus der alten mit Ausnahme an der Stelle s, wo der Wert mit dem Wert des Ereignisses e, also ωe(e) belegt wird (vereinfachter Fall).

<u>und</u>

Mit dem Öffnen einer Stelle s', also der Realisierung eines Stellenübergangs (s, s') werden unter Umständen auch Schreibereignisse des Systems „eröffnet". Dies ist dann der Fall, wenn der Stelle s' eine Ereignismenge zugeordnet ist, die auch Ereignisse der Typen O_S oder M_S enthält.
In diesem Falle wird über die (s, s') zugeordnete Propagationsfunktion die Entscheidung gefällt, welches der möglichen Ereignisse als nächstes an der Stelle s' stattfinden soll. Hier wird der Sicht des Modells Rechnung getragen, dass auch das Schreiben des Systems an Ausgabestellen und Hilfsstellen über Ereignisse mit entsprechenden Zustandübergängen abgehandelt werden, wobei die Abarbeitung dieser Ereignisse unmittelbar nach dem Öffnen der Stelle, also synchron erfolgt.
Wenn nun mehrere Stellenübergänge bei einem δ-Übergang erfolgen, muss über eine Ordnungsfunktion auf den Stellenpaaren in RO entschieden werden, in welcher Reihenfolge die Abarbeitung der Schreibereignisse erfolgen soll.

Diese Ordnung ergibt dann die Reihenfolge im Tupel enext der propagierten Ereignisse:
$enext = (e_0 , .. e_i, .. e_m),\ 0 <= i <= m$, mit $i,m \in N$,

falls | S' | = m+1

mit $e_i = (p_{(sROsi)}(\omega_S), s_i, t_i)$, mit $s_i \in S'$, $t_i \in \{ O_S, M_S \}$

falls $p_{(sROsi)}$ definiert,

also $\pi_{RO}(sROs_i) = p_{(sROsi)} \neq null$

$e_{(s,si)} = null$, sonst

und $ord_{RO}(s,s_i) = i$

ord_{RO} sei eine geeignete Ordnungsfunktion auf den Paaren $(s_k,s_l) \in RO$, die jeweils allen Paaren mit gleicher Ausgangsstelle s_k eine eindeutige Ordnungszahl zuordnet.

und $t_i = O_S$, falls s_i Dialogstelle, $t_i = M_S$, sonst

enext = null, sonst

und

Bleiben noch S' und S'' zu definieren.
Ein δ-Übergang kann das Öffnen oder Schließen von Stellen bewirken und damit die Menge der möglichen Ereignisse erweitern und/oder einschränken. In der nächsten Klammerung werden die Kriterien für das Öffnen einer Stelle und in der darauf folgenden die für das Schließen festgelegt. Dies geschieht in Form der Definition zweier Stellen-Teilmengen S' bzw. S'' bzw. zweier Relationen RO' und RC', die RO bzw. RC einschränken:

(

1.Definition S'
Zu Öffnende Stellen S': Voraussetzung für das Öffnen einer Stelle s' ist, dass die Stelle s, an der das Ereignis stattfindet, über RO in Beziehung steht. Es müssen jedoch noch weitere Bedingungen erfüllt sein, die zu einer weiteren Einschränkung der Übergänge s nach s' in RO führen.
Wir definieren also S', respektive ROH':

$s' \in S' \Leftrightarrow (s,s') \in ROH' \Leftrightarrow$

(

$((s,s') \in RO'$

oder

$(\exists (s,s''') \in RO'$ und $\exists (s''',s') \in HSStart$

d.h. s' ist Startknoten von s'''

HSStart, eine Stelle u ist Startstelle von t, sei dabei wie folgt definiert:

und

$(t,u) \in HSStart \Leftrightarrow$

(

$\exists$ k-Tupel (s_0, ..s_i, .. s_k) mit $s_0, s_i, s_k \in S$, $i, k \in N$,

und $s_0 = t$ und $s_k = u$

und $(s_i, s_{i+1}) \in HS$ für alle i = 0, .., k-1,

d.h. s_{i+1} ist Teilstelle von s_i

und $s_i \in S_0$ für alle i = 1, .., k

d.h. s_i ist aus der Menge der Startstellen S_0

)

)

)

Definition von RO':
und (s,s') ∈ RO' ⇔
(
(s,s') ∈ RO
und

Ereignis e muss aus der Menge der auslösenden Ereignisse stammen, die dem Übergang s,s' zum Öffnen zugeordnet ist, also:

$e \in \varepsilon_{RO}(s,s')$
und

Die dem potentiellen Übergang s,s' aus RO zugeordnete Condition muss bei der aktuell gültigen Wertebelegung ωs den Wert TRUE liefern:
$c(\omega_{akt}) = TRUE$, mit $c = \kappa_{RO}(sROs')$
und $\omega_{akt} = \omega s_{|Sc}$,
wobei $\omega s_{|Sc}$ die aktuelle Wertezuordnung ωs, eingeschränkt auf die für c definierte Stellenmenge S_c
)
) *Ende Definition S', ROH' mit aktuellen Übergängen s,s' zum Öffnen s'*

und
(
2. Definition S'':
Zu schließende Stellen S''. Ähnlich zu 1. muss nun ein potentieller Übergang s,s'' aus RCH' existieren, der durch RC sowie die Teilstellenhierarchie HS definiert ist und auch noch weitere Bedingungen erfüllt.
Definition S'', respektive RCH':

(s,s'') ∈ RCH' ⇔
(((s,s'') ∈ RC'
oder
(∃ (s,s''') ∈ RC' und ∃ (s''',s'') ∈ HS*
d.h. s'' ist Teilstelle von s'''
)
)
und

s'' ∈ S'' ⇔ (s,s'') ∈ RC' ⇔
((s,s'') ∈ RC
und
$e \in \varepsilon_{RC}(s,s'')$
und

$c'(\omega_{akt}) = TRUE$, mit $c' = \kappa_{RC}(sRCs')$
und $\omega_{akt} = \omega s_{|Sc'}$,

wobei $\omega s_{|Sc'}$ die aktuelle Wertezuordnung
ωs, eingeschränkt auf die für c definierte
Stellenmenge $S_{c'}$

)

) *Ende Definition S'', RCH' mit aktuellen Übergängen s,s'' zum Schließen s''*

) *Ende Definition Übergangsfunktion*

Anhang C) Induktion von Ereignishierarchien über Stellenhierarchie

Es macht in vielen Fällen Sinn, eine Folge bzw. Teilfolge von Ereignissen wieder als ein Ereignis zu betrachten. In diesem Verständnis können über die im Grundmodell Kap. 3 angegebenen Ereignisfolgen Hierarchien von Ereignissen konstruiert werden.
Eine mögliche Unterteilung in Teilfolgen erreichen wir, wenn wir alle Ereignisse in einer Ereignisfolge, die an derselben Stelle stattfinden, als Teilfolge betrachten. Alle Ereignisse dieser Teilfolge können wir als ein zusammengesetztes Ereignis interpretieren.
Existiert nun eine Hierarchie über den Stellen, können so gewonnene zusammengesetzte Ereignisse an Teilstellen wiederum zu einem Ereignis einer übergeordneten Stelle zusammengefasst werden usw.

Eine Hierarchie der Ereignisse H_f bezüglich einer Ereignisfolge f kann über eine Hierarchie von Stellen $X \subset (S \otimes S)$ wie folgt konstruiert werden:

$$(e_1,e_2) \in H_f \Leftrightarrow \quad (\sigma(e_1), \sigma(e_2)) \in X \ \wedge \ e_1 = \zeta(\sigma(e_1), f) \quad \text{oder} \quad \sigma(e_1) = \sigma(e_2) \ \wedge \ e_1 = \zeta(\sigma(e_1), f)$$

wobei ζ die Funktion (Konstruktionsfunktion) darstellen soll, die jeder Stelle s bezüglich einer Folge f genau ein zusammengesetztes Ereignis, das Gesamtereignis an der Stelle s bezüglich f, zuordnet. Das zusammengesetzte Ereignis entsteht dabei so, dass alle an einer Stelle und deren Teilstellen in einer Ereignisfolge f stattfindenden Ereignisse zu einem der Stelle zugeordneten Gesamtereignis zusammengefasst werden:

$$\zeta \mid (S, F) \rightarrow E \ , \qquad \zeta \text{ partiell}$$

ζ ist partiell definiert, da ggf. nicht in allen Ereignisfolgen aus F an allen Stellen aus S Ereignisse auftreten.

H kann nun z.B. definiert werden als die Relation (eine Polyhierarchie), die alle durch die Stellenhierarchie X induzierten Hierarchien H_f vereint:

$$H = \bigcup_{f \in F} H_f$$

Anhang D) RadioButton-Modell implementiert mit einem und vier Widgets

Tabellarische Gegenüberstellung zur Implementierung der Stellen des Ereignisstellenmodells eines RadioButtons (Kap.6) mit vier Widgetinstanzen und einer Instanz JRadioButton:
Die Operationen, die zur Realisierung der logischen Ereignisse S_0 bis S_3 bei der Implementierung mit einer einzigen RadioButton-Instanz (Abschnitt 6.1) angegeben sind, sind nur notwendig, wenn ein Übergang von einer Stelle außerhalb zur jeweiligen Stelle S_0, S_1, S_2 oder S_3 erfolgt. Übergänge zwischen den Stellen S_0, S_1, S_2 und S_3 sind interne Übergänge innerhalb der RadioButton-Instanz und werden automatisch durch den Code des RadioButtons geleistet.

Stelle abstrakt	*Ereignis*		*Widget konkret*		*Ereignis*	*Attribut*		*Operationen*	
	Typ	***Wert***	***Klasse***	***Instanz-ID***	***Typ***	***Typ/Name***	***Wert***	***Öffnen***	***Schließen***
S_0	**I**	**„An“**	JButton	„OnButton“	Mausklick			Widget kreieren, sichtbar setzen, sensitiv setzen	Widget insensitiv setzen, oder unsichtbar setzen, oder löschen
S_0	**I**	**„An“**	JRadioButtom	„MyRadioB“	MausPress	isSet	true	Widget kreieren, sichtbar setzen, sensitiv setzen, isSet mit false vorbelegen	isSet mit true belegen, oder Widget insensitiv setzen, oder unsichtbar setzen, oder löschen
S_1	**O**	**„Aus“**	JLabel	„OutLabel“	Ausführen Operation zum Setzen Attribut-wert	LabelIcon	„LampeAn“	Widget kreieren, sichtbar setzen, “LampeAn”-Icon setzen	„LampeAus“-Icon setzen, oder Widget unsichtbar setzen, oder löschen
S_1	**O**	**„Aus“**	JRadioButton	„MyRadioB“	Ausführen Operation zum Setzen Attribut-wert	isSet	false	Widget kreieren, sichtbar setzen, isSet false setzen	isSet true setzen, oder Widget unsichtbar setzen, oder löschen
S_2	**O**	**„An“**	JLabel	„OnLabel“	Ausführen Operation zum Setzen Attribut-wert	LabelIcon	„LampeAn“	Widget kreieren, sichtbar setzen, „LampeAn“-Icon setzen	“LampeAus”-Icon setzen, oder Widget unsichtbar setzen, oder löschen
S_2	**O**	**„An“**	JRadioButton	„MyRadioB“	Ausführen Operation zum Setzen Attribut-wert	isSet	true	Widget kreieren, sichtbar setzen, isSet true setzen	isSet false setzen, oder Widget unsichtbar setzen, oder löschen
…	**…**	**…**	Fortsetzung	Nächste	Seite	…	…	…	…

Anhang D)

Stelle abstrakt	***Ereignis***		***Widget konkret***		***Ereignis***	***Attribut***		***Operationen***	
	Typ	***Wert***	***Klasse***	***Instanz-ID***	***Typ***	***Typ/Name***	***Wert***	***Öffnen***	***Schließen***
S_3	**I**	**„Aus“**	JButton	„OutButton“	Mausklick			Widget kreieren, sichtbar setzen, sensitiv setzen	Widget insensitiv setzen, oder unsichtbar setzen, oder löschen
S_3	**I**	**„Aus“**	JRadioButton	„MyRadioB“	MausPress	isSet	false	Widget kreieren, sichtbar setzen, sensitiv setzen, isSet mit true vorbelegen	isSet false setzen oder Widget insensitiv setzen, oder unsichtbar setzen, oder löschen

Anhang E) Formale Darstellung der Exemplarvariation

Es folgt eine Einführung der Exemplarvariation in das Ereignisstellenmodell in formaler Darstellung am Beispiel der inkrementellen Variation einer Exemplarstelle und Variation des Aspekts Kardinalität .
Dazu wird das in Kap.5 vorgestellte Ereignisstellen-Modell erweitert. Den Kern der Erweiterung stellt die Einführung einer Funktion „variation“ dar, die dynamisch das Modell in verschiedene so genannte Modellkonfigurationen abändert.

Wir erweitern die Übergangsfunktion δ, die wir zum Unterschied von δ aus dem Ereignisstellen-Modell ESM in Kap.5 nun δdyn nennen wollen und die nun zusätzlich die dynamische Änderung unseres Modells hinsichtlich folgender Modellelemente aus ESM Kap.5 ausdrückt:

$$S, E, RO, RC, HS, \varepsilon_{RO}, \varepsilon_{RC}, \kappa_{RO}, \kappa_{RC}, \pi_{RO}$$

Wir führen bei einer Variation häufig neue Stellen (Stellenmenge S) in unser System ein, die wir aus einem ideal gesehen unendlichen Stellenreservoir SUniv schöpfen.
Dass sich die Menge der Ereignisse E ändern kann, liegt schon daran, dass neue Stellen in unser System eingeführt werden.
Bei Eintreten eines Ereignisses, das die Berechnung einer Variation auslöst, können ggf. zusätzlich Stellen zu öffnen (RO) oder zu schließen (RC) sein. Die Stellen, von denen aus Stellenübergänge erfolgen, können sich ändern. Die hierarchische Struktur HS der Teilstellen kann variieren. Mit neuen Stellenübergängen gibt es auch neue Mengen auslösender Ereignisse und damit geänderte Zuordnungen ε_{RO}, ε_{RC}.
Prinzipiell wollen wir auch erlauben, dass die Menge der akzeptierten Ereignisse einer Stelle dynamisch bestimmt werden kann sowie die Menge der einen Übergang auslösenden Ereignisse, die über ε_{RO}, ε_{RC} zugeordnet werden.

Ebenso ändern können sich die Zuordnungen von Conditions zu Stellenübergängen κ_{RO} und κ_{RC} sowie die von Propagationsfunktionen π_{RO}.
Dies liegt ebenfalls schon daran, dass neue Stellenübergänge definiert werden, denen entsprechende Conditions und Propagationsfunktionen zuzuordnen sind.
Schließlich ist ωStart der ggf. geänderten Stellenmenge anzupassen.

Vars sei nun die Menge der Variationen (oder Variationsvorschriften) var $\in$ Vars,
mit var = (VarEvents, VarS, VarAspects, varType, varFunc)

VarEvents: Menge der Ereignisse, VarEvents $\subseteq$ SUniv $\otimes$ T $\otimes$ W
VarS: Menge der Variationsstellen als Gegenstände der Variation VarS $\subseteq$ SUniv
VarAspects: Menge der zu variierenden Aspekte, VarAspects $\subseteq$ Aspects,
wobei Aspects = { Card, Id, PS, Value }, Menge der möglichen Aspekte
Card: Konstante, die für Variation der Kardinalität steht
Id: Konstante, die für Variation der Stellenidentität steht
PS: Konstante, die für Variation der Teilstellen steht (PartialStructure)
Value: Konstante, die für Variation des Stellenwertes steht

VarType $\in$ { inc, abs }, wobei VarType den so genannten Variationstyp bezeichnet und

inc: eine Konstante, die für eine inkrementelle Variation steht
abs: eine Konstante, die für eine so genannte absolute Variation steht

varFunc ist eine so genannte Variationsfunktion aus einer Menge von Funktionen der Form:

varFunc | E ⊗ MConfigs → MConfigs

Wir gehen davon aus, dass sich die Variationen mit Eintritt eines Ereignisses ebenso ändern können. Dies z.B. schon deshalb, weil sich durch die Hinzunahme weiterer Stellen die Menge der eine Variation var auslösenden Ereignisse (VarEvents) ändern kann.

Mit der Folge der Ereignishistorie $ehist_0$, ... $ehist_i$, ..., wobei $ehist_i = e_0, e_1, ... e_i$, für $i \in N$, (e_0 sei das Ereignis des Starts, $e_1, ... e_i \in E$) erhalten wir somit eine Folge von Modellkonfigurationen

$$mConfig_i = (S_i, SStart_i, E_i, RO_i, RC_i, HS_i, Vars_i, \varepsilon RO_i, \varepsilon RC_i, \kappa RO_i, \kappa RC_i, \pi RO_i, q_i, \omega S_i),$$

wobei gilt:

$$\delta dyn\ (e_i, mConfig_i) = (mConfig_{i+1}, enext_{i+1}),\ \text{mit } enext_{i+1} \in E^*$$

Die mit dem Index versehenen Tupelelemente einer Modellkonfiguration sind dabei entsprechend den Tupelelementen ohne Index im ESM-Modell Kapitel 5 definiert, abgesehen vom neu hinzugekommenen Element $Vars_i$, das oben beschrieben ist. Dabei ist S aus ESM jeweils durch SUniv zu ersetzen.
$SStart_i$ ist dabei die Menge der lokalen und globalen Startstellen (im ESM Kap.5 mit S_0 bezeichnet). $SStart_i$ darf nicht mit S_0 in $mkonfig_0$ verwechselt werden. Die $SStart_i$ sind jeweils Teilmenge von S_i und ändern sich von Konfiguration zu Konfiguration, weil z.B. ggf. neue Stellen eingeführt werden und/oder die HS_i variieren.

Das sich ergebende erweiterte ESM-Modell das wir ESMD (ESM-Dynamisch) nennen wollen, stellt sich als folgendes Tupel dar:

ESMD = (SUniv, T, W, FMConfigs, C, P),

wobei FMConfigs die Folge der Modellkonfigurationen $mConfig_i$, $i = 0, 1, ... i \in N$ darstellt, mit $mConfig_0$ als fest vorgegebener Startkonfiguration.
T, W, C, P seien wie im ESM Kap.5 definiert.

Änderungen der Funktion δdyn gegenüber der δ-Funktion des ESM aus Kapitel 5.

Bevor der Übergang von q nach q‘ und der Übergang von ωS zu ωS‘ durchgeführt wird, wird eine Variation der Teilkonfiguration (S_i, $SStart_i$, E_i, RO_i, RC_i, HS_i, $Vars_i$, εRO_i, εRC_i, κRO_i, κRC_i, πRO_i, ωS_i) vorgenommen, falls das Ereignis e eine Variation auslöst.

Dies bedeutet:

$\delta dyn\ (e_i, mConfig_i) =$

((S_{i+1}, $SStart_{i+1}$, E_{i+1}, RO_{i+1}, RC_{i+1}, HS_{i+1}, $Vars_{i+1}$, εRO_{i+1}, εRC_{i+1}, κRO_{i+1}, κRC_{i+1}, πRO_{i+1}, $\omega Start_{i+1}$ q_{i+1}, ωS_{i+1}), $enext_{i+1}$),
mit $(q_{i+1}, \omega S_{i+1}, enext_{i+1}) = \delta core(e_i, variation(e_i, mConfig_i))$,

$\delta core$ sei die Funktion δ aus ESM Kap.5, die statt der beiden Parameter q und ωS als Parameter eine gesamte Modellkonfiguration mConfig aus MConfigs, der Menge aller Modellkonfigurationen, enthält:

$\delta core$ Funktion $\delta core \mid (E \otimes MConfigs) \rightarrow ((QE \otimes WSHist) \otimes E^*)$

$\delta core$ entsteht aus der Funktion δ aus Kap.5, indem die dort verwendeten Bezeichner S, S_0, E, RO, RC, HS, εRO, εRC, κRO, κRC, πRO als Bezeichner der Elemente einer Modellkonfiguration mConfig gesehen werden, die als Parameter übergeben wird. Ansonsten gilt entsprechend die Beschreibung in Kap.5.

variation ist eine Funktion variation | $(E \otimes MConfigs) \rightarrow (MConfigs)$,
die abhängig von einem Ereignis e und einer Modellkonfiguration mConfig eine neue Modellkonfiguration mConfig' bestimmt.

Die Funktion variation sei wie folgt definiert:
Für $e \in E$, mConfig = (S, SStart, E, RO, RC, HS, Vars, εRO, εRC, κRO, κRC, πRO, q, ωS),
mConfig' = (S', SStart', E', RO', RC', HS', Vars', $\varepsilon RO'$, $\varepsilon RC'$, $\kappa RO'$, $\kappa RC'$, $\pi RO'$, q', $\omega S'$)
und variation Funktion variation | $(E \otimes MConfigs) \rightarrow (MConfigs)$ gilt

variation (e, mConfig) = mConfig' $\Leftrightarrow$
(
falls nicht gilt: $e \in VarEvents$
mit var = (VarEvents, VarS, VarAspects, VarType, varFunc) für ein $var \in Vars$
dann mConfig = mConfig',

andernfalls:
(
Wir nehmen zunächst an, es gibt genau ein var, für das $e \in VarEvents$ gilt.
Außerdem nehmen wir an, dass diese Variation vom Typ „inkrementell" ist, also VarType = inc, dass VarAspects = { Card } und dass | VarS | = 1, also die Variation genau eine Exemplarstelle besitzt.

Dann bestimmen sich die Elemente der Konfiguration mConfig' wie folgt:

S':
$S' = S \cup Sneu \cup SSubneu$,
mit $Sneu \subset SUniv$, Sneu endlich, projS (varConfig) = SvarConfig und
varConfig = varFunc(e,mConfig)
und Sneu = SvarConfig \ S,

projS sei die Projektion von S, definiert auf der Menge der Konfigurationen MConfigs:
projS | MConfigs → Potenzmenge(SUniv)

(Wir wollen im Folgenden generell die Projektionen der einzelnen Tupelelemente einer Konfiguration mit proj<Tupelelement> bezeichnen, also projE, projRO usw.)

Für jede Stelle aus Sneu, die nicht bereits in S enthalten ist (echte neue Stelle), wird dieselbe Teilstellenstruktur angenommen, die auch die Exemplarstelle besitzt. D.h. es werden in die Modellkonfiguration entsprechende neue Teilstellen aufgenommen.

SSubneu ist die Erweiterung der Stellen aus Sneu um ihre Teilstellen entsprechend der Hierarchie der Exemplarstelle svar. D.h. es gibt jeweils eine bezüglich der Hierarchie strukturtreue bijektive Abbildung der Menge der Teilstellen $SSub_{svar}$ von svar auf jeweils die Menge der Teilstellen $SSub_{sneu}$ für ein bestimmtes sneu aus Sneu.

$SSub_s$ ist wie folgt definiert: $SSub_s = \{ s' , (s,s') \text{ aus } HS'^* \}$, dabei definiere HS'* die transitive und reflexive Hülle von HS'

HS':
HS' ist die in der neuen Konfiguration mConfig' sich ergebende hierarchische Relation, die durch Variation von HS entsteht.

$HS' = HS \cup HSubneu \cup HSuperneu$,
HSuperneu ist die hierarchische Einordnung der Stellen aus Sneu unter die mit der Exemplarstelle svar gemeinsamen übergeordneten Stelle, falls diese vorhanden:

$HSuperneu = \{ (s,s'), \text{ mit } s' \in Sneu \text{ und } (s,svar) \in HS \text{ und } svar \in VarS \}$

und für HSubneu gilt:
$(s',s'') \in HSubneu \Leftrightarrow \exists\, s \in Sneu \text{ mit } s',s'' \in SSub_s \text{ und } (origSneu(s'),origSneu(s'')) \in HS$

origS ist eine bezüglich der Hierarchie strukturtreue Abbildung für alle Teilstellen von s aus Sneu , die jeder der Teilstellen von Stellen aus Sneu inklusive der Stellen aus Sneu selbst die in der hierarchischen Struktur entsprechende exemplarische Teilstelle bzw. Stelle aus $SSub_{svar}$ zuordnet:

$$origSneu \mid \bigcup_{s \in Sneu} SSub_s \rightarrow SSub_{svar},$$

Es gilt $(s,s') \in HS'^* \Leftrightarrow (origSneu(s),origSneu(s')) \in HS^*$ für alle $s,s' \in \bigcup_{s \in Sneu} SSub_s$

Im Folgenden sei $SSubneu := \bigcup_{s \in Sneu} SSub_s$,
die Vereinigung aller Teilstellen von Sneu (inkl. Stellen selbst).
Außerdem wollen wir die Funktion origSneu im Definitionsbereich und Wertebereich erweitern zu einer Funktion orig | SUniv → SUniv, die für alle nicht neuen Stellen die Identität liefert. Dies erlaubt uns in den folgenden Ausführungen eine verkürzte Schreibweise.
Also: orig(s) = s , falls s nicht ∈SSubneu
orig(s) = origSneu(s) sonst

E':
$E' = E \cup Eneu$,

mit Eneu = { (sneu, τ(evar), ωe(evar)), evar $\in$ σ^{-1}(orig(sneu)) und sneu $\in$ SSubneu}

RO':
RO' = RO $\cup$ ROneu,
mit ROneu = { (s, sneu) } $\cup$ { (sneu, s') }, mit
sneu $\in$ Sneu und (s, orig(sneu)),(orig(sneu),s') $\in$ RO,
s, s' $\in$ SUniv,

d.h. alle neuen Stellen erhalten die Stellenübergänge zum Öffnen wie die Exemplarstelle bzw. deren Teilstellen.
Entsprechendes gilt für die Schließe-Relation:

RC':
RC' = RC $\cup$ RCneu,
mit RCneu = { (s, sneu) } $\cup$ { (sneu, s') }, mit
sneu $\in$ SSubneu und (s, orig(sneu)), (orig(sneu), s') $\in$ RC,
wobei s,s' $\in$ SUniv

Mit den Öffne- und Schließe-Beziehungen der Exemplarstellen werden auch die Conditions und Propagationsfunktionen an neue Stellenübergänge entsprechend übernommen. Hier muss allerdings berücksichtigt werden, dass die Parameterversorgung der Conditions und Propagationsfunktionen an den neuen Stellenübergängen gegenüber den alten Übergängen variieren kann. Dies bedeutet insbesondere, dass Werte von anderen Stellen gelesen werden. Formal hatten wir in Kap.5 für Conditions und Propagationsfunktionen als Definitionsbereich selbst wieder eine Menge von Funktionen definiert, nämlich die Menge der Wertezuordnungen zu einer definierten Teilmenge von Stellen. Diese Stellenmenge, die die mögliche Parameterversorgung einer Condition beeinflusst, ändert sich nun in bestimmten Fällen.
Wir wollen uns hier nur auf einen einfachen aber sinnvollen Fall der Änderung der Werteversorgung beschränken, nämlich auf den Fall, dass der Wert von der neuen Stelle gelesen wird, wenn er vorher von der ursprünglichen Exemplarstelle abgelesen wurde. Wir führen dazu formal eine Funktion ein, die eine entsprechende Vertauschung der Stellen bezüglich der Werteversorgung berücksichtigt, die wir jeweils κ'_{RO}, κ'_{RC} und π'_{RO} nennen wollen:

κ'_{RO} | RO' $\rightarrow$ (C $\cup$ $\varnothing$),
κ'_{RO}((s,s')) = c' mit (s,s') $\in$ RO', c' $\in$ C,

wobei die Condition c' $\in$ C sich von c = κ_{RO}(orig(s),orig(s')) dadurch unterscheidet, dass bei allen Wertezuordnungen im Definitionsbereich die Stellen orig(s) bzw. orig(s') ersetzt werden durch die Stellen s bzw. s', falls orig(s) bzw. orig(s') dort vorkommen.
Entsprechend wie κ'_{RO} sind auch κ'_{RC} und π'_{RO} zu interpretieren.

Damit ergibt sich nun für $\kappa_{RO'}$:

$\kappa_{RO'}$((s,s')) = κ'_{RO}((s,s')), für (s,s') $\in$ RO'

Handelt es sich bei s und s' um keine neuen Stellen, gilt κ'_{RO}((s,s')) = κ_{RO}(orig(s),orig(s')), da dann orig(s) = s und orig(s') = s' und ein Vertauschen von Stellen in den Wertezuordnungen des Definitionsbereichs der Condition wirkungslos ist.

Ist eine der beiden Stellen s,s', o.B.d.A. s, eine neue Stelle wird in den Wertezuordnungen, die die Elemente des Definitionsbereichs von $c = \kappa_{RO}(orig(s),orig(s'))$ darstellen, ein Vorkommen der entsprechenden Exemplarstelle orig(s) gegen s ausgetauscht. Somit ergibt sich dann per Definition die Condition $c' = \kappa'_{RO}((s,s'))$.
Sind beide Stellen neue Stellen, gilt entsprechend, dass die Vorkommen beider Exemplarstellen orig(s) und orig(s') gegen s bzw. s' ausgetauscht werden.

In gleicher Weise gilt für $\kappa_{RC'}$:

$\kappa_{RC'}((s,s')) = \kappa'_{RC}((s, s'))$, für $(s,s') \in RC'$

und für $\pi_{RO'}$:

$\pi_{RO'}((s,s')) = \pi'_{RO}((s,s'))$, für$(s,s') \in RO'$.

Für die auslösenden Ereignisse der Stellenübergänge gilt:

$\varepsilon_{RO'}$:
$\varepsilon_{RO'}(s,s') = \varepsilon_{RO}(s,s')$, falls $(s,s') \in RO$

d.h. die auslösenden Ereignisse der alten Stellenübergänge aus RO bleiben erhalten.

$\varepsilon_{RO'}(s,s') = \varepsilon_{RO}(s,orig(s'))$, falls $s' \in Sneu$ und $(s,orig(s')) \in RO$

d.h. es bleiben ebenso die Auslöseereignisse bei Übergängen von alten Stellen zu neuen Stellen dieselben.

$\varepsilon_{RO'}(s,s') = \{ (s, \tau(evar), \omega e(evar)), evar \in \varepsilon_{RO}(orig(s),orig(s'))\}$,
falls $s \in Sneu$ und $(orig(s),orig(s')) \in RO$
d.h. bei Übergängen von neuen Stellen zu neuen oder alten Stellen muss die jeweilige Exemplarstelle gegen die neue Stelle ausgetauscht werden.

Entsprechendes gilt bei $\varepsilon_{RC'}$:
$\varepsilon_{RC'}(s,s') = \varepsilon_{RC}(s,s')$, falls $(s,s') \in RC$
$\varepsilon_{RC'}(s,s') = \varepsilon_{RC}(s,orig(s'))$, falls $s' \in Sneu$ und $(s,orig(s')) \in RC$
$\varepsilon_{RC'}(s,s') = \{ (s, \tau(evar), \omega e(evar)), evar \in \varepsilon_{RC}(orig(s),orig(s'))\}$,
falls $s \in Sneu$ und $(orig(s),orig(s')) \in RC$

SStart':

$SStart' = SStart \cup SStartneu$, mit $SStartneu = \{ s, s \in SSubneu$ und $orig(s) \in SStart \}$

d.h. die Menge der Startstellen wird um die neuen Stellen erweitert, deren entsprechende Exemplarstellen Startstellen der Vorgängerkonfiguration sind.

Vars':

Die Variationen bleiben beim Übergang von einer Konfiguration zu einer anderen im Falle einer Kardinalitätsvariation bis auf eine eventuelle Erweiterung der jeweiligen Ereignismengen VarEvents erhalten.

Es gilt: var' = (VarEvents', VarS, VarAspects, VarType, varFunc) $\in$ Vars' $\Leftrightarrow$
var = (VarEvents, VarS, VarAspects, VarType, varFunc) $\in$ Vars'
mit VarEvents' = VarEvents $\cup$ VarEventsneu
und VarEventsneu = { (sneu, τ(evar), ωe(evar))}, mit evar $\in$ VarEvents, σ(evar) = orig(sneu)) und sneu $\in$ SSubneu}

d.h. eine Ereignismenge VarEvents' wird jeweils um die Ereignisse erweitert, die sich an neuen Stellen ereignen können, für die exemplarisch ein entsprechendes Ereignis in VarEvents an einer relevanten Exemplarstelle definiert ist.

ωS':
ωS' ist die Erweiterung von ωS bezüglich der neuen Stellen
ωS'(s) = ωS(s) für s $\in$ S'\SSubneu
ωS'(s) = ωS(orig(s)) für s $\in$ SSubneu

q': q' = q
Durch die Funktion variation wollen wir die Menge der geöffneten Stellen, die durch q bzw. q' beschrieben wird, nicht verändern. Dies geschieht erst durch Anwendung der Funktion δcore auf die durch variation veränderte Konfiguration mConfig'.
) Ende Definition mConfig' in variation, falls e aus VarEvents für ein var aus Vars
) Ende Definition Funktion variation