Definieren statt Programmieren

Wolfgang. K. Epple

Zusammenfassung

Das Vordringen des Computers in immer neue Anwendungsgebiete und die dabei aufgetretenen Akzeptanzprobleme haben dafür gesorgt, daß die **M**ensch - **M**aschine - **S**chnittstelle (MMS) zunehmend in den Mittelpunkt der Betrachtung gerückt ist.

In diesem Beitrag wird ein Werkzeug vorgestellt, das die interaktive Erstellung von benutzerorientierten Kommando - Dialog - Systemen ermöglicht. Bei Verwendung des Werkzeugs reduzieren sich die vom Dialogentwickler durchzuführenden Tätigkeiten auf die Definition der Dialogsprache und auf die Modifikation von automatisch erzeugten Masken.

Kern des Systems ist eine Datenbasis, in der die Kommandos und Parameter einer Dialogsprache abgelegt werden. Die Kommandos können interaktiv definiert werden. Sie werden bereits bei der Definition analysiert. Jedem Kommando können darüber hinaus freiwählbare Zeichenketten zugeordnet werden, die bei korrekter Eingabe eines Kommandos an das korrespondierende Anwendungsprogramm übergeben werden. Außerdem können umgangssprachliche Erklärungstexte festgelegt werden.

Ein wesentlicher Vorteil ist, daß bei Verwendung des Werkzeugs der Implementierungsaufwand drastisch gesenkt wird. Es ist völlig ausreichend zur Entwicklung eines Dialogsystems lediglich die Kommandosprache zu definieren. Das daraus automatisch erzeugbare Dialogsystem ist zugleich Prototyp und endgültiges System.

Überblick

1. Einleitung

Das Vordringen des Computers in immer neue Anwendungsgebiete und die dabei aufgetretenen Akzeptanzprobleme haben dafür gesorgt, daß die **M**ensch - **M**aschine - **S**chnittstelle (MMS) zunehmend in den Mittelpunkt der Betrachtung gerückt ist. Während früher den mehr oder weniger stark 'EDV - motivierten' Menschen zugemutet wurde, sich an vorhandene Mensch - Maschine - Schnittstellen zu gewöhnen, wird heute von Mensch - Maschine - Schnittstellen gefordert, daß sie sich den wechselnden Bedürfnissen des Menschen, der sie benutzt, dynamisch anpassen oder zumindest statisch darauf abgestimmt sind.

Umfangreiche Studien haben gezeigt, daß die geforderte Flexibilität der Dialogsysteme kein triviales Problem ist und daß akzeptable Lösungen einen großen Implementierungsaufwand und damit zwangsläufig einen hohen finanziellen Aufwand erfordern. Darüber hinaus wird die MMS wie keine andere Softwarekomponente einer besonders kritischen Betrachtung unterzogen, denn jeder Benutzer eines Systems muß sich mit der MMS direkt auseinandersetzen. Als Folge davon setzt seine Kritik zunächst bei der MMS ein.

2. Problem

Die Softwareentwicklung im allgemeinen und die Entwicklung von Dialogsystemen im besonderen ist gegenwärtig vorallem durch fehlende Validierungsmöglichkeiten gekennzeichnet. Unter Validierung sei hier nicht die strenge Verifikation, der formale Nachweis von gewünschten Eigenschaften verstanden, sondern die schwächere vertrauensbildende Maßnahme, die belegt, daß ein (Software-) Produkt mit der Intention und den Vorstellungen des Auftraggebers übereinstimmt. Typisch für die Validation ist, daß das Bezugsobjekt, gegen das das Prüfobjekt validiert wird, i.d.R. nicht oder zumindest nicht vollständig formal dargestellt werden kann. Es existiert in der Vorstellung eines Auftraggebers.

Ein Kernproblem der gegenwärtigen Software - Entwicklung ist die fehlende Möglichkeit, bereits zu einem sehr frühen Zeitpunkt der Entwicklung zu zeigen, daß die - in manchen Bereichen formal darstellbare - Beschreibung der Anforderungen, die Vorstellungen eines Auftraggebers vollständig erfüllt. Diese Eigenschaft wird als **externe Vollständigkeit** oder **Problemadäquatheit** bezeichnet. Das Fehlen dieser Eigenschaft ist häufig bei der Abnahme von Systemen die Ursache von Aha - Effekten der Art : **Ja so haben wir uns das aber nicht vorgestellt.**

Bei der Entwicklung von Dialogsystemen ist das Fehlen dieser Eigenschaft besonders prekär, weil hier, wie in keinem anderen Bereich, die Notwendigkeit besteht, sehr früh einen Eindruck von dem Verhalten und von den verfügbaren Eigenschaften des Dialogsystems zu gewinnen.

Durch den zunehmenden Einsatz des Rechners in neuen Anwendungsbereichen und den damit verbundenen Anstieg der Anforderungen an Dialogsysteme gerät die Entwicklung auch hier verstärkt unter Druck. Es müssen standardisierte Schnittstellen und Programmbausteine entwickelt werden, mit deren Hilfe Dialogsysteme systematisch und effizient aufgebaut werden können.

3. Lösungsalternativen

Die konstruktiven Ansätze zur Verbesserung der Situation sind zahlreich. Einerseits wurde eine Vielzahl von **Anforderungen** und **Empfehlungen** aufgestellt, deren Beachtung zur Entwicklung von "guten" Dialogsystemen führt. Sie betreffen vorwiegend die psychologischen Aspekte eines Dialogsystems. Eine

gute Zusammenfassung dieser Anforderungen ist in [9] enthalten. Andererseits existieren **Beschreibungsmittel** und **Werkzeuge**, die die systematische Entwicklung von Dialogen unterstützen. Eine Zusammenstellung der Werkzeuge und Beschreibungsmittel findet sich in [4].

Die verfügbaren Werkzeuge lassen sich grob in folgende Klassen einteilen :

1. Masken- bzw. Formulargeneratoren,
2. Tabellengesteuerte Interpreter und
3. Dialogsystemgeneratoren.

Masken- und Formulargeneratoren ermöglichen es, interaktiv Bedienerschnittstellen für verschiedene Anwendungsprogramme zu erzeugen. Eine Maske besteht in der Regel aus Feldern, wobei man zwischen Eingabe- und Ausgabefeldern unterscheidet. In ein Eingabefeld kann vom Bediener Information eingetragen werden wohingegen Ausgabefelder ausschließlich durch ein Programm beschrieben und verändert werden. Mit Formulargeneratoren lassen sich einerseits Datenstrukturen z.B. dadurch definieren, daß dür die Felder einer Maske u.a. der Typ, Grenzwerte und die maximale Länge angegeben werden. Andererseits kann man damit gleichzeitig die Form der Masken, d.h. die Lage der einzelnen Felder auf dem Bildschirm festlegen. Darüber hinaus kann nach Vorgabe von Grenzwerten für Eingabefelder die Eingabe automatisch auf ihre Plausibilität überprüft werden. Ein Teil der verfügbaren Generatoren ermöglicht es außerdem, die Reihenfolge der 'Abarbeitung' der einzelnen Felder festzulegen.

Bei **tabellengesteuerten Interpretern** wird der Dialogablauf, d.h. die Folge der möglichen Kommandos sowie deren Parameter mit Hilfe von endlichen Automaten beschrieben. Einer der ersten Ansätze in dieser Richtung ist in [7] dargestellt. Die möglichen Zustände und Zustandsübergänge werden codiert und in Tabellenform (Automatentafel) abgelegt. Es wurden Standardinterpreter entwickelt, die die tabellarisch repräsentierten Dialogbeschreibungen interpretieren und dabei unter Verwendung von Standardbausteinen Bedienereingaben analysieren, Fehlermeldungen erzeugen, Hilfsinformation anbieten und entsprechende anwendungsspezifische Ausführungsroutinen aufrufen. Einige Vertreter dieser Gruppe sind in [1], [2], [3] und [6] beschrieben. Der bekannteste Vertreter dieser Gruppe dürfte der in [11] dargestellte Transition Diagram Interpreter (TDI) sein. Eine Realisierung eines ähnlichen Ansatzes wird in [10] vorgestellt.

Ein **Dialogsystemgenerator**, wie er in diesem Beitrag vorgestellt wird, stellt eine Synthese aus Maskengenerator und tabellengesteuertem Interpreter dar. Er gestattet die interaktive Definition von kommandoorientierten Dialogsystemen. Es genügt bereits, die Syntax der Dialogsprache zu definieren. Der Generator erzeugt daraus einerseits ein Automatenmodell, das den Ablauf eines Interpreters steuert und andererseits eine oder mehrere Masken, mit deren Hilfe Bedienereingaben abgefragt und analysiert werden können. Im Gegensatz zu den genannten Repräsentanten der tabellengesteuerten Interpreter ist es hier nicht erforderlich, die Zustände und Zustandsübergänge 'von Hand' zu codieren. Es ist bereits ausreichend, wenn die Syntax der Kommandosprache definiert wird. Das System extrahiert daraus die benötigte Information und überführt sie in eine interne Darstellung.

4. Lösungskonzept

4.1 Systemstruktur

Um ein hohes Maß an Flexibilität zu erreichen und den Einsatz des Dialogsystemgenerators in Verbindung mit verschiedenartigen Anwendungsprogrammen zu ermöglichen, war die Entwicklung eines modularen Systems eine wesentliche Randbedingung. Der Entwurf des Systems erfolgte unter Verwendung des Prinzips der Datenabstraktion. Nach der Identifikation von unbedingt erforderlichen und wünschenswerten Funktionen wurden diese zu Gruppen zusammengefaßt. Die Daten, die von den jeweiligen Funktionen gelesen oder modifiziert werden, stellten dabei das Klassifikationskriterium dar. Im Prinzip wurden die Operationen eines abstrakten Datentypen definiert.

Die Operationen lassen sich in zwei Klassen zerlegen : Entweder sie liefern irgendwelche Werte eines Objektes von einem bestimmten Typ (**value** - Funktionen) oder sie ändern den Wert eines Objektes (**operate** - Funktionen). Wesentlich dabei ist, daß die Struktur der zugrundeliegenden Daten von außen nicht sichtbar ist. Es gibt keinerlei Seiteneffekte. Die Veränderung eines Datums ist ausschließlich durch festgelegte Zugriffsfunktionen möglich. Die derart entwickelten Einheiten, im folgenden Module genannt, sind damit weitgehend unabhängig voneinander : Das Lesen und Verändern von Daten erfolgt ausschließlich durch die Aktivierung von Zugriffsfunktionen.

Die Anpaßbarkeit des Systems an verschiedene Anwendungsprogramme wurde folgendermaßen erreicht: Die Dialogsprache eines beliebigen Anwendungsprogramms läßt sich interaktiv mit Hilfe eines sog. Metadialogsystems definieren. Das Ergebnis des Definitionsprozesses wird in einer Datenbasis abgelegt. Die Definition einer Dialogsprache basiert auf dem Modell eines endlichen Automaten, wobei Parameter, die in verschiedenen Kommandos verwendet werden, lediglich einmal zu definieren sind. Funktionen zur Interpretation von Bedienerkommandos und zur Unterstützung des ungeübten Benutzers entscheiden aufgrund der in der Datenbasis abgelegten Beschreibung der Dialogsprache über die Zulässigkeit von Bedienereingaben. Syntaktisch korrekte und plausible Kommandos werden akzeptiert, in ein definiertes Standardformat überführt und an das korrespondierende Anwendungsprogramm übergeben. Neben den Interpretationsfunktionen existiert eine globale Systemsteuerung. Wesentlich ist, daß keine der Funktionen, die letztendlich durch Programme realisiert sind, verändert werden muß, wenn die Dialogsprache eines Anwendungsprogramms verändert wird, oder wenn ein mit Hilfe des Dialogsystemgenerators erstelltes Paket um zusätzliche Anwendungsprogramme erweitert wird. Das Gesamtsystem ist in Bild 1 dargestellt. Eine detaillierte Beschreibung ist in [5] enthalten.

Das Gesamtsystem besteht aus :

- einem dialogführenden Teil,
- einem Generator und
- einer Datenbasis mit Zugriffsfunktionen.

Der dialogführende Teil analysiert die Bedienereingaben, bereitet diese für das Anwendungsprogramm auf und leitet sie an dieses weiter. Ein Kommando- und Maskeninterpreter sowie die Dialogsystem - Steuerung sind Teilkomponenten des dialogführenden Teils.

Der **Kommandointerpreter** analysiert Bedienereingaben. Er entscheidet durch Vergleich der Eingabe mit der in der Kommando - Datenbasis gespeicherten Information über die Zulässigkeit von Eingaben. Er arbeitet dabei zeichenweise. Jedes vom Bediener eingetippte Zeichen wird unmittelbar analysiert. Der Interpreter akzeptiert ausschließlich potentiell zulässige Zeichen. Nicht zulässige Zeichen werden abgelehnt. Alternativ dazu können Eingaben des Bedieners mit

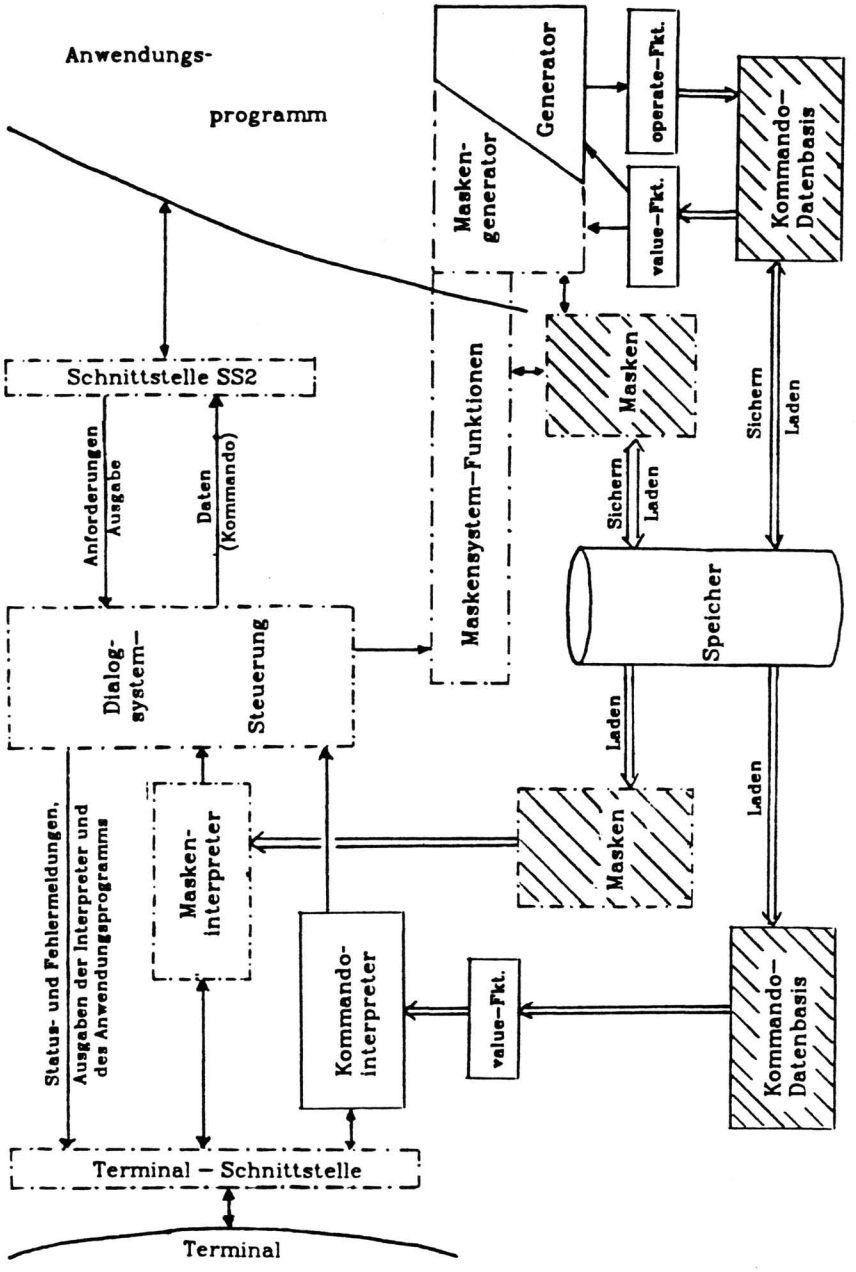

Bild 1 : Systemstruktur [5]

Hilfe des **Maskeninterpreters** analysiert werden. Der Bediener erhält dabei die für ein Kommando erforderlichen Parameter angezeigt und hat dann die Möglichkeit die Werte in entsprechende Felder einzutragen. Der Maskeninterpreter verwendet dazu die aus der Kommando - Datenbasis gewonnenen Masken.

Die **Dialogsystem - Steuerung** analysiert und verarbeitet Eingaben, die den globalen Ablauf betreffen. Das Umschalten zwischen Masken- und Kommandomodus, die Definition und das Ausführen von Makros (Kommandosequenzen) sind Beispiele dafür. Um diese Kommandos genauso wie die eigentlichen (anwendungsspezifischen) Kommandos interpretieren zu können, wurde deren Beschreibung ebenfalls in der Datenbasis abgelegt.

Der **Generator** ermöglicht den interaktiven Aufbau der Datenbasis für ein Anwendungsprogramm. Er ermöglicht die Erstellung neuer und die Modifikation bereits vorhandener Datenbasen. Es stehen infolgedessen Kommandos zur Definition, Modifikation und zum Löschen von Kommando- und Parameterbeschreibungen zur Verfügung. Jede Datenbasis wird nach der Definition oder nach einer Modifikation auf Konsistenz und Vollständigkeit untersucht. Ist eine Datenbasis fehlerfrei und somit zur Steuerung des Kommandointerpreters geeignet, so wird sie auf einer Datei gesichert. Der Maskengenerator erzeugt für jede fehlerfreie Datenbasis entsprechende Masken. Für jedes mit Hilfe des Generators definierte Kommando wird eine Maske generiert. Die Beschreibung der Masken wird ebenfalls auf einer Datei gesichert.

Die **Datenbasis** enthält die Zustandsmodelle von allen definierten Dialogsprachen. Sie zerfällt in drei unabhängige Teile, die

- Kommandos der Anwendungsprogramme,
- Kommandos für interne Funktionen des Dialogsystems und
- Kommandos zum Starten der Anwendungsprogramme

enthalten. Darüber hinaus besteht die Möglichkeit, sowohl für Kommandos als auch für Parameter umgangssprachliche Erklärungstexte festzulegen. Verweise auf diese Texte werden gemeinsam mit den Kommandos in der jeweiligen Teilkomponente der Datenbasis gespeichert.

4.2. **Systemverhalten**

Die Programmodule des Systems werden von der Dialogsystem - Steuerung angestoßen. Sie geben nach Ihrer Beendigung die Kontrolle an die Steuerung zurück. Beim Start des Systems wird nach einer Initialisierungsphase der Kommandointerpreter gestartet. Der Bediener kann nun ein beliebiges Anwendungsprogramm starten. Sobald dies geschehen ist, wird die zu diesem Anwendungsprogramm gehörende Datenbasis geladen. Die korrespondierenden Maskenbeschreibungen werden erst dann geladen, wenn per Kommando der Maskenmodus eingeschaltet, d.h. der Maskeninterpreter aktiviert wurde. Nach dem erfolgreichen Laden der entsprechenden Datenbasis wartet die Dialogsystem - Steuerung solange, bis das Anwendungsprogramm über eine Schnittstellenprozedur seine Anforderungen mitteilt. Erwartet das Anwendungsprogramm Bedienereingaben, so startet die Systemsteuerung den zuletzt aktiven Interpreter (Kommando- oder Maskeninterpreter), der die Analyse und Aufbereitung der Bedienerkommandos übernimmt. Die korrekten Bedienereingaben werden anschließend an das wartende Anwendungsprogramm übergeben und die Systemsteuerung wartet auf eine neue Anforderung.

Wenn das Anwendungsprogramm eine andere Funktion der Systemsteuerung angefordert hat, so wird diese ausgeführt und das Ende dem wartenden Anwendungsprogramm mitgeteilt.

Es ist möglich, daß der Bediener, während er mit einem Anwendungsprogramm arbeitet, ein weiteres startet. In diesem Fall werden alle für das aktuelle Anwendungsprogramm relevanten Informationen solange gekellert bis der Bediener das neugestartete Programm wieder beendet hat. Das gesamte System kann erst dann terminiert werden, wenn alle aktivierten Anwendungsprogramme beendet wurden.

5. Resümee

Das System wurde auf einem PCS - Arbeitsplatzrechner (CADMUS 9200) unter MUNIX in Pascal implementiert. Es zeigte sich dabei, daß die nahezu vollständige (Tipp-) Fehlersicherheit, die dadurch erreicht wird, daß im Komandomodus jedes vom Bediener eingetippte Zeichen einzeln, unmittelbar nach Eingabe analysiert wird, eine wünschenswerte, jedoch zeitintensive Funktion von interaktiven Systemen ist. Bei der gewählten Realisierung bewegten sich die Wartezeiten im Sekundenbereich.

Abgesehen von dieser zeitlichen Einschränkung stellt der vorgestellte Dialogsystemgenerator ein ideales Werkzeug dar, das den Implementierungsaufwand von Dialogsystemen drastisch reduziert. Es ist völlig ausreichend zur Entwicklung der Dialogschicht eines Anwendungsprogrammes lediglich die Kommandosprache zu definieren. Das daraus automatisch erzeugbare Dialogsystem ist zugleich **Prototyp** und **endgültiges System**. Es übernimmt vollständig die Analyse der Bedienereingaben und die Koordination der verschiedenen Programmteile.

Der hier vorgestellte Generator für Dialogsysteme ist das Ergebnis einer längeren Entwicklung, zu deren erfolgreichem Abschluß mehrere Personen beigetragen haben. Dank gebührt in erster Linie U. Geiger und J. Mehlis, die im Rahmen Ihrer Diplomarbeit [5] das System mit unermüdlichem Eifer realisierten. Auch die ehemaligen Mitglieder des Arbeitskreises "Systematische Entwicklung von Realzeitsystemen" an der Universität Karlsruhe haben in zahlreichen Diskussionen die Entwicklung positiv beeinflußt.

6. Literaturverzeichnis

[1] CHERITON, David, R. :"Man - Machine Interface Design for Time - Sharing Systems", Proc. ACM Nat. Conf., 1976, pp. 362 - 380

[2] DENERT, Ernst :"SPECIFICATION AND DESIGN OF DIALOGUE SYSTEMS WITH STATE DIAGRAMS' in Proc. of the International Computing Symposium 1977, Liege, Belgium, 4 - 7 April 1977

[3] ENCARNACAO, J.; HANUSA, H.; STRASSER, W. :"Tools and Techniques for the Description, Implementation, and Monitoring of Interactive Man - Machine Dialogues" in [8]

[4] EPPLE, Wolfgang K. : "Ein Verfahren zur Entwicklung von Bedienerdialogen", Dissertation, Universität Karlsruhe, 1985

[5] GEIGER, U. und MEHLIS, J. :"Konzipierung und Realisierung eines interaktiven Dialogsystemgenerators", Diplomarbeit am Lehrstuhl für Prozeßrechentechnik der Universität Karlsruhe, 1984

[6] HAUER, Karl - Heinz; LAWRENCE W. Bernhard; SCHNUPP, Peter H. :"A SPECIFICATION LANGUAGE FOR DIALOG USER INTERFACE", in [8], pp. 81 - 86

[7] PARNAS, David L. :"ON THE USE OF TRANSITION DIAGRAMS IN THE DESIGN OF A USER INTERFACE FOR AN INTERACTIVE COMPUTER SYSTEM", in Proc. National ACM Conference, 1969, pp. 379 - 385

[8] PROCEEDINGS 1982 INTERNATIONAL ZURICH SEMINAR ON DIGITAL COMMUNICATIONS, MAN - MACHINE INTERACTION, March 9 / 11, 1982; IEEE Catalog, No. 82, CH 1735 - 0

[9] SCHMITT, Alfred :"Dialogsysteme", Bibliographisches Institut, Reihe Informatik/40, April 1983, 270 S.

[10] SCHMITT - LADEMANN, F.P. und SCHMITT, A. :"Kommandoschnittstellen mit normiertem Benutzerkomfort und ihre Implementierung", Notizen zum Interaktiven Programmieren, Heft 6, März 1981, S. 3 - 21

[11] WASSERMAN, Anthony, I. :"User Software Engineering and the Design of Interactive Systems", in IEEE Proc. 5th International Conference on SOFTWARE ENGINEERING, March 9 - 12, 1981, San Diego, California

Wolfgang K. **Epple**
Lehrstuhl für Prozeßrechentechnik
Prof. Dr.-Ing. U. Rembold
am Institut für Informatik III
Universität Karlsruhe, Postfach 6380, 7500 Karlsruhe