DIE BENUTZERSCHNITTSTELLE EINER INTEGRIERTEN SOFTWAREENTWICKLUNGSUMGEBUNG

W. Schäfer, Osnabrück

Zusammenfassung: Thema dieses Papiers ist die integrierte Gestaltung und der Entwurf der Benutzerschnittstelle einer Softwareentwicklungsumgebung. Zunächst wird erläutert, was der Begriff 'integriert' in diesem Zusammenhang bedeutet und wie sich daraus eine konkrete Benutzerschnittstelle entwickeln läßt. Beim Entwurf des Ein-/Ausgabesystems werden die Punkte Portabilität und Adaptabilität der entstehenden Implementierung besonders berücksichtigt. Dies führt zu einer Entwurfsstrategie, die für die Entwicklung beliebiger interaktiver Systeme eingesetzt werden kann.

1 Integration in IPSEN

Die in diesem Papier vorgestellten Ergebnisse entstanden im Rahmen des Forschungsprojekts **IPSEN** (**I**ncremental **P**rogramming **S**upport **En**vironment), das an der Universität Osnabrück durchgeführt wird (/8/). IPSEN ist der Prototyp einer integrierten Softwareentwicklungsumgebung, die zum Ziel hat, die Erstellung, Ausführung, Analyse und Wartung der Dokumente, die während der einzelnen Phasen des Softwarelebenszyklus bzw. lebenszyklusbegleitend entstehen, durch geeignete Werkzeuge zu unterstützen. Die Aktivitäten bei der Entwicklung der einzelnen Dokumente unterteilen wir analog zu anderen Projekten (z.B. /7/) in die drei Bereiche Programmieren im Großen, Programmieren im Kleinen, Projektorganisation / Projektmanagement.

Integriert wird in IPSEN in verschiedenen Bedeutungen verwendet. Zum ersten betrifft es den **Leistungsumfang** der Werkzeuge. Im einzelnen heißt das:

- Der größte Teil des Lebenszyklus wird durch die Werkzeuge abgedeckt.
- Die Werkzeuge arbeiten dokumentenübergreifend, d.h. es gibt Werkzeuge, die in unterschiedlichen Dokumenten vorhandene Informationen bereitstellen und modifizieren.
- Konsistenzbedingungen innerhalb und zwischen Dokumenten werden abgeprüft.

Zweitens setzt sich der **integrierte** Ansatz in der **internen Realisierung** fort. Im einzelnen heißt das:

- Alle Dokumente werden intern durch eine einheitliche graphähnliche Datenstruktur repräsentiert. Zu jedem Dokument existiert genau ein Graph. Die einzelnen Graphen sind durch zusätzliche Kanten untereinander verbunden. So ist eine übersichtliche Darstellung aller notwendigen Informationen über die einzelnen Dokumente und ihre Beziehungen untereinander möglich.
- Das Arbeiten mit einem Werkzeug bedeutet intern immer die Veränderung des (der) entsprechenden Graphen durch das Werkzeug. (Die Funktionalität aller Werkzeuge kann so durch einen formalen Kalkül spezifiziert werden. Dieser formale Kalkül sind Graphgrammatiken /9/.)

Die dritte Bedeutung des Begriffs **Integration** ist das zentrale Anliegen dieses Papiers. Hierbei geht es um die Gestaltung der **IPSEN-Benutzerschnittstelle.** In diesem

Zusammenhang heißt Integration:
- Alle Werkzeuge arbeiten kommandogestützt, inkrementell und syntaxgestützt.
- Ihre Bedienung durch den Benutzer ist einheitlich, d. h. unabhängig von ihrem Leistungsumfang oder den Dokumenten, auf denen sie arbeiten.
- Es existieren keine sogenannten expliziten Modi mehr (z. B. Editormodus, Testmodus, Ausführungsmodus, ...). Kein Werkzeug muß somit durch ein spezielles Kommando aktiviert und deaktiviert werden, sondern der Benutzer kann immer die in der aktuellen Situation sinnvollen Kommandos aller Werkzeuge aufrufen.
- Die Bildschirmgestaltung ist einheitlich für alle Werkzeuge und alle Dokumente.

Die ersten beiden Bedeutungen des Begriffs Integration sind hier nur schlagwortartig angerissen worden. Sie werden nicht weiter vertieft, sondern sind Thema anderer Veröffentlichungen (vgl. /2/, /4/, /6/). Im folgenden Abschnitt werden die wesentlichen Konzepte der Benutzerschnittstelle von IPSEN näher erläutert und mit einigen Beispielen aus dem Bereich Programmieren im Kleinen illustriert. Diese Beispiele können aber genauso aus den anderen Bereichen gewonnen werden. Im zweiten Teil des Papiers schildern wir die wichtigsten Entwurfsüberlegungen und -entscheidungen bei der Realisierung des IPSEN Ein-/Ausgabesystems. Dieses E/A - System ist die Grundlage für die Implementierung der vorher erläuterten Konzepte. Wir werden aber andeuten, daß sowohl die hinter dem Entwurf stehenden Überlegungen als auch die Implementierung selbst allgemein in Dialogsystemen Verwendung finden können. Am Schluß gehen wir kurz auf den Stand der vorliegenden Implementierung ein.

2 Eine integrierte Benutzerschnittstelle

2.1 Inkrementelle Arbeitsweise

Die während der Softwareentwicklung entstehenden unterschiedlichen Dokumente haben normalerweise sehr unterschiedliche **Repräsentationen** (in Form von Text oder Graphik); weiterhin möchte man eventuell verschiedene Repräsentationen des gleichen Dokuments haben (z. B. Quelltext und Struktogramm). Die Werkzeuge, mit denen diese Dokumente erstellt werden, orientieren sich in vielen herkömmlichen Systemen an dieser externen Repräsentation (z. B. zeilenorientierte Editoren oder Graphikeditoren).

Der logische (und korrekte) Aufbau eines Dokuments wird dagegen in IPSEN aus einer zugrundeliegenden abstrakten Syntax abgeleitet. Diese Definition ist weitgehend unabhängig von einer konkreten Repräsentation. Die logischen Einheiten einer solchen syntaktischen Beschreibung nennen wir in IPSEN **Inkremente.** Inkremente sind beispielsweise in einem Programm eine Anweisung oder eine Typdeklaration, in einer Modulspezifikation der Spezifikationsanteil für eine Zugriffsoperation, in einer Anforderungsdefinition ein Kapitel usw..
Bei der Anzeige eines Dokuments ist ein solches Inkrement immer als das aktuelle Inkrement des Dokuments ausgezeichnet. Es wird analog zu einem sonst üblichen Bildschirmcursor durch eine spezielle Darstellung (z.B. leuchtintensiv) von den anderen abgehoben.

Durch dieses aktuelle Inkrement ist festgelegt, welche Kommandos vom Benutzer

in dieser spezifischen Situation eingegeben werden dürfen. Wir nennen sie die für ein aktuelles Inkrement **gültigen Kommandos.** Gültige Kommandos sind beispielsweise die für dieses Inkrement sinnvollen Edierkommandos (Einfügen, Löschen, Ändern, ...) und die möglichen Instrumentierungs-, Ausführungs- und Testkommandos. Instrumentierungskommandos für eine einzelne Anweisung sind das Eintragen einer Zusicherung oder eines Unterbrechungspunktes. Eine Schleife kann zusätzlich durch einen Schleifenzähler instrumentiert werden. Testkommandos für eine Anweisung sind das Ansehen aller Variablenwerte der in der Anweisung verwendeten Variablen oder das Ansehen des Speicherzustands. Diese Kommandos werden sinnvollerweise nach Ausführung dieser Anweisung in einem Testlauf aufgerufen.

Diese **inkrementelle kommandogestützte Arbeitsweise** hat folgende Vorteile: (1) Kommandos beziehen sich immer auf Inkremente, d.h. logische Einheiten des Dokuments. Sie sind unabhängig von jeder Repräsentation. Der Benutzer muß nicht zwischen Kommandos unterscheiden, die einerseits nur Repräsentation modifizieren, andererseits den logischen Aufbau des Dokuments manipulieren. (2) Konkrete Syntax kann automatisch generiert werden. (3) Übliche Korrektionszyklen für vollständige Dokumente entfallen. Zum Beispiel entfällt der Zyklus Edieren - Übersetzen - Edieren bis ein Programm syntaktisch korrekt ist, weil bei der Eingabe neuer Inkremente die kontextfreie und kontextsensitive Syntax sofort überprüft werden kann.

Eine solche Arbeitsweise wurde in der Literatur zunächst für Editoren für das Programmieren im Kleinen vorgeschlagen **(syntax-gesteuerte Editoren)** (vgl. z. B. /1/, /10/). Sie wurde aber nicht oder nur zum Teil auf andere Bereiche übertragen (vgl. z. B. /7/). In IPSEN wird sie konsequent auf die Bearbeitung aller Dokumente angewendet.

2.2 Kommandoeingabe

Die Eingabe der oben skizzierten Kommandos ist auf zwei Weisen möglich. Man kann ein Kommando aus einem Menü selektieren oder man gibt es als Zeichenkette ein (eine sogenannte Kommandokurzbezeichnung). Diese Möglichkeiten unterstützen das unterschiedliche Verhalten der verschiedenen denkbaren **Benutzergruppen** (Anfänger, Experte).

Der **Menüaufbau** ist übersichtlich, d.h. Menüs enthalten nicht zu viele Alternativen. Wir erreichen das, indem wir Kommandos in Kommandogruppen zusammenfassen, z.B. in Edierkommandos, Instrumentierungskommandos, usw.. Bei der Anzeige eines Menüs ist immer nur eine solche Kommandogruppe expandiert, d.h. alle ihre Alternativen zu sehen. Die jeweils expandierte Kommandogruppe wird solange nicht gewechselt wie nur Kommandos aus dieser Gruppe selektiert werden. Dies soll die übliche Bearbeitung eines Dokuments unterstützen, zunächst ein Dokument zu erstellen, dann ggf. zu instrumentieren und anschließend zu testen.
Das Erlernen der Kurzbezeichnungen unterstützen wir, indem wir in den Menüs die Kommandokurzbezeichnungen mit anzeigen.

Der **Aufbau** der **Kurzbezeichnungen** orientiert sich an der Kommandogruppeneinteilung. Die Bezeichnungen sind aus drei Buchstaben zusammengesetzt. Der erste

Buchstabe gibt die Kommandogruppe an, der zweite (und falls zur Signifikanz nötig) ggf. dritte Buchstabe identifiziert das Kommando in dieser Kommandogruppe. In vielen Fällen ist bereits der zweite Buchstabe signifikant. Dies ermöglicht einen weiteren Komfort. Der Benutzer kann bei der Eingabe den ersten Buchstaben weglassen, wenn er die gleiche Kommandogruppe benutzt wie bei der Eingabe des vorhergehenden Kommandos. Da in vielen Fällen bereits der zweite Buchstabe ein Kommando in einer Kommandogruppe eindeutig identifiziert, ist in diesen Fällen die Kommandoeingabe über einen einzigen Tastendruck möglich.

Alle die bisher erläuterten Eingabemöglichkeiten für Kommandos existieren parallel nebeneinander. Es gibt also keinen explizit aufzurufenden "Menümodus" oder "Tastaturmodus", bei dem dann die Eingabe nur auf diese eine Art möglich ist. Kommt ein Benutzer aber ohne Menüeingabe aus, verschwenden diese Menüs unnötig Platz auf dem ohnehin kleinen Bildschirm. Für diesen Fall sehen wir einen einfachen **Übergang** zwischen **Anzeigen** und **Nichtanzeigen** von **Menüs** vor. Menüs werden dem Benutzer solange automatisch angezeigt, bis er einmal auf eine Selektion in einem Menü verzichtet und sein Kommando durch eine Kurzbezeichnung eingibt. Will er dann später wieder Menüs angezeigt bekommen, erreicht er das, indem er ein "?" statt eines Kommandos eingibt. Dieser einfache Übergang unterstützt eine weitere große Benutzergruppe. Viele Benutzer kennen die Kommandos einer Kommandogruppe besser als die einer anderen Gruppe. Solange sie sich nur in der ihnen gut bekannten Kommandogruppe bewegen, verzichten sie auf die Menüs. Sie haben dann bei einem Wechsel der Kommandogruppe eine einfache Möglichkeit, zu erreichen, daß wieder Menüs angezeigt werden.

Weitere **automatische Adaptionen** des Systems analog dem Wechsel von der automatischen Menüanzeige zum Nichtanzeigen sind vorgesehen. Bei der sequentiellen Eingabe eines Programms wird automatisch eine Cursorbewegung zum nächsten einzugebenden Inkrement ausgeführt. (Noch nicht eingegebene Inkremente werden in diesem Fall durch eine Lücke im Text bzw. durch ein Metasymbol auf dem Bildschirm dargestellt.) Erfolgt die Kommandoeingabe über Menüs, so werden ggf. benötigte weitere Parameter in eine Maske eingegeben, die erläuternden Kommentar enthält.

2.3 Bildschirmaufbau

Als letztes wollen wir in diesem Abschnitt an einem Beispiel zeigen, wie der Bildschirminhalt bei der Bearbeitung eines Dokuments mit IPSEN aufgebaut ist (vgl. Fig. 1).

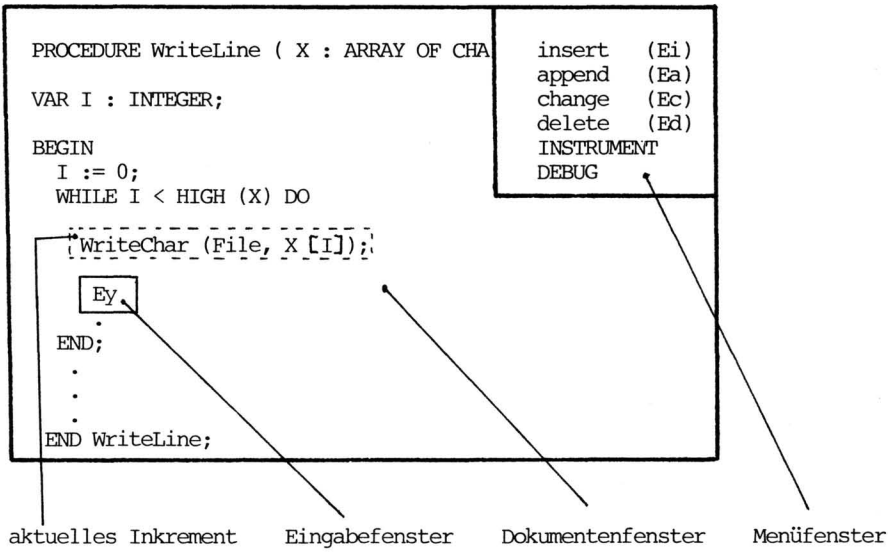

Fig.1: Bildschirmaufbau in IPSEN

Der Bildschirminhalt setzt sich zusammen aus (sich eventuell überlappenden) **Fenstern** vier verschiedener **Typen.** Diese Typen sind Dokumentenfenster (sie dienen zur Repräsentation von Dokumenten), Menüfenster (sie dienen zur Darstellung von und Selektion in Menüs), Eingabefenster (sie dienen zur Eingabe von Kommandos und Parametern) und Nachrichtenfenster (sie dienen zur Ausgabe von System- und Fehlermeldungen). Von jedem Typ können mehrere aber auch gar kein Fenster angezeigt sein, wie es in obigem Beispiel beim Nachrichtenfenster der Fall ist. Nur ein Dokumentenfenster ist sinnvollerweise immer angezeigt.

Jeder Fenstertyp charakterisiert spezifische Entscheidungen über den Bildschirmaufbau. Im einzelnen heißt das:
Dokumentenfenster: Das Fenster überdeckt den ganzen Bildschirm. Das aktuelle Inkrement wird leuchtintensiv dargestellt. Ein neues aktuelles Inkrement wird durch einen Cursor selektiert, der von einer Maus gesteuert wird.
Menüfenster: Das Fenster wird immer soweit rechts oben als möglich angezeigt. Dadurch wird so wenig wie möglich von dem anderen Text verdeckt. Die Alternativen stehen untereinander. Die Selektion erfolgt ebenfalls über den Mauscursor. Menüs sind andersfarbig gekennzeichnet.
Eingabefenster: Sie sind ebenfalls farblich abgehoben. Ihre Anzeige erfolgt immer unter dem aktuellen Inkrement. (Hierauf bezieht sich das Kommando oder der Parameter, der in dem Fenster eingegeben wird, normalerweise.)
Nachrichtenfenster: Nachrichten werden immer rot dargestellt und unten auf dem Bildschirm angezeigt. Sie werden erst gelöscht, wenn der Benutzer durch einen Mausclick in dem Fenster bestätigt, daß er sie gesehen hat.

3 Das Ein-/Ausgabesystem

3.1 Anforderungen

In diesem Abschnitt beschreiben wir den **Entwurf** des Ein-/Ausgabesystems von IPSEN. Mit Hilfe dieses Systems lassen sich die im letzten Abschnitt beschriebenen Konzepte in eine konkrete Implementierung übertragen. Der Entwurf ist in einem Modulkonzept beschrieben, das ebenfalls im Rahmen des Projekts IPSEN entwickelt wurde (/6/).

Das angestrebte Ziel bei der Entwicklung des Entwurfs ist, eine möglichstgroße **Portabilität** und **Adaptabilität** der Implementierung zu erreichen. Portabilität heißt hier die leichte Übertragbarkeit auf unterschiedliche Bildschirmgeräte. Adaptabilität heißt die leichte Anpassung an neue Anforderungen, insbesondere die Gestaltung der Benutzerschnittstelle betreffend. Diese Ziele sollten aber für beliebige interaktive Systeme Gültigkeit haben, sodaß die hier angewendeten Entwurfsstrategien und die entstehende Systemarchitektur auf diese Systeme übertragbar sind.

3.2 Ein allgemeines Fensterverwaltungssystem

Obige Ziele erreicht man, indem man (1) alle Entwurfsentscheidungen, die keine anwendungsspezifischen (in unserem Fall IPSEN spezifischen) Dinge betreffen in eigenen Teilsystemen verkapselt, (2) die Entwurfsentscheidungen soweit wie möglich lokalisiert, um bei Änderungen möglichst nur die Implementierung eines kleinen Teilsystems bzw. Moduls ändern zu müssen (vgl. /5/).

Dies bedeutet für unser Ein-/Ausgabesystem zunächst die Einteilung in zwei Teilsysteme. Das erste ist ein allgemeines Fenstersystem zur Verwaltung sich **überlappender Fenster** und deren Inhalt (vgl. Fig. 2). Es stellt Zugriffsoperationen zur Verfügung wie z.B. elementare Operationen auf Fenstern (Öffnen, Schließen, Verschieben, ...), elementare Operationen auf den Fensterinhalten (Lesen/Schreiben von Zeichenketten und Schreiben von Graphiksymbolen, Verwaltung von Cursorbewegungen, ...), die Festlegung von speziellen Fenstergeometrien (Form und Aussehen des Rahmens, Lage der Fenster auf dem Bildschirm, ...).

Portabilität des allgemeinen Fensterssystems erreicht man, indem man die Implementierung auf den Ressourcen eines sogenannten **virtuellen Terminals** aufbaut. Dieses virtuelle Terminal stellt unabhängig von der zugrundeliegenden Hardware elementare Ein-/Ausgabeoperationen zur Verfügung (Schreiben von Zeichenketten, Lesen von Tastatureingaben,...). Bei einer Portierung des Fenstersystems müssen dann nur diese Operationen neu implementiert werden.

Ein solches virtuelles Terminal in sehr großer Allgemeinheit wird durch das **graphische Kernsystem** (GKS vgl. /3/) realisiert. Verwendet man eine GKS-Implementierung als Basis für ein solches Fenstersystem und geht davon aus, daß GKS in nächster Zukunft auf vielen Terminals zur Verfügung steht, vermeidet man sogar die Neuimplementierung der elementaren Operationen bei der Portierung.
Die große Allgemeinheit, die das GKS zur Verfügung stellt, ist durchaus sinnvoll

eingesetzt, wenn man sich überlegt, daß Dokumente auch in graphischer Repräsentation dargestellt und manipuliert werden (Modulspezifikationen, Struktogramme, usw.).

Ein weiterer Vorteil der Verwendung von GKS ist, daß das GKS-Konzept "Segmentspeicher" zur Implementierung des Fenstersystems gut eingesetzt werden kann. Die Implementierung eines solchen Fenstersystems erfordert nämlich im wesentlichen **zwei Datenstrukturen** zur Speicherung aller notwendigen Informationen. Die Inhalte der einzelnen Fenster müssen gespeichert werden, damit sie regenerierbar sind, wenn ein Fenster gelöscht wird, das ein anderes teilweise oder ganz überdeckt hat. Desweiteren muß die geometrische Position der Fenster auf dem Bildschirm und ihre Größe gespeichert werden, um Lese- und Schreiboperationen relativ zu Fenstergrenzen durchführen zu können. Diese beiden Datenstrukturen (deswegen werden sie in Datenmodulen verkapselt) werden in den beiden Modulen WINDOW DESCRIPTION und WINDOW CONTENTS verkapselt, die wieder mit Ressourcen des GKS und hier mit Zugriffen auf den Segmentspeicher realisiert sind (vgl. Fig. 2).

Der Modul WINDOW MANAGER koordiniert die Zugriffsoperationen auf diese Module bzw. auf den Bildschirm. Seine Implementierung sorgt dafür, daß die Änderung des aktuellen Bildschirminhalts ebenfalls in den beiden internen Datenstrukturen gespeichert wird. An seiner Schnittstelle exportiert der Modul die oben skizzierten Zugriffsoperationen. Da er keine weiteren Datenstrukturen verkapselt, bezeichnen wir ihn als Funktionsmodul.

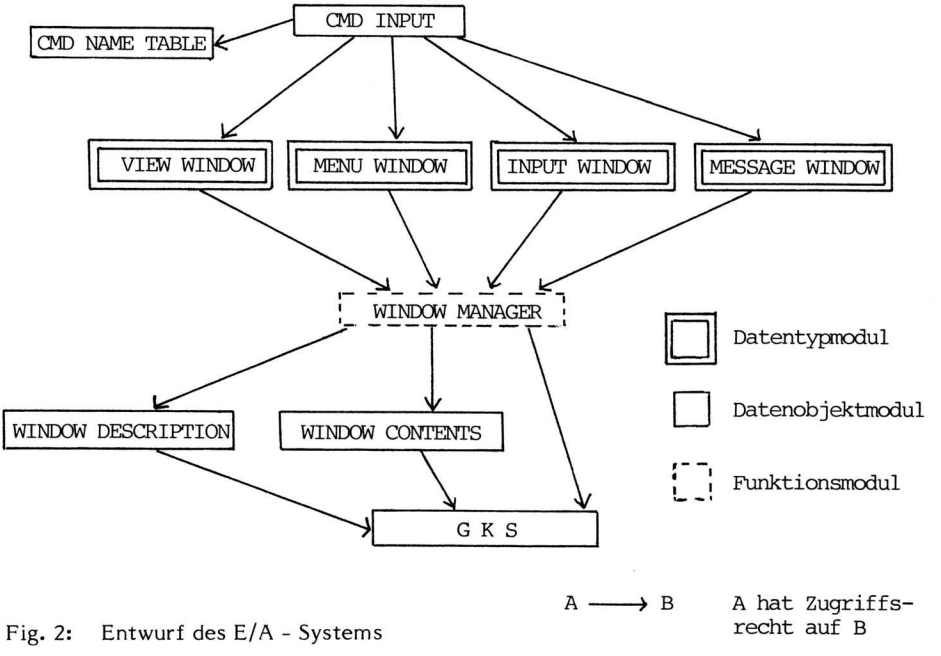

Fig. 2: Entwurf des E/A - Systems

3.3 Anwendungsspezifische Ein-/Ausgabe

Mit Hilfe des bisher beschriebenen Systems können die anwendungsspezifischen Entwurfsentscheidungen implementiert werden. Da solche Entscheidungen soweit wie möglich lokalisiert werden sollen, ergeben sich für die Implementierung von IPSEN zunächst vier Module, die die Entscheidungen über die Gestalt der vier in 2.3 beschriebenen Fenstertypen verkapseln (vgl. Fig. 2). Diese Module sind Datentypmodule um anzugeben, daß es möglich ist, mehrere Fenster desselben Typs gleichzeitig zu verwalten.

Im einzelnen exportieren die Module beispielhaft folgende Zugriffsoperationen:
VIEW WINDOW: Ausgabe von speziellen Zeichenketten (spezielle Größe, spezielle Schrift) oder Symbolen (Dreieck, Kreis), um die Repräsentation von Inkrementen zu erzeugen, Ausgabe von Darstellungsattributen (fett, invers), Ermitteln des aktuellen Inkrements.
MENU WINDOW: Anzeigen von Menüs, Selektion in Menüs.
INPUT WINDOW: Ausgabe von Zeichenketten in einer bestimmten Darstellung (z.B. für erläuternden Kommentar), Einlesen von Benutzereingaben mit Echo im Fenster.
MESSAGE WINDOW: Ausgabe von Fehlermeldungen in einer bestimmten Darstellung, Löschen der Meldung nach Bestätigung durch den Benutzer.

In den letzten beiden noch zu erläuternden Modulen von Fig. 2 sind die Entscheidungen über die Möglichkeiten der Kommandoeingabe implementiert. CMD INPUT exportiert im wesentlichen nur eine Ressource. Diese liest ein Kommando ein und reicht einen internen Kommandonamen weiter. Außerdem gibt es nur noch Ressourcen für das Setzen und Lesen der aktuellen Kommandogruppe. CMD NAME TABLE enthält eine Tabelle, die jedem internen Kommandonamen die dem Benutzer bekannten Namen zuordnet, d.h. die Kurzbezeichnungen und die einzelnen Alternativen in den Menüs.

3.4 Zusammenfassung

Zum Schluß sollen die wesentlichen Punkte und Vorteile des entstandenen Entwurfs kurz zusammengefaßt werden. Die Portabilität des gesamten Systems ist durch die zugrundeliegende GKS - Schnittstelle gewährleistet. In dem allgemeinen Fenstersystem sind keine anwendungsspezifischen Entscheidungen über die Benutzerschnittstelle implementiert. Diesbezügliche Änderungen haben deshalb keinen Einfluß auf die Schnittstelle oder die Implementierung des Fenstersystems. Vielmehr kann bei einer vorliegenden Implementierung dieses Systems die anwendungsspezifische Benutzerschnittstellengestaltung in Form eines **rapid prototyping** mit dem Fenstersystem getestet werden. Ein solches System läßt sich somit immer einsetzen, wenn eine Dialogschnittstelle mit überlappenden Fensten und komfortablen Eingabemöglichkeiten entwickelt werden soll.

Die anwendungsspezifische Fensterschicht ist änderungsfreundlich, da alle Entwurfsentscheidungen über die Gestalt der vier Fenstertypen in unterschiedlichen Modulen liegen. Änderungen eines Typs bleiben somit auf einen Modul beschränkt. Diese Schicht kann außerdem durch die Implementierung **weiterer Fenstertypen**

erweitert werden, falls man mit den vorgeschlagenen Typen allein nicht auskommt. So ist auch diese Schicht allgemeiner als nur in IPSEN einsetzbar.

Die spezifische Form der **Kommandoeingabe** ist **änderbar** allein durch Änderung bzw. Austausch des Moduls CMD INPUT. Noch einfacher ist natürlich die Änderung, wenn nur die Namen der externen Kommandos ausgetauscht werden sollen. In diesem Fall braucht man nur die Tabelle im Modul CMD NAME TABLE zu ändern.

4 Implementierung

Die Implementierung des E/A - Systems erfolgte auf einem IBM PC XT mit Farbbildschirm in der Programmiersprache Modula-2. Eine Implementierung des GKS liegt allerdings bisher nicht vor. Es wurden selbst Ressourcen geschrieben, die denen entsprechen, die das GKS sonst zur Verfügung stellen würde, sodaß eine spätere Umstellung auf eine dann ggf. vorliegende GKS - Implementierung möglich ist. Unsere vorliegende Implementierung ist aber keineswegs eine umfassende GKS - Implementierung. Der vorhandene Satz an Funktionen beschränkt sich auf die Ausgabe von Zeichen (-ketten) und Linien sowie deren Darstellungsattributen. Weiter sind Funktionen vorgesehen, die Zeichenketten einlesen und die Position eines Cursors (Locators) zurückliefern. Es gibt keine Implementierung des Segmentspeichers. Hier wird das zugrundeliegende Dateisystem des vorhandenen Betriebssystems benutzt.

Danksagung Für viele Diskussionen und Anregungen zu diesem Papier bedanke ich mich bei Prof. M. Nagl und meinen Kollegen G. Engels und C. Lewerentz.

5 Literatur

/1/ Donzeau-Gouge, M. et.al.: Programming Environments Based on Structured Editors - The MENTOR Experience, Techn. Bericht 26, INRIA, Frankreich 1980

/2/ Engels, G./Gall, R./Nagl, M./Schäfer, W. : Software Specification using Graph Grammars, Computing 31, Springer 1983, 317 - 346

/3/ Encarnacao/Strasser (ed.): Geräteunabhängige graphische Systeme, München: Oldenbourg 1981

/4/ Engels, G./Schäfer, W. : Graph Grammar Engineering: A Method Used for the Development of an Integrated Programming Support Environment, in TAPSOFT conference proceedings, Berlin, LNCS 186, Springer 1985, 179 - 193

/5/ Engels, G./Schäfer, W. : Design of an Adaptive and Portable Programming Support Environment, in Proceedings of ICS 85, Florenz, North Holland, 297 - 308

/6/ Lewerentz, C./Nagl, M.: Incremental Programming in the Large: Syntax- aided Specification Editing, Integration and Maintenance, in Proceedings of the 18th Hawaii International Conference on System Science 1985, 638 - 649

/7/ Habermann, N. et.al.: The Second Compendium of GANDALF Documentation , Techn. Bericht, Dept. of Computer Science, Carnegie-Mellon University, Pittsburgh 1982

/8/ Nagl, M.: An Incremental Programming Support Environment, in Computer Physics Communications, North-Holland 1984

/9/ Nagl, M.: Graph-Grammatiken, Theorie, Anwendung, Implementierung,Vieweg 1979

/10/ Teitelbaum, T./Reps, T.: The Cornell Programm Synthesizer - A syntax-directed Programming Environment, CACM 24, 9, 1981, 563 - 573

/11/ Nievergelt, J.: Die Gestaltung der Mensch-Maschine Schnittstelle, in Kupka, I. (Hrsg.): 13. Jahrestagung der GI, Informatik - Fachberichte Bc. 73, Berlin, Springer Verlag 1983, 41 - 50

Wilhelm Schäfer
Angew. Informatik, FB 6
Universität Osnabrück
Postfach 4469
4500 Osnabrück