Benutzer-Schnittstelle

Autoren: Matthias Schneider
M. Herczeg
Hermann Wenzel

Kümmerte sich vor zwanzig Jahren noch kaum ein Programmierkünstler um die Schnittstelle zum Anwender, so hat der Einzug des Computers in den Alltag neue Anforderungen an die Software hervorgerufen. Das Attribut „Anwenderfreundlichkeit" ist allerdings nicht ganz so einfach zu realisieren, wie es als Werbeargument benutzt wird.
Wer als Hersteller nicht „auf der Strecke bleiben" will, räumt der Mensch-Maschine-Schnittstelle jedoch hohen Stellenwert ein.

Matthias Schneider*

In der Forschungsgruppe INFORM an der Universität Stuttgart entstand in den letzten vier Jahren eine Reihe von prototypischen Systemen zur Verbesserung der Mensch-Computer-Kommunikation. Ein wichtiges Ergebnis dieser Forschungsarbeit läßt sich so formulieren: Eine qualitative Verbesserung der Mensch-Computer-Schnittstelle läßt sich dadurch erreichen, daß man den Rechner mit Wissen über die Umgebung, in der er eingesetzt wird, ausstattet.
Bei der Entwicklung dieser Systeme wurde eine Reihe von Design-Kriterien und Prinzipien erarbeitet, die sich auf den Entwurf von endbenutzer-programmierbaren Systemen übertragen lassen. Einige dieser wünschenswerten Eigenschaften von Programmsystemen sollen in den folgenden Abschnitten beschrieben werden.

Vom „Wie" zum „Was"

Zur Definition eines Vorgangs in herkömmlichen Programmiersprachen (PASCAL, COBOL etc.) muß der Programmierer sowohl den zeitlichen Ablauf als auch die funktionale Zerlegung dieses Vorgangs beschreiben. Ein Beispiel einer solchen Beschreibung ist eine Anleitung zum Ausfüllen eines bestimmten Formulars. Hierbei wird eine Reihenfolge vorgegeben, die zwar sinnvoll, aber nicht unbedingt notwendig ist.
Zusätzlich dazu muß er in der Lage sein, die von ihm verwendeten Daten (Objekte seiner Anwendungsumgebung) geeignet zu beschreiben. Vor allem die vollständige und zeitlich eindeutige Beschreibung eines Ablaufs ist ohne geeignete Schulung und Programmiererfahrung sehr schwierig.
Für Anwender mit nur geringer Programmiererfahrung ist es meist einfacher, ihr Problem in bezug auf die folgenden Fragen zu definieren:
1. Wie sehen die Randbedingungen aus? Beispiel: Die Gesamtsumme eines Finanzplans soll einen bestimmten Wert nicht überschreiten.
2. Welche Beziehungen existieren zwischen den Variablen des Problems? Beispiele: Ein bestimmtes Formularfeld soll immer die Summe einer Reihe weiterer Felder enthalten. Zwischen der Anzahl der Verwaltungsangestellten und der der Wissenschaftler in einem Projekt soll ein bestimmtes Verhältnis herrschen.
3. Was ist das Gesamtziel der Bearbeitung? Beispiel: Alle relevanten Felder eines Finanzplans sollen ausgefüllt sein.
Versucht man, eine derartige Problembeschreibung direkt in ein Programm zu übertragen, enthält dieses die folgenden Konzepte:

- globale Abhängigkeitsbeziehungen, die die Randbedingungen festlegen
- lokale Beziehungen zwischen den Variablen des Systems
- Regeln, die die möglichen Veränderungen der Variablen beschreiben. Dadurch muß keine vollständige zeitliche Reihenfolge der Veränderungen festgelegt werden. Es ist deshalb auch nicht notwendig, den gesamten Problemlöseprozeß zeitlich zu ordnen.

Ein Programm-Interpreter versucht nun, durch An-

*Matthias Schneider ist Mitarbeiter am Forschungsprojekt INFORM, Universität Stuttgart

Designkriterien für benutzermodifizierbare Systeme

wendung der Regeln eine Variablenbelegung zu finden, die die globalen und lokalen Bedingungen des Problems erfüllt. Das Problem ist dann gelöst, wenn keine Widersprüche mehr gefunden werden können. (Ein – allerdings rudimentäres – Beispiel für eine derartige Programmiermethode ist das MULTIPLAN-System Multiplan 84).
Aus diesen Überlegungen läßt sich ein erstes Kriterium für den Entwurf endbenutzer-programmierbarer
Systeme formulieren:

1. Kriterium: *Der Benutzer eines Programmsystems sollte seine Probleme in der Terminologie des „Was möchte ich erreichen?" formulieren können und nicht gezwungen werden, die Antwort auf die Frage „Wie kann ich es erreichen?" explizit zu formulieren.*

Repräsentation von Wissen

Ein Problem bei der Erstellung von Programmen besteht darin, das Wissen, das der Anwender über sein Programm und dessen Lösungen besitzt, in eine maschinell interpretierbare Form zu bringen. Der Begriff „Wissen" umfaßt in diesem Zusammenhang sowohl Programme (funktionales Wissen) als auch Daten (deskriptives Wissen). Je nach Wahl der Repräsentation für Daten und Programme lassen sich diese Bereiche nicht mehr eindeutig trennen.
Diese Übertragung läßt sich dadurch vereinfachen, daß man eine Repräsentation im Computer schafft, die dem Modell des Benutzers von seinem Problem und dem Lösungsraum möglichst exakt entspricht. Die Daten- und Programmrepräsentation in herkömmlichen Programmiersprachen (mit ihren Datentypen String, Integer, Real etc.) ist dazu wenig geeignet. In der Sprache des Problembereichs werden die Daten meist anders bezeichnet und sind häufig sehr komplex und hierarchisch geordnet (Beispiele: Forschungsantrag, Finanzplan, Formular, Formularfeld etc.). Dasselbe gilt für die Repräsentation eines Lösungsweges.
Es liegt nahe, die Daten im System so zu repräsentieren, daß die Objekte der Problemumgebung mit ihren Beziehungen in der dem Anwender vertrauten Form darstellbar sind. Zu diesem Zweck wurden *objektorientierte* Programmiersprachen entwickelt, deren wichtigste Komponenten *Objekte* mit ihren Methoden sind. Beispiele für objektorientierte Programmiersprachen sind Smalltalk und ObjTalk. Objekte sind aktive Datenstrukturen mit Verhaltensattributen, die entlang einer Generalisationshierarchie von einem Objekt auf andere vererbt werden können. Wenn ein Objekt eine Nachricht erhält, zeigt es ein bestimmtes Verhalten, das durch Methoden festgelegt wird. Dies kann in der Veränderung seines lokalen Zustands, dem Senden von Nachrichten an weitere Objekte oder der Rückgabe einer Antwort bestehen.

2. Kriterium: *Der Benutzer eines Programmsystems sollte die Datenobjekte seines Problems in einer problemangemessenen Repräsentation definieren können. Der Aufwand zur Umstrukturierung der Daten sollte für den Anwender möglichst klein sein.*

Mensch-Maschine-Schnittstelle

Die Mensch-Maschine-Schnittstelle herkömmlicher Computersysteme wurde weitgehend durch die vorhandene Hardware bestimmt. Mit zeichenorientierten Bildschirmen mit einem festen Zeichenraster wurden meist nur teletypeartige Dialogsysteme gestaltet.
Mittlerweile ist es möglich geworden, mit Hilfe von hochauflösenden Bitmap-Graphikbildschirmen mit Zeigegeräten, wie z. B. der Maus, der Mensch-Maschine-Schnittstelle eine neue Qualität zu geben. Die Graphikfähigkeiten der Bildschirme erlauben es, Objekte in einer dem Anwender vertrauten Form darzustellen und Beziehungen zwischen ihnen zu visualisieren, die mit teletypeorientierten Schnittstellen nicht oder nur in sehr abstrakter Form darstellbar sind. Da diese Abstraktion gerade für Benutzer mit geringer Programmierpraxis sehr schwer nachvollziehbar ist, sollte eine Schnittstelle für Endbenutzer die graphischen Darstellungsmöglichkeiten moderner Bildschirme ausnutzen. Ein Beispiel für eine derartige Schnittstelle zeigt Abbildung 1.
Statt des Dialogs mittels einer Kommandosprache (mit meist schwer zu erlernender Syntax) kann der Dialog jetzt über Menü-Auswahl stattfinden. Der Anwender wird nicht mehr gezwungen, die verfügbaren Kommandos auswendig zu wissen bzw. sie in einem Manual oder mittels eines Hilfesystems abzufragen, sondern kann sich die verfügbaren Kommandos darstellen lassen und mit einem Zeigegerät das gewünschte Kommando auswählen.
Auch die Programmierung von Anwenderprogrammen kann so vereinfacht werden. Der Benutzer wählt aus einem Satz von Beispielprogrammen bzw. Programmteilen diejenigen aus, die er für seine Problemlösung benötigt und verändert sie gegebenenfalls für seine Aufgabenstellung. In einer objektorientierten Programmiersprache bedeutet dies beispielsweise, daß die Programmierung durch die Modifikation bereits vorhandener Objekte stattfindet.
Die Verwendung eines Fenstersystems ermöglicht es, zu einem Kontext gehörige Ausschnitte des Mensch-Computer-Dialogs zusammenhängend darzustellen. Während in teletype-orientierten Dialogen der Benutzer beim Wechsel zwischen verschiedenen Prozessen sich die Zusammenhänge selbst klarmachen muß, können mit ei-

nem Fenstersystem alle Interaktionen, die in einem bestimmten Kontext stattfinden, in einem Fenster zusammengefaßt werden.

3. Kriterium: *Beim Entwurf von endbenutzer-programmierbaren Systemen muß der Programmierung der MC-Schnittstelle besondere Aufmerksamkeit gewidmet werden. Durch Verwendung moderner Bildschirme und Interaktionstechniken wie Menüs und Fenstersystemen läßt sich die Aufgabe des Endbenutzers vereinfachen.*

Direkte Manipulation

Die Objekte eines Anwendungsgebietes müssen im Rechner repräsentiert werden. Die Aufgabe des Anwenders ist es nun, diese Objekte zu verändern, neue Objekte zu kreieren, Objekte zu löschen und Beziehungen zwischen Objekten herzustellen.
Diese Manipulation von Datenobjekten kann entweder auf der an die Arbeitsweise des Rechners angepaßten Repräsentation stattfinden oder auf einer zweiten, dem Benutzer leichter verständlichen Darstellung erfolgen. Es liegt nahe, eine dem Benutzer vertraute graphische Darstellung zu suchen und eine Abbildung zwischen der internen und der externen Darstellung zu schaffen, die jede Veränderung durch das Programm (d. h. in der internen Repräsentation) nach außen hin sichtbar macht und jede Modifikation durch den Benutzer (in der externen Repräsentation) auf die interne Repräsentation überträgt.
Der Benutzer kann jetzt auf die Objekte auf seinem Bildschirm zeigen und Operationen auf diese Objekte anwenden, d. h. ihr Erscheinungsbild und ihre internen Eigenschaften verändern. Diese Interaktionsform bezeichnet man als *direkte Manipulation.* Sie befreit den Endbenutzer von der Aufgabe, die Syntax der internen Repräsentationssprache zu erlernen.
Der Designer eines endbenutzer-modifizierbaren Systems muß für direkte Manipulation folgende Eigenschaften vorsehen:

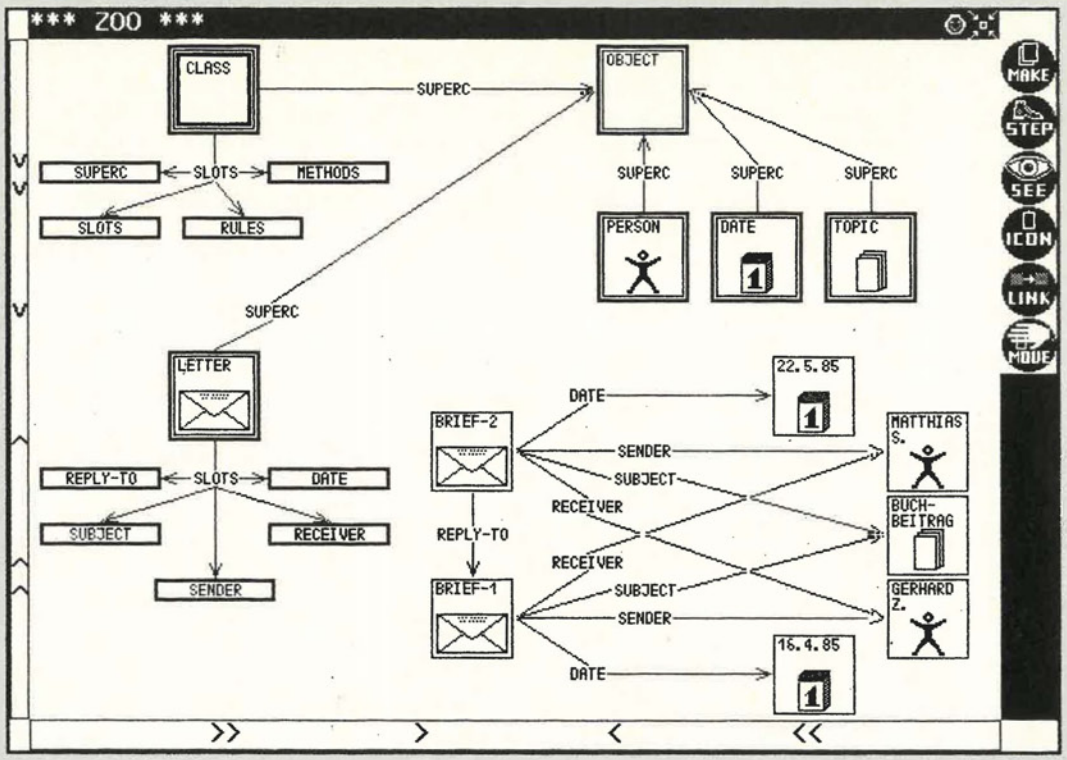
- Im System muß Wissen über mögliche graphische Repräsentationen und die verfügbaren Operationen auf dieser Darstellung vorhanden sein. Dem System müssen die dargestellten Objekte und ihre Position auf dem Bildschirm bekannt sein, da sonst die Auswahl mit dem Zeigeinstrument nicht sinnvoll ist.
- Die Beziehung zwischen der externen und internen Repräsentation muß eindeutig sein.
- Der Designer muß Methoden zur Veränderung der nicht direkt sichtbaren Eigenschaften von Objekten definieren. (Eine häufig verwendete Lösung für dieses Problem besteht in der Verwendung von *Property-Sheets*).

Zusätzlich zur direkten Manipulation sollten in einem Programmsystem alternative Formen der Interaktion möglich sein, um beispielsweise externe Programme anschließen zu können. Systeme ohne diese Möglichkeit lassen sich meist nur sehr schwer vom Benutzer verändern. (Ein Beispiel für ein solches System, das sich kaum in ein Anwendungssystem einbinden läßt, ist die – derzeitige – Benutzeroberfläche des Xerox-Star-Systems).

4. Kriterium: *Der Benutzer sollte in der Lage sein, die Objekte seiner Anwendungsumgebung direkt zu manipulieren, ohne sich um die Repräsentation im Programmsystem kümmern zu müssen.*

Hilfe- und Erklärungskomponenten

Die Komplexität von Programmsystemen macht es notwendig, dem Benutzer Unterstützung während der Benutzung zu geben. Dabei ist ein Manual in Buchform sicherlich nicht ausreichend. Erforderlich ist vielmehr Online-Unterstützung



Das System ZOO wurde von Wolf-Fritz Riekert im Rahmen der Arbeiten der Forschungsgruppe INFORM entwickelt. In diesem System wird jedes Objekt des Anwendungsgebiets graphisch dargestellt. Der Benutzer kann mit Hilfe eines Zeigegeräts (Maus) zuerst Objekte und danach die auf sie auszuführenden Operationen auswählen. Auch Beziehungen zwischen Objekten lassen sich so herstellen. Das System erzeugt aus diesem Dialog die Datenobjekte, die dann vom System verwendet werden (siehe Abb. 2).

Abbildung 1: Direkte Manipulation in System ZOO

```
toplevel
94:(ask Brief-2 redefining-form:)
TIME NEEDED: 83
(ask letter
     remake:
     Brief-2
     with:
     (reply-to = ,Brief-1)
     (sender = ,|Matthias S.|)
     (receiver = ,|Gerhard Z.|)
     (date = ,|22.5.85|)
     (subject = ,Buch-Beitrag)
     (picture = ,<some_object-picture>))
95:
```

Diese Datenstruktur kann vom System verwendet werden, während der Anwender die Repräsentation auf der Benutzeroberfläche verändert.

Abbildung 2: Die entstehende ObjTalk-Repräsentation eines Objekts

mit Hilfesystemen, die sich an den Kenntnisstand und die Erfahrung des Benutzers anpassen können. Da Endbenutzer häufig nicht den gesamten Funktionalitätsumfang eines Systems kennen, genügen *passive* Hilfekomponenten, die auf Anforderung des Benutzers Hilfeinformationen geben, nicht. *Aktive* Hilfesysteme können aus dem Dialog des Benutzers mit dem System Schlußfolgerungen über fehlendes Wissen des Benutzers ziehen und ihn gezielt auf ihm unbekannte Operationen hinweisen.

Beim Vorgang der Programmierung sollte der Anwender von einem Dokumentationssystem unterstützt werden, das seine Designentscheidungen und das entstehende Programm beschreiben kann. Auch diese Informationen sollten nicht nach der Programmierung in Papierform, sondern bereits während des Programmiervorgangs auf dem Rechner zur Verfügung stehen. Nur so kann die gesammelte Information bereits während der Programmierung eingesetzt werden. Computersysteme haben eine Komplexität erreicht, die es unmöglich macht, daß ihr Verhalten von einem Benutzer leicht nachvollzogen werden kann. Es ist deshalb notwendig, daß der Rechner seine Vorgehensweise erklären kann. Diese Forderung hat einen Einfluß auf die Art und Weise, in der Code geschrieben wird. Annahmen und Verfahren, die in *implizierter* Form im Code versteckt werden, können dem Benutzer nur schwer verständlich gemacht werden. Wenn die Entscheidungskriterien des Systems in einer Wissensbasis (siehe Abschnitt 2) gespeichert sind, kann dem Benutzer Information über diese Kriterien vermittelt werden.

5. Kriterium: *Ein Programmsystem, das vom Endbenutzer modifiziert werden soll, sollte Komponenten besitzen, die die Funktionalität des Systems erläutern, die Veränderungen durch den Benutzer dokumentieren, und das Verhalten des Systems rechtfertigen können.*

Anmerkung

Die ursprüngliche Fassung dieses Artikels erscheint als Beitrag zum Expertenkolloquium „Softwarenutzung am Arbeitsplatz und berufliche Weiterbildung" in der Schriftenreihe der Forschungsgruppe „Verwaltungsautomation" an der Gesamthochschule Kassel.

Einzelpreis DM 10,–

D 1518 E

& Software Magazin

computer magazin

10'85

Oktober 1985, 14. Jahrgang