# Ein modellbasierter Ansatz zur Dialogsteuerung in Benutzungsoberflächen

Friedrich Strauß, Stefan Hügel, Kirsten Winter, Britta Schinzel
Universität Freiburg

**Zusammenfassung**
User Interface Management Systeme (UIMS) ermöglichen eine getrennte Entwicklung von Benutzungsoberfläche und Applikation. Für die Konstruktion von direkt manipulativen Oberflächen ist jedoch die Integration von Anwendungswissen in die Dialogsteuerung notwendig.
Dieser Beitrag zeigt auf, wie anwendungsabhängiges Wissen durch eine modellbasierte Beschreibung der Dialogsteuerung in das UIMS aufgenommen werden kann. Dabei wird unsere situationsorientierte Sichtweise der Benutzungsschnittstelle vorgestellt, die Grundlage für eine modulare Dialogbeschreibung ist. Die Zerlegung der Oberfläche in Teilschnittstellen wird durch die situationsorientierte Konstruktion der Dialogsteuerung erreicht. Die Beschreibung der Anwendungsabhängigkeiten innerhalb bzw. zwischen den Teilschnittstellen ist so in einfacher Weise möglich. Die realisierte Dialogsteuerung erlaubt außerdem die Steuerung von Präsentationsoberflächen mit variabler Anzahl von Objekten und Fenstern und bietet so eine weitergehende Unterstützung für direkt manipulative Oberflächen als bisherige Ansätze.
Ein derartiges Modell der Anwendung wird zusätzlich zur Erzeugung kontextsensitiver Hilfe genutzt. Darüberhinaus kann die Dialogbeschreibung die Analyse der Dialogsteuerung vereinfachen und ihre Adaptierbarkeit verbessern.

## 1 Einleitung

Graphisch interaktive Benutzungsoberflächen (GUIs) mit direkter Manipulation als primärer Interaktionsform bilden die geeignete Grundlage für werkzeugartige Anwendungen. Die Programmierung dieser Oberflächen ist häufig genauso komplex und umfangreich, wie die Programmierung der eigentlichen Anwendung. Nach verschiedenen empirischen Untersuchungen werden bis zu 88% des Codes für die Gestaltung der Oberfläche aufgewendet (vgl. [10]). Oberflächeneditoren (Interface Builder), die auf objektorientierte Fenstersysteme aufbauen, vereinfachen die Programmierung der Präsentationskomponente erheblich, indem standardisierte Oberflächenelemente (Widgets) graphisch interaktiv zu einem Rumpf für die Benutzungsoberfläche zusammengesetzt werden [11]. Problematisch ist unter softwaretechnischen wie softwareergonomischen Aspekten die Programmierung der eng mit der Oberfläche verzahnten Dialogsteuerung innerhalb der Anwendung [11].

Mit dem Seeheim-Modell wurde eine Architektur für User Interface Management Systeme (UIMS) vorgeschlagen, die eine konzeptuelle Trennung von Oberfläche (inkl. der Dialogsteuerung) und Anwendung vorsieht. Diese Aufteilung bietet eine Reihe von Vorteilen: die Oberflächenprogrammierung kann durch spezielle Sprachen und Werkzeuge unterstützt werden, die Konsistenz der Präsentation und der Dialogstruktur kann erhöht werden, kontextsensitive Hilfen können einfacher zur Verfügung gestellt werden etc.

Die direkte Manipulation [8], die sich in allen modernen graphisch interaktiven Anwendungen wiederfindet, widerspricht allerdings der sequentiellen Struktur der frühen UIMS-Ansätze, die eine getrennte lexikalische, syntaktische und semantische Bearbeitung vorsehen. Das Prinzip der direkten Manipulation beruht u.a. wesentlich auf direkter *syntaktischer und semantischer Behandlung* von Interaktionen. Bei direkt manipulativen Oberflächen bietet sich häufig die Darstellung beliebig vieler Objekte in einem Fenster an, die bearbeitet und z.B. auch zwischen Fenstern verschoben werden sollen. Ein graphisch interaktiver Dateimanager ist eine typische Anwendung mit solchen Anforderungen, die im folgenden auch als Illustrationsbeispiel verwendet wird.
Die frühen auf dem Seeheim-Modell basierenden Dialogsteuerungen können solche Oberflächen nicht unterstützen. Eine ausführliche Diskussion früher und aktueller (amerikanischer) Ansätze findet sich in [13]. Neuere Ansätze zur Dialogsteuerung erlauben direkt manipulative Interaktionen, stellen aber häufig keine Mittel zur Verfügung, eine beliebige Anzahl von Objekten oder Fenster zu verwalten (z.B. [9]). Andere Ansätze stellen eher eine abgespeckte objektorientierte Programmiersprache zur Verfügung (vgl. [19]), deren Repräsentation dann allerdings u.a. den Einsatz von Hilfemodulen und Analysewerkzeugen erschwert, oder nutzen Constraints zur Beschreibung graphischer Abhängigkeiten.
Modellbasierte UIMS-Ansätze basieren auf einem funktionalen Modell der Anwendung, das zur automatischen Konstruktion der graphischen Darstellung und/oder zur Steuerung der Oberfläche genutzt wird. Wesentlich ist hier die Ausdrucksmächtigkeit und Adäquatheit der Modellbeschreibungssprache. Die meisten Ansätze nutzten eine objektorientierte Sichtweise zur Beschreibung des Anwendungsmodells, die allerdings durch ihre uniforme Beschreibung von Interaktionen Beschränkungen bzgl. der Ausdrucksmächtigkeit von semantischen Abhängigkeiten implizieren (vgl. Abschnitt 2.4). Wir schlagen deshalb neben einer objektorientierten Realisierung der Präsentationskomponente zur graphischen Umsetzung der direkt manipulativen Interaktionen eine *logisch orientierte* Beschreibung der Interaktionen bzw. Funktionen einer Anwendung (der Anwendungslogik) vor, da so eine größere Ausdrucksmächtigkeit für semantische Eigenschaften zur Verfügung gestellt und eine explizite Dialogbeschreibung realisiert werden kann, die für weitere Werkzeuge zur Verfügung steht. Dieser Ansatz ist auf einer höheren Abstraktionsebene angesiedelt als graphische Constraints, die im wesentlichen zur direkten Umsetzung graphischer Abhängigkeiten dienen.

Im folgenden Abschnitt stellen wir den SUSI-Ansatz vor, der diese logisch- und situations-orientierte Steuerung von Oberflächen erlaubt. Als erste realisierte Erweiterung skizzieren wir Abschnitt 3 die kontextsensitiven Hilfefunktionen, die mit dem zur Dialogsteuerung realisierten Hilfemodul zur Verfügung gestellt werden und einer von mehreren Aspekten darstellt, die erst durch eine explizite Dialogbeschreibung ermöglicht werden. Abschließend werden mögliche Erweiterungen der SUSI-Dialogsteuerung diskutiert. Neben Evaluation und Analyse der Dialogsteuerung erscheint uns die dynamische Anpassung der Steuerung durch das Hinzufügen weiterer Aktionsregeln zur Dialogbeschreibung besonders hilfreich.

# 2 Die SUSI-Dialogsteuerung

In diesem Abschnitt wird SUSI (Situation Oriented User Interface Model) vorgestellt, ein auf Aktionsregeln und logischen Beschreibungen basierendes Konzept zur Dialogsteuerung in Benutzungsschnittstellen (vgl. [6, 7, 16]).

In SUSI wird der Dialog in strukturierter Weise durch *Aktionsregeln* beschrieben, deren Ausführung vom Eintreten von Ereignissen (repräsentiert durch *Trigger*) und der Gültigkeit von Vorbedingungen abhängig ist. Bei der Verarbeitung einer Aktionsregel können Operationen in der Anwendung ausgelöst, die Darstellungsschicht verändert, und das Wissen über den aktuellen Zustand beeinflußt werden.

## 2.1 Modularisierung der Beschreibung durch Situationen

Graphische Benutzungsschnittstellen sind meist aus mehreren Teilschnittstellen zusammengesetzt, die _ beispielsweise in einer Datenbankanwendung _ verschiedenen Benutzerprozessen oder unterschiedlichen Sichten auf den Datenbestand zugeordnet sind. In der Regel werden diese Teilschnittstellen durch verschiedene *Fenster* visualisiert, Bereichen, in denen zum Teil unabhängig voneinander gearbeitet wird (z.B. nebenläufige Prozesse) und die zum Teil miteinander kommunizieren (z.B. Verzeichnisfenster, zwischen denen Dateien verschoben werden).
Bei der Beschreibung einer solchen Schnittstelle ist es damit einerseits sinnvoll, unabhängige Teilschnittstellen unabhängig voneinander zu beschreiben; gleichzeitig muß jedoch eine Kommunikation zwischen ihnen möglich bleiben.

In SUSI werden die einzelnen Sichten durch *Situationstypen* repräsentiert. Jedem Fenster ist eine *Situation* als Ausprägung eines Situationstyps zugeordnet; gleichartige Fenster, wie z.B. Verzeichnisfenster eines Dateimanagers, haben den gleichen Situationstyp, d.h. eine festgelegte Mengen von *Aktionsregeln*, wie z.B.

*trig (drag (X, Y)), fileviewer(Y), Y:cwd(Z), movable (X, Z), move (X, Z)*
-->
*not (fileIcon (X)), trig (window2:fileviewer display (X));*

die das dynamische Verhalten bzw. die Reaktion auf Benutzereingaben festlegen und eine festgelegte Menge von *Nebenbedingungen* (constraints), wie z.B.

*file (A), folder (B), user (C), owns (A, C), owns (B, C) --> movable (A, B);*

die die Definition von anwendungsbezogenen Konzepten ermöglichen, die in Vorbedingungen von Aktionsregeln genutzt werden können. Der momentane Zustand einer Situation wird durch eine Menge von *Fakten* (Relationen im Sinne der Prädikatenlogik erster Stufe) beschrieben, die sich mit der Zeit verändert. Die Kommunikation zwischen den Situationen erfolgt durch das Versenden von Triggern; dadurch kann in einer anderen Situation die Ausführung einer Aktionsregel angestoßen werden. Das Zusammenwirken ist in Abb. 1 dargestellt.

## 2.2 Der Dialogmanager

Eine Benutzungsschnittstellen kann in eine Darstellungsschicht (Präsentation und lexikalische Behandlung von Interaktionen), Dialogsteuerungsschicht (syntaktische und semantische Behandlung von Interaktionen) und Anwendungsschnittstelle (Ausführung von Anwendungsfunktionen) unterteilt werden. Im folgenden wird beschrieben, wie diese Schichten in SUSI realisiert sind. Zusätzlich wird kurz der Aspekt der Darstellung der Anwendungsdaten angesprochen.

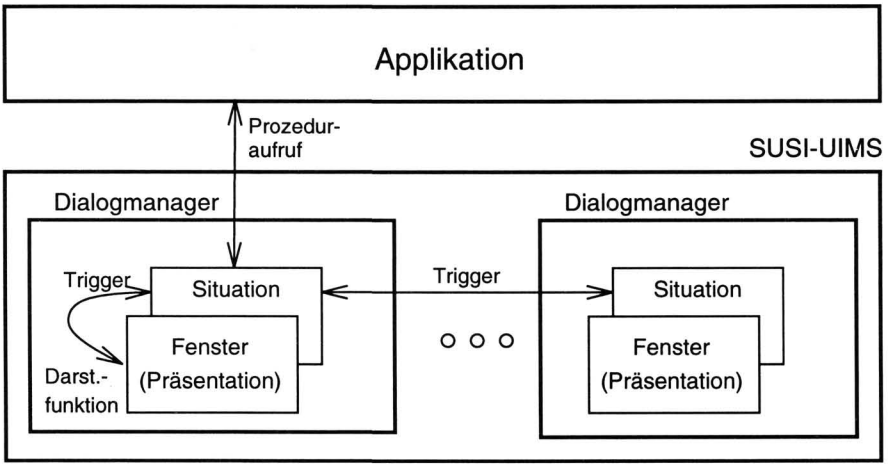


Abb. 1: Präsentationsumgebung und Situationen in der Benutzungsoberfläche zur Laufzeit.

Durch die lexikalische Verarbeitung werden Benutzeraktionen an der Oberfläche registriert und _ entsprechend codiert _ an die Dialogsteuerung weitergeleitet. Dies erfolgt in SUSI durch die Übersetzung in Trigger: Relationen (Literale), die Ereignisse repräsentieren. Aktionsregeln werden aktiviert, indem der übergebene Trigger mit einem ausgezeichneten Literal der Regel unifiziert wird; im Erfolgsfall wird diese Regel weiter bearbeitet. Bei der Unifikation können gleichzeitig Variablen instantiiert werden; auf diese Weise erfolgt eine Parameterübergabe.

Ist ein von der Darstellungsschicht erzeugter Trigger mit dem entsprechenden Literal einer Aktionsregel unifizierbar, so wird diese Regel weiter verarbeitet. Zunächst wird die *Vorbedingung* abgeleitet, eine Konjunktion von Relationen, die entweder innerhalb der aktuellen Situation definiert sind, d.h. Aspekte des Zustands dieser Situation beschreiben, die in einer anderen Situation definiert sind, d.h. sich auf deren Zustand beziehen und von *prozeduralen* Fakten, die durch den Aufruf einer Funktion abgeleitet werden und ihren Wert abhängig vom Erfolg dieses Funktionsaufrufs erhalten.

Konnte die Vorbedingung erfolgreich abgeleitet werden, d.h., wurde die Interaktion ausgeführt, so wird die *Nachbedingung* behandelt. Sie ist ebenfalls eine Konjunktion von Literalen und beschreibt den Zustand nach der Ausführung der Aktionsregel. Dazu werden die in ihr enthaltenen positiven Literale etabliert, d.h. der Faktenmenge werden entsprechende Fakten hinzugefügt, und es werden in der Nachbedingung enthaltene negative Literale aus der Faktenmenge gelöscht. Gleichzeitig werden ggf. Seiteneffekte erzeugt (s.u.). Außerdem können durch die Ausführung der Aktionsregel neue Trigger erzeugt werden, die die Auswahl weiterer Regeln bestimmen.

An der Anwendungsschnittstelle werden die in der Dialogsteuerung erzeugten Befehle in Operationen der Anwendung umgesetzt. Dies erfolgt durch prozedurale Fakten in den Vorbedingungen von Regeln, die als Seiteneffekte die entsprechenden Anwendungsaktionen aktivieren. Das Kopieren einer Datei beispielsweise wird durch eine solche Funktion ausgeführt; die Relation ist *wahr*, wenn das entsprechende Betriebssystemkommando erfolgreich abgeschlossen wurde. Ansonsten ist die Relation *falsch*; durch das Prüfen dieses Wertes kann so entschieden werden, ob die Aktionsregel erfolgreich ausgeführt werden kann oder ob eine (andere) Regel zur Fehlerbehandlung aktiviert werden muß.

Rückmeldungen von Zustandsänderungen in der Darstellungsebene erfolgen durch eine Kopplung einzelner Fakten an Objekte der Oberfläche. Werden bestimmte Fakten bei der Verarbeitung der Nachbedingung einer Aktionsregel etabliert oder aus der Faktenmenge entfernt, werden ihnen zugeordnete Funktionen aufgerufen, die die gewünschten Effekte auf die Darstellungsebene realisieren. Ist beispielsweise eine Datei in der Dialogsteuerung durch ein Fakt und auf der Präsentationsebene durch ein Sinnbild (Icon) repräsentiert, so wird durch solche Funktionsaufrufe bewirkt, daß

das Sinnbild dargestellt wird, wenn das Fakt gültig wird und wieder verschwindet, wenn das Fakt ungültig wird.

## 2.3 Ein Anwendungsbeispiel

Anhand der zuvor dargestellten Aktionsregel bzw. Nebenbedingung soll die Arbeitsweise verdeutlicht werden: Wird eine Datei *file1* aus dem Fenster *window1* in das Fenster *window2* (beide vom Typ *fileviewer*) gezogen, ergibt dies einen Trigger *drag (file1, window2)*, der die Aktionsregel (siehe 2.1) aktiviert. Dabei werden *X* und *Y* entsprechend instantiiert.

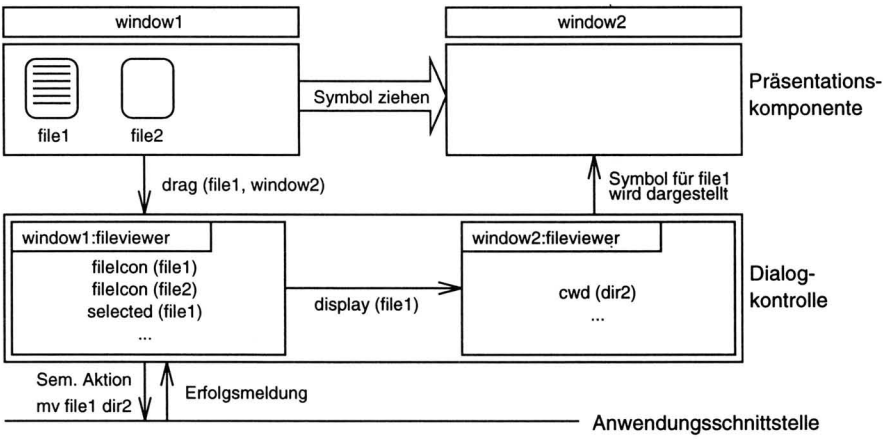


Abb. 2: Der Kontrollfluß in SUSI: Aus *window1* soll eine Datei durch Ziehen mit der Maus auf *window2* verschoben werden.

Nachdem das aktuelle Verzeichnis *dir2* aus der Situation *window2* ausgelesen wurde, wird mit Hilfe der Nebenbedingung das Literal *movable (file1, window2)* abgeleitet; dies geschieht durch Resolution. Die Auswertung des prozeduralen Fakts *move (file1, dir2)* löst die Dateiverschiebung im Betriebssystem aus. Dann wird durch die erste Nachbedingung das Fakt *fileIcon (file1)* gelöscht, das das Sinnbild in *window1* repräsentiert. Als Seiteneffekt wird gleichzeitig dieses Sinnbild von der Oberfläche entfernt. Zuletzt wird ein Trigger erzeugt, der in der Situation *window2* eine Regel zur Darstellung aktiviert. Den Kontrollfluß bei dieser Operation verdeutlicht Abb. 2.

## 2.4 Die Dialogbeschreibung als funktionales Modell der Anwendung

Ein Ziel bei der Entwicklung der Dialogbeschreibungssprache ist, mit der Beschreibung der Dialogsteuerung auch ein funktionales Modell der Anwendung (auf Interaktionsebene) aufzustellen. Wir sehen im Gegensatz zu anderen Ansätzen, wie HUMANOID, GENIUS oder UIDE davon ab, die Oberfläche automatisch aus einem abstrakten (aufgaben- bzw. funktionsorientierten) Modell der Anwendung zu generieren (vgl. [18]). Ein wesentliches Problem ist bei solchen Ansätzen die adäquate Erzeugung der Präsentationskomponente zu der Wissen über den Anwendungsbereich und fundierte Kenntnisse über die (nicht formalisierbaren) Regeln zur softwareergonomischen Gestaltung notwendig ist [20].

Das funktionale Modell der Anwendung _ die Anwendungslogik _ kann durch die Situationstypen und die dazu durch das Triggerkonzept realisierte Kommunikation einfach modularisiert werden und ermöglicht eine explizite Repräsentation durch Aktionsregeln und Nebenbedingungen. Da durch Aktionsregeln beliebige Fakten innerhalb einer Situation etabliert werden können und zusätzlich Konzepte durch Hornformeln definiert werden können, sind auch komplexe Zusammenhänge zwischen verschiedenen Anwendungsfunktionen in den Vorbedingungen der Aktionsregeln darstellbar. Ob bestimmte Informationen als Fakten innerhalb der Situation dauerhaft vorliegen oder bei Bedarf durch prozedurale Fakten bestimmt werden, kann anwendungsabhängig umgesetzt werden. So werden z.B. im Dateimanager die Informationen über die Zugriffsrechte und Besitzrechte von Dateien aus Effizienzgründen als Fakten in der Situation repräsentiert.

Die meisten modellbasierten Ansätze stellen eine objektorientierte Darstellung von Anwendungsfunktionen und Interaktionen zur Verfügung, die allerdings nur eine eingeschränkte Formulierung von semantischen Bedingungen erlaubt. Zum Beispiel können in UIDE, analog zu unseren Vorbedingungen, semantische Bedingungen an die Argumente einer Interaktionen bzw. einer Funktion geknüpft werden [18]. Als Bedingungen lassen sich Eigenschaften an das Argument formulieren (Typ des Objektes, Ungleichheit zu einem anderen Argument). In UIDE ist es jedoch nicht möglich, Eigenschaften an Attribute eines Argumentes zu formulieren, wie es in unserem Dateimanager mit dem Konzept *movable* als Vorbedingung für eine Verschiebe-Aktionsregel realisiert wurde (pers. Kommunikation N. Sukaviriya).

# 3 Kontextsensitive Hilfe

Handhabbarkeit und Benutzerfreundlichkeit von Anwendungssystemen wird zum großen Teil durch die Selbstbeschreibungsfähigkeit der Benutzungsschnittstelle bestimmt. Unter diesem Begriff fordert die DIN-Norm, da *"[...] jeder einzelne Dialogschritt unmittelbar verständlich ist oder der Benutzer auf Verlangen zu dem*

*jeweiligen Dialogschritt entsprechende Erläuterungen erhalten kann. [...] Beschreibungen sollen situationsabhängig gegeben werden.*" [4]. Die Verständlichkeit einer graphischen Oberfläche wird häufig durch ein einsichtiges Design suggeriert. Die gestalterische Ausdrucksmöglichkeit ist dabei durch die statischen Gegebenheiten des Interaktionsbereichs beschränkt, hingegen kann die situationsgebundene Dynamik einer Interaktion auf diese Weise nur unzureichend beschrieben werden. Diese (dynamische) Selbstbeschreibungsfähigkeit läßt sich jedoch durch ein Hilfesystem verbessern. Die Aufgabe eines Hilfesystems ist, auf Anfrage Einsatzzweck, Leistungsumfang, Funktionsweise und Handhabung des Systems wiederzugeben (vgl. [12]).

Im SUSI-Konzept werden diese Anforderungen an ein Hilfesystem durch folgende Eigenschaften erfüllt: Vermittelt wird ein *benutzerorientiertes Modell* des Gesamtsystems, das durch die Repräsentation der Anwendungslogik gestützt wird. Zusätzlich wird das Hilfesystem mit kontextsensitiven Eigenschaften versehen. *Kontextsensitivität* bedeutet, daß Fragestellungen in Abhängigkeit vom aktuellen Zustand des Systems beantwortet werden und damit auch die vorhandenen Informationsmenge auf den Ausschnitt begrenzt wird, der im aktuellen Zustand für Benutzer und Benutzerin Relevanz hat. Dargestellt werden die aktuellen Möglichkeiten und deren Auswirkungen. Auf diese Weise werden Benutzer und Benutzerin beim Aufbau eines mentalen Modells vom System und seiner Funktionsweise unterstützt.

## 3.1 Funktionalität eines Hilfemoduls

Für den Entwurf eines Hilfesystems müssen Designentscheidungen bzgl. verschiedener Fragen getroffen werden: Wie bzw. wann können Fragen an das Hilfesystem gerichtet werden? Was soll das Hilfesystem beschreiben? In welcher Weise wird das Vorwissen der Fragestellenden berücksichtigt? Wie werden Erklärungen erzeugt?
Hinsichtlich der Problemstellungen, die hier nur ausschnittweise aufgelistet sind, legt das SUSI-Konzept folgende Gestaltung nahe (siehe auch [7, 21]):

Entsprechend den Anforderungen der DIN-Norm [4] und der im vorigen Abschnitt explizierten Forderung nach Kontextsensitivität kann das Hilfesystem Funktionsweise und Handhabung des Anwendungssystems sowohl unabhängig als auch abhängig vom aktuellen Systemzustand beschreiben. Kontextsensitive Erklärungen können u.a. in Bezug auf ein bestimmtes Widget gestellt werden und beschreiben die im aktuellen Zustand möglichen oder unmöglichen Aktionen.
Der Vorgang des Erklärens wird zunächst als ein Kommunikationsprozeß aufgefaßt, der auf einer Sequenz von Fragen basiert (siehe Abbildung 3). Darin haben Benutzer und Benutzerin die Möglichkeit weiterführende bzw. vertiefende Fragen zu einem Hilfetext zu stellen, sofern ihr Erklärungsbedürfnis nicht befriedigt werden konnte. Auf diese Weise wird der Wissensstand der Fragenden berücksichtigt. Diese Sichtweise auf den Vorgang legt eine Hypertext-ähnliche Struktur der Hilfetexte nahe.

Kontextsensitivität kann nur erreicht werden, falls Wissen über die Anwendungslogik adäquat repräsentiert ist. Im SUSI-Konzept wird die Dialogbeschreibung als Wissensbasis genutzt. Die Erzeugung von Hilfetexten ist daher vom Aufbau der Dialogbeschreibungssprache geleitet. Fragestellungen _ insbesondere weiterführende Fragestellungen _ beziehen sich implizit auf die Konzepte der Dialogbeschreibung. Durch diese enge Beziehung zur Dialogbeschreibung wird eine automatische Generierung von Hilfetexten möglich. Wie das Hilfemodul zur Erzeugung von kontextsensitiver Hilfe konzipiert ist, stellt der nächste Abschnitt dar.

## 3.2 Konzeption des Hilfemoduls

Das Hilfesystem adaptiert ein Modell des Anwendungssystems, indem die Struktur der Dialogbeschreibung aufgenommen wird: Die Frage nach möglichen Aktionen läßt sich anhand ableitbarer Aktionsregeln beantworten. Die Nichtausführbarkeit einer bestimmten Aktion läßt sich mit Hilfe der nichterfüllten Fakten oder Nebenbedingungen in der betreffenden Aktionsregel begründen. Das Sprachkonzept der Nebenbedingung in SUSI bietet für die Hilfe den Vorteil, daß für die Anwendung wesentliche Eigenschaften von Objekten zusammengefaßt und erklärt werden können. Damit bilden die Sprachkonzepte der Dialogbeschreibung elementare Erklärungseinheiten für die zu generierende Hilfe. Zu diesem Zweck wird jedem Sprachelement vor Laufzeit des Systems ein Erklärungstext zugeordnet, der den Begriff zunächst unabhängig vom Systemzustand beschreibt.
Die Kontextsensitivität des Hilfesystems, die Bezugnahme zum aktuellen Systemzustand ist dadurch gewährleistet, daß das Hilfesystem Anfragen an den Dialoginterpreter richten kann. Der Dialogmanager stellt zwei Funktionen zu Verfügung, um auf die Anfragen zu reagieren:

- *eine Routine zur "Simulation" einzelner Aktionen*, die die Ausführbarkeit einer Aktion im aktuellen Kontext testet, ohne die Aktion auszuführen;
- *eine Routine zur Evaluation von Nebenbedingung*, die überprüft, ob eine Nebenbedingung im aktuellen Systemzustand erfüllt ist.

Bei der automatischen Generierung von Hilfetexten verknüpfen Generierungsmethoden zur Laufzeit die passenden Erklärungseinheiten entsprechend der Fragestellung zu einem Hilfetext und nehmen die Aktualisierung der zugehörigen Erklärungstexte vor. Der Bezug zum aktuellen Systemzustand wird durch die Anfragen an den Dialogmanager hergestellt.

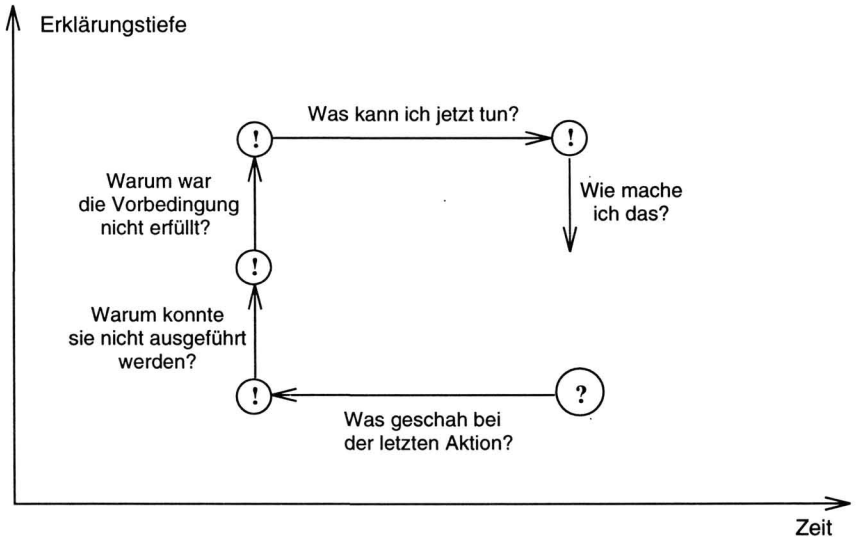


Abb. 3: Sequenz von Fragestellungen

Diese Hilfe unterstützt u.a. den Benutzer beim Aufbau eines mentalen Modells von der Funktionsweise der Anwendung, indem Abhängigkeiten der Interaktionen vom aktuellen Systemzustand erläutert werden und setzt zum Beispiel auch die Hemmschwelle zum Explorieren herab, indem die Auswirkungen einzelner Funktionen skizziert werden (vgl. *Sichten* und *Neutral-Modus* in [14]).

# 4 Realisierung

Zur SUSI Dialogbeschreibungssprache wurde ein Dialogmanager realisiert, der die hier vorgestellten Funktionen realisiert und als graphische Komponente in der aktuellen Version den Common Lisp Interface Manager (CLIM) nutzt [7]. Der Dateimanager ist als Anwendungsbeispiel mit diesem System realisiert worden; eine Adreßdatenbank, die komplexere semantische Beziehungen zwischen den einzelnen Anwendungsobjekten erlaubt, ist als nächste Anwendung vorgesehen. Geplant ist außerdem eine C++ Version des Dialogmanagers, der dann in Verbindung mit den üblichen auf C oder C++ aufsetzenden Interface Managern als UIMS-System genutzt werden kann. Die Konstruktion einer Dialogsteuerung, die Eingabe der als Hypertextschablone organisierten Hilfetexte oder auch die Umsetzung einer aufgabenorientierten Beschreibung in eine funktionale Beschreibung durch Situationstypen (vgl. Sichten in [1]) wird momentan noch nicht durch spezielle Werkzeuge unterstützt.

# 5 Mögliche Erweiterungen

Die situationsorientierte Dialogbeschreibung stellt ein funktionales Modell zur Anwendung zur Verfügung. Im folgenden wollen wir zwei Aspekte skizzieren, die durch diese expliziten Dialogbeschreibung ermöglicht werden. Die Dialogbeschreibung kann auf Mängel hin analysiert werden und kann Daten zur Evaluation der Anwendung bereitstellen. Zudem kann die dynamische Anpassung der Anwendung durch eine Ergänzung der Dialogbeschreibung vorgenommen werden.

## 5.1 Evaluation und Analyse

Ein Vorteil expliziter Dialogsteuerungen innerhalb eines UIMS ist die Möglichkeit, die Dialogstruktur zu evaluieren und zu analysieren. Für die Evaluation einer Anwendung und einer Oberfläche werden die Interaktionen des Benutzers mit der Anwendung abgespeichert. Da neben den primitiven Ereignisse der einzelnen Widgets auch die angestoßenen Anwendungsfunktionen (Aktionsregeln) von der Dialogsteuerung abgespeichert werden können, können effizienzorientierte Keystroke-Level- und GOMS-Analysen (siehe [3]) einfach durchgeführt werden. Die Zusammenfassung mehrerer Interaktionen zu einzelnen (kognitiven) Handlungen kann auch hier nicht automatisch geleistet werden, da innerhalb SUSI kein aufgabenorientiertes Modell vorliegt (vgl. [2]).

Eine Analyse der Dialogsteuerung kann mehrere Zielsetzungen haben. Fragestellungen nach der Erreichbarkeit von bestimmten Systemzuständen oder der Interaktionslänge für bestimmte Aufgaben können durch eine Analyse der Aktionsregeln beantwortet werden. Softwareergonomisch motivierte Fragestellungen, wie die der max. Anzahl der Schritte um bestimmten Aktionen ausführen zu können (Undo, Hilfe, Ende etc.) oder die Länge von ununterbrechbaren Sequenzen, können ebenfalls untersucht werden. Durch Rückwärtsverkettung von Aktionsregeln zur Untersuchung von Situationsveränderungen kann die kombinatorische Explosion für die Analysen eingeschränkt werden.

## 5.2 Adaptierbare Oberflächen

Adaptierbare Steuerungen werden z.B. bei Dateimanagern eingesetzt. Hier können weitere Dateitypen und diesen Typen zugeordnete Aktionen definiert werden. Tabellenkalkulationen sind eine weitere Klasse von Anwendungen, die von den Benutzern erfolgreich selber programmiert bzw. adaptiert werden. Da in SUSI eine explizite und strukturierte Darstellung der Dialogsteuerung vorliegt, die nicht kompiliert, sondern interpretiert wird, ist auch mit SUSI eine dynamische Adaptierung der Steuerung in weitreichendem Maße möglich. Durch Veränderung oder Erweiterung der Aktionsregeln können weitere oder komplexere Interaktionen durch den Benutzer definiert werden. Da die Interaktionen zu einem Widget zur Interpretation an die Dialogsteuerung weitergegeben werden, kann die Funktionalität

der Anwendung auf einfache Weise erweitert werden, indem Aktionsregeln eingeführt oder verändert werden, die diese Interaktion bearbeiten. Solange die zur Verfügung gestellte Oberfläche (die Präsentationskomponente) und die Anwendungsfunktionen für die Erweiterungen ausreichend sind, ist dies im Gegensatz zu klassisch realisierten Anwendungen durch die Veränderung der Dialogbeschreibung einfacher möglich. Ein gutes Verständnis der Funktionsweise der genutzten Anwendungsfunktionen sowie der Dialogsteuerung ist allerdings nötig, damit Benutzer selbständig die Oberfläche erweitern können. Durch eine Markierung sollten wesentliche Aktionsregeln und kritische Anwendungsfunktionen vor Benutzerveränderungen geschützt, und so der Anpassungsbereich eingeschränkt werden.

# 6 Zusammenfassung

User Interface Management Systeme stellen einen wesentlichen Fortschritt für die Entwicklung von Oberflächen dar, indem sie eine adäquate Auftrennung zwischen Oberflächenentwicklung und Anwendungsentwicklung ermöglichen und so den unterschiedlichen Anforderungen an Oberflächenprogrammierung und Anwendungsprogrammierung Rechnung getragen werden kann. Die hier vorgestellte Dialogbeschreibung ermöglicht die Integration von Anwendungswissen (insbesondere der Anwendungslogik) in die Dialogsteuerung und ist in der Lage, auch komplexe *direkt manipulative* Interaktionen adäquat in der Oberfläche zu bearbeiten.

Durch die Faktenbasis und die Nebenbedingungen wird die objektorientierte Präsentationskomponente um eine logische Sicht zur Beschreibung von Oberfläche und Anwendungszustand erweitert. Die Interpretation von Interaktionen erfolgt durch Aktionsregeln, die durch die Nachbedingungen die Zustandsveränderungen von Anwendung und Oberfläche beschreiben. Dadurch stellt die Dialogbeschreibung ein funktionales Modell der Anwendung zur Verfügung. Diese logische Sicht ist eng verwandt mit den stärker objektorientierten Modellbeschreibungen, wie in UIDE oder HUMANOID, erlaubt jedoch eine einfachere bzw. ausdrucksstärkere Formulierung von semantische Bedingungen und der zugrundeliegenden Umgebung (Situation) zur Interpretation von Interaktionen. Allerdings verfolgen wir mit unserem Ansatz bewußt keine automatische Konstruktion der Präsentationskomponente aus einem Modell der Anwendung. Wir haben außerdem gezeigt, wie diese Dialogbeschreibung für ein kontextsensitives Hilfemodul genutzt wird und skizziert, wie Analyseverfahren und dynamische Anpassungen der Applikation auf solch eine Dialogsteuerung aufsetzen können. Interessante Erweiterungsmöglichkeiten sind das automatische Sperren und Entsperren von Interaktionen anhand der Ausführbarkeit der Aktionsregeln und die Einbindung von graphischen Constraints, um u.a. eine vereinfachte Realisierung von semantischen Rückkopplungen zu ermöglichen.
Die modulare Beschreibung der Oberfläche durch Situationen, die über Ereignisse kommunizieren können, sind u.E. ein gutes Beschreibungsmittel für den Übergang von einer aufgabenorientierten Beschreibung der Schnittstelle zu einer rechnerorientierten bzw. schnittstellenorientierten Beschreibung ihrer Funktionalität. Im Rahmen

einer größeren Anwendung ist die weitere Evaluation der praktischen Nutzbarkeit und Beschreibungsfähigkeit der Dialogbeschreibung sowie des Hilfesystems durchzuführen und SUSI um Entwicklungswerkzeuge zu erweitern, die das Design und die Konstruktion einer Schnittstelle unterstützen.

# 7 Literaturverzeichnis

[1] Astrid Beck, Christian Janssen: Vorgehen und Methoden für Aufgaben- und Benutzerangemessene Gestaltung von graphischen Benutzungsschnittstellen, in: Wolfgang Coy, Peter Gorny, Ilona Kopp, Constantin Skarpelis (Hrsg.): Menschengerechte Software als Wettbewerbsfaktor. Forschungsansätze und Anwenderergebnisse aus dem Programm "Arbeit und Technik", Teubner, Stuttgart, 1993, S. 200-221.

[2] Michael Byrne, Scott Wooe, Piyawadee Sukaviriya, James Foley, David Kieras: Automating Interface Evaluation, in: Proccedings of CHI '94, Addison Wesley, Ney York, 1994, S. 232-237.

[3] Stuart K. Card, Thomas P. Moran, Alan Newell: The Psychology of Human-Computer Interaction, Hillsdale, NJ, Lawrence Erlbaum Associates, 1983.

[4] DIN-Norm 66234 Teil 8, 1991.

[5] James D. Foley, Won Chul Kim, Srdjan Kovacevic, Kevin Murray: UIDE - An Intelligent User Interface Design Environment, in: J. Sullivan, S. Tyler (Hrsg.): Architectures for Intelligent Interfaces: Elements and Prototypes, Addison Wesley, MA, 1991.

[6] Stefan Hügel: Logische Modellierung von Benutzungsschnittstellen - Realisierung einer Sprache zur Beschreibung von Kontexten und Aktionen, Diplomarbeit, Institut für Informatik und Gesellschaft, Universität Freiburg und Institut für Logik, Komplexität und Deduktionssysteme, Universität Karlsruhe, 1994.

[7] Stefan Hügel, Kirsten Winter, Friedrich Strauß: SUSI - Situationsorientierte Modellierung von Benutzungsschnittstellen mit integrierter kontextsensitiver Hilfe, IIG-Bericht 9/94, Institut für Informatik und Gesellschaft, Universität Freiburg, 1994

[8] Rolf Ilg, Jürgen Ziegler: Direkte Manipulation, in: Helmut Balzert, Heinz Hoppe, Reinhard Oppermann, Helmut Peschke, Gabriele Rohr, Norbert Streitz (Hrsg.): Einführung in die Software-Ergonomie, de Gruyter, 1988.

[9] Christian Janssen: Dialognetze zur Beschreibung von Dialogabläufen in graphisch-interaktiven Systemen, in: [15], S. 67-76.

[10] Brad A. Myers: Creating User Interfaces by Demonstration, Boston, MA, 1988.

[11] Brad A. Myers: State of the Art in User Interface Software Tools, in: H. Rex Hartson, Deborah Hix (Hrsg.): Advances in Human-Computer Interaction, Vol. 4, Ablex Publishing Corporation, Norwood, New Jersey, 1993.

[12] Reinhard Oppermann, Bernd Murchner, Harald Reiterer, Manfred Koch: Softwareergonomische Evaluation: Der Leitfaden EVADIS II, de Gruyter, Berlin, 1992.

[13] Dan R. Olsen Jr.: User Interface Management Systems: Models and Algorithms, Morgan Kaufmann Publishers, San Mateo, California, 1992.

[14] Hansjürgen Paul: Das Explorative Modell als konzeptioneller Ansatz zur Gestaltung interaktiver Systeme, in: [15], S. 77-86.

[15] Karl-Heinz Rödiger (Hrsg.): Software-Ergonomie '93 - Von der Benutzungsoberfläche zur Arbeitsgestaltung, Teubner, Stuttgart, 1993.

[16] Friedrich Strauß: Situation Oriented Description of User Interfaces, in: Patrick Brezillon (Hrsg.): Proceedings of the IJCAI'93 (13th International Joint Conference on Artificial Intelligence) Workshop on "Using knowledge in its Context", Rapport Interne du LAFORIA 93/13, Institut Blaise Pascal, Paris, 1993.

[17] Friedrich Strauß: Contextsensitive Help-facilities in GUIs through Situations, in: T. Grechenig, M. Tscheligi (Hrsg.): Proceedings of the VCHCI'93 (Vienna Conference on Human Computer Interaction), Lecture Notes in Computer Science 733, Springer-Verlag, 1993, S. 79 - 90.

[18] Piyawadee "Noi" Sukaviriya, James Foley, Todd Griffith: A Second Generation User Interface Design Environment: The Model and The Runtime Architecture, in: Proceedings of Interchi '93, 1993, S. 375-382.

[19] Bernhard Trefz: Diamant - A User Interface Management System for Object Oriented Interfaces, in: Margaret Galer, Susan Harker, Jürgen Ziegler: Methods and Tools in User-Centered Design for Information Technology, North Holland, Amsterdam, 1992, S. 319-343.

[20] Jean Vanderdonckt, Missiri Ouedraogo, Banta Ygueitengar: A Comparison of Placement Strategies for Effective Visual Design. In: G. Cockton, S.W. Draper, G.R.S. Weir: Proceedings of HCI '94, Cambridge University Press, 1994, S. 125 - 144.

[21] Kirsten Winter: Kontextsensitive Benutzerunterstützung in graphischen Benutzerschnittstellen, Diplomarbeit, Institut für mathematische Maschinen und Datenverarbeitung, Universität Erlangen-Nürnberg, 1994.

Friedrich Strauß, Stefan Hügel, Kirsten Winter, Britta Schinzel
Universität Freiburg
Institut für Informatik und Gesellschaft, Abt. Modellbildung und soziale Folgen
Friedrichstraße 50, D-79098 Freiburg i. Brsg., Telefon 0761/203-4954
{frieder, stefan, kirsten, britta}@modell.iig.uni-freiburg.de