# Wissensbasiertes Generieren von Benutzerschnittstellen

Klemens Waldhör, Nürnberg

**Zusammenfassung**

Der Beitrag schildert den im Rahmen der Softwarearchitektur BASAR entwickelten Dialogmanager. Der Dialogmanager besteht aus einer Menge von Experten, die Wissen in Form von Regeln und statischen Wissensbasen beinhalten. Diese Experten kooperieren, um dem Benutzer eine optimale Mensch-Maschine-Schnittstelle zur Verfügung zu stellen. Die Realisierung der für das Benutzermodell und ergonomische Wissen zuständigen Experten wird geschildert.

## 1. Einleitung

Die Programmierung von Benutzerschnittstellen benötigt bei der Entwicklung von Programmsystemen und Prototypen einen nicht unwesentlichen zeitlichen und personellen Aufwand. Dies führt dazu, daß jeder Programmierer seine eigenen Techniken und Module zur effizienten Realisierung solcher Schnittstellen entwickelt und verwendet. Daraus resultiert eine für den Benutzer verwirrende Verschiedenartigkeit von Bedienungsoberflächen. Moderne Entwicklungsumgebungen bieten zwar einen hohen Standard für die Erstellung von Benutzeroberflächen (Fenstertechnik, direkte Manipulation etc.), stellen aber bedingt durch diese Möglichkeiten eine große Anzahl von Ein/Ausgabemöglichkeiten zur Verfügung (z.B. Macintosh, Symbolics, X-Windows und deren Toolkits). Sie tragen also nicht generell zur Verminderung des Programmieraufwandes bei, sondern erhöhen ihn sogar durch die zur Verfügung gestellten Möglichkeiten.

Nun gibt es aber bereits umfangreiche Literatur zum Thema, wie der Programmierer gute Benutzerschnittstellen erstellen kann (z.B. Shneiderman, 1987; Heeg, 1988; Lauter, 1987; Norman & Draper, 1986; DIN 66234/1, DIN 66290/8). In der Praxis werden diese Richtlinien aber kaum gelesen oder beachtet, vor allem nicht von der Zielgruppe, für die sie bestimmt sind. Dies liegt unter anderem in der Allgemeinheit und Abstraktheit der Richtlinien, die hier aufgestellt werden. Für die praktische Anwendung müßte der Programmierer diese wieder auf einer sehr viel tieferen Ebene operationalisieren. Die meisten Programmierer vertrauen daher lieber mehr auf ihre Erfahrung und Fingerspitzengefühl als auf solch vage Richtlinien. Es fällt ihnen meist nicht schwer zu begründen und ist auch für jeden, der selbst Programme erstellt, einsichtig, warum diese Richtlinien nicht eingehalten werden können. Daher sind solche Richtlinien meist nur in Prototypen von Forschungslaboratorien realisiert und der Anwender bekommt diese nie zu Gesicht.

## 2. Der Dialogmanager

### 2.1 Interaktionsaufträge und Interaktionsobjekte

Um den Entscheidungsprozeß des Programmierers bei der Auswahl geeigneter Interaktionsformen zu unterstützen oder diesen Prozeß möglichst weitgehend zu automatisieren, wurde im Rahmen des vom BMFT geförderten WISDOM-Projektes die **Software-Architektur BASAR** entwickelt (**BA**sic **S**oftware **AR**chitecture, Beschreibungen von BASAR finden sich in Märtin & Waldhör, 1988; Waldhör & Rupietta, 1987; Waldhör & Rosenow, 1988), die eine abstrakte und von einem Fenstersystem unabhängige Spezifikation von Benutzerschnittstellen ermöglicht und den Programmierer in großem Umfange vom Generieren der Benutzerschnittstelle entlastet. Dies geschieht mittels **Dialogaufträge**. Dialogaufträge verwenden **Interaktionsobjekte**, die vom Dialogmanager in die entsprechenden E/A-Befehle des Zielrechners (bzw. Zielsprache) transformiert werden. Interaktionsobjekte (z.B. Herczeg, 1986) sind z.B. Fenstertypen, Eingabetechniken, allgemein alle Kommunikationsformen zwischen Benutzer und Applikation. Der Programmierer gibt nur an, welche Ein/Ausgabetechniken er

benötigt, nicht aber wie diese realisiert werden sollen. Der Programmierer muß in einem Dialogauftrag nicht alle vorgesehenen Möglichkeiten spezifizieren, fehlende Informationen werden automatisch generiert und in den Auftrag eingefügt. Der Programmierer soll sogar sowenig als möglich spezifizieren, um eine mit anderen Anwendungen konsistente Oberfläche zu erhalten.

## 2.2 Repräsentation des Dialogmanagers

Um die Einbindung von Richtlinien zur Gestaltung von Benutzeroberflächen zu ermöglichen, werden Teile des Dialogmanagers in Form von Regeln und notwendiges Faktenwissen in Wissensbasen gespeichert. Der Programmierer braucht daher die Richtlinien nicht selbst zu kennen oder zu operationalisieren. Er verwendet diese bei der Benutzung von Dialogaufträgen implizit. Durch die einfache Darstellung in Regelform kann das Systemverhalten an neue Situationen und an den Benutzer angepaßt werden, ohne daß der Anwendungsprogrammierer Änderungen in seiner Applikation vornehmen muß.

Durch die Einbindung ergonomischen Wissens in Form von **Dialogregeln** wird versucht, dem Programmierer die Entscheidung, welches spezielle Interaktionsobjekt er in einer entsprechenden Situation verwenden soll, zu erleichtern bzw. abzunehmen. Er soll bereits während der Programmierung Unterstützung über spezielle Interaktionsobjekte erhalten. Andererseits wird versucht, während der Programmlaufzeit ein für den Benutzer optimales Interaktionsobjekt auszuwählen (z.B. ob ein Menü mit oder ohne Auswahlzwang verwendet wird, ob und wie Menüeinträge sortiert werden). BASAR stellt auf diese Weise dem Benutzer eine einheitliche und konsistente Benutzeroberfläche zur Verfügung.

Der Dialogmanager besteht aus einer Menge von Interaktionsexperten. Jeder Experte selbst besteht wieder aus vier verschiedenen Unterexperten - **Benutzermodell, Dialogwissen, Systemselbstbild und Interaktionsmethoden.** Der Dialogmanager entscheidet bei einem Interaktionsauftrag, ob ein Experte etwas zum Auftrag beitragen kann oder nicht. Jeder dieser Unter-

experten besteht aus einer Menge von Regeln und entsprechenden Interaktionsklassen, die in einer Wissensbasis repräsentiert sind. Diese Wissensbasis enthält einerseits eine Taxonomie der möglichen Interaktionsobjekte als auch in den einzelnen Interaktionsklassen (diese repräsentieren die Interaktionsobjekte) die Beschreibung der Interaktionsobjekte in Form von Frames. Durch die Änderung dieser Wissensbasis können neue Interaktionsobjekte eingeführt werden (z.B. ein neues Eingabegerät integriert werden).

Abbildung 1 zeigt die Repräsentation des Interaktionsobjektes **Fenster:** Die Attribute Spezialisierung und Generalisierung werden bei der Generierung automatisch ermittelt und sind hier der Übersicht halber eingefügt. Die Regelliste wird bei der Definition der Regel (getrennt vom Interaktionsobjekt) festgelegt und wird Interaktionsobjekt getrennt verwaltet. Abbildung 2 zeigt ein Beispiel eines solchen Fensters.

**Interaktionsobjekt:** Fenster

| Attribut: | Wert: | Kommentar: |
|---|---|---|
| area-id: | <String> | Interne Kennung |
| Titel: | <String> | |
| Höhe: | <Pixel> | |
| Breite: | <Pixel> | |
| Eingabe: | <Fenster> | Verhalten bei Eingabe |
| Iconfunkt.: | <Icons> | Definition, welche Fensteroperationen unterstützt werden (verkleinern ...) |
| Mausaktionen: | <Aktionen> | Verhalten bei Verwendung der Maus |
| Regel: | <Regelliste> | Die mit diesem Interaktionsobjekt verbundenen Regeln |
| Spezial.: | <Special> | Verfeinerungen (z.B. statische, dynamische Fenster |
| General.: | Interaktions-objekt | Generalsierung dieses Interaktionsobjektes |

**Abbildung 1:** Interaktionsobjekt Fenster

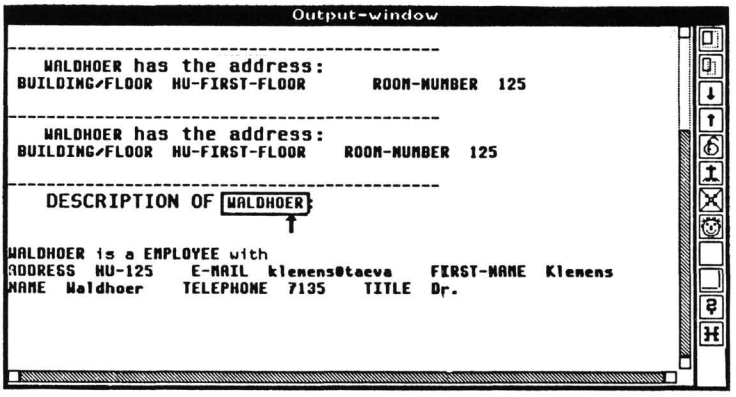


**Abbildung 2:** Instanz des Interaktionsobjektes Fenster

## 2.3 Benutzermodell

Das Benutzermodell enthält Regeln, wie ein Auftrag auf Grund von Informationen über den Benutzer ausgeführt werden soll. Dabei werden zwei Typen unterschieden: Informationen, die durch explizite Anweisungen des Benutzers generiert werden (der Benutzer kann z.B. die aktuelle Benutzeroberfläche abspeichern und später wieder verwenden) und in einer Benutzerwissensbasis gespeichert werden, und (prozedurales) Wissen, das auf Grund der im Benutzermodell vorhandenen Regeln verwendet wird. So wird z.B. der Typ des Fensters, das dem Benutzer in einer Applikation dargeboten wird, durch folgenden Regelkomplex (Abbildung 3) auszugsweise beschrieben.

Zu beachten ist, daß sich die Regeln des Benutzermodells nur auf Objekte beziehen, die vom Dialogmanager verwaltet werden können. Applikationsspezifische benutzerabhängige Festlegungen, die durchaus auch Auswirkungen auf Interaktionsobjekte haben können, werden nicht mitverwaltet. Der Grund dafür ist der Unterschied in der Repräsentation. Während die Benutzerschnittstelle relativ abstrakt spezifiziert wird, werden die meisten Applikationen konventionell auf Anweisungsebene programmiert. Eine explizite

Darstellung der Applikation auf einem höheren Niveau (etwa in Form von Plänen, über die ein entsprechender Planungsprozeß ablaufen kann) könnte hier Abhilfe schaffen.

| **Benutzermodell-Regel:** Auswahl des Fenstertyps |
|---|
| Wenn in mehr als 60% aller Fälle die vom Benutzer explizit festgelegten Fenster vom Typ Scrollfenster sind, dann verwende ein Scrollfenster. |
| Wenn in weniger als 40% aller Fälle Fenster mit Scrollbalken verwendet werden, verwende ein Fenster ohne Scrollbalken. |
| Wenn eine Änderung des Fenstertyps erfolgen soll, teile dies dem Benutzer mit und frage ihn, ob er damit einverstanden ist. |
| **Abbildung 3:** Fenstertypauswahl |

Prinzipiell können zwei Arten von Benutzermodellregeln unterschieden werden: **Allgemeine Regeln** wie die oben geschilderte Regel, die für alle Benutzer zutreffen und die das Benutzermodell anzuwenden versucht, und **spezielle Regeln**, die nur für ganz gewisse Benutzer zutreffen. Dies kann etwa durch eine Generalisierung oder Spezialisierung von allgemeinen Regeln erfolgen. Nehmen wir an, daß der Benutzer "Mayer" bei seinen Applikationen immer Menüs verwenden will, die alphabetisch sortiert sind. Verwendet Mayer nun eine neue Applikation, so versucht das Benutzermodell dieses Wissen zu verallgemeinern und folgende neue Regel zu generieren, die aber nur für diesen Benutzer gilt:

| **Neue Benutzermodell-Regel:** Sortierung der Menü-Einträge |
|---|
| Wenn für den Benutzer Mayer ein Menü angezeigt werden soll, dann sollen die Menüeinträge alphabetisch sortiert werden. Dies muß von Mayer bestätigt werden. |
| **Abbildung 4:** Generieren einer neuen Regel |

Diese neuen Regeln können durch die Anwendung einer Metaregeln generiert werden. Diese lautet stark vereinfacht:

| **Meta-Benutzermodell-Regel:** Neue Benutzer-Regel |
|---|
| Wenn ein Benutzer für ein bestimmtes Interaktionsobjekt und ein bestimmtes Attribut dieses Interaktionsobjektes häufig denselben Wert verwendet und dies über mehrere Applikationen konsistent auftritt, generiere für dieses Attribut des Interaktionsobjektes eine für diesen Benutzer spezifische Regel. |
| Wenn eine neue benutzerspezifische Regel generiert wird, muß der Benutzer immer um Bestätigung dieser Regel gefragt werden. |
| **Abbildung 5:** Regel zur Generierung von Regeln |

Ähnliche Regeln existieren für andere Interaktionsobjekte. Änderungen, die durch das Benutzermodell durchgeführt werden sollen (z.B. Wechsel der Interaktionsform), müssen immer vom Benutzer bestätigt werden. Das System führt dazu einen expliziten Dialog mit dem Benutzer. In obiger Regel fragt es z.B. nach, ob der Benutzer mit dieser Änderung einverstanden ist. Wenn nicht, kann der Benutzer selbst bestimmen, was geschehen soll. Dem unerfahrenen Benutzer wird zusätzlich noch erklärt, was die Operation bedeutet, z.B. welche Operationen er in einem Fenster ohne Scrollbalken nicht durchführen kann. Als unerfahrener Benutzer wird ein solcher eingestuft, der noch nie mit BASAR gearbeitet hat. Es ist klar, daß diese Definition nicht unbedingt zutreffend ist, aber wägt man ab, ob man lange Dialoge über die Erfahrung des Benutzers mit Computern führt oder lieber einem bereits erfahrenen Benutzer mit einer Hilfetafel belästigt, erscheint der zweite Weg plausibler. Der Benutzer kann die Hilfe-Option abschalten, falls er sie nicht benötigt.

### 2.4 Dialogwissen

Das ergonomische Wissen des Experten Dialogwissen wird in Situationen verwendet, in denen aus dem Benutzermodell keine Informationen generiert werden können. Generell haben Regeln des Benutzermodells eine höhere Priorität als die des Dialogwissens. In Situationen, in denen sowohl das Benutzermodell als auch das Dialogwissen zu einem Auftrag etwas beitragen können, wird eine

Kontrollregel verwendet, die obige Konfliktlösung durchführt und sich für das Benutzermodell entscheidet, da davon ausgegangen wird, daß der Benutzer die letzte Entscheidung über seine Schnittstelle treffen will. Ergonomisches Wissen ist ebenfalls in Form von Regeln abgespeichert, z.B.

| **Dialogwissen-Regel:** Formulareinträge kennzeichnen |
|---|
| Wenn ein Formular angezeigt werden soll, invertiere die Felder, in denen Einträge gemacht werden können. |
| **Abbildung 6:** Beispiel einer Dialogwissen-Regel |

Es können natürlich auch konkurrierende Regeln innerhalb eines solchen Experten vorhanden sein. So kann z.B. eine Regel zur Größengestaltung eines Fensters auf Grund optischer Kriterien mit einer anderen Regel kollidieren, die besagt, daß sich zwei Fenster möglichst wenig überlappen sollen, um dem Benutzer immer möglichst den ganzen Fensterinhalt anzeigen zu können. Die Konfliktlösung wird von einer speziellen Komponente des Dialogmanagers - dem **Scheduler** - auf Grund von Kontrollregeln getroffen. Diese Kontrollregeln sind von den oben schon erwähnten Kontrollregeln zu unterscheiden, da dort die Kontrollregeln zwischen verschiedenen Experten verwendet werden, während sie hier innerhalb eines Experten zwischen Subexperten verwendet werden.

Die beiden restlichen Experten sind für die Realisierung der Interaktion auf dem Bildschirm verantwortlich und in diesem Zusammenhang weniger wichtig.

## 3. Vorteile wissensbasierter Benutzerschnittstellen

Aus der regelbasierten Darstellung des ergonomischen Wissens und des Benutzermodells ergeben sich mehrere Vorteile. Als erstes lassen sich die Regeln leicht ändern und an neue Gegebenheiten anpassen. Ebenso lassen sich neue Interaktionsobjekte durch eine Definition (als Frames) in der Dialogwissensbasis einführen und deren Verhalten durch Regeln beschreiben. Durch die in BASAR

vorhandene Möglichkeit, die Kontrollregeln zu verändern, die die Anwendung verschiedener Regeln steuern, können verschiedene Benutzeroberflächen und deren Verhalten leicht generiert, gesteuert und verglichen werden. Durch die explizite Repräsentation der Benutzerschnittstelle im Systemselbstbild kann auf gewisse Vorlieben des Benutzers geschlossen werden und daraus neue Regeln generiert und in das Benutzermodell integriert werden.

## 4. Implementierung des Dialogmanagers

Das hier beschriebene System wurde auf SYMBOLICS in COMMON-LISP implementiert und im Rahmen zweier prototypischer Systeme getestet. ELO-QUERY ist eine wissensbasierte Abfrageschnittstelle, die dem Benutzer Zugang zu vielfältigen Informationen über eine Organisation ("Elektronisches Organisationshandbuch") ermöglicht (Rosenow-Schreiner & Waldhör, 1988). Im Rahmen des Planungssystems LUPINO, das Vorgänge im Bürobereich plant, wurde das Konzept ebenfalls verwendet. Beide Tests der Funktionalität des Konzeptes verliefen erfolgreich.

In der derzeitigen Implementierung sind die Regeln noch direkt durch LISP-Funktionen abgebildet, die Interaktionsobjekte selbst in einer frameartigen Struktur beschrieben. Um eine leichtere Änderbarkeit der Regeln und der Wissensbasis zu ermöglichen, soll der Dialogmanager mit Hilfe der TA-Wissensrepräsentation LUIGI und der von ihr zur Verfügung gestellten Produktionsregelsprache modelliert werden. Dies betrifft hauptsächlich das Benutzermodell und das Dialogwissen, während die anderen beiden Experten in LISP repräsentiert bleiben. Eine Implementierung des Systems auf UNIX unter X-Windows ist ebenfalls geplant.

## 6. Literatur

**DIN 66234/8 (1988).** Bildschirmarbeitsplätze. Grundsätze der Dialoggestaltung, Beuth.

**DIN 66290/1 (1986).** Gestaltung von Maskenorientierten Dialogsystemen, Entwurf, Beuth.

**Heeg, F.J. (1988).** Empirische Software-Ergonomie, Springer, Berlin.

**Herczeg, M. (1986).** Eine objektorientierte Architektur für wissensbasierte Benutzerschnittstellen, Dissertation, Fakultät für Mathematik und Informatik der Universität Stuttgart.

**Lauter, B. (1987).** Software-Ergonomie in der Praxis, Oldenbourg, München.

**Märtin, C., Waldhör K. (1988).** BASAR - A Blackboard Based Software-Architecture, in: Kodratoff, Y. (Ed.) (1988). Proc. of the 8th ECAI Conference, Pitmann, London.

**Norman, D.A., Draper, S.W. (Eds.) (1986).** User Centered System Design, Lawrence Erlbaum, London.

**Rosenow-Schreiner, E. , Waldhör, K. (1988).** ELO-QUERY - Eine Abfrageschnittstelle für eine Organisationswissensbasis, Arbeitsbericht AB-TA-88-05, TA Triumph-Adler AG, Nürnberg, Juli 88.

**Shneiderman, B. (1987).** Designing the User Interface, Addison Wesley, Amsterdam.

**Waldhör, K., Rosenow, E. (1988).** Spezifikation einer Schnittstelle zwischen Anwendungs- und Dialogschicht, Arbeitsbericht AB-TA-87-02, TA Triumph-Adler AG, Nürnberg, Mai 88.

**Waldhör, K., Rupietta W. (1987).** Auftragsbearbeitung in der Dialogschicht, Arbeitsbericht AB-TA-87-03, TA Triumph-Adler AG, Nürnberg, August 87.

Anschrift des Autors:
**Dr. Klemens Waldhör**
TA Forschung / EF13
TA Triumph Adler AG
Hundingstr. 11b
D-8500 Nürnberg
e-mail:klemens%taeva@unido.uucp

---

Diese Forschungsarbeit entstand im Rahmen des vom BMFT geförderten WISDOM-Projektes.