User Interface Management Systeme

Michael Herczeg, Klemens Waldhör

Zusammenfassung

Der Beitrag schildert den Aufbau von modernen User Interface Management Systemen. Nach einer allgemeinen Beschreibung von Anforderungen an moderne Benutzerschnittstellen werden zwei Arten von Benutzerschnittstellenarchitekturen vorgestellt: Das Blackboardsystem BASAR, das verschiedene Wissensquellen zur Generierung einer Benutzerschnittstelle verwendet und der Benutzerschnittstellenbaukasten USIT, mit dem verschiedene komplexe Interaktionsobjekte und Techniken nach einem einfachen Schema zusammengestellt werden können.

1. Einleitung

Wie eine Vielzahl anderer Computeranwendungen werden auch Bürosysteme in Zukunft als hochinteraktive Systeme bereitgestellt werden. Dies gestaltet sich für den Anwendungsprogrammierer aus den folgenden Gründen als umfangreiche und komplexe Aufgabe:

1) Die Anforderungen an Benutzerschnittstellen sind kaum formal spezifizierbar und ändern sich in Abhängigkeit von Benutzer und Anwendungssystem sehr schnell.

2) Die dezentrale Programmierung von Ein-/Ausgabevorgängen führt zu unübersehbar inkonsistenten Benutzerschnittstellen, die dem Benutzer die Bedienung und dem Programmierer die Änderung schwer machen.

3) Die Programmkonstrukte zum Bau von fortgeschrittenen Benutzerschnittstellen sind sehr elementar. Daher ist die Programmierung ein zeitraubender Vorgang. Fehler sind sehr schwer zu finden und die Seiteneffekte von Änderungen sind oft unüberschaubar.

4) Direkt manipulative Benutzerschnittstellen, die für viele Büroanwendungen anzustreben sind, erfordern eine Vielzahl von Abhängigkeiten zwischen den externen Darstellungen und den internen Anwendungsstrukturen.

5) Solange Benutzerschnittstellen von Anwendungsprogrammierern "miterstellt" werden, bildet sich keine Expertise zur Erstellung qualitativ hochwertiger und effizienter Benutzerschnittstellen.

6) Die peripheren Geräte zur Kommunikation mit dem Benutzer (typischerweise Bildschirm, Tastatur und inzwischen oft Maus und Scanner) variieren sehr stark in ihren Fähigkeiten und Schnittstellen. Standards

hinken den technischen Möglichkeiten meist um fünf bis zehn Jahre hinterher.
7) Benutzerschnittstellen müssen an immer neue Endgeräte angepaßt werden. Dies ist selten ohne große Änderung der Benutzerschnittstelle möglich.

Daraus resultierende, oft unzulängliche Implementierungen gehen neben dem Softwareersteller vor allem zu Lasten der Benutzer. Viele dieser Probleme (insbesondere 2 und 3) sind seit längerer Zeit bekannt und es wurde versucht hier Abhilfe zu schaffen. Die erste Teillösung war die Entwicklung von Programmierhilfsmitteln zur Erstellung stark standardisierter Benutzerschnittstellen, wie **Menü- und Maskengeneratoren**. Der Softwareerstellungsprozeß ließ sich dadurch wesentlich beschleunigen. Der Preis war neben der Erstellung der Hilfsmittel die von da ab geltende Dominanz unmotiviert uniformer, geradezu monotoner Benutzerschnittstellen. Derartige Hilfsmittel prägen auch heute noch das äußere Erscheinungsbild vieler Bürosysteme. Die Benutzer plagen sich durch unübersichtliche Menü- und Maskenhierarchien, die außerdem auch durch zu kleine Bildschirme und unzureichende Hardwareressourcen vielen früheren manuellen Arbeitsmöglichkeiten geradezu anachronistisch nachstehen müssen. Dennoch ist die Konsistenz derartig teilweise automatisch erzeugter Benutzerschnittstellen bezüglich der Darstellung und Interaktionsform als ein erster Fortschritt gegenüber bis dahin "handcodierten" Benutzerschnittstellen anzusehen.

Durch Systeme wie dem Xerox STAR Bürosystem™ (/Smith et al. 82/) und seinen nachempfundenen PC-Lösungen wie dem APPLE Macintosh™ (/Williams 84/) wurden Designmöglichkeiten demonstriert, die heute in vielen - wenn auch sicher nicht allen - Details als richtungsweisende Lösungen bei der Gestaltung zukünftiger Bürosysteme anzusehen sind. Der konzeptionelle Ansatzpunkt ist die Anreicherung der existierenden Menü- und Maskensysteme durch die Einführung reichhaltiger graphischer Oberflächenmodellierungen, die bis zur Nachbildung manueller Schreibtischumgebungen reichen (Desktop-Metapher). Die dialogtechnische Methode wird heute mit dem Begriff **direkte Manipulation** zusammengefaßt und umschreibt den Eindruck des Benutzers mehr oder weniger direkt die Anwendungswelt zu manipulieren (/Shneiderman 83/; /Hutchins et al. 86/). Leider bedeuteten diese Systeme im Vergleich zu den Menü- und Maskensystemen zunächst einen softwaretechnischen Rückschritt. Solche Systeme erforderten einen äußerst hohen Programmieraufwand. Beim Xerox STAR System waren dazu mehr als hundert Personenjahre Entwicklungsaufwand nötig (/Lipkie et al. 82/). Doch auch hier hat man es inzwischen geschafft, durch die Bereitstellung von Programmierhilfsmitteln, die eher Baukastencharakter hatten (sogenannte **User Interface Toolkits**), den Erstellungsaufwand deutlich zu reduzieren, wenn auch mit entsprechenden Einschränkungen an der Vielgestaltigkeit der graphischen Darstellungen und Interaktionskonzepten.

Als derzeit letzter Schritt der Entwicklung spricht man nun in Anlehnung an DBMS (Data Base Management System) von **UIMS (User Interface Management Systemen)**, wobei der Begriff für oft sehr unterschiedliche Zielsetzungen gebraucht wird (/Pfaff 85/; /Hayes et al. 85/; /Draper 86/; /Hill 86/). Die wichtigsten Ziele sind:

- Konsistenz von Benutzerschnittstellen innerhalb einer Anwendung und über mehrere Anwendungen hinweg,
- Formale Spezifikationen und Programmgeneratoren für Benutzerschnittstellen,
- Konzeption der Benutzerschnittstelle als eine Art Softwaremodul mit klar definierten Schnittstellen zum Anwendungssystem,
- Wiederverwendbarkeit von Benutzerschnittstellen,
- Heranbildung von Benutzerschnittstellen-Programmierern mit entsprechender Expertise als Ergänzung zu Anwendungs Programmierern und
- Einbindung unterstützender Komponenten wie History-Mechanismen (Stornieren und Wiederholen von Aktionen), Hilfesystemen, Benutzermodellen (benutzerabhängiges Verhalten des Systems) und Metasystemen (Umgestaltung der Benutzerschnittstelle durch den Benutzer).

Insbesondere durch die Einbeziehung umfangreichen und heterogenen Wissens zum Beispiel beim Bau von Hilfesystemen oder bei der benutzerabhängigen konsistenten Wahl von Interaktionstechniken, spricht man nun auch von **wissensbasierten UIMS**.

Die meisten der den Zielsetzungen auch nur annähernd gerecht werdenden Arbeiten im Bereich von UIMSs sind derzeit noch Forschungsgegenstand und werden im allgemeinen nur für einfache Anwendungsbeispiele eingesetzt. Die zur Erstellung von Benutzerschnittstellen am ehesten nutzbaren Hilfsmittel findet man in Form von Baukastensystemen, die eine sinnvolle softwaretechnische Grundlage für UIMSs darstellen.

Im Projekt WISDOM wurde mit dem im ersten Teil dieses Beitrags beschriebenen System **BASAR** (/Waldhör, Rosenow 88b/) ein UIMS-Rahmensystem erstellt, in das verschiedene Wissensquellen eingebettet werden können. Als programmtechnische Grundlage wurde der ebenfalls im Rahmen von WISDOM entwickelte Benutzerschnittstellen-Baukasten **USIT** (/Herczeg 88/) verwendet, der im Abschnitt 3 dieses Beitrags beschrieben wird. Da das WISDOM-Projekt in der Zeit zwischen 1984 - 1988 durchgeführt wurde, war es nicht möglich auf sich in der Zwischenzeit etablierende Fensterstandards und Toolkits wie X-Window, Open Look, Motif etc. zurückzugreifen. BASAR und USIT dienten als Hilfsmittel zum **Rapid Prototyping** einer Reihe von Büro-Anwendungssystemen (Abfragekomponente ELO-QUERY - siehe Chrapary et al., Vorgangseditor VED - siehe Bauer et al. und der Mailfilter Editor - siehe von Kleist-Retzow et al., alle in diesem Band).

2. Das UIMS-Rahmensystem BASAR

2.1 Allgemeine BASAR-Architektur

Im Rahmen des **WISDOM-Projektes** lautete eines der Ziele, eine Software-Architektur zu entwickeln, die zum einen den Anwendungsprogrammierer in optimaler Weise unterstützt, zum anderen sich auf die individuellen Wünsche des Benutzers einstellt. Das zu entwickelnde System sollte aber nicht nur auf der Benutzerschnittstellenebene stehen bleiben, sondern auch die Integration unterschiedlichster Applikation in einem einheitlichen Rahmen erlauben. So sollte vor allem die Verwendung wissensbasierter Applikationen und "Standard"-Applikationen möglich sein. Zusätzlich sollte ein einfacher Mechanismus zum Austauschen von Daten etc. zwischen solchen Applikationen möglich sein. Wenn sich eine Applikation an einen bestimmten Formalismus (dem Auftragsformalismus) hält, sollte sie integrierbar sein, unabhängig davon, wie die Applikation intern strukturiert ist.

(1) Intelligente und flexible Benutzerschnittstelle

Die Schnittstelle soll sich an den Benutzer anpassen können und von ihm an seinen Arbeitsstil angepaßt werden können. Sind keine entsprechenden Benutzerinformationen vorhanden, soll das System selbst eine ergonomische Benutzerschnittstelle aufbauen können. Die Schnittstelle soll auch leicht erweiterbar sein. Die Einbindung zusätzlicher Interaktionsarten (Sprachein- und ausgabe, diverse Zeigeinstrumente, natürlichsprachliche Eingabe ...) soll ohne größeren Schnittstellenaufwand möglich sein. Wird ein bestimmtes Format für die Kommunikation mit der Benutzerschnittstelle eingehalten, sollte eine neue Interaktionsform ohne Rücksicht auf ihre eigentliche Realisierung integrierbar sein.

(2) Unterstützung der Kombination unterschiedlicher Applikationen

Die Architektur soll zwar primär wissensbasierte Applikationen (Expertensysteme ...) unterstützen, soll aber auch für die Einbindung von Standardapplikationen (Statistikprogramme, Kalkulationsprogramme ...) offen ein. Die Verwendung von wissensbasierten Applikationen in Standardsystemen und umgekehrt soll über eine entsprechende Schnittstelle möglich sein.

(3) Unterstützung von Multitasking und Multiprocessing

Da auch bei Personalcomputern multitaskingfähige Systeme mit neuen Betriebssystemen (OS/2, UNIX) eingesetzt werden, sollte die Architektur diese Möglichkeiten unterstützen. Konkret sollten mehrere Applikationen gleichzeitig lauffähig sein und der Datenaustausch zwischen Applikationen möglich sein.

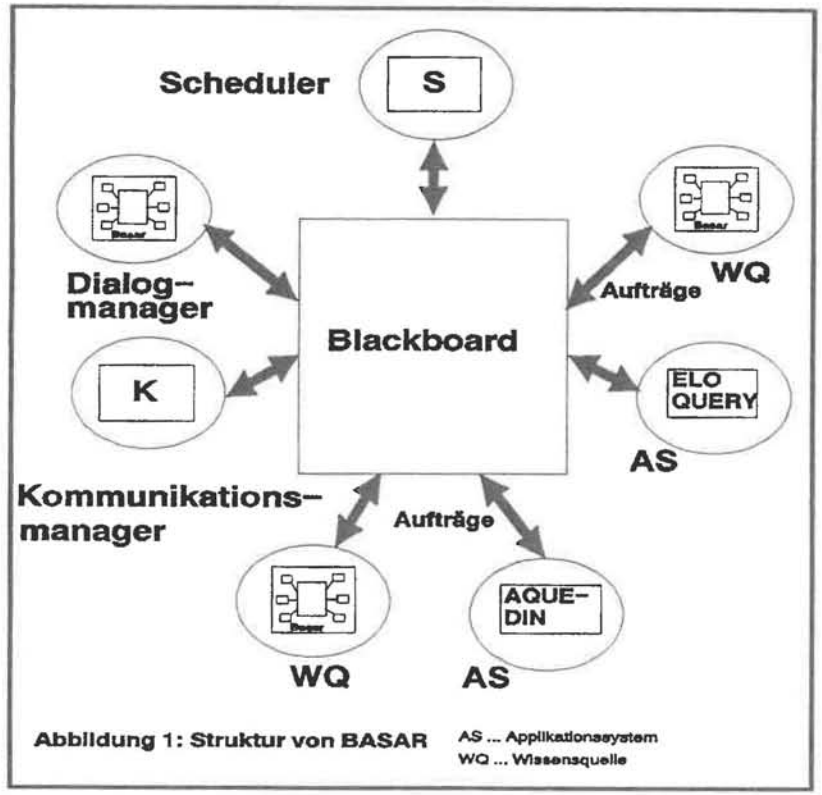

Abb. 1: Struktur von BASAR

(4) Lauffähigkeit auf unterschiedlichen Hard- und Softwaresystemen

Die Architektur soll ohne größeren Aufwand auf unterschiedlicher Hardware (PC, SUN, Symbolics ...) und auf unterschiedlichen Betriebssystemen (MS/DOS, UNIX, Symbolics) lauffähig sein. Zusätzlich sollte die Kommunikation zwischen unterschiedlichen Systemen möglich sein. Ein Benutzer kann an einer bestimmten Maschine seine Eingabe durchführen, während die Applikation auf einer anderen Maschine abläuft. Der Benutzer sollte dies aber nicht explizit durchführen müssen, sondern das System sollte auf Grund entsprechender Wissensbasen diese Zuteilung von Maschine zu Maschine selbst durchführen können.

Zur Realisierung dieser Anforderungen wählten wir einen **Blackboard-Ansatz** Dieser Ansatz hat sich vor allem bei wissensbasierten Systemen als erfolgreich herausgestellt. Das erste System, das nach diesem Paradigma entwickelt wurde, entstand im Rahmen des HEARSAY-II

Projektes (/Engelmore, Morgan 88/). Dieses System war zur Erkennung gesprochener Sprache konstruiert worden. /Balzert 87a/ beschreibt ein Blackboard-Modell zur Realisierung software-ergonomischer Anforderungen, das aus mehreren Schichten aufgebaut ist. BASAR kann als Nachfolgemodell dieses (nur theoretisch beschriebenen) Ansatzes verstanden werden. Balzert beschreibt drei gleichwertig neben einander liegende Schichten, während BASAR, von einer Top-Level-Blackboard ausgehend, dies konsequent als generisches Modell zur Erzeugung beliebig tief geschachtelter Blackboards weitertreibt. Daraus resultiert eine einfachere und vor allem übersichtlichere Architektur nach einem einheitlichen Beschreibungsschema. Im folgenden soll kurz der Aufbau von Blackboard-Systemen geschildert werden.

Blackboard-Systeme bestehen aus einem **zentralem Informationsmedium** (der Blackboard) und einer **Menge von Experten**, die über diese Blackboard miteinander kommunizieren können (Abbildung 1). Unter einem Experten wollen wir ein Programm, eine Menge von Regeln, Funktionen ... etc. verstehen, die für einen (eventuell sehr kleinen) Problembereich eine Lösung finden können oder diesen Problembereich in kleinere Bereiche zerlegen können, sodaß andere Experten in der Lage sind, diesen Teilproblembereich zu bearbeiten. Der Ablauf eines Problemlösezyklus sieht folgendermaßen aus: Ein Experte schreibt ein Problem, das in einer uns hier nicht interessierenden Weise repräsentiert sein soll (etwa als Frames), in die Blackboard. Die anderen Experten erhalten nun Zugriff auf die Blackboard und können das Problem begutachten und versuchen das Problem oder Teile des Problems zu lösen. Dazu kann es notwendig sein, daß ein Experte beim Lösungsversuch ein Teilproblem identifiziert und dieses wieder in die Blackboard stellt. Auf diese Weise kann ein Problem inkrementell gelöst werden.

Blackboardsysteme weisen unter anderem folgende Vorteile auf (die Zahlen in Klammern geben die entsprechende Referenz zu unserer Anforderungsliste an):

Problemlösungsart:

Verschiedenartige Wissensquellen können am Problemlösungsprozeß teilnehmen (1,2).

Problemlösungsvorgehensweise:

Die Lösung eines Problems kann strukturiert werden (durch Blackboardhierarchien) und so der Denkweise eines Experten angepaßt werden (1,2).

Implementierung der Experten:

Jede Wissensquelle (Experte) kann in der Sprache implementiert werden, die ihrem Problembereich angepaßt ist, z.B. Statistikprozeduren in FORTRAN, Regeln in einer regelbasierten Sprache (1,2).

Flexibilität der Ablaufsteuerung:

Verschiedene Kontrollstrategien zur Auswahl von Experten können implementiert und damit experimentiert werden (1).

Programmierung:

Parallele Programmiertechniken können ohne großen Aufwand verwendet werden, da die Wissensquellen entkoppelt sind und der globale Informationsfluß über die Blackboard festgelegt ist. Die zentrale Datenstruktur - die Blackboard - und der Verwaltungsmechanismus lassen sich leicht an die Gegebenheiten der zur Verfügung stehenden Datenstrukturen anpassen (Listen in LISP und C, COMMON-Blöcke in FORTRAN) (3,4).

Ein wichtiges Problem bei der Anwendung der Experten besteht darin, festzulegen a) welche Experten überhaupt zum Problem beitragen können und b) in welcher Reihenfolge deren Beiträge abgearbeitet werden. Es gibt mehrere Ansätze, wie dieses Problem gelöst werden kann. Wir wollen uns hier auf die Lösung mittels eines **Schedulers** beschränken. Ein Scheduler ist eine spezielle Applikation im System, die für die Verwaltung der Expertenbeiträge und der Blackboard zuständig ist. Dazu wird die Blackboard in zwei Partitionen geteilt, die **Domain- und die Controlpartition** (/Hayes-Roth 85/). Die Domainpartition enthält die Problemdefinitionen, während in der Controlpartition die Informationen über die einzelnen Experten gesammelt und bewertet werden.

In BASAR werden die **Problemdefinitionen** als **Aufträge** bezeichnet und als Framestruktur in die Domainblackboard eingetragen. Dazu verwendet eine Applikation oder Wissensquelle einen **"send-order"-Befehl** (Schritt 1 in Abbildung 2). Dieser Befehl spezifiziert das zu lösende Problem.

Ein Frame (/Fikes, Kehler 85/) ist eine Struktur, die aus einer Menge von Slot - Slotwerte Paaren besteht. Wird ein solcher Auftrag in die Domainpartition gestellt, so hat als erster der Scheduler Zugriff auf diesen Auftrag (Schritt 2 in Abbildung 2). Der Scheduler generiert nun einen **Activation-Record** (Schritt 3) in der Controlpartition und gibt dann den Zugriff auf den Auftrag für die anderen Experten frei (Schritt 4). Ein solcher Experte besteht meist aus einem <if>-Teil und einer Liste möglicher Aktionen. Wie schon gesagt, ist aber der Aufbau eines solchen Experten nicht eingeschränkt auf diese Möglichkeit. Der Experte vergleicht nun seinen <if>-Teil mit der Problembeschreibung in der Domainblackboard. Abhängig von diesem Ergebnis, trägt er nun seine möglichen Aktionen im Activation-Record ein.

Haben alle Applikationen und Wissensquellen ihre Überprüfungen beendet, sperrt der Scheduler den Zugriff auf den Auftrag und erzeugt eine **Prozeßumgebung** für diesen Auftrag (Schritt 5). Alle Methoden, Regeln, Funktionen oder Programme, die von den einzelnen Experten in

den Activation-Record geschrieben wurden, werden nun exekutiert. Dazu generiert der Scheduler einen **Execution-Plan.** In diesem Execution Plan wird festgelegt, in welcher Reihenfolge die Beiträge ausgeführt werden. Die aktuelle Implementierung verwendet noch feste Pläne, in der die Ausführungsreihenfolge festgelegt ist. Dies soll aber durch zur Laufzeit festgelegte Pläne ersetzt werden. Der Scheduler soll dabei durch die Inspektion der Beiträge und darin vorkommender Variable festlegen, wann ein Beitrag exekutiert werden kann. Dies kann z.B. durch die Reihenfolge der Variablenbindungen festgelegt werden.

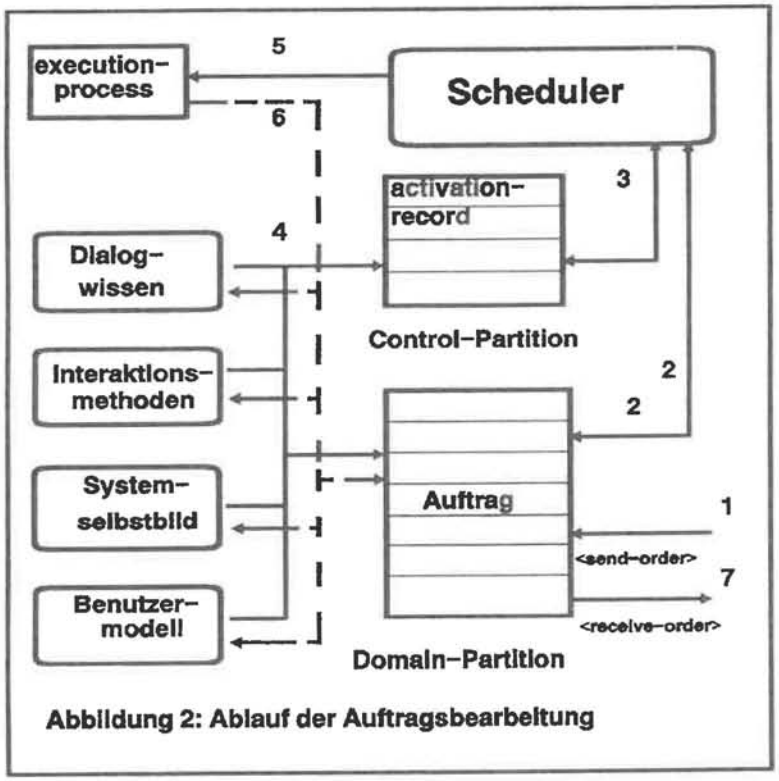

Abbildung 2: Ablauf der Auftragsbearbeitung

Abb. 2: Ablauf der Auftragsgenerierung

Sind alle Beiträge in der Prozeßumgebung abgelaufen, löscht der Scheduler den Activation-Record und schreibt die Ergebnisse in den Auftrag zurück (es können natürlich von den einzelnen Beiträgen temporäre Ergebnisse erzeugt werden und von anderen Beiträgen weiterverarbeitet werden, diese werden aber im allgemeinen nicht vom Sender des Auftrags

benötigt). Der Sender des Auftrags kann nun den Auftrag mit den Ergebnissen mit einem **"receive-order"-Befehl** (Schritt 7 in Abbildung 2) zurückholen.

Eine Applikation kann beliebig viele Aufträge gleichzeitig in die Blackboard stellen. Die Reihenfolge, in der sie fertige Aufträge abholt, bleibt ihr überlassen. Die Synchronisation und Kontrolle der Aufträge in der Blackboard liegt aber immer beim Scheduler. Er sorgt auch dafür, daß immer der richtige Empfänger Zugriff zu einem fertigen Auftrag erhält, da auch mehrere Applikationen gleichzeitig auf der Blackboard operieren können. Ein nicht fertig bearbeiteter Auftrag kann nicht von der Applikation aus der Blackboard geholt werden.

Wie weiter oben erwähnt, kann ein einfacher Experte (Wissensquelle) aus einer Menge von Regeln bestehen. Ein Experte kann aber auch ein beliebiges Applikationssystem sein. Oft ist es aber möglich und notwendig, Experten zu logischen Gruppen zusammenzufassen, um a) die Übersichtlichkeit und Fehlersuche zu erleichtern und b) die Wartbarkeit eines Systems zu erhöhen. Um dies zu ermöglichen unterstützt BASAR beliebig viele Blackboard-Ebenen (Abbildung 1). Ausgehend von einem Top-Level-BASAR-System können die einzelnen Experten wieder aus BASAR-Systemen bestehen (Meta-Level-Architektur). So ist z.B. der Dialogmanager wieder als eigenes BASAR-System ausgebildet. Die Kommunikation zwischen verschiedenen Ebenen erfolgt ebenfalls mittels Aufträgen.

Die **Programmentwicklung** und **Testung** wird bei BASAR durch einen **Blackboard-Spy** unterstützt. Durch Setzen eines Flags kann der Programmierer den Ablauf von Aufträgen in der Blackboard steuern und gegebenenfalls Aufträge in der Blackboard ändern (z.B. durch Ändern von Slots in Aufträgen). Der Spy zeigt dabei den aktuellen Zustand der Domain- und Controlpartition an, welche Wissensquellen und Applikationen aktiv sind und wie der Zustand der Wissensquellen ist. Der Spy ist nicht für den eigentlichen Anwender bestimmt, da sehr viel Information generiert wird und der Benutzer gar nicht bemerken wird, daß in seiner Applikation Veränderungen aufgetreten sind. Eine detaillierte Schilderung des Ablaufs des Schedulers findet sich in /Waldhör 89a/.

2.2 Der Dialogmanager

Der Dialogmanager ist für die Verbindung der Applikationen und Wissensquellen zuständig. Applikationen verwenden **Interaktionsaufträge** um mit dem Benutzer kommunizieren zu können. Dialogaufträge referieren wiederum **Interaktionsobjekte**; Interaktionsobjekte sind z.B. Fenstertypen, Formulare, Eingabetechniken, kurz alle Kommunikationsformen zwischen Benutzer und Applikation. Der Anwendungsprogrammierer sollte nur mehr angeben können, welche Interaktionstechniken er benötigt, nicht wie diese realisiert werden sollen. Der Programmierer

muß in einem Interaktionsauftrag nicht mehr alle möglichen Informationen angeben, die ein Interaktionsobjekt bietet, sondern kann durch eine Minimalangabe ein entsprechendes Objekt erhalten. Er muß z. B. nicht mehr definieren, daß er ein Fenster benötigt, dieses an einer bestimmten Stelle auf dem Bildschirm angezeigt werden soll und dann auf eine Eingabe vom Benutzer warten soll, sondern er verwendet den Interaktionsauftrag **input-demand** eventuell mit der Angabe, wie die Eingabe beschaffen sein soll (Zeichenkette, Symbol, Zahl ..) und der Dialogmanager generiert daraus ein Fenster, plaziert es am Schirm und fordert eine Eingabe vom Benutzer.

Der Dialogmanager besteht aus einer Menge von **Interaktionsexperten**, die wiederum aus jeweils **vier Dialogexperten** dem **Benutzermodell**, dem **Dialogwissen**, dem **Systemselbstbild** und den **Interaktionsmethoden**, aufgebaut sind. Die Auswahl eines Interaktionsexperten für einen bestimmten Interaktionsauftrag wird durch ein eigenes BASAR-System realisiert. Die Auswahl der entsprechenden Methoden wird auch hier durch einen Scheduler gesteuert. So werden für den schon erwähnten Auftrag input-demand ein Fenster- und ein Eingabeexperte zuständig sein. Es müssen nicht für jedes Interaktionsobjekt alle vier Unterexperten existieren. In einigen Fällen ist z.B. kein Benutzermodell oder kein Dialogwissen notwendig oder dieser Bereich nicht entsprechend modellierbar.

Das **Benutzermodell** besteht aus einer Menge von Regeln und einer Wissensbasis, in der benutzerspezifische Informationen abgelegt sind. Eine grundlegende Regel lautet, daß, falls für ein bestimmtes Interaktionsobjekt Informationen aus dem Benutzermodell existieren, diese auch angewandt werden sollen. Hat etwa der Benutzer definiert, daß er nur Fenster mit Scrollbars verwenden will, so wird ein solches verwendet. Andere Experten (z.B. Dialogwissen) können daran keine Änderungen mehr vornehmen, da davon ausgegangen wird, daß die Wünsche des Benutzers oberste Priorität haben. Sind in der Benutzerwissensbasis keine entsprechenden Informationen verfügbar, wird auf allgemeinere Regeln zurückgegriffen, wie etwa (die Regeln werden hier natürlichsprachlich aufgeführt, sie sind über LISP-Funktionen realisiert):

Benutzermodell-Regel: Fenstertypauswahl
Wenn der Benutzer in mehr als 60 % aller Fälle angegeben hat, daß er Fenster mit Scrollbalken verwenden will, generiere ein Fenster mit Scrollbalken.
Wenn eine Änderung des Fenstertyps erfolgt, frage beim Benutzer nach, ob er damit einverstanden ist.

Die letzte Regel soll garantieren, daß das System sein Verhalten nicht in für den Benutzer unvorhersehbarer Weise ändert. Der Vorteil in der regelbasierten Darstellung solchen Wissens liegt darin, daß in einfacher Weise verschiedene Benutzeroberflächen und verschiedenes Systemverhalten durch Änderungen in einer oder mehreren Regeln erzeugt werden können. Zusätzlich kann man durch entsprechende Meta-Regeln dem System eine gewisse Lernfähigkeit ermöglichen. Der Benutzer muß also nicht selbst festlegen (durch eine Konvention etwa), daß er nur Fenster mit Scrollbalken verwenden will, das System kann es von sich aus erkennen und entsprechend handeln (natürlich nur nach Rücksprache mit dem Benutzer). Diese Regel wird dann in das Benutzermodell übernommen. So kann das System, wenn es feststellt, daß der Benutzer immer wieder Fenster mit Scrollbalken verwenden will, eine neue generelle Regel generieren, die lautet:

Neue Benutzermodell-Regel: Benutzer-Mayer-Fenstertyp
Wenn für den Benutzer "Mayer" ein Fenster erzeugt werden soll, verwende ein Fenster mit Scrollbalken.

Im Gegensatz zum Benutzermodell enthält das **Dialogwissen ergonomisches Wissen** über die Interaktion (/Waldhör 89b/). Dieser Experte tritt meist dann in Aktion, wenn im Benutzermodell keine Informationen für ein Interaktionsobjekt vorhanden sind. Durch die Einbindung ergonomischen Wissens in Form von Dialogregeln wird der Programmierer bei seiner Entscheidung unterstützt, welches Interaktionsobjekt er in einer bestimmten Situation verwenden soll. Diese Entscheidung kann auch vom System selbst getroffen werden, wenn der Programmierer nichts spezifiziert hat. Der Programmierer wird in zweifacher Weise unterstützt: a) während der Programmerstellung erhält er Unterstützung zu einem bestimmten Interaktionsobjekt (z.B. welche Slots weist das Objekt auf) und b) das System versucht zur Laufzeit ein optimales Objekt auszuwählen.

In Situationen in denen sowohl das Benutzermodell als auch das Dialogwissen anwendbar ist, wird das Benutzermodell durch eine entsprechende Kontrollregel des Dialogmanagers bevorzugt. Der Benutzer sollte immer die letzte Entscheidung bei der Verwendung von Interaktionsobjekten haben.

Ergonomisches Wissen wird ebenfalls in Form von Regeln verwendet. So kann z.B. eine Regel in einem Formular folgendermaßen lauten:

Dialogwissens-Regel: Formularauswahl
Wenn ein Formular angezeigt werden soll, verwende ein mit einem Scanner eingelesenes Formular.
Wenn kein gescanntes Formular verfügbar ist, generiere ein synthetisches Formular.

Diese Regel geht von der Tatsache aus, daß ein Benutzer ein ihm bekanntes Formular lieber verwenden wird, als ein Formular, das auf dem Bildschirm neu generiert wird und mit seinem ihm vertrauten keine Ähnlichkeit mehr aufweist.

Regeln im Dialogwissen können miteinander konkurrieren. Es kann z.B. eine Regel besagen, daß ein Fenster immer ein gewisses Höhen-zu-Breiten-Verhältnis aufweisen soll. Dies kann mit einer Regel konkurrieren, die verhindern soll, daß sich zwei Fenster stark überlappen. In diesem Fall ist wieder eine Kontrollregel vorgesehen, die diesen Konflikt auflöst. Der Vorteil der regelbasierten Repräsentation liegt darin, daß man durch einfaches Ändern einer solchen Kontrollregel das Systemverhalten ohne großen Aufwand verändern kann. Weiterhin kann diese Wissensquelle einfach durch Hinzufügen neuer Regeln inkrementell erweitert werden, sodaß nicht bei der Einführung eines neuen Interaktionsobjektes (z.B. ein neues Eingabegerät wird angeschlossen) alle Regeln bekannt sein müssen, sondern erst im Laufe einer Testphase erweitert werden können.

Die beiden restlichen Experten - **Systemselbstbild** und **Interaktionsmethoden** - entsprechen konventionellen Ein-/Ausgabeanweisungen. Sie erzeugen aus dem durch die beiden anderen Experten gewonnenen Informationen die Interaktion mit dem Benutzer. Das Systemselbstbild enthält Information über den aktuellen Zustand des Systems (welche Fenster existieren, wie ist deren Zustand, welche Bereiche eines Fensters sind maussensitiv, welche Formulare werden angezeigt, welcher Benutzer ist gerade aktiv ...) und liefert diese Information an die Interaktionsmethoden weiter. Der Experte Interaktionsmethoden wählt auf Grund der Anforderungen ein entsprechendes Interaktionsobjekt aus der Zielsprache aus oder generiert ein solches selbst. Dieser Teil des Dialogmanagers ist sehr maschinenabhängig und muß für jede Maschine neu programmiert werden. Eine SYMBOLICS LISP Maschine verwendet z.B. sehr viel abstraktere und mächtigere E/A-Anweisungen als C. Solche Unterschiede sollen durch diesen Experten ausgeglichen werden.

Die **Interaktionsmethodenexperten** des Dialogmanagers bauen auf den Interaktionsobjekten des INFORM-Fenstersystems auf (siehe Koller et al. in diesem Band). Eine erste Implementierung des Dialogmanagers verwendete die SYMBOLICS-eigenen Interaktionsobjekte. Im Rahmen

des zweiten Prototypen von BASAR wurde dann das INFORM-Fenstersystem verwendet. Hier zeigten sich die Vorteile der durch den Blackboard-Ansatz gegebenen Modularität der einzelnen Wissensquellen. Es mußte nur der Interaktionsmethodenexperte in einem gewissen Umfang neu implementiert oder umgeschrieben werden, während die anderen Dialogexperten nicht verändert werden brauchten.

In der aktuellen Version von BASAR werden alle notwendigen Interaktionsobjekte, die für die Programmierung von wissensbasierten Applikationen benötigt werden, unterstützt. Weitere Interaktionsmethoden können ohne großen Aufwand in das System eingebracht werden.

Folgende Interaktionsaufträge stehen derzeit zur Verfügung:

- Generierung und Verwalten von Fenstern mit und ohne Scrollbalken. Jedes Fenster enthält eine Menüleiste mit der der Benutzer das Fenster manipulieren kann (Verkleinern/Vergrößern, Verschieben, Löschen, Hilfefunktionen, applikationsspezifische Funktionen, Iconumwandlung ... etc.). Spezifiziert der Programmierer keine eigene Menüleiste, so wählt das System aus der Dialogwissensbasis entsprechend sinnvoll Menüeinträge aus.
- Verschiedene Menütypen (ebenfalls mit der oben angesprochenen Menüleiste)
- Verschiedene Graphikmöglichkeiten (Anzeigen von gescannten Bildern, Ausgabe von Graphiken, graphische Funktionen wie Kreise etc.)
- Verschiedene Ausgabemöglichkeiten (Texte, Icons ...). Diese können vom Anwendungsprogrammierer als maussensitiv definiert werden und können dann vom Benutzer bei Eingaben verwendet werden.
- Verschiedene Eingabemöglichkeiten (Tastatur, Maus ...). Die Koordinierung der Eingabe wird vom Dialogmanager übernommen, sodaß der Programmierer selbst nur mehr wenige Parameter spezifizieren muß, mit denen er komplexe Eingabeoperationen definieren kann.

2.3 Schlußfolgerungen

Für den Anwendungsprogrammierer stellt die Programmierung in BASAR eine wesentliche Erleichterung gegenüber der konventionellen Programmierung dar. Anwendungsprogrammierer werden weitgehend von der Programmierung der Benutzeroberfläche entlastet. Damit können auch Programmierer, die sich nicht mit allen Feinheiten eines Fenstersystems beschäftigen wollen, mit BASAR Benutzeroberflächen für ihre Applikationen programmieren. Sie erhalten zudem auch bei verschiedenen Applikationen für den Benutzer einheitliche Oberflächen, was die Bedienung erleichtert.

Ein weiterer Vorteil ist die enorme Einsparung von Code gegenüber einem konventionellen Programm. Bei einer früheren Version der Abfragekomponente, die für das Elektronische Organisationshandbuch in LISP programmiert war, benötigte der Anwendungsprogrammierer ca. 7

Seiten Code nur für den Eingabeauftrag, der in BASAR mit 7 Zeilen programmiert werden kann und das LISP-Programm in seiner Funktionalität übertrifft.

Insgesamt läßt sich sagen, daß mit BASAR ein System zur Verfügung steht, das für die Anwendungsprogrammierer eine gute Unterstützung bei der Programmierung ergonomischer Benutzerschnittstellen bedeutet. Für die Benutzer stellen sich die in BASAR erstellten Applikationen einheitlicher und damit leichter erlernbar und bedienbar dar.

3. Der Benutzerschnittstellen-Baukasten USIT

Im folgenden wird der im Rahmen des WISDOM-Projekts erstellte Benutzerschnittstellen-Baukastens **USIT (User Interface Toolkit)** vorgestellt werden. Der Baukasten dient zur Realisierung verschiedener Prototypen mit innerhalb eines gewünschten Rahmens einheitlichen Bedienoberflächen.

3.1 Aufgabe eines Benutzerschnittstellen-Baukastens

Zur Realisierung einer Interaktionstechnik bedarf es der Kontrolle zweier Vorgänge. Zum einen müssen Daten des Anwendungssystem auf den Ausgabegeräten der Anwendung und dem Benutzer angemessen aufbereitet ausgegeben werden. Dies können Darstellungen auf Bildschirmen sein, wie Texte, Bitmuster, Linien, Flächen und räumliche Graphiken oder auch akustische oder tastbare Ausgaben. Zum anderen müssen Eingaben des Benutzers erfaßt werden, die ebenfalls über unterschiedliche periphere Geräte wie z.B. Tastatur, Maus, Touch-Screen oder Scanner eingegeben werden können.

Eine Interaktionsmethode ist nun eine spezielle Art Ausgaben zu erzeugen und Eingaben zu erfassen sowie insbesondere deren Sequenzierung und Zuordnung zueinander festzulegen. Ein Benutzerschnittstellen-Baustein ist ein Softwaremodul zur Realisierung einer solchen Interaktionsmethode. Ein Benutzerschnittstellen-Baukasten muß die für eine oder meist mehrere Anwendungen benötigten Bausteine enthalten. Da für neue Anwendungen erfahrungsgemäß immer wieder neue oder veränderte Interaktionsformen benötigt werden, muß der Baukasten auch erweiterbar sein.

3.2 Anwendungsneutralität

Ein UIMS und als Teil sein Benutzerschnittstellen-Baustein sollen möglichst viele Anwendungssysteme unterstützen. Man spricht auch von **Anwendungsneutralität** eines solchen Systems. Diese hängt im wesentlichen von zwei seiner Eigenschaften ab:

- Welcher Art ist die Programmierschnittstelle zwischen Anwendungsprogramm und UIMS?
- Welche Interaktionstechniken werden unterstützt?

Die erste Frage wird durch das UIMS-Rahmensystem entschieden, die zweite Frage betrifft den Benutzerschnittstellen-Baukasten.

Zur Erzielung einer universellen Einsetzbarkeit eines UIMS ist es wichtig, möglichst alle häufig eingesetzten und bewährten Interaktionstechniken bereitzustellen. Dies sind derzeit vor allem Textmenüs, Formulare und Tabellen. Andere Techniken werden in letzter Zeit durch die Verfügbarkeit graphikfähiger Bildschirme zunehmend genutzt, wie z.B. Icons, graphische Menüs, editierbare Netze, Anzeigeinstrumente und Property-Sheets.

Im USIT-System wurden alle diese Interaktionsformen als Bausteine bereitgestellt (Netze (/Stenger 86a/) und Instrumente (/Stenger 86b/) wurden wegen des hohen Entwicklungsaufwands als eigene Subsysteme entwickelt).

Alle Bausteine sind reich attributiert und können in vielen Details gesteuert durch das UIMS oder das Anwendungsprogramm variiert werden. Auch dies ist entscheidend für die allgemeine Nutzbarkeit eines solchen Systems. Als Beispiel für die Gestaltungsparameter im folgenden eine unvollständige Eigenschaftsliste von Textmenüs:

- Font zur Darstellung der Textelemente (gleich für alle Elemente, spezifisch für jedes einzelne Element)
- Art des Auswahlfeedbacks bei den Menüelementen (Invertieren, Einrahmen, spezieller Mauscursor, kein Feedback)
- Layout (horizontal oder vertikal, ein- oder mehrspaltig, Zeilen- und Spaltenabstand, Menüelemente linksbündig/zentriert/rechtsbündig)
- Umrahmung des gesamten Menüs (Linien, freier weißer Raum)
- Verschwinden des Menüs (auf Programminitiative, auf Benutzerinitiative, nach Auswahl, nach Verlassen des Menüs mit dem Mauscursor)
- Darstellung des inaktiven Menüs oder einzelner inaktiver Menüelemente
- Hilfetext für einzelne Menüelemente oder das ganze Menü
- Verhalten bei Auswahl eines Textelements (Speichern der Auswahl, Initiieren einer Aktion)
- Größe des Menüs (Sichtbarkeit aller Menüelemente oder Scrolling)
- Darstellung des Menüs in Normal- oder Inversdarstellung

Das Spektrum der mit dem USIT-Baukasten erstellbaren Systeme reicht von traditionellen menü- und formularorientierten Lösungen bis hin zu direktmanipulativen Desktops.

3.3 Implementierung eines Benutzerschnittstellen-Baukastens

Bei der Realisierung eines Benutzerschnittstellen-Baukastens ist insbesondere auf die folgenden Anforderungen zu achten:

Abstrakte Beschreibung:

Die Bausteine sollten nicht nur konzeptionell sondern auch programmtechnisch als Einheiten betrachtet werden können, die einfach änderbare Attribute und klare Schnittstellen nach außen besitzen.

Spezialisierbarkeit:

Vorhandene Bausteine sollten zur begrenzten Änderung ihres Aussehens oder Verhaltens mit geringem Aufwand spezialisierbar sein.

Aggregierbarkeit:

Vorhandene Bausteine sollten zu komplexeren Bausteinen zusammengesetzt werden können. Dies kann in Art einer hierarchischen Strukturierung erfolgen. Die neuen, komplexen Bausteine sollten wieder als Einzelelemente im Sinne der oben genannten abstrakten Beschreibung betrachtet werden können.

Erweiterbarkeit:

Selbst bei reichhaltiger "Grundausstattung" des Baukastens werden durch neue Anwendungssysteme neue Anforderungen an den Baukasten zu stellen sein. Ein UIMS und seine Subsysteme sollten als offene Systeme konzipiert werden, die Hilfsmittel bereitstellen neue Bausteine herzustellen, die nicht durch Spezialisierung oder Aggregierung vorhandener Bausteine erzeugt werden können.

Zur Implementierung von Benutzerschnittstellen-Baukästen bietet sich die Verwendung einer objektorientierten Programmiersprache an (/Herczeg 86a/). Nahezu alle fortgeschrittenen derartigen Baukastensysteme wurden mittels objektorientierter Sprachen realisiert oder wenn eine solche nicht zur Verfügung stand, deutlich objektorientiert konzipiert (/Lipkie et al. 82/; /Goldberg, Robson 83/; /Goldberg 84/; /Sibert et al. 86/; /Symbolics 86a/; /Symbolics 86b/).

Die Gründe dafür liegen in den folgenden Eigenschaften objektorientierter Modellierungstechniken:

- Objekte sind Einheiten mit änderbaren Attributen (Slots) und funktionalen Verhaltensbeschreibungen (Methoden). Es ist natürlicher, Bausteine durch Objekte zu beschreiben. Ein solches Objekt repräsentiert dann zum Beispiel ein Icon, ein Menü oder ein Formular.
- Die Kommunikation mit Objekten erfolgt ausschließlich mittels Botschaften. Die zulässigen Botschaften (die Schnittstellendefinition) erfolgt durch die Festlegung von Methodenfiltern. Eine derartige Schnittstellendefinition ist gut lokalisiert und damit bei Bedarf leicht einsehbar und änderbar. Eine Botschaft ist beispielsweise die Aufforderung an ein Menü sich auf dem Bildschirm darzustellen oder die Aufforderung an ein Icon sich invertiert darzustellen.
- Klassen beschreiben Mengen gleichartiger Objekte durch Festlegung ihrer Attribute und Methoden. Sie dienen gleichzeitig als Generatoren von Instanzen, d.h. im Fall von Benutzerschnittstellen generieren sie die vom UIMS für eine bestimmte Anwendung benötigten Bausteine. Durch die Änderung von sogenannten Klassenslots kann das Verhalten einer ganzen Menge von existierenden Bausteinen geändert werden (z.B. das Feedbackverhalten von Menüelementen wird global geändert oder der Font für Menüelemente aller Textmenüs wird einheitlich festgelegt).
- Klassen können in Vererbungsverbände eingeordnet werden. Dadurch ererben Klassen die Attribute und Methoden ihrer Superklassen, die von der erbenden Klasse erweitert oder redefiniert werden können. Auf diese Weise lassen sich mehr oder weniger spezialisierte Bausteine definieren. Die Repräsentation ist sehr redundanzarm. So sind Text- und Graphikmenüs zwei spezialisierende Subklassen einer allgemeinen Klasse, die Menüs beschreibt.

Der Baukasten USIT wurde unter Ausnutzung der genannten Mechanismen in der objektorientierten Programmiersprache ObjTalk (/Rathke 86/; /Girgensohn, Rathke 88/) für SYMBOLICS LISP Maschinen implementiert. Frühere Versionen dieses Baukastens waren Bestandteil des Benutzerschnittstellen-Entwicklungssystems WLISP (/Böcker, Fabian, Lemke 85/; /Herczeg 85a-b/; /Herczeg 86a/) in einer VAX/UNIX-Rechnerumgebung.

3.4 Uniforme Repräsentation von Interaktionstechniken

Bei der Entwicklung unterschiedlicher Benutzerschnittstellen-Bausteine über einen Zeitraum von mehr als 7 Jahren haben sich immer wieder Gemeinsamkeiten in der Attributierung, der strukturellen Beschreibung und der Beschreibung des Interaktionsverhaltens abgezeichnet. Beispielsweise sollten Texte in Icons oder Menüs genauso editierbar sein wie Texte in Formularfeldern und Formulare sollten ähnliche Layoutformen

ermöglichen wie Menüs. Darüber hinaus zeigte sich die Notwendigkeit vielerlei Mischformen von Interaktionstechniken bereitzustellen, so zum Beispiel Menüs, die als Alternative zur Menüauswahl mit der Maus ein Texteingabefeld besitzen, in dem die Auswahl textuell eingegeben werden kann. Ein anderes Beispiel sind Formulare, die an der Stelle eines oder mehrerer Texteingabefelder ein graphisches Menü anbieten.

Als Folge dieser Anforderungen wurde unter das Baukastensystem USIT ein weiteres Baukastensystem gelegt, das allgemeine Basisbausteine bereitstellt, um Benutzerschnittstellen-Bausteine aufzubauen. Derartige Basisbausteine sind:

Interaktionsobjekte:

Interaktionsobjekte (sogenannte Intels) dienen als Repräsentanten einer bestimmten Interaktionsform und besitzen im allgemeinen Verweise auf untergeordnete Intels oder Objekte mit anderen Aufgaben (Darstellungsobjekte, Readerobjekte, Layoutobjekte, Verhaltensobjekte, Protokollobjekte, Objekte eines Hilfesystems). Sie wirken in vielen Fällen als Aggregatoren für untergeordnete Bausteine und definieren die Schnittstelle eines neuen, komplexeren Bausteins.

Darstellungsobjekte:

Es wurden Darstellungsobjekte (sogenannte Dispels) für Texte, Bitmaps und graphische Elemente definiert. Mehrere dieser Dispels dienen zum Aufbau komplexerer Darstellungen. So besteht ein Textmenü aus einer Reihe von Textdispels, ein Bitmapmenü aus Bitmapdispels. Diese Dispels sind teilweise editierbar. Formularfelder werden üblicherweise aus editierbaren Textdispels aufgebaut.

Readerobjekte:

Readerobjekte (sogenannte Reader) lesen vom Benutzer eingegebene Zeichenströme von der Tastatur und bilden höherwertige Zeichen oder führen bei entsprechend definierten Texteingabesequenzen beliebige Aktionen aus. Solche Reader werden beispielsweise an Textdispels angekoppelt, um diese editierbar zu machen.

Layoutobjekte:

Die Anordnung von Dispels wird durch Layoutobjekte (sogenannte Clusters) kontrolliert. Durch die Änderung von Attributen eines Clusters oder durch Ersetzen eines Clusters durch einen anderen kann das Layout von Darstellungselementen einer bestimmten Interaktionsform auch dynamisch geändert werden.

Ausgabebereiche:

Dispels und durch Intels erzeugte Aggregationen von untergeordneten Intels und Dispels werden in Ausgabebereichen, einer Verallgemeinerung von Fenstern (Windows) ausgegeben. Gegenüber "normalen" Fenstern sind Ausgabebereiche meist von sehr viel einfacherer Natur. Diese Lightweight-Windows werden in großer Zahl benötigt, für ein Formular mit 10 Feldern werden ca. 30 bis 50 Ausgabebereiche benötigt. Diese Ausgabebereiche werden für USIT durch das Subsystem INFWIN (/Fabian 87/) bereitgestellt.

Fonts, Bitmaps und Sounds:

Fonts (Zeichensätze) und Bitmaps werden als eigenständige Objekte realisiert, die von vielen anderen Objekten gemeinsam genutzt werden können. Auch akustische Elemente (Sounds) können so für die verschiedensten Einsatzbereiche bereitgestellt werden. Diese Objekte repräsentieren vor allem Ausgabefunktionalität. So besitzen Fonts beispielsweise die Funktionalität zur Formatierung und Ausgabe von Texten in Ausgabebereichen. Bitmaps dienen vor allem zur Ausgabe von Hintergrundmustern sowie von Inhalten für Icons.

Verhaltensobjekte:

Verhaltensobjekte (sogenannte Reactivities) entscheiden innerhalb eines Intels mit Hilfe regelartigen Wissens wie Ereignisse, die durch Benutzereingaben oder Subintels entstehen, zu interpretieren sind.

Protokolle:

Zum Aufbau einer Dialogablaufgeschichte können Protokolle auf verschiedenen Ebenen erstellt werden. Auf diese kann dann bei Bedarf z.B. für History- oder Hilfefunktionen zurückgegriffen werden. Eine andere Nutzung dieser Protokolle ist die Analyse von Benutzerverhalten bei der Evaluation einer Benutzerschnittstelle.

Objekte eines Hilfesystems:

Während dem Ablauf einer Interaktionstechnik (z.B. einer Menüauswahl) kann ein Hilfesystem aktiviert werden, das dem Benutzer kontextabhängig die Interaktionstechnik oder auch Konzepte des Anwendungssystem erklärt.

Durch diese feinkörnige Architektur zusammen mit einer - wenn auch noch einfachen - Spezifikationssprache ist es möglich, sehr spezielle Interaktionstechniken zusammenzustellen. Dazu werden die einzelnen Typen von Objekten spezifiziert und zusammengebunden. Dies geschieht durch Definition einer neuen Intel-Klasse. Durch Instantiierung dieser Klasse entstehen dann Inkarnationen der neuen Interaktionstechnik zum Aufruf durch das UIMS. Eine solchen Konfigurierbarkeit von Inter-

aktionstechniken verhindert auf vordefinierte Interaktionstechniken festgelegt zu sein und öffnet gewissermaßen den Weg zu einem **Interaktionskontinuum** ohne die Möglichkeit des Vordefinierens häufig benötigter Interaktionstechniken zu nehmen.

3.5 Konsistenz von Benutzerschnittstellen

Die Verfügbarkeit eines Interaktionsspektrums anstatt festgelegter Interaktionsmethoden stellt sich auf dem ersten Blick konträr zur Zielsetzung konsistenter und damit leichter bedienbarer Bedienoberflächen dar.

Genauer betrachtet und richtig angewandt bietet die Methodik jedoch die Möglichkeit anwendungsgerechte Bedienoberflächen zu erstellen, die viele ausgefeilte Details zu einer effizienten Bedienung von Anwendungssystemen ermöglichen. Dies muß beim Benutzer nicht zwangsläufig ein Bild der Inkonsistenz verursachen. Gerade durch die Verwendung einheitlicher Basisbausteine kann eine sehr grundlegende Konsistenz dargeboten werden, die durch entsprechende Programmierkonventionen allein nicht erreichbar wäre. Texte können zum Beispiel immer auf dieselbe Art zu editieren sein, die Nutzung des Zeigeinstruments kann immer einheitlich erfolgen und Visualisierungen können durch zentral festgelegte Attribute gesteuert werden. Die Konsistenz auf höherer Ebene zu wahren ist jedoch in erster Linie die Aufgabe des UIMS-Rahmensystems, das sich des Benutzerschnittstellen-Baukastens bedient. Der Benutzerschnittstellen-Baukasten selbst sichert diese Konsistenz über Standardeinstellungen nur zu einem bestimmten Grad ab, da alles weitergehende auf Kosten der Flexibilität ginge.

In vielen Fällen wird man bei der Auswahl von Bausteinen auf Standardbausteine zurückgreifen, die ein einheitliches Erscheinungsbild schaffen und gleichartig zu bedienen sind. Änderungen an diesen Bausteinen werden sich in allen Anwendungen konsistent auswirken. Dies ist die Folge der Isolierung von Eigenschaften in den Klassendefinitionen, also der geringen Redundanz der Repräsentation.

3.6 Die Meta-Benutzerschnittstelle

Ist eine Benutzerschnittstelle erzeugt, gibt es selbst zur Laufzeit des Systems noch die Möglichkeit Attribute einzelner Interaktionsobjekte oder ihrer untergeordneten Objekte zu ändern, um so die Darstellung oder das Verhalten einer Interaktionsmethode zu beeinflussen. Dies kann prinzipiell auch auf Initiative des Benutzers erfolgen. Man spricht in diesem Fall von **adaptierbaren Benutzerschnittstellen**.

Damit diese Änderungsvorgänge wiederum interaktiv möglich sind, bedarf es einer Benutzerschnittstelle, die die vorhandene "Anwendungs"-Benutzerschnittstelle als Anwendungssystem betrachtet. Wir sprechen hierbei von der **Meta-Benutzerschnittstelle**. Zu ihrem Bau wird man sich natürlich wieder des Benutzerschnittstellen-Baukastens bedienen. Diese

Vorgehensweise ermöglicht dann auch die rekursive Anwendung der Meta-Benutzerschnittstelle auf sich selbst, so daß weitere Metaebenen nicht benötigt werden.

Der Baukasten USIT verfügt über eine solche Meta-Benutzerschnittstelle, die derzeit allerdings so ausgebildet ist, daß sie den Systementwicklern als Hilfsmittel zum **Rapid Prototyping** dient. Mit diesem Hilfsmittel können einfach und schnell Gestaltungsalternativen entwikkelt werden.

Durch die deskriptive Änderbarkeit der Benutzerschnittstelle durch Änderung von Attributwerten, schafft man darüber hinaus die Basis für **adaptive Benutzerschnittstellen**. Dabei geht die Initiative zur Umgestaltung vom UIMS-Rahmensystem aus, das auf Grund seines sich ändernden Wissens über den Benutzer, vorhandenen Gestaltungsregeln und daraus resultierenden Schlußfolgerungen eine solche Änderung selbsttätig durchführt. Der Benutzer kann dabei durchaus um seine Zustimmung befragt werden. Ob derartige adaptive Benutzerschnittstellen überhaupt nützlich und erwünscht sind, wird derzeit noch durch kontroverse Diskussionen und erste Prototypen in Fachgremien erarbeitet.

4. Ausblick

Der Beitrag schilderte die im WISDOM-Projekt durchgeführten Aktivitäten auf dem Gebiet der Benutzerschnittstellen. Es gilt, die dabei gemachten Erkenntnisse und Erfahrungen auf Systeme zu portieren, die nicht nur in akademischen Institutionen oder Forschungsbereichen vorhanden sind, sondern auch dem Normalbenutzer (Bürobereich ...) zu gute kommen zu lassen.

Die aktuelle Version von BASAR ist in Common LISP auf einer Symbolics implementiert. Durch eine Reimplementierung der einzelnen Wissensquellen in LUIGI, der TA-Wissensrepräsentation (siehe Kindler et al. in diesem Band) können sowohl der Scheduler und dessen Kontrollstrategien als auch die einzelnen Interaktionsexperten besser an die Gegebenheiten wissensbasierter Systeme angepasst werden. Durch die Einführung und Veränderung von neuen Regeln und Kontrolltaktiken sollen verschiedene Techniken leichter getestet werden können. Es bietet sich vor allem an, die derzeit im LISP-Code implizit vorhandenen Dialog- und Benutzermodellregeln in LUIGI zu repräsentieren. Durch die Repräsentierung des USIT in LUIGI kann eine zur Laufzeit generierte Benutzerschnittstelle optimal an den Benutzer angepaßt werden. BASAR soll zusätzlich auf UNIX portiert werden, wobei der Dialogmanager auf X-Windows (siehe dazu auch den Beitrag von Märtin in diesem Band) zurückgreifen wird. Ein wichtiger Punkt wird die Unterstützung des Anwendungsprogrammierers sein. Er soll entsprechende Werkzeuge zur Benutzerschnittstellengenerierung verwenden können, die ihn bei der Auswahl der Interaktionsobjekte beraten. Ein weiterer Punkt betrifft die

Repräsentation der eigentlichen Applikation. Hier besteht ein Trade-off zwischen der abstrakten Repräsentation der Benutzerschnittstelle und der Anwendung. Es muß versucht werden, Applikationen auf einem ähnlich hohen Abstraktionsniveau zu beschreiben. Dies kann. z.B. in Form von Plänen geschehen, die zur Laufzeit ausgewertet werden und zusammen mit der Benutzerschnittstelle dem Benutzer erweiterte Möglichkeiten wie Selbsterklärungsfähigkeit, UNDO/REDO (/Waldhör 87/) zur Verfügung stellen.