Integrierte Erstellung von Aufgabenmodell und Dialogspezifikation interaktiver Benutzungsschnittstellen

Khatoun Shahrbabaki
Universität - GH Paderborn
Fachbereich Mathematik/Informatik
D-33098 Paderborn
Tel.: 05251 60-3073
cat@uni-paderborn.de

Gerd Szwillus
Universität - GH Paderborn
Fachbereich Mathematik/Informatik
D-33098 Paderborn
Tel.: 05251 60-2077
szwillus@uni-paderborn.de

Zusammenfassung

Benutzerorientierter Entwurf interaktiver Systeme besteht aus der Iteration einzelner in sich abgeschlossener Phasen, wie Aufgabenanalyse, -modellierung und -spezifikation, die durch Beschreibungsnotationen unterstützt werden, die auf die jeweiligen Ansprüche zugeschnitten sind. Da die Aussagekraft dieser Notationen meistens auf die einzelnen Phasen beschränkt bleibt, gehen vielfach Informationen an den Übergängen zwischen den einzelnen Phasen verloren. Entwurfsentscheidungen können aus diesem Grund von anderen Entwicklungsphasen nicht nachvollzogen werden. In diesem Artikel stellen wir eine Kombination von zwei Notationen (MAD und DSN) vor, die durch ihre Struktur nicht nur Entwicklungsentscheidungen zurückverfolgen lassen, sondern dank ihrer Verflechtung eine konsistente und korrekte Übertragung der Entwurfsentscheidungen gewährleisten.

1 Einleitung

Viele Methoden für den Entwurf von Benutzungsschnittstellen enthalten die zyklischen Phasen Aufgabenanalyse, Aufgabenmodellierung, Dialogspezifikation und Prototyping. Um eine Problemstellung formal spezifizieren zu können und die Erkenntnisse des jeweiligen Entwurfs festzuhalten, sind verschiedene Notationen mit Betonung unterschiedlicher Aspekte entwickelt worden. Notationen wie **UAN** (Hartson et al 1990 [6]), **GOMS** (Card et al 1983 [3]), setzen den Schwerpunkt mehr auf Beschreibung des Benutzerverhaltens, während **PPS** (Olsen 1990 [8]) und **DSN** (Curry und Monk 1991 [2]) das dynamische Verhalten einer Applikation beschreiben. Andere Notationen wie **MAD** (Sebillotte 1992 [10]), und Green's Notation (Green 1981 [4]) modellieren nicht die Interaktionen des Benutzers, sondern formalisieren die Aufgabenstruktur. Da diese Notationen für spezielle Aspekte einzelner Phasen entworfen sind, bereiten die Übergänge zwischen den

verschiedenen Entwicklungsphasen und ihren dezidierten Methoden die meisten Probleme im Hinblick auf Korrektheit und Konsistenz.

In einem Beispielprojekt (Shahrbabaki 1993 [11]) haben wir untersucht, wie zwei, originär nicht "füreinander" entwickelte Methoden (nämlich MAD bei der Aufgabenmodellierung und DSN bei der Dialogspezifikation), elegant und konzeptionell klar miteinander verknüpft werden können, wodurch ein fließender Übergang von einer benutzerorientierten Sicht auf die Aufgaben zu einer systemorientierten Sicht auf Kontrollaspekte der Benutzungsschnittstelle ermöglicht wird. Als algorithmisch einfaches Problem mit hinreichend komplexer Benutzungsschnittstelle wurde ein **elektronischer Terminkalender** gewählt.

In dem vorliegenden Papier betrachten wir zunächst die Aufgabenmodellierung mit MAD und ihre Anwendung in unserem Beispielprojekt. Anschließend führen wir kurz in die Dialogspezifikation mit DSN ein und zeigen an unserem Beispielprojekt auf, wie diese Methode bei der Erstellung einer Spezifikation durch eine vorangegangene Aufgabenmodellierung mit MAD effizient unterstützt wird. Zum Schluß diskutieren wir die aufgetretenen Synergieeffekte der beiden Notationen, indem wir von unserem konkreten Anwendungsbeispiel abstrahieren.

2 Aufgabenmodell

In vielen Fällen ist die Erstellung eines Aufgabenmodells eine naheliegende Vorstufe der Entwicklung einer interaktiven Applikation. Typischerweise beschreibt sie in einer semi-formalen Notation die Aufgaben, die der zukünftige Benutzer an oder mit einem System durchführen möchte. Benutzeranforderungen und andere applikationsrelevante Informationen finden dementsprechend dort ihren Niederschlag. Zusätzlich beschreiben einige Modelle die Aufgabe in Szenarien (Curry und Monk 1991 [2]), (Robertson 1995 [9]). Die Verwendung dieser Szenarien ist sinnvoll, da die Entwickler eine realistische Vorstellung des späteren Einsatzumfelds gewinnen können. Dies und die Aufgabenanalyse bilden die Basis eines benutzerzentrierten Entwurfs. Als alleinige Beschreibungsnotation kommen sie wegen ihrer inhärenten Ungenauigkeit nicht in Frage, so daß zusätzliche Notationen für eine formale Spezifikation notwendig werden. In unserem Projekt wurden Szenarien nur als Vorstufe der Modellierung verwendet.

Andere Ansätzen wie **UAN** (Hartson et al 1990 [6]), Command Language Grammar **CLG** (Moran 1981 [7]) integrieren die Benutzeraktionen bereits in das Aufgabenmodell. Damit zieht eine Änderung des Modells meistens auch eine Änderung der Interaktionen nach sich, was die Bearbeitung des Aufgabenmodells

erschwert, da beide Aspekte zu eng miteinander verknüpft behandelt werden. Außerdem kann mit diesen Notationen nicht festgestellt werden, in welchem Zustand das System sich gerade befindet. Wir schlagen hier die Verwendung der **MAD**-Notation (Sebillotte 1992 [10]) vor, die eine hierarchische Modellierung der Benutzeraufgaben erlaubt. In unserem Ansatz wurde diese Notation teilweise erweitert und ergänzt.

Im folgenden werden wir zunächst die Grundzüge von MAD erläutern, danach führen wir als Anwendungsbeispiel einen **elektronischen Terminkalender** ein und stellen beispielhaft einige wesentliche Aspekte des Entwicklungsprozesses dar.

2.1 MAD-Methode

MAD ist eine graphische Notation, die ähnlich wie Green's Methode (Green 1981 [4]) die Reihenfolge der Aufgaben mit Vor- und Nachbedingungen festlegt, jedoch zusätzlich mit einer graphischen Darstellung der Aufgabenhierarchie unterstützt wird. Durch ihre graphische Repräsentation und Einführung einer Struktur ist MAD anschaulich und leicht nachvollziehbar. Den Kern des Ansatzes bildet ein Hierarchieplan von zu erledigenden Aufgaben. Jede Aufgabe wird solange in Teilaufgaben zerlegt, bis keine weitere Zerlegung mehr möglich ist. Dadurch entsteht ein hierarchischer Baum mit Knoten als Teilaufgaben, Blättern als atomaren Zielen und Kanten, die die Zusammensetzung einzelner Teilaufgaben *(Task Structure)* widerspiegeln.

Für jeden Baumknoten der MAD-Hierarchie wird ein Schema mit Werten gefüllt, welches Felder besitzt, in denen verschiedene relevante Daten vermerkt werden (siehe Tabelle 1).

- TASK: Bezeichnung der jeweiligen Teilaufgabe
- Lfd. Nr.: Hierarchische Numerierung der Felder
- GOAL: Ziele, die mit dieser Teilaufgabe verfolgt werden
- PRECONDITIONS: Vorbedingungen, die zur Ausführung der Aufgabe erfüllt sein müssen.

TASK:	Lfd. Nr.:
INITIAL STATE:	FINAL STATE:
GOAL:	
PRECONDITIONS:	POSTCONDITIONS:
UPPER LEVEL:	COMPOSITE TASK & STRUCTURE:
ELEMENTARY TASK:	

Tabelle 1: Datenfelder eines Aufgabenobjekts im MAD-Formalismus

- POSTCONDITIONS: Nachbedingungen, welche erst nach dem Erledigen der Aufgabe zutreffen.

- TASK STRUCTURE: Es werden sechs unterschiedliche Strukturierungsarten von MAD zur Verfügung gestellt (LOOP, OPT, OR, PAR, SEQ, SIM). Sie geben an in welcher Reihenfolge die in diesem Knoten involvierten Teilaufgaben ausgeführt werden müssen, dabei haben die im Baum höher angesiedelten Knoten bei der Ausführung eine höhere Priorität. Dies stellt eine unserer Erweiterungen dar. In der ursprünglichen MAD-Version kann eine Aufgabe nur in Teilaufgaben zerlegt werden, die in gleicher Beziehung zueinander stehen. Deshalb ist auch keine Festlegung einer Ausführungsordnung (mit Ausnahme von sequentiellen Aufgaben) notwendig. Wir erlauben in unserer Erweiterung die Entstehung inhomogener Aufgabenstrukturen, die eine definierte Ordnung erforderlich machen.

  - Wiederholende Ausführung (**LOOP**)
  - Optionale Ausführung (**OPT**)
  - Gegenseitiger Ausschluß (**OR**)
  - Beliebige Reihenfolge (**PAR**)
  - Sequentielle Reihenfolge (**SEQ**)
  - Gleichzeitige Ausführung (**SIM**)

- UPPER LEVEL: In diesem Feld wird auf die nächsthöhere Baumebene verwiesen.

- COMPOSITE TASK: Komponenten der zusammengesetzten Teilaufgaben.
- ELEMENTARY TASK: Wenn dieser Knoten ein Blatt darstellt, wird hier 'Ja', sonst 'Nein' eingetragen.

2.2 Das Aufgabenmodell des Terminkalenders

Das gesamte Aufgabenmodell unserer Beispielanwendung umfaßt eine Vielzahl von MAD-Diagrammen. Als repräsentativen Ausschnitt des Modells stellen wir in diesem Papier die Termineingabe vor. Dabei verwenden wir zur optischen Präsentation und zum leichteren Verständnis einen Layoutvorschlag (Abbildung 1), der nach vollständiger Aufgabenmodellierung des Terminkalenders (Shahrbabaki 1993 [11]) und entsprechender Integration der Teilspezifikationen resultierte.

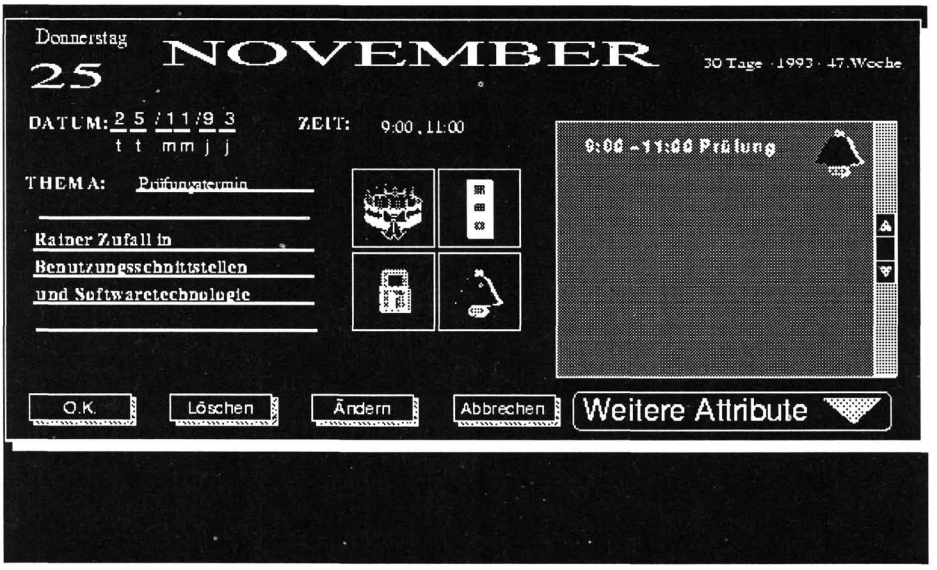


Abbildung 1: Ein mögliches Layout für Termineintragung nach dem Aufgabenmodell

Zu einem vollständigen Termin gehören die Eingaben Datum, Zeit und Thema (Beschreibung des Termins). Ein Aufgabenmodell für die Termineingabe repräsentiert die Abbildung 2 .

Wie man anhand des Modells (Abbildung 2) erkennt, können Datum, Zeit und Thema in beliebiger Reihenfolge angegeben werden (PAR-Konstrukt). Dagegen ist die Eingabe von Attributen (wie z.B. das Einstellen eines Alarms) nicht zwingend

(OPT). Das Quittieren erfolgt als letzte Aktion und steht deshalb als sequentiell (SEQ) zu allen vorherigen Aktionen; die Reihenfolge der Teilaufgaben von oben nach unten ist also, anders als im ursprünglichen MAD-Modell, signifikant. Wir werden später in der Dialogmodellierung sehen, daß eine im Sinne dieser Vorgaben "falsche" Benutzereingabe (z.B. Eintragung eines Termins ohne Datum) zu Fehlermeldungen des Systems führt.

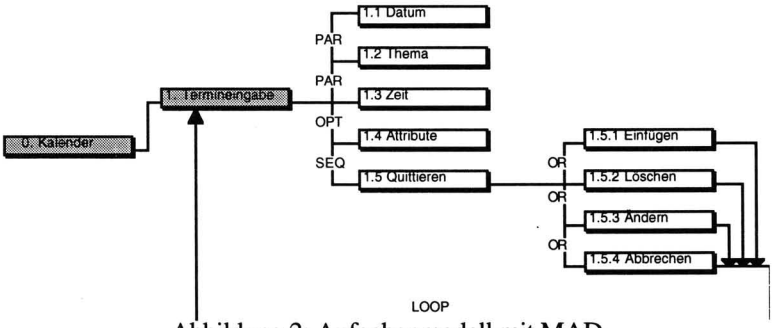


Abbildung 2: Aufgabenmodell mit MAD

3 Dialogspezifikation

Angesichts der Komplexität der Interaktionen zwischen Mensch und Computer benötigen die Entwickler eine Beschreibungsmethode zur Festlegung der Benutzereingabe- und Verzweigungsmöglichkeiten die im Laufe eines Dialogs mit dem System entstehen. In diesem Sinne versteht sich eine formale Dialogspezifikation als eine präzise, eindeutige Beschreibung der Interaktionen zwischen Mensch und Computer. Eine derartige Beschreibung muß zwar einerseits systemorientiert sein, andererseits sollte sie den Entwickler nicht zwingen, alle oder zu viele Details der Interaktionen festlegen bzw. überhaupt betrachten zu müssen.

Es gibt Dialogspezifikationen in verschiedenen Detaillierungsgraden, denen allerdings durchweg eine Betonung der Beschreibung aus der Sicht des Systems - nicht des Benutzers - gemeinsam ist. Hat man vorher, bei der Aufgabenmodellierung die Denk- und Entscheidungsabläufe des Benutzers beschrieben, so behandelt die Dialogspezifikation nun die Eingabemöglichkeiten und die resultierenden Zustandsänderungen des Systems.

Diese Informationen lassen sich nicht mehr mit der MAD-Notation erfassen, da sie nicht Teil eines Aufgabenmodells sind. Wir setzen nun die DSN-Methode ein, um die aus der Aufgabenmodellierung resultierenden Eigenschaften des Dialogs zu

beschreiben. DSN, welches unabhängig von MAD entwickelt wurde, eignet sich sehr gut als Anschluß an eine MAD-Analyse.

Mit Hilfe von DSN spezifizieren wir den Dialog, der in unserer Beispielmodellierung bei der Eingabe eines Termins entsteht.

3.1 Dialogspezifikation mit DSN

Die **Dialogue Specification Notation** (DSN) wurde von Curry und Monk entwickelt (Curry und Monk 1991, [2]). DSN modelliert einen Dialog durch ein Produktionssystem, dessen Hauptkomponenten aus dem **Propositional Production System** (**PPS**), einer von Olsen (Olsen 1990 [8]) entwickelten Notation, übernommen sind.

In diesem Produktionssystem werden die gewünschten "Ziele" erreicht, wenn sich durch Benutzeraktionen in einem bestimmten "Kontext" Produktionsregeln ableiten lassen. **Zustandsvektor**, **Ereignisse** und **Regeln** bilden die Hauptkomponenten der DSN, die im folgenden erklärt werden:

- Zustandsvektor

  Der Kontext einer Regel ist mit Bedingungen gleichzusetzen, die bei der Auslösung einer Produktionsregel erfüllt sein müssen. Diese Bedingungen spiegeln sich in den Systemzuständen wider.

  Sämtliche Zustände eines Systems sind mit XOR* in verschiedenen Mengen *(State-Space Conditions)* gruppiert. Der Zustandsvektor setzt sich aus genau einem Zustand aus jedem dieser Mengen zusammen; mathematisch wird also ein Kreuzprodukt gebildet. Elemente des Zustandsvektors werden als "aktive" Zustände bezeichnet. Somit bilden die aktiven Zustände des Systems (oder der Zustandsvektor) den zum "Feuern" einer Regel erforderlichen Kontext. Alle Zustände werden mit dem Präfix # von anderen Komponenten der Notation unterschieden.

---

*) Diese Relation drückt hierbei aus, daß sich das System in genau einem Zustand aus dem Vektor befindet, der mit aktivem Zustand bezeichnet wird. Beispielsweise werden die Zustände "Mauseingabe" und "Tastatur" in einem Vektor zusammengefaßt, wenn das System nicht gleichzeitig von Maus und Tastatur eine Eingabe erhalten sollte.

- Ereignisse

  In DSN wird zwischen den vom Benutzer und den vom System ausgelösten Ereignissen differenziert. Erstere stellen die Benutzeraktionen dar und werden mit dem Präfix "↓" eingeleitet. Ähnlich wie die Systemzustände werden sie in einem Vektor *User Events* zusammengefaßt. Letztere repräsentieren die Rückmeldungen des Systems und werden mit einem nach oben gerichteten Pfeil (↑) gekennzeichnet. Sie werden in dem Vektor *Application Events* zusammengefaßt.

- Regeln

  Man kann die Regeln abstrakt als eine Abbildung sehen, die das zu bearbeitende Problem auf eine mögliche Lösung abbildet.

  Auf der linken Seite einer Produktion werden die Parameter aufgeführt, die als Vorbedingung einer möglichen Lösung des Problems gelten sollen. Darunter fallen aktive Zustände, Benutzeraktionen und Systemmeldungen. Auf der rechten Seite einer Regel werden Konsequenzen des Systems in Form von Zustandsänderungen eingetragen.

Zu DSN gehört ein Simulationstool **DDT** (Dialogue Design Tool), das zu jedem gegebenen Zeitpunkt die Regel, die "gefeuert" werden muß, und die mögliche Benutzerangaben anzeigt. Außerdem kann der Entwickler Skizzen anfertigen, die er einzelnen Regeln zuordnet, welche die in der Regel beschriebene Situation optisch repräsentiert. Damit kann der Entwickler sehr grob einzelne graphische Aspekte der entworfenen Applikation schon im Vorfeld einer Weiterentwicklung testen. Die gebotenen Möglichkeiten des existierenden Werkzeugs sind jedoch noch weit von einem echten Prototyping entfernt.

3.2 Die Dialogspezifikation des Terminkalenders

Die Spezifikation der Teilaufgaben beginnt mit der Beschreibung der Benutzeraktionen. Da zu einem Zeitpunkt immer nur ein Ereignis vom Anwender ausgelöst werden kann, ist ein gleichzeitiges Eintreten verschiedener Benutzereingaben ausgeschlossen. Aus diesem Grund werden alle Eingaben in einem Vektor *User-Event* (F1) zusammengefaßt. Der Vektor F1 kann im Laufe der Entwicklung immer wieder um neue Eingabemöglichkeiten erweitert werden. Des weiteren können Benutzereingaben erst "grob" und später feiner granuliert werden. Beispielsweise kann die Eingabe eines Datums zunächst abstrakt als (↓*Datum*) bezeichnet werden, um später auf eine speziellere Eingabe, wie z.B. Tag-,

Monat- und Jahreingabe verfeinert zu werden. Wir bleiben vorerst bei folgenden abstrakten Benutzereingaben:

USER EVENTS:
F1 (↓datum, ↓zeit, ↓thema, ↓ok, ↓abbrechen, ↓löschen, ↓ändern)

Die möglichen Reaktionen der Applikation werden in einem Vektor *System-Feedback-Event* (F2) zusammengefaßt. Auch diese können zunächst abstrakt und später detailliert formuliert werden.

SYSTEM FEEDBACK EVENTS:
F2 (↑OK, ↑ERROR)

Betrachten wir nun das in Abschnitt 2.2 entworfene Aufgabenmodell. Die Dialogspezifikation in DSN läßt sich direkt aus den in MAD modellierten Teilaufgaben ableiten. Dort wurde die Teilaufgabe **Termineingabe** zunächst in (**Datum, Thema, Zeit, Attribute** und **Quittieren**) zerlegt. Die ersten drei Eingaben wurden mit der Option (*PAR*) zusammengefaßt, die eine Ausführung dieser Teilaufgaben in beliebiger Reihenfolge erlaubt (siehe Abbildung 3).

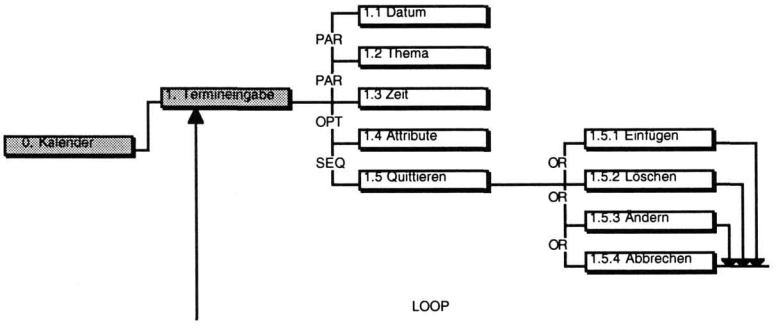


Abbildung 3: Teilaufgaben der Termineingabe

Wir zeigen nun am Beispiel auf, wie eine derartige Teilstruktur in entsprechende DSN-Konstrukte umgesetzt werden kann.

Als erstes werden für jede einzelne Teilaufgabe, die eine "*PAR*-Struktur" besitzt, zwei komplementäre Zustände erzeugt. Im Falle der Teilaufgabe *Datum* sind dies die

Zustände #Datum und #KeinDatum, die aufgrund gegenseitigen Ausschlußes in einem Zustandsvektor (F4) zusammengefaßt werden können.

F4 (#Datum, #KeinDatum)

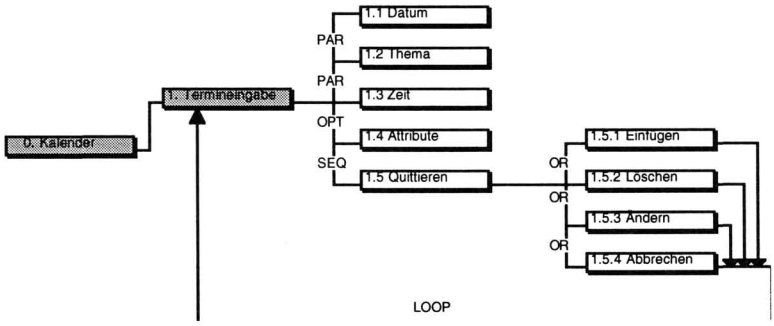


Abbildung 4: Teilaufgaben des Quittierens

Entsprechend verfahren wir mit weiteren Zuständen, die mit *SIM* oder *OPT* strukturiert sind.

Die Teilaufgaben (**Einfügen, Löschen, Ändern** und **Abbrechen**) schließen eine gleichzeitige Selektion vom Benutzer aus, weshalb sie im Aufgabenmodell mit der Option *OR* verbunden sind (siehe Abbildung 4). Deshalb können sie nicht auf der gleichen Art und Weise in den Dialogspezifikation einfließen, wie die vorher behandelten Teilaufgaben. Da solche Zustände sich per Definition ausschließen, brauchen keine komplementären Zustände eingeführt werden. Sie können direkt in einem Zustandsvektor zusammengefaßt werden.

F8 (#Einfügen, #Löschen, #Ändern, #Abbrechen)

Es resultieren nun folgende Zustandsvektoren:

SYSTEM STATES

F3 (#Eingabe, #KeineEingabe)

F4 (#Datum, #KeinDatum)

F5 (#Thema, #KeinThema)

F6 (#Zeit, #KeineZeit)

F7 (#Attr, #KeinAttr)

F8 (#Einfügen, #Löschen, #Ändern, #Abbrechen)

Die Produktionsregeln beschreiben nun die bei der Anwendung erzielten Reaktionen bzw. das dynamische Verhalten der Applikation. Jede Regel ist mit einer Nummer gekennzeichnet, die kanonisch aus der Numerierung des entsprechenden Aufgabenobjektes hervorgeht. Wir wollen nun anhand der folgenden Abbildung den Dialogverlauf nachvollziehen.

```
Start ( #KeineEingabe, #KeinDatum, #KeineZeit, #KeinThema, #Abbrechen )
1.1. #KeinDatum ↓datum Æ #Datum
1.2. #KeinThema ↓thema Æ #Thema
1.3. #KeineZeit ↓zeit Æ #Zeit
1.4. #KeinAttr ↓att Æ #Attr
1.5.0 #KeineEingabe #Zeit #Datum #Thema ↑OK Æ #Eingabe
1.5.1 #Eingabe ↓ok Æ
        #Einfügen #KeineEingabe #KeinDatum #KeinThema #KeineZeit #KeinAttr
1.5.2 #Eingabe ↓löschen Æ
        #KeineEingabe #Löschen #KeinDatum #KeinThema #KeineZeit #KeinAttr
1.5.3 #Eingabe ↓ändern Æ
        #KeineEingabe #Ändern #KeinDatum #KeinThema #KeineZeit #KeinAttr
1.5.4 #Abbrechen ↓abbrechen Æ
        #KeineEingabe #KeinDatum #KeinThema #KeineZeit #KeinAttr
```

Die Regeln 1.1 bis 1.4 zeigen, daß der Benutzer in beliebiger Reihenfolge Zeit, Thema, Datum angeben kann. Erst wenn alle drei Angaben eingetragen sind, erlaubt das System die Eintragung (Regel 1.5.0). An dieser Stelle kann man durch entsprechende Regeln die Systemreaktionen um sinnvolle Fehlermeldungen erweitern. Beispielsweise können wir die Regeln 1.5.0.1, 1.5.0.2, 1.5.0.3 einfügen, die auf unvollständige Eingaben hinweisen und keine Eintragung zulassen.

```
1.5.0.1 #KeineEingabe #KeineZeit #Datum #Thema ↑ERROR Æ
     #KeineEingabe {Zeit fehlt}
1.5.0.2 #KeineEingabe #Zeit #KeinDatum #Thema ↑ERROR Æ
     #KeineEingabe {Datum fehlt}
1.5.0.3 #KeineEingabe #Zeit #Datum #KeinThema ↑ERROR Æ
     #KeineEingabe {Thema fehlt}
```

Wir können in diesem Papier nicht auf alle Einzelheiten von DSN und des Anwendungsbeispiels eingehen und beschränken uns daher auf die Aussage, daß wir die oben aufgeführten Regeln mit DDT animiert und damit ihr korrektes Verhalten verifizieren konnten. Zur einer tiefergehenden Betrachtung verweisen wir auf die Arbeiten von Monk und Curry (Curry und Monk 1991 [2]).

3.3 Allgemeine Transformationsregel zwischen MAD und DSN

Beispielhaft haben wir eine Übertragung von MAD nach DSN im vorigen Abschnitt vollzogen. An dieser Stelle geben wir einen kurzen Einblick in die formalen Transformationsregeln, die wir bei der Übertragung von MAD nach DSN eingesetzt haben. Auf eine ausführlichere Diskussion der Regeln verzichten wir aus Platzgründen, da dies eine detaillierte Erklärung der DSN erforderlich machen würde. Der Beitrag dieses Abschnitts zielt damit nur auf die Vermittlung der Grundideen der vereinfachten Transformationsregeln und erhebt keinen Anspruch auf Vollständigkeit. Er soll es dem Leser ermöglichen, eine bessere Vorstellung von der schematisch vorgenommenen Übertragung zwischen den Notationen zu gewinnen. Wir behandeln nacheinander die Aspekte Ereignisse (*User Events & Application Events*), Zustände und Produktionsregeln.

- **Ereignisse**, also Eingaben des Benutzers und Reaktionen des Systems, werden in einer MAD-Spezifikation syntaktisch nicht betrachtet. Hier kann eine Transformation nicht schematisch vorgehen - vielmehr besteht hier Entwurfsspielraum bzw. -bedarf bei der endgültigen Dialogmodellierung. Dies sind Sachentscheidungen, die in jedem Transformationsfall individuell und aus der Semantik heraus festgelegt werden müssen.

- **Zustände** werden in den Transformationsregeln schematisch eingeführt, um die Semantik der in MAD festgelegten Zeitabläufe im DSN-Modell nachzuvollziehen. Wir ordnen Aufgaben DSN-Zustände zu, die den Beginn bzw. den Abschluß der Durchführung einer Aufgabe repräsentieren; abhängig vom gewählten MAD-Konstruktor gehen wir verschieden vor:

  - *OR* oder *SEQ* -verknüpfte Aufgaben $a_1,...,a_n$ werden durch einen Vektor von sich gegenseitig ausschließenden Zuständen $(\#a_1,...,\#a_n)$ repräsentiert.

  - *OPT*, *SIM* oder *PAR* -verknüpfte Aufgaben $a_1,...,a_n$ werden durch n Vektoren sich jeweils paarweise ausschließender Zustände $(\#a_1, \#\overline{a_1}),\ldots,$ $(\#a_n, \#\overline{a_n})$ dargestellt.

- *LOOP* -verknüpfte Aufgaben wirken sich lediglich in den Produktionsregeln aus, indem der Kontext wiederhergestellt wird, der vor Ausführung der Schleife gültig war.

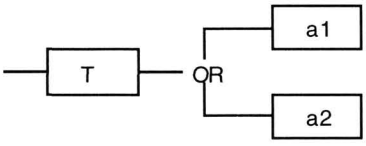
- **Produktionsregeln** stellen im DSN-Modell die Beschreibung der Übergänge zwischen Zuständen sicher. Dabei lassen sich die beteiligten Zustandsmengen schematisch ableiten, um dem MAD-Modell zu genügen; die Festlegung von Benutzereingaben und Systemreaktionen, die in den Produktionsregeln ausgewertet werden, ist eine Entwurfsentscheidung bei der Dialogmodellierung. Die allgemeinen Transformationsregeln, die wir hier behandeln wollen, können daher nur die Zustandstransformationen betreffen und nicht die auslösenden Ereignisse behandeln.

  Bei der Übertragung des Aufgabenmodells zu Produktionsregeln spielt die Ordnung der MAD-Knoten (graphisch von oben nach unten), die wir in das Modell eingeführt haben, eine entscheidende Rolle. Dadurch werden die inhomogenen Teilstrukturen möglich, die das ursprüngliche MAD-Konzept nicht enthielt. Gemäß dieser Ordnung wird zuerst die Beziehung zwischen den zwei jeweils ersten Knoten ausgewertet; dann werden diese mit dem jeweils nächsten Knoten verknüpft.

  In den folgenden Schemata haben wir aus Gründen der einfacheren Darstellung Benutzeraktionen als das Signal gewählt, welches jeweils das Ende einer Sequenz markiert. Im konkreten DSN-Modell kann dies jedoch sowohl eine vom System ausgelöste Reaktion als auch eine Aktivierung eines Zustands sein. Außerdem betrachten wir der Einfachheit halber lediglich Paare von Aufgaben.

  Seien im folgenden also T, a_1 und a_2 und Knoten eines MAD-Modells. Wir wollen einige der MAD-Konstruktoren anführen, bei denen die formale Transformation besonders gut nachzuvollziehen ist. Es gelte also die Beziehung $T = a_1 \oplus a_2$, wobei $\oplus \in \{OR, PAR, SIM\}$

  ($\oplus$ = OR): Wir erzeugen die Zustandsvektoren F0 und F1 und die Produktionsregeln R1 und R2.



$F0 = (\#T, \#\overline{T})$

$F1 = (\#a_1, \#a_2)$

R1) $\#T\#a_1 \downarrow a_1 \rightarrow \#T\#a_1$

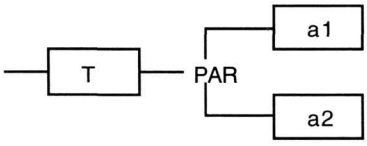
R2) $\#T\#a_2 \downarrow a_2 \rightarrow \#T\#a_2$

$\#\overline{T}$ repräsentiert den Zustand, "T ist durchzuführen".

$\#T$ entspricht dem Zustand "T ist durchgeführt".

Das Eintreffen der Benutzereingabe a_1 repräsentiert das Ereignis, daß die Aufgabe a_1 abgeschlossen ist; analog ist a_2 zu verstehen.

($\oplus$ = PAR oder SIM): Hier werden F0, F1 und F2 erzeugt, um zu symbolisieren, inwieweit die Gesamtaufgabe T, und die Teilaufgaben a_1 und a_2 initiiert bzw. abgeschlossen sind. Es ergeben sich folgende Zustände und Regeln:



$F0 = (\#T, \#\overline{T})$

$F1 = (\#a_1, \#\overline{a_1})$

$F2 = (\#a_2, \#\overline{a_2})$

R1) $\#\overline{T}\ \#\overline{a}_1\#\overline{a}_2 \downarrow a_1 \rightarrow \#T\#a_1\#\overline{a}_2$

R2) $\#\overline{T}\ \#\overline{a}_1\#\overline{a}_2 \downarrow a_2 \rightarrow \#T\#\overline{a}_1\#a_2$

R3) $\#\overline{T}\ \#\overline{a}_1\#a_2 \downarrow a_1 \rightarrow \#T\#a_1\#a_2$

R4) $\#\overline{T}\#a_1\ \#\overline{a}_2 \downarrow a_2 \rightarrow \#T\#a_1\#a_2$

3.4 Integration von Aufgabenmodell und Dialogspezifikation

In praktischen allen Bereichen der Softwareentwicklung hat sich die hierarchische Dekomposition als wesentliches Strukturierungsprinzip sowohl für die erzeugten Artefakte, als auch für die Aufgliederung der Arbeitsphasen etabliert. Aus der Theorie komplexer Systeme (wie z.B. beschrieben in (Booch 1991 [1])) ist bekannt, daß dieses Prinzip nicht künstlich aufgesetzt ist, sondern vielmehr als den

verschiedenen Problemen inhärent angesehen werden kann. Daher reflektieren die Notationen zur Beschreibung von Systemen typischerweise diese hierarchischen Strukturen, was ein Vorgehen nach dem *Divide-&-Conquer*-Prinzip erlaubt, bzw. sogar nahelegt. Im Bereich der Entwicklung von interaktiven Systemen zeigt sich das z.B. an der Baumstrukturierung von GOMS-Spezifikationen, welche sich nach Auffassung der Erfinder von GOMS an den postulierten hierarchischen Speicher- und Lernstrukturen des menschlichen Handelns orientieren.

Auch die MAD-Methode setzt daher naheliegend Hierarchien ein, um komplexe Aufgabenstrukturen in einfachere Teile zu zerlegen. Die nachfolgende Phase der Dialogspezifikation, also der Modellierung von Benutzereingaben, Ausgaben an den Benutzer, Unterbrechungen und Zustandsübergängen, manchmal auch bezeichnet als das sogenannte *Kontrollmodell*, bricht jedoch im Kern mit der hierarchischen Betrachtungsweise: Das kanonische Beschreibungsmittel, der endliche Automat, entspricht in seinen Strukturierungsmöglichkeiten gerichteten Graphen und nur in Teilaspekten Baumstrukturen. Man erkennt dies unmittelbar an der wohlbekannten Veranschaulichung endlicher Automaten durch Graphen mit Kreisen als Zustandsrepräsentationen und Pfeilen als gerichteten Übergängen dazwischen*.

Ein wesentlicher Anteil der Probleme beim Übergang von Aufgabenmodellierung zu Dialogspezifikation besteht in diesem Wechsel des Beschreibungskalküls: Im hierarchischen Aufgabenmodell finden sich nicht explizit Zustände und Übergänge dazwischen - im automatenorientierten Dialogmodell sind die Zustände im wesentlichen flach nebeneinander angeordnet und besitzen keine hierarchische Struktur. Die Kombination der Methoden MAD und DSN erlaubte es nun, durch leichte Modifikationen diesen Übergang systematisch durchzuführen und die Sicherstellung der Konsistenz zwischen diesen beiden Phasen deutlich zu erleichtern.

Eine wichtige Voraussetzung für die Durchführung der systematischen Transformation war eine Modifikation der MAD-Methode zur reicheren Gestaltung der Aufgabenstrukturen. MAD, wie in (Sebillotte 1992 [10]) definiert, erlaubt nur eine homogene Strukturierung der Teilaufgaben: alle Unteraufgaben einer Aufgabe stehen in einer einzigen der angebbaren Beziehungen (PAR, SEQ, ...). In der praktischen Durchführung zeigt sich schnell, daß diese Festlegung sehr unflexibel ist und dazu führt, daß aus rein technischen Gründen in der Aufgabenmodellierung

*) Statecharts (Harel 1988 [5]) kombinieren diese Grapheigenschaft in eleganter Weise mit Hierarchien - das beherrschende Stilelement bei der Spezifikation von Kontrolle in Benutzungsschnittstelle ist jedoch der unstrukturierte Übergang von einem Zustand zum nächsten.

Zwischenknoten eingeführt werden müssen, um Teilaufgaben homogen zu strukturieren. Resultat sind unangemessene Strukturierungen, die zunehmend unlesbar werden, und damit ihren Anspruch der Semiformalisierung umgangssprachlich formulierter Aufgabenstrukturen nicht mehr erfüllen.

In der modifizierten Fassung kann man Teilaufgaben erstens zeitlich anordnen, zweitens jede einzelne Teilaufgabe mit dem angemessenen Konstrukt versehen, und drittens neben den von MAD gegebenen Konstruktoren auch optionale Teilaufgaben mit OPT spezifizieren. In der graphischen Darstellung repräsentiert die Reihenfolge von "oben nach unten" eine entsprechende zeitliche Reihenfolge und die Konstruktoren an den Kanten werden an die Eingänge in die Teilaufgaben, statt an den Ausgang bei der übergeordneten Aufgabe notiert. Diese moderaten Änderungen erlauben ein wesentlich "flüssigeres" Verwenden der Notation - viel weniger als bei der urprünglichen Methode muß man über die Notwendigkeiten der Notation nachdenken. Gleichzeitig erreicht man mit den neuen Mitteln eine Anreicherung des Aufgabenmodells um Anfänge einer Verhaltensbeschreibung, die die Dialogspezifikation vorbereitet, ohne von der in dieser Phase durchzuführenden Aufgabenbeschreibung abzulenken. Es ist sogar sinnvoll denkbar, mit einer Modellierung ohne zeitliche Reihenfolgen und Konstruktoren auszugehen und diese Informationen schrittweise anzureichern.

Beginnt man nun mit der Dialogmodellierung, muß man in die Abläufe aus Benutzersicht die Eingaben an das System und die Ausgaben des Systems integrieren. Wie bereits gesagt, kommt hier zur Spezifikation das Zustandskonzept hinzu, das sich im Aufgabenmodell nur sehr indirekt bzw. überhaupt nicht widerspiegelt. Das Dialogmodell strukturiert seine Darstellung analog der Beschreibung von Zustandsübergängen und Ausgaben abhängig von den Benutzereingaben in endlichen Automaten. Diese Beschreibung muß möglichst in die Darstellung des Anwendungs eingebettet sein und es andererseits erlauben, die große Fülle von möglichen Interaktionstechniken zu beschreiben und einzusetzen. Nach unseren Erfahrungen ist DSN hierfür hervorragend geeignet, da es - ohne auf bestimmte Techniken oder Abstraktionsebenen festgelegt zu sein - erlaubt, komplexe Zustandsübergänge und Systemreaktionen zu beschreiben. Anders als die endlichen Automaten selbst, sind die Beschreibungen bei DSN strukturierter und damit kompakter, da die Kreuzproduktbildung als wesentliches Konstruktionsmerkmal verwendet wird.

Wie im vorangegangenen Abschnitt exemplarisch beschrieben, kann das gemäß unserer MAD-Variante fomulierte Aufgabenmodell sehr naheliegend und nachvollziehbar in eine DSN-Spezifikation transformiert werden. Damit hat man den Übergang von der hierarchischen zur automatenorientierten Beschreibung

durchgeführt und kann anschließend auf DSN-Ebene selbst weiter verfeinern, bis man zu den verfeinerten Interaktionstechniken kommt. Zentral hierbei ist die Einbettung der einen Beschreibungsform in die andere; überläßt man den Transformationsaufwand alleine dem Anwender von Notationen ohne eine entsprechende Einbettung anzubieten, entstehen eine Reihe von Problemen: Vergessen von Ideen aus einer Phase in der nächsten, Fehler beim Übertragen von Abläufen der einen Phase in die nächste, Verpflichtung zur Sicherstellung der Konsistenz zwischen den Phasen und mangelhafte Dokumentation von Entwurfs aus einer Phase in der darauf folgenden. Bei der Kombination von MAD und DSN kommt hinzu, daß das Werkzeug DDT eine sehr frühe Animation der DSN-Spezifikation zuläßt - damit läßt sich, bei der engen Kopplung von Aufgabenmodell und Dialogmodell, auch das Aufgabenmodell indirekt überprüfen.

Ein anderer Ansatz zur Lösung der Probleme der Übergänge zwischen Entwurfsphasen könnte in einer einheitlichen Notation für die Modellierung verschiedener Aspekte bestehen. Allerdings liegt die Betonung in verschiedenen Phasen auf derart unterschiedlichen Aspekten, daß kaum eine Notation in der Lage ist, alle diese zu umfassen. Die singulären Notationen sind ohne diesen Anspruch entwickelt wurden, was vielleicht auch darauf zurückzuführen ist, daß ein globaler Überblick auf das gesamte interaktive System durchaus nicht immer im Vordergrund stehen muß. Diese globale Erkenntnis von Notationen notiert Whitehead in seinem fundamentalen Werk über Mathematik

> *"By relieving the brain of all unnecessary work, a good notation sets it free to concentrate on more advanced problems"* (Whitehead 1958 [12])

und bringt damit zum Ausdruck, daß man Notationen aufgabengerecht ausstatten soll, damit sie den Anwender von unnötigen Details befreien und ihm erlauben, auf der "richtigen" Abstraktionsebene zu denken und Entscheidungen zu treffen. Die für die Phasen Aufgabenmodellierung und Dialogspezifikation entwickelten Notationen MAD bzw. DSN sind gut geeignet, um in ihrer jeweiligen "Welt" eingesetzt zu werden. Da man verschiedene Probleme angeht, sollen und müssen solche Notationen verschieden sein, was zu den erwähnten Problemen bei den Übergängen zwischen zwei derartigen Spezifikationen führt. Die Erfahrung, die wir in diesem Papier vermitteln wollen, besteht darin, daß gerade diese beiden Methoden auf Grund ihrer Struktur sehr gut zueinander "passen" und einen kanonischen Übergang von der Aufgabenmodellierung zur Dialogspezifikation erlauben.

4 Ausblick

Das vorliegende Papier stellt einen Versuch dar, existierende moderne Methoden der Entwicklung von Benutzungsschnittstellen miteinander zu kombinieren, um die notwendigen Phasenübergänge innerhalb des Entwicklungsprozesses zu erleichtern. Dies kann nur ein Anfang sein, da einerseits diese Methodik keineswegs in allen Fällen anwendbar sein kann, andererseits aber auch die umliegenden Phasen, insbesondere die vorangehende Aufgabenanalyse und das sich anschließende Prototyping (Simulation) in die Betrachtungen einbezogen werden müssen. Es sollte aber hier einmal der Versuch gemacht und dokumentiert werden, wie man die inhärenten Mängel von "Insellösungen" angehen kann.

Literatur

[1] Booch, G.: *Object Oriented Design with Applications*, Benjamin Cummings, Redwood City, CA 1991

[2] Curry, M. B.; Monk, A. F.: *Dialogue modelling of graphical user interfaces with a production system,* Technical Report, Department of Psychology, University of York, 1991

[3] Card, S. K.; Moran, T. P.; Newell, A: *The Psychology of Human Computer Interaction,* Erlbaum, Hillsdale, NJ, 1983

[4] Green, M.: *A methodology for the specification of graphical user interfaces,* Computer Graphics, Vol. 15, No.3, 1981

[5] Harel, D.: *On Visual Formalisms*, Communications of the ACM, Vol 31 No.5, pp 514-540, 1988

[6] Hartson, H.R.; Siochi, A.C.; Hix, D.: *The UAN: A User- Oriented Representation for Direct Manipulation Interface Designs*, ACM Transactions on Information Systems, Vol. 8, pp 181-203, 1990

[7] Moran, T.P.: *The Command language Grammar: A representation for the user interface of interactive computer systems*, International Journal of Man-Machine Studies, pp 15-51, 1981

[8] Olsen, D.R.: *Propositional Production Systems for dialogue description*, Human Factors in Computer Systems, CHI '90 Conference Proceedings, pp 57-63, 1990

[9] Robertson, S.P.: *Generating object-oriented design representations via scenario queries*, In: J. Caroll (Hrsg.) *Scenario-based design: Envisioning Work in technology and system design*, (im Druck)

[10] Sebillotte, S.: *Task Analysis and Formalization according to MAD: Hierarchical Task Analysis Method of Data Gathering and Examples of Task Descriptions,* Technical Report, 1992

[11] Shahrbabaki, K.: *Beiträge zu einer partizipativen Entwurfsmethodik von Benutzungsschnittstellen*, Diplomarbeit, Universität - GH Paderborn, Fachbereich Mathematik/Informatik, 1993

[12] Whitehead, A.: *An Introduction to the Mathematics,* New York, Oxford University Press, 1958