Ansätze zu einer empirisch begründeten Entwurfsmethodik für graphische Benutzerschnittstellen

Rainer Gimnich, Gabriele Rohr, Heidelberg

Zusammenfassung

Für den Entwurf geeigneter Benutzerschnittstellen zu komplexen Anwendungssystemen werden Ansätze zu einer Methodik vorgestellt, die die Analyse der Anwendung, konzeptuelle Modellierung, formale Spezifikation und Implementation von Benutzerschnittstellen unterstützt. Analyse und Modellierung basieren dabei auf Ergebnissen empirischer Untersuchungen und legen eine formale Beschreibung der Benutzerschnittstelle nahe, bei der über verschiedene Interaktionsarten abstrahiert wird und die im Zuge ihrer Konkretisierung durch geeignete Werkzeuge interpretiert - d.h. schnell prototypisiert - werden kann.

1. Einleitung

Formale Spezifikationsmethoden beim Entwurf von Software-Systemen legen zum Zeitpunkt des Entwurfs fest, in welche Konzepte das jeweilige System gegliedert wird. Leider gibt es bis heute noch keine Methode, die sicherstellt, daß die Konzepte und ihre Gliederung denen des Benutzers entsprechen. Auch bei graphischen, direkt-manipulativen Benutzerschnittstellen ist die Entscheidung darüber, was graphische Objekte, was textuelle Menüs oder was direkt-manipulative Funktionen sind, häufig arbiträr.

Im folgenden soll versucht werden, auf der Basis bei uns durchgeführter Untersuchungen ein Analyse-Schema zu entwickeln, das zumindest hilft, eine globale Entscheidung über die für den Benutzer optimale Repräsentationsform zu fällen. Weiterhin wird versucht, eine Methode zur Spezifikation benutzerseitiger Semantik zu entwickeln (angelehnt an Jackendoff, 1983), die zumindest eine Berücksichtigung in formalen Methoden zur Software-Spezifikation ermöglicht.

2. Analyseverfahren

Einer der wichtigsten ersten Schritte für die Analyse eines Anwendungssoftware-Systems ist die Bestimmung der als Objekte zu fassenden Einheiten. So konnte Rohr (1987) z.B. zeigen, daß das Erkennen von Objekten und Objektklassen in einem System wesentlich zur besseren Handhabung dieses Systems durch den Benutzer beiträgt. Ein Problem ist allerdings dabei, wie man zu der Definition solcher Objekte kommt. Ein mögliches Verfahren ist, die Sätze von Attributklassen zusammenzufassen, die auf der Benutzerseite bestimmte Objekte mit bestimmten Bedeutungen konstituieren, wie es zur Zeit beim Datenbankentwurf mithilfe des Entity-Relationship-Modells oder anderen Verfahren schon getan wird. Die konkreten Attributwerte

definieren dann die Eigenschaften des Einzelobjekts. Weiterhin sind die Beziehungen der Objekte zueinander zu bestimmen und die auf den Objekten ausführbaren Operationen.

Die Darstellungsform einer so definierten Applikations-"Welt" hängt nun von mehreren Faktoren ab. Zu unterscheiden sind die globalen Operationsebenen und die innerhalb oder zwischen ihnen bestehenden Aufgabencharakteristiken.

Gerstendörfer und Rohr (1987) stellten vier verschiedene Grundtypen von Aufgabencharakteristiken fest, für die eine jeweils spezifische Repräsentationsform die Aufgabenlösung optimal erleichtert. Diese optimale Repräsentation ist unabhängig vom individuellen Problemlösestil.

Für Teilaufgaben, die primär detaillierte Sichten auf strukturelle Zusammenhänge erfordern, zeigte die bildlich-räumliche Repräsentation der Objektstruktur den größten Vorteil, was auch durch frühere Untersuchungen von Rohr (1986) schon vermutet wurde, wo bildliche Repräsentationsformen zu einem besseren strukturellen Verständnis der Gesamtapplikation führten.

Bei Teilaufgaben, die das Zusammenfassen und Zu- und Umordnen von Elementemengen betreffen, führt eine tabellarische Repräsentationsform (z.B. *Spreadsheet*) zu den besten Ergebnissen. Während rein lineare Transformationsprozesse mit vorgegebener sequentieller Abfolge am besten textlich repräsentiert werden, stellt sich die Rekonstruktion oder freie Bestimmbarkeit sequentieller Abläufe als insgesamt sehr schwierig dar. Sie erfordert die parallele Darstellung der beteiligten Objekte/Prozeduren und die Entscheidung über ihre sequentielle Abfolge (siehe dazu auch Arend in diesem Band). Nach den Ergebnissen von Gerstendörfer und Rohr (1987) und auch denen von Dadam und Rohr (1988), in einem anderen Zusammenhang, scheint die Vermutung nahezuliegen, daß eher abstrakte, global unterschiedliche Repräsentationen der Objekte mit Merkmalsträgern, die die Logik der Sequenz nahelegen (Metaphern), als günstigste Darstellungsform angesehen werden können. Die Operationsebene der globalen Suche oder Orientierung enthält meistens eine solche Charakteristik.

Die Operationsebene der detaillierten Analyse von und Suche nach Beziehungen von Objekten und ihren Eigenschaften ist meistens durch die Aufgabencharakteristik "strukturelle Sichten" gekennzeichnet. Die Ebene der Operationen auf dem Analyse- oder Such-Resultat kann dagegen jede Aufgabencharakteristik enthalten.

Während die Analyse der Operationsebenen und deren Aufgabencharakteristiken zu einer Entscheidung über die globale Repräsentationsform von Zuständen von Objektstrukturen führt, wird mit der Analyse der Operationscharakteristik die Repräsentationsart der Operation, nämlich Kommandowort oder direkte Manipulation (räumliche Operation), festgelegt. Rohr (1986) konnte zeigen, daß nach Jackendoff (1983) spezifizierte, von der Semantik her räumliche Funktionen (z.B. Einfügen, Verschieben) besser durch Bildoperationen, dagegen existentielle (z.B. Auslöschen, Definieren) oder zusammenfassende Funktionen besser durch Wortkommandos repräsentiert werden. Zusammenfassende Funktionen werden als Funktionen verstanden, die sich wiederholende komplexe Operationsfolgen als eine Funktion bezeichnen.

Einen Überblick über die Analysestufen gibt Tabelle 1.

Die bisher gezeigte Analysemethode hilft zwar der globalen Gliederung und Zustandsbestimmung einer Applikation und deren Grundpräsentationsform, aber noch nicht der konkreten Ausgestaltung der Objektbeziehungen und der darauf ablaufenden Operationen. Dieses muß gesondert behandelt werden, mit einer Spezifikationsmethodik, die es erlaubt, die Semantik zu beschreiben.

Tabelle 1: Stufen der Analyse einer Applikation und mögliche Repräsentation

	Analyse-Stufe	Repräsentation
1.	**Analyse der Applikationskomponenten**	
	- Objekte	- Graphische Objekte
	- Eigenschaften	- Wortlisten/Form
	- Beziehungen zw. Objekten	- räumliche Anordnung
	- Operationen auf Objekten	- Wort/direkte Manipul.
2.	**Analyse der globalen Operationsebenen der Applikation**	
	- Globale Suche/Orientierung	- bildlich abstrakt
	- Detaillierte Analyse und Suche	- graphisch räumlich
	- Operationen auf dem Analyse-/Such-Resultat	- je nach Aufgabe
3.	**Analyse der Aufgabencharakteristiken auf jeder Operationsebene**	
	- Strukturelle Sichten	- graphisch räumlich
	- Rekonstruktion von Sequenzen	- bildlich abstrakt
	- Lineare Transformationsprozesse	- Sätze
	- Zu- und Umordnen und Zusammenfassen von Elementemengen (Kalkulation)	- formale Tabelle
4.	**Analyse der Operationscharakteristik der Einzeloperationen innerhalb einer bestimmten Aufgabencharakteristik**	
	- räumliche Funktionen	- direkte Manipulation
	- existentielle Funktionen	- Wort
	- zusammenfassende Funktionen	- Wort

3. Spezifikation der benutzerseitigen Semantik: konzeptuelles Modell

Wichtig für einen Spezifikationsansatz der Benutzersemantik ist, speziell wenn er graphische Benutzerschnittstellen berücksichtigen soll, daß er eine derart allgemeine Beschreibung liefert, die es erlaubt, abstraktere Konzepte in räumliche Konzepte umzusetzen. Räumliche Konzepte sind die Grundlage für graphische Repräsentationen. Dieses leistet der Ansatz von Jackendoff (1983), der ursprünglich für die Beschreibung der Semantik natürlicher Sprachen entwickelt wurde. Aus diesem Grund können wir uns auch auf einen Teilausschnitt seiner Beschreibungssprache beschränken, da in den von uns zu betrachtenden Applikationssystemen (wir schließen erst einmal Expertensysteme aus) z.B. Konditionale und Futur nicht ausgedrückt werden.

Grundsätzlich besteht der Beschreibungsansatz aus der Bestimmung konzeptueller Primitive von Einheiten und Funktionen und ihrer formalen Beschreibung. Auf der Seite der Einheiten werden Dinge (THING) und Eigenschaften (PROPERTY) unterschieden. Zwar hat Jackendoff (1983) zwei unterschiedliche Notationen für PROPERTY, aber nur diese Notation erlaubt es, auch Eigenschaftsklassen zu spezifizieren.

ENTITY:

[TOKEN] <THING> ,

[TYPE] <**THING**> '

[TOKEN] <**PROP**> '

[TYPE] <**PROP**>

Beide Konzepte haben eine gemeinsame Teilmenge von Funktionen und sind miteinander kombinierbar. Ein wesentlicher Unterschied besteht jedoch: Während ein Individuum einer Klasse von Dingen mehrere Individuen einer anderen Klasse von Dingen besitzen kann, kann es nicht mehrere Individuen einer Eigenschaftsklasse besitzen (z.B. kann ein Gegenstand nicht gleichzeitig von der Form rund und eckig sein). Dieser Unterschied hilft auch bei der Bestimmung von Objekten und Attributen in der Stufe 1 des Analyseverfahrens.

Weiterhin werden generelle Ereignisklassen unterschieden wie **STATE(BE), EVENT(GO)** und **ACTION(DO)**. Diese sind für die Beschreibung von Interaktionen mit dem Anwendungssystem von wesentlicher Bedeutung. Sie ergeben, kombiniert mit den im folgenden beschriebenen Funktionsprimitiven und den damit verknüpften Komponenten, eine relativ mächtige Beschreibungssprache.

Bei den Funktionsprimitiven lassen sich zwei Hauptgruppen unterscheiden: solche, die auf Platzfunktionen transformiert werden können, und solche, die es nicht können. Im allgemeinen gilt, daß Funktionen der Form

FUNCTION(ENTITY, ENTITY)

in Platzkonzepte übersetzt werden können, aber nicht Funktionen der Form

FUNCTION(ENTITY)

Zu der Gruppe der transformierbaren Funktionen gehören

EXAMPLIFIED_BY(ENTITY, ENTITY)

INSTANCE_OF(ENTITY, ENTITY)

POSSESS(ENTITY, ENTITY)

Zu der Gruppe der nicht (oder nur über Hilfskonstrukte) transformierbaren Funktionen gehören

EXIST(ENTITY)

NOT_EXIST(ENTITY)

Eine Sonderstellung nimmt das Konzept **AMOUNT** ein. Es ist nur im Zusammenhang mit Platzkonzepten räumlich darstellbar.

Die Schlüsselfunktion für graphische Benutzerschnittstellen nimmt die **PLACE_FUNCTION** ein. Ihre Besonderheit ist, daß sie immer mit einem Parameterwert aus einer Liste räumlicher Parameter besetzt ist, und daß das referierte Objekt als Platz fungiert:

PLACE_FUNCTION:par (ENTITY, PLACE:ENTITY)

par : in, on, under, beneath, behind, in_front_of, ...

Diese speziellen Eigenschaften der Platzfunktion machen sie besonders geeignet, als Projektion für andere Funktionssemantiken zu dienen. Die konkreten Platzparameter können und müssen dabei empirisch bestimmt werden.

Ausgedrückt als Zustandsbeschreibung können Besitzfunktion und Instanziierungsfunktion in der folgenden Form auf Platzfunktionen projiziert werden (siehe dazu auch Rohr 1988):

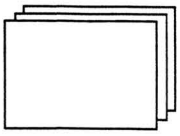
(1) BE INSTANCE_OF ([TOKEN] <THING> , [TYPE] <THING>)

→ BE PLACE_FUNCTION: In ([TOKEN] <THING> , PLACE: [TYPE] <THING>)

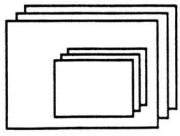
(2) BE POSSESS ([TOKEN] <THING> , [TYPE] <THING>)

→ BE PLACE_FUNCTION: on ([TOKEN] <THING> , PLACE: [TYPE] <THING>)

Umsetzbar als abstrakte Graphik in der folgenden Form:



(1)



(2)

In gleicher Weise können so auch ACTIONs und EVENTs beschrieben werden.

Nehmen wir uns ein Beispiel, hier die Anwendung Datenbanken, um die Konsequenzen der Beschreibungssprache besser nachzuvollziehen.

Angenommen, wir befinden uns auf der Operationsebene der "detaillierten Analyse und Suche" mit der Aufgabencharakteristik "strukturelle Sichten". Wir benötigen also eine graphisch räumliche Repräsentation. Nehmen wir weiterhin an, wir hätten drei Objektklassen identifiziert: Abteilungen, Angestellte und Geräte (auf die Modellierung der Eigenschaften verzichten wir hier aus Platzgründen). Als nächstes müssen wir die Beziehungen zwischen den Objekten identifizieren. Wir haben folgende Aussagen vorliegen:

(1) Abteilungen gehören zu einer Organisation

(2) Es gibt viele Abteilungen, Angestellte, Geräte

(3) Eine Abteilung hat viele Angestellte

(4) Ein Angestellter gehört zu einer Abteilung

(5) Ein Angestellter hat viele Geräte

(6) Ein Gerät hat viele Angestellte (als Benutzer)

Als drei mögliche Sichten für die Analyse, kann man folgende feststellen:

A. Die Organisation aus der Sicht der Abteilungen

B. Die Organisation aus der Sicht der Angestellten

C. Die Organisation aus der Sicht der Geräte

Die Hauptcharakteristik der Aufgabe sei hier, die Sichten für die Analyse der Objektstruktur zu wechseln.

Die Spezifikation der Semantik der oben aufgeführten Aufgaben ergibt:

(1) BE INSTANCE_OF ([TYPE] <THING> : Abteilung, [TYPE] <THING> : Organisation)

(2) BE INSTANCE_OF (AMOUNT: many ([TOKEN] <THING> : organisation), [TYPE] <THING> : Organisation) etc.

(3) BE POSSES ([TOKEN] <THING> : abteilung, AMOUNT: many ([TOKEN] <THING> : angestellte))

(4) BE POSSESS ([TOKEN] <THING> : angestellte, AMOUNT: one ([TOKEN] <THING> : abteilung))

etc.

A. BE EXAMPLIFIED_OF ([TYPE] <THING> : Organisation, [TYPE] <THING> : Abteilung)

B. BE EXAMPLIFIED_OF ([TYPE] <THING> : Organisation, [TYPE] <THING> : Angestellte)

etc.

Das Ganze kann dann in PLACE_FUNCTIONS transformiert werden.

Um die Sicht zu wechseln, muß eine Aktion gestartet werden, die entweder "indirekter" durch die Folge

DO FUNCTION(..)

GO PLACEFUNCTION: on (..)

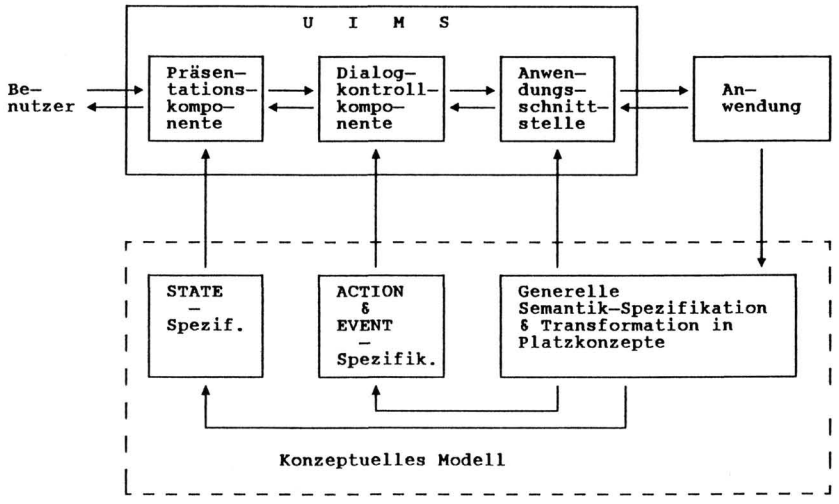
oder "direkter" durch eine räumliche Operation

DO PLACE_FUNCTION: on (..)

modelliert werden kann. Allgemein ergibt sich aus dem Abgeleiteten eine Forderung für eine zu implementierende Funktion mit dieser Bedeutung des Sichtwechsels.

4. Logische Gesamtarchitektur

Eine Benutzerschnittstelle ist ein Software-Teilsystem, das aufgrund formaler Beschreibungen aller relevanten Interaktionsaspekte implementiert wird. Die durch die Analyse der Anwendung gewonnene Beschreibung des konzeptuellen Modells fließt dabei in die implementationsnähere formale Spezifikation der Benutzerschnittstelle ein. Man versucht, auf der Basis formaler Beschreibungen Benutzerschnittstellen (zumindest teilweise) automatisch zu erzeugen und vorhandene anwendungsunabhängige Softwarebausteine wiederzuverwenden. In diesem Zusammenhang wird auch der Begriff **User Interface Management System (UIMS)** gebraucht. Mit dem Entwurf unseres UIMS zielen wir u.a. auf eine Abstraktion über verschiedene Interaktionsarten ab. Dabei baut die logische Architektur auf dem "Seeheim-Modell" auf (vgl. Pfaff, 1985):



Die **Präsentationskomponente** ist für die externe Präsentation der Benutzerschnittstelle verantwortlich. Auf dieser Ebene werden die lexikalischen Dialogeinheiten (Ein-/Ausgabe-*Tokens*) abstrakt definiert und zu physikalischen Repräsentationen in Beziehung gesetzt (*internal-external mapping*).

Die **Dialogkontrollkomponente** definiert die Syntax des Dialogs zwischen Benutzer und Anwendungsprogramm. Dies erfordert die Speicherung und Kontrolle der benutzer- und systemseitigen Abarbeitungszustände. Die von der Präsentationskomponente gelieferten abstrakten Eingabesequenzen werden hier analysiert, und anwendungsrelevante Informationen werden an die Anwendungsschnittstelle weitergeleitet. Umgekehrt werden in der Dialogkontrollkomponente auch Ergebnisse von Anwendungsfunktionsaufrufen interpretiert und entsprechende Informationen zur Ausgabe an die Präsentationskomponente gegeben. Bestimmte Benutzereingaben (z.B. bei Metadialogen) können bereits in der Dialogkontrollkomponente abschließend behandelt werden, ohne daß Anwendungsfunktionen eingeschaltet werden müssen.

Die **Anwendungsschnittstelle** ist eine Repräsentation der Anwendung aus der Sicht des Benutzers. Für die Implementierung wird die formale Spezifikation dieser Komponente zu den durch die Anwendung vorgegebenen Objekten und Operationen in Beziehung gesetzt.

Durch die vorgestellte Ebenentrennung wird eine weitgehende Entkopplung der verschiedenen Interaktionsaspekte erreicht. Dies hat den Vorteil, daß für neue Anwendungen u.U. nur die Anwendungsschnittstelle redefiniert werden muß.

5. Spezifikation der Komponenten

Die Beschreibungsmethoden und -sprachen für die einzelnen Komponenten sollen die Erstellung formaler Spezifikationen unterstützen. Abstrakte Beschreibungen sollen dabei durch schrittweises Verfeinern in konkrete Realisierungen umgesetzt werden können, wobei eine

schnelle Prototypentwicklung (*Rapid Prototyping*) auf verschieden hohen Entwicklungsstufen angestrebt wird.

5.1 Spezifikation der Präsentationskomponente

In unserem Vorgehen werden Präsentationskomponente und Anwendungsschnittstelle in einer einheitlichen Notation spezifiziert. Es handelt sich um eine formale Spezifikationssprache auf der Grundlage der Wiener Methode (VDM, *Vienna Development Method*, vgl. z.B. Bjørner und Jones, 1982; Jones, 1986). In der verwendeten Sprache (vgl. Gimnich und Ebert, 1987) werden Präsentations- und Anwendungsobjekte zusammen mit den auf ihnen wirkenden Operationen als abstrakte Datentypen spezifiziert. Dies geschieht auf konstruktive Weise, d.h. durch die direkte Angabe eines mathematischen Modells, zu dessen Konstruktion bestimmte vorgegebene Grundkonzepte (z.B. Mengen, Listen, Abbildungen) verwendet werden.

Beispiel:

Zu einem vorgegebenen Datenbanksystem als Anwendung soll eine Benutzerschnittstelle entworfen werden, die eine (ggf. parallele) Verwendung geeigneter Interaktionsarten realisiert. Die Analyse der Anwendung legt eine hierarchische Präsentation der Datenobjekte nahe, wobei die Hierarchie dynamisch restrukturierbar ist, d.h. aus der "Sicht" verschiedener in der Darstellung enthaltener Objekte aufgebaut werden kann.

Die Präsentationskomponente kann nun durch ein mathematisches Modell beschrieben werden, das eine logische Bildschirmaufteilung in einen Bereich zur direkten Bearbeitung von Datenobjekten (data_object) und einen Bereich mit Kontrollobjekten (control_object) für Layout, Farbgebung, Hilfefunktionen, usw. enthält. Dabei werden die Datenobjekte als baumartige Hierarchien rekursiv definiert:

```
module  presentation

definitions

    present   := record
                    data_object :   node ;
                    control_object :  { help_object, scroll_object, ... } ;
                 end;

    node      := record
                    id :  string ;
                    attributes  subset  { marked, modified, ... } ;
                    successors :  set of node ;
                 end;
```

Die auf diesem Präsentationsmodell gewünschten Operationen werden getrennt spezifiziert, ihre Syntax in mathematischer Funktionsnotation, ihre Semantik mittels Vor- und Nachbedingungen. Z.B. kann eine einfache Funktion mark_data_object zum Auszeichnen eines beliebigen Objekts in der dargestellten Hierarchie folgendermaßen festgelegt werden:

```
operations
    mark_data_object : present × node → present
    pre-mark_data_object (p, n) == true
    post-mark_data_object (p, n; p')
        == p' = p # < find(p.data_object, n).attributes ∋ marked >
```

Die Vorbedingung besagt, daß die Funktion mark_data_object stets angewendet werden darf; die Nachbedingung beschreibt die Wirkung der Funktion: das entsprechende Objekt soll nach der Funktionsanwendung das Attribut marked enthalten.

In der gleichen Notation werden alle weiteren Funktionen dieses Moduls spezifiziert, u.a. eine Funktion restrict_data_object zum Eintragen von Restriktionen für die Werte eines Datenobjekts und eine Funktion change_view zum Restrukturieren der Hierarchie gemäß den Anforderungen (vgl. Abschnitt 3).

Eine derartige Beschreibung der Präsentationskomponente erlaubt eine Abstraktion über verschiedene Interaktionsarten: Erst in der schrittweisen Konkretisierung der Spezifikation wird festgelegt, ob beispielsweise die obige Markierungsfunktion in der Präsentation durch ein Kontrollobjekt (z.B. ein Menü oder einen *Screen-Button*) oder durch eine bereichsabhängige Maustastenbelegung aufrufbar ist, ob die Angabe des Zielobjekts durch eine Zeigeoperation oder Eingabe seines Namens oder beides möglich ist. Dadurch ist eine einheitliche Betrachtung mehrerer, ggf. parallel verwendeter Interaktionsarten möglich.

5.2 Spezifikation der Anwendungsschnittstelle

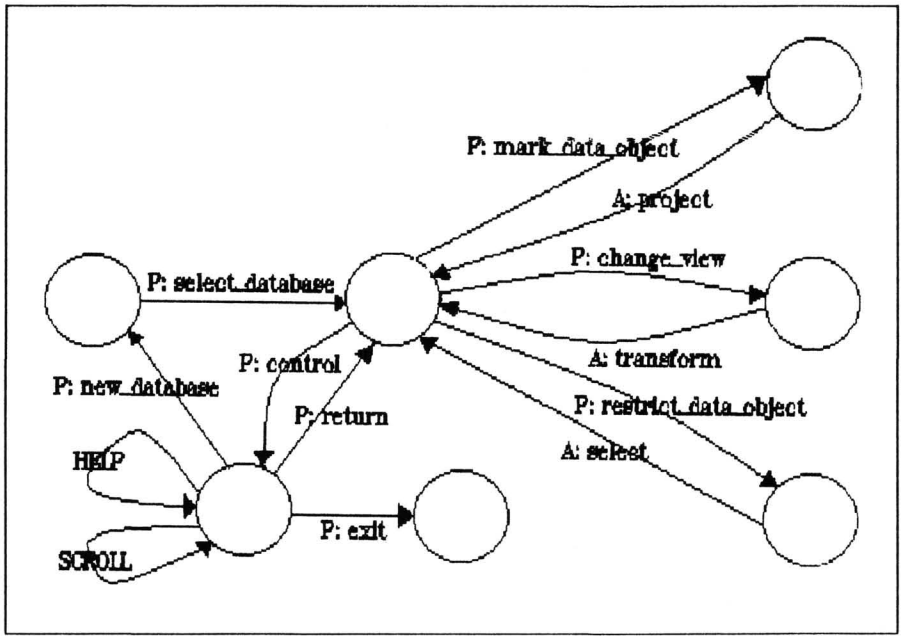
Mit der gleichen Spezifikationssprache läßt sich die Anwendungsschnittstelle beschreiben. Im Beispiel des Datenbanksystems als Anwendung wird eine der Darstellung baumartiger Hierarchien auf der Präsentationsseite entsprechende Semantik definiert. Hierzu werden komplexe Datenobjekte im Sinne eines erweiterten NF^2 *(non first normal form)* Datenmodells (vgl. z.B. Pistor und Traunmüller, 1986) spezifiziert, einschließlich darauf wirkender Operationen. Diese Objekte sind als Mengen von Tupeln komplexer Feinstruktur zu verstehen, d.h. die Tupelkomponenten können selbst wieder Mengen von Tupeln sein. Eine Spezifikation der Anwendungsschnittstelle zu dieser Beispiel-Anwendung findet sich in Gimnich (1989).

Auch die Beschreibung der Anwendungsschnittstelle wird weiter konkretisiert und schließlich zu den vorgegebenen Datenstrukturen und Operationen des Anwendungssystems in Beziehung gesetzt.

5.3 Spezifikation der Dialogkontrollkomponente

Um das Konzept des Abarbeitungszustandes auf abstrakten Beschreibungsebenen für Dialogstrukturen explizit zu fassen, wird hier von hinreichend mächtigen Zustands-Übergangs-Netzwerken ausgegangen (vgl. Jacob, 1983, 1986).

Die Dialogkontrollkomponente für die Datenbankbenutzerschnittstelle läßt sich z.B. folgendermaßen beschreiben:



Hier werden drei Arten von Übergangsmarkierungen unterschieden:

- Operationen der Präsentationskomponente (mit **P:** versehen)
- Operationen der Anwendungsschnittstelle (mit **A:** versehen)
- Kontrollfluß zwischen mehreren Netzwerken, z.B. Aufruf von Unternetzen (in GROSSBUCHSTABEN geschrieben).

Komplexere Dialogstrukturen werden entsprechend mächtige Kontrollstrategien, z.B. ein Koroutinen-Konzept (Jacob, 1986), erfordern. Je nach Zielumgebung kann es sich im Verlauf der Konkretisierung der Dialogstrukturbeschreibung als sinnvoll erweisen, zu *event*-orientierten Konzepten überzugehen; entsprechende Konversionsalgorithmen finden sich z.B. in Green (1986).

6. Einsatz der Entwurfsmethodik

Erste Tests der Anwendbarkeit dieses Ansatzes auf tatsächliche alternative Entwürfe von Benutzerschnittstellen für unterschiedliche Applikationen wie

- Datenbankabfragen

- Erstellung von Business-Graphiken
- Hardware-Konfiguration von Betriebssystemen

(in mehr oder weniger breiten Teilaspekten) haben zu überraschend "guten" und schnellen prototypischen Lösungen geführt, die im ersten Schritt schon zu einer wesentlichen Verständnisverbesserung der modellierten Anwendungsfunktionen auf der Benutzerseite führten.

Literatur

Bjørner, D., Jones, C.B. (1982). Formal specification and software development. Prentice-Hall, Englewood Cliffs, NJ.

Dadam, P., Rohr, G. (1988). Advanced information management. in: Blaser, A. (ed.), Heidelberg Scientific Center bi-annual report 1986/87 - research areas and projects, IBM, Heidelberg.

Gerstendörfer, M., Rohr, G. (1987). Which task in which representation on what kind of interface? in: Bullinger, H.-J., Shackel, B. (eds.), Human-Computer Interaction - INTERACT '87, North-Holland.

Gimnich, R., Ebert, J. (1987). Constructive formal specifications for rapid prototyping. in: Bullinger, H.-J., Shackel, B. (eds.), Human-Computer Interaction - INTERACT '87, North-Holland.

Gimnich, R. (1989). Implementing direct manipulation query languages using an adequate data model, in: Tauber, M.J., Gorny, P. (eds.), Visualization in Human-Computer Interaction, Springer-Verlag, Berlin/Heidelberg/New York (im Druck).

Green, M. (1986). A survey of three dialogue models. ACM Transactions on Graphics 5 (1986, 3), 244 - 275.

Jackendoff, R. (1983). Semantics and cognition. MIT Press, Cambridge, MA.

Jacob, R.J.K. (1983). Using formal specifications in the design of a human-computer interface. Communications of the ACM 26 (1983, 4), 259 - 264.

Jacob, R.J.K. (1986). A specification language for direct-manipulation user interfaces. ACM Transactions on Graphics 5 (1986, 4), 283 - 317.

Jones, C.B. (1986). Systematic software development using VDM. Prentice-Hall, London.

Pfaff, G.E. (1985). User interface management systems. Springer-Verlag, New York.

Pistor, P., Traunmüller, R. (1986). A database language for sets, lists, and tables. Information Systems 11 (1986, 4), 323 - 336.

Rohr, G. (1986), Using visual concepts. in: Chang, S.-K., Ichikawa, T., Ligomenides, P.A. (eds.), Visual languages, Plenum Press, New York.

Rohr, G. (1987). How people comprehend unknown system structures: Conceptual primitives in systems' surface representations. in: Tauber, M.J., Gorny, P. (eds.), Visualization in Human-Computer Interaction, Springer-Verlag, Berlin/Heidelberg/New York.

Rohr, G. (1988). Graphical user languages for querying information: where to look for criteria? Proceedings IEEE Workshop on Visual Languages, Pittsburgh, PA.

Rainer Gimnich, Gabriele Rohr
IBM Wissenschaftliches Zentrum Heidelberg
Tiergartenstr. 15
6900 Heidelberg