Visualisierungen beim Entwicklungsprozeß experimenteller Programmsysteme

Volker Haarslev und Ralf Möller, Hamburg

Dieser Beitrag beschreibt einen neuen Ansatz, den aufgabenorientierten Entwurf von experimentellen Programmsystemen graphisch zu unterstützen. Als Teil dieses Ansatzes haben wir die visuelle Programmierumgebung VIPEX implementiert. VIPEX definiert eine visuelle Sprache, mit der Experimentalsysteme erzeugt werden können, die auf einer Datenflußarchitektur basieren. Als Strukturierungsmittel werden Hierarchien und Gruppen sowie Vererbungsmechanismen angeboten.

1 Einleitung

Es gibt seit einigen Jahren eine Forschungsinitiative, die sich mit dem Bereich "Visualisation in Scientific Computing" [*McCormick et al. 87*] (kurz: Visualisierung) befaßt. Hierbei bezeichnet 'Visualisierung' den Vorgang, aus symbolischen Daten geometrische zu erzeugen und diese als Graphiken darzustellen. Die Visualisierung vereint bisher weitgehend unabhängige Bereiche wie Computer-Graphik, Bildverarbeitung, maschinelles Sehen, rechnergestützter Entwurf (CAD), Signalverarbeitung und Entwurf von Benutzeroberflächen. Das Ziel der Visualisierung besteht darin, existierende wissenschaftliche Methoden zu verbessern und neue zu entwickeln, indem neue wissenschaftliche Einsichten durch visuelle Methoden vermittelt werden.

Die Motivation für die in diesem Beitrag vorgestellte Arbeit ist darin zu sehen, daß es zur Zeit weder allgemein verfügbare Programmiersprachen noch Betriebssysteme gibt, die Visualisierungstechniken in angemessener Weise ermöglichen oder selber nutzen. Alle existierenden großen Betriebssysteme unterstützen überwiegend eine symbolische und numerische Verarbeitung. Erste Ansätze existieren nur für Arbeitsplatzrechner (z.B. Macintosh™) und im Bereich der *visuellen Programmiersprachen* [*Shu 86, Chang 87*].

Wir beschreiben nachfolgend einen Ansatz, den Entwurf von experimentellen Programmsystemen durch Visualisierungen zu unterstützen. Die experimentelle Entwicklung von Programmen ist in vielen Forschungsbereichen ein interaktiver Vorgang. Um diesen Vorgang zu intensivieren, muß dem Entwickler die Möglichkeit gegeben werden, seine Berechnungen interaktiv und möglichst in Echtzeit zu steuern. Es werden dabei beispielsweise Parameter, Auflösung oder Darstellung der Verarbeitungsprozesse verändert. Eine sofortige visuelle Rückmeldung ist notwendig, um die resultierenden Auswirkungen zu inspizieren und neue Erkenntnisse oder Anomalien zu entdecken.

In diesem Kontext sind somit zwei wichtige Ziele zu nennen. Das erste Ziel besteht darin, dem Benutzer Werkzeuge anzubieten, die die graphische Darstellung von Daten erleichtern. Der Systemverlauf kann dann leichter beobachtet und eine fehlerhafte Verarbeitung möglichst frühzeitig erkannt werden. Dieser Punkt ist insbesondere in Bereichen mit langen Antwortzeiten bei der Verarbeitung von Bedeutung (z.B. Simulation, Bildverarbeitung).

Das zweite Ziel ist darin zu sehen, dem Benutzer einen effektiven Einfluß auf den Systemablauf anzubieten. Wichtige Interaktionsziele sind dabei die Kontrolle des Datenflusses, die Veränderung der Programmarchitektur und die Steuerung von Programmkomponenten durch Parameter. Der Zustand und die Struktur eines Programmsystems sollten deshalb in Form einer graphischen Darstellung der beteiligten Verarbeitungsprozesse sowie ihrer Parameter visualisiert werden.

Wir haben deshalb das System VIPEX (*Visual Programming of Experimental Systems*) entwickelt, das einen ersten Versuch darstellt, die beiden oben genannten Ziele im Rahmen einer Programmierumgebung zu verwirklichen. VIPEX entstand aus der Erfahrung mit ODISA (*Object-oriented Dialog System for Image Sequence Analysis*) [*Haarslev 87a, Haarslev 87b*]. ODISA wurde für eine Klasse von Bildfolgenauswertesysteme entwickelt, die mithilfe von GENESYS (*Generic Experimental Systems*) [*Faasch 87*] erzeugt werden können. ODISA und GENESYS sind als Teil einer objektorientierten Programmierumgebung für Bildfolgenauswertesysteme anzusehen [*Faasch & Haarslev 88*].

VIPEX erweitert und verallgemeinert den bei ODISA zugrunde gelegten Ansatz in verschiedenen Bereichen. VIPEX arbeitet in einer Lisp-Umgebung und stellt einen Rahmen zur Dialoggestaltung zur Verfügung, der von der jeweiligen Anwendungsdomäne unabhängig ist. Eine Anwendung von VIPEX auf den Bereich der Bildverarbeitung findet sich in *Haarslev & Möller 88a* und *Haarslev & Möller 88b*.

VIPEX definiert eine visuelle Programmiersprache, die dem Benutzer die Realisierung eines Programmsystems in Form eines Datenflußnetzes ermöglicht. Die elementaren Knoten des Netzes bestehen aus *Verarbeitungsobjekten*, die über *Datenleitungen* miteinander verbunden werden. Der Benutzer kann die Verarbeitungsfunktion von Knoten definieren, indem Lispfunktionen an die Verarbeitungsobjekte gebunden werden.

Der hierarchische Programmentwurf und die visuelle Programmierung werden durch *Kompositionsobjekte* und *Gruppenobjekte* unterstützt. Diese virtuellen Objekte dienen dazu, komplexe Objekte zu erstellen oder Teilnetze bzw. beliebige Gruppen zu repräsentieren. Objekte verfügen über eigene *Parameter*, die eine Kontrolle ihrer Operationen (d.h. ihrer Lispfunktionen) erlauben. Abhängigkeiten zwischen Objektparametern werden mithilfe von *Vererbungsgraphen* beschrieben.

VIPEX stellt fortlaufend die aktuelle Architektur von Experimentalsystemen graphisch dar und visualisiert den momentanen Zustand des Systems. Dabei werden der Verarbeitungsfortschritt von Operationen sowie der Datenfluß ständig angezeigt. Die graphische Darstellung von Objekten besteht aus *Objektpiktogrammen*. Der Benutzer erhält weiterhin Visualisierungen von Daten, die während des Programmablaufs entstehen. VIPEX verwendet als Ausgabegerät ein Farbrastergraphiksystem sowie eine Maus und eine Tastatur als Eingabegeräte [*Möller 88*].

2 Visualisierung mentaler Bilder

Aufgrund der Erfahrung vieler Wissenschaftler finden das menschliche Denken und die damit verbundenen Problemlösungsprozesse überwiegend in visuellen Kategorien statt [*Benzon 85, Tauber 87*]. Die Strukturierung von Experimentalsystemen ist ein Bestandteil der Systementwicklung und unterliegt damit auch dem menschlichen Denken. Um den Entwickler bei diesen Vorgängen zu unterstützen, erscheint es sinnvoll, seine mentalen Bilder von Experimentalsystemen graphisch darzustellen.

Wir sind der Meinung, daß für eine bestimmte Klasse von Experimentalsystemen ein gemeinsames mentales Bild beschrieben werden kann. Dieses Bild resultiert aus der Architektur solcher Systeme und ergibt sich aus der Anwendung von Programmentwicklungstechniken wie strukturierter Systementwurf, Modularisierung, Verbergen von Information und Datenabstraktion. Der Benutzer entwirft ein Experimentalsystem als eine Kombination von funktionalen Einheiten, die flexibel miteinander verknüpft werden können.

Eine *funktionale Einheit* (FE) bezeichnet dabei eine 'elementare' Systemkomponente (z.B. Funktion, Modul, Objekt, Programm, Prozeß). Sie kann als Abbildung beschrieben werden, die bestimmte Eingangsdaten $(E_1, \ldots, E_l)$ unter Berücksichtigung der Parameter $(P_1, \ldots, P_n)$ in die Ausgangsdaten $(A_1, \ldots, A_m)$ abbildet: $\mathrm{FE}_{P_1,\ldots,P_n} : (E_1, \ldots, E_l) \longmapsto (A_1, \ldots, A_m)$.

Der nachfolgende Abschnitt beschreibt genauer, wie VIPEX ein derartiges mentales Bild graphisch darstellt.

3 Objektpiktogramme

Objektpiktogramme dienen dazu, bestimmte vom Benutzer zugeordnete Teile eines Experimentalsystems graphisch zu repräsentieren. Ein Objektpiktogramm (siehe Abb. 1) wird nachfolgend auch als *Zustandsdarstellung* eines Objektes bezeichnet. Dieses Piktogramm setzt sich aus einer Kombination von Unterpiktogrammen, Fenstern und Text zusammen. Die Leitungsanschlüsse werden als kleine Fenster dargestellt, die sich auf der linken Seite (als Eingänge) bzw. der rechten (als Ausgänge) befinden. Ein in der oberen linken Ecke alloziertes Unterpiktogramm (als *Funk-*

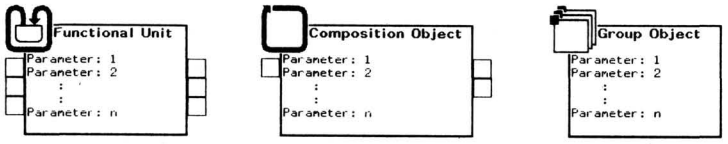

Abbildung 1: Piktogramme für Verarbeitungs-, Kompositions- und Gruppenobjekte

tionssymbol bezeichnet) charakterisiert das funktionale Verhalten des dargestellten Objektes. Das Hauptfenster enthält den Objektnamen und optional eine Liste von Objektparametern.

3.1 Verarbeitungsobjekte

Funktionale Einheiten werden als *Verarbeitungsobjekte* modelliert. Sie führen die eigentliche Verarbeitung durch, indem die vom Benutzer formulierten und mit Verarbeitungsobjekten assoziierten Lispfunktionen mit den aktuell gültigen Werten der Objektparameter sowie den bereitstehenden Eingangsdaten evaluiert werden.

Der Benutzer verfügt über zwei einfache Mechanismen, um den Systemablauf zu kontrollieren. Er kann den Datenfluß zwischen Objekten steuern, indem er Leitungen hinzufügt oder entfernt. Weiterhin ist er in der Lage, den Verarbeitungszustand von Objekten (mithilfe von Maus und Menü) zu kontrollieren. Der Verarbeitungszustand kann den Wert `STOP` (keine Verarbeitung), `STEP` (genau einen Satz von Daten) oder `CYCLE` (ständig verarbeitungsbereit) haben, der durch die Farbe des Funktionssymbols (rot, gelb oder grün) angezeigt wird. Ein Objekt kann seine Verarbeitung allerdings nur dann beginnen, wenn alle Dateneingänge belegt und alle -ausgänge leer sind.

Neben der Zustandsdarstellung eines Objektes existiert weiterhin auch eine *Strukturdarstellung*, die sich der Benutzer ebenfalls jederzeit darstellen lassen kann. Bei den Verarbeitungsobjekten enthält diese Strukturdarstellung ein Fenster des Lisp-Editors, in dem die zugehörige Lispfunktion angezeigt wird. Der Benutzer kann somit leicht die Verarbeitungsvorschrift eines Verarbeitungsobjektes inspizieren, ändern und erneut im laufenden Experiment erproben.

Der Datenfluß zwischen Objekten kann ebenfalls inspiziert werden. Dafür muß der Benutzer die Leitungsanschlüsse von Objektpiktogrammen "öffnen", um sich die Struktur der eingehenden oder ausgehenden Daten innerhalb eines *Anschlußfensters* anzeigen zu lassen. Die Form dieser Strukturdarstellung ist jedoch in hohem Maße von dem Datentyp abhängig. Abbildung 3 zeigt eine Beispielkonfiguration für ein Problem aus der Bildverarbeitung (Objekterkennung in Bildfolgen, genauer in *Haarslev & Möller 88c*).

3.2 Datenleitungen

Datenleitungen verbinden Objekte miteinander. Sie werden als Linien dargestellt, die von Ausgangs- zu Eingangsanschlüssen gezogen werden. VIPEX zieht die Linien automatisch unter Verwendung eines Algorithmus', der für den Schaltungsentwurf entwickelt wurde. Die von VIPEX verwendete Version mußte jedoch modifiziert werden, um für Benutzer akzeptable Leitungswege zu erhalten.

Unsere Experimente haben gezeigt, daß Benutzer kurze, sich kreuzende Leitungen gegenüber langgezogenen, kreuzungsfreien Leitungen vorziehen. Dies liegt wohl darin begründet, daß Benutzer zum Verständnis eines Leitungsverlaufs diesen mit den Augen nachvollziehen müssen. Weiterhin werden Leitungswege vorgezogen, die mit einem gewissen Mindestabstand zu Objekten oder zentriert zwischen diesen verlaufen. Das Abrunden von Leitungsecken verhindert Mehrdeutigkeiten und erhöht die Akzeptanz beim Benutzer.

3.3 Kompositionsobjekte

Beim Zusammenfassen und Gruppieren von Objekten sollte der Benutzer im Rahmen des Entwicklungsprozesses ebenfalls unterstützt werden. In Analogie zu Petrinetzen erscheint es sinnvoll, Hilfsmittel zur hierarchischen Strukturierung vorzusehen. VIPEX bietet deshalb *Kompositionsobjekte* an. Der Benutzer kann ein Kompositionsobjekt interaktiv erzeugen, indem er ein zu repräsentierendes Teilnetz spezifiziert. In der graphischen Darstellung des Datenflußnetzes ersetzt das Piktogramm eines Kompositionsobjektes (d.h. dessen Zustandsdarstellung) das ihm hierarchisch untergeordnete Teilnetz.

Die Zustandsdarstellung eines Kompositionsobjektes ähnelt der eines Verarbeitungsobjektes. Die Anzahl der notwendigen Leitungseingänge und -ausgänge wird dynamisch aus dem Teilnetz abgeleitet. Sie ergibt sich aus den in das Teilnetz hinein- bzw. hinausführenden Leitungen (siehe Abb. 2). Abbildung 1 zeigt u.a. die Zustandsdarstellung eines entsprechenden Kompositionsobjektes. Die Strukturdarstellung eines Kompositionsobjektes besteht aus dem repräsentierten Teilnetz (siehe Abb. 3). Damit kann der Benutzer jederzeit auf eine detaillierte Darstellung eines Kompositionsobjektes zurückgreifen.

3.4 Objektparameter

Ein beim Entwurf experimenteller Algorithmen allgemein angewendetes Prinzip besteht darin, die zu entwickelnden Algorithmen mit Parametern zu versehen, um interaktiv eine detaillierte Kontrolle auf die Wirkungsweise der Algorithmen ausüben zu können. VIPEX bietet dem Benutzer deshalb die Möglichkeit, *Objektparameter* zu spezifizieren, die als ein Bestandteil der Zustandsdarstellung dargestellt werden (siehe Abb. 1, 3).

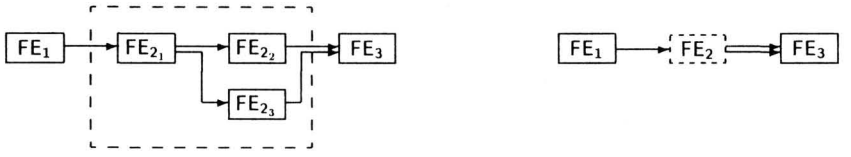

Abbildung 2: Vollständiges Netz mit Teilnetz und neues Netz mit Kompositionsobjekt FE_2

Bei den Verarbeitungsobjekten werden deren Parameter durch die Parameter der vom Benutzer formulierten Lispfunktion definiert. Die Parameter von Kompositionsobjekten werden von den hierarchisch untergeordneten Objekten des repräsentierten Teilnetzes bestimmt. Der Benutzer kann Parameter untergeordneter Objekte an ihr übergeordnetes Kompositionsobjekt binden. Dies bedeutet, daß diese Parameter in der Zustandsdarstellung des Kompositionsobjektes auftreten und auch nur dort inspiziert oder geändert werden können (z.B. in Abb. 3).

Durch diesen Mechanismus erhält der Benutzer die Möglichkeit, sich eine abstrakte Sicht eines Teilnetzes zu schaffen. Er kann das Kompositionsobjekt mit seinen Parametern als virtuelles Verarbeitungsobjekt ansehen, ohne sich um die Details des untergeordneten Teilnetzes kümmern zu müssen. Er kann auch Parameter von unterschiedlichen untergeordneten Objekten an denselben Parameter ihres Kompositionsobjektes binden. Dadurch ist der Benutzer in der Lage, in einfacher Weise eine konsistente Änderung mehrer Parameter gleichzeitig durchzuführen.

3.5 Gruppenobjekte

Neben der hierarchischen Strukturierung bietet VIPEX noch ein weiteres Hilfsmittel zum Systementwurf. Der Benutzer kann beliebige, nicht notwendigerweise hierarchisch geordnete Objekte mithilfe eines *Gruppenobjektes* zusammenfassen. Objekte können in mehreren Gruppenobjekten zugleich Mitglied sein. Die Zustandsdarstellung eines Gruppenobjektes enthält somit keine Leitungsanschlüsse, sondern nur Parameter (siehe Abb. 1). Diese Parameter ergeben sich analog zu den Kompositionsobjekten aus den Parametern der in der Gruppe enthaltenen Objekte (z.B. der Parameter "Histogram Style" in Abb. 3).

Das Setzen des Verarbeitungszustandes eines Gruppenobjektes bewirkt immer, daß der Verarbeitungszustand aller Gruppenmitglieder ebenfalls auf diesen Wert gesetzt wird, da dieser Zustand automatisch an die Gruppenmitglieder vererbt wird. Dadurch wird das gleichzeitige Verändern des Verarbeitungszustandes einer Menge von Objekten unterstützt (z.B. 'suspendiere alle Verarbeitungsobjekte'). Weiterhin gestatten Gruppenobjekte, zusätzliche Sichten von Objekten zu schaffen (z.B. mit einer Auswahl von Parametern) und benutzerspezifische Merkmale an Objekte zu binden (z.B. Voreinstellungen für Parameterwerte).

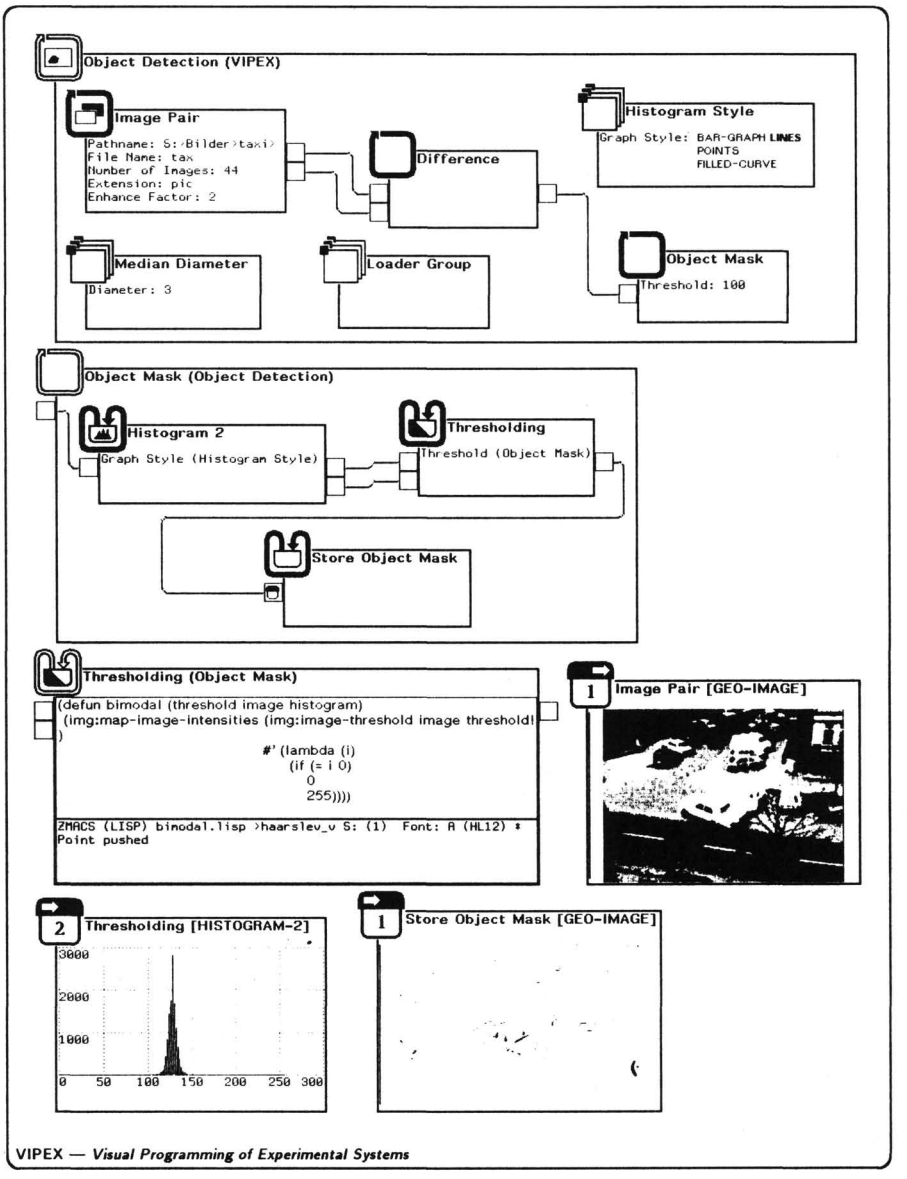

Abbildung 3: Beispielkonfiguration aus der Bildverarbeitung

4 VIPEX und andere Systeme

In der neueren Literatur finden sich zahlreiche Ansätze zur Visualisierung beim Programmierprozeß. Beispielsysteme sind u.a. die Programmierumgebung BALSA-II [*Brown 88*], die Lisp-

Programmierumgebung TINKER [*Lieberman 84*], das PICT-System zur bildhaften Programmierung [*Glinert & Tanimoto 84*] und das PVS-System zur Visualisierung dynamischer Prozesse [*Foley & McMath 86*]. Eine Übersicht und eine Diskussion mehrerer Systeme, die visuelle Programmiersprachen verwenden, findet sich in *Shu 86*.

VIPEX sollte mit zwei Ansätzen zur bildhaften Programmierung und mit einer Mehrsprachumgebung verglichen werden. Das GARDEN-System [*Reiss 87*] erlaubt den Entwurf mehrerer visueller Sprachen. Als Anwendungsbeispiele sind endliche Automaten, Flußdiagramme und Datenflußgraphen zu nennen. HI-VISUAL [*Hirakawa et al. 86*] ist eine Sprache, die die visuelle Programmierung mit Piktogrammen unterstützt. Als Anwendungsdomäne wird die Bildverarbeitung vorgestellt. Die Mehrsprachumgebung DAISY-II [*Fisher 88*] bietet ein Werkzeug zum Erzeugen von Blockdiagrammen an. Es werden Flußdiagramme von Programmen erzeugt, die über Unix-Leitungen (pipes) miteinander verknüpft sind. Parameter dieser Programme können durch Kontrolltableaus inspiziert und modifiziert werden.

Unser Ansatz unterscheidet sich gegenüber HI-VISUAL, GARDEN und DAISY-II insbesondere dadurch, daß VIPEX eine wesentlich detailliertere Darstellung der Objekte und ihres Datenflusses erlaubt. Es wird ein einheitliches Format für Objektpiktogramme und Datenelemente verwendet. Die mit Absicht klein gehaltene Menge von Piktogrammen soll dem Benutzer die Bildung eines Systemmodells erleichtern. Weiterhin bietet VIPEX Mechanismen, um experimentelle Systeme mithilfe virtueller Objekte zu strukturieren und um Vererbungsgraphen für Objektparameter zu definieren. Der Datenfluß kann durch die direkte Manipulation der Objektzustände und ihrer Leitungsverschaltung beeinflußt werden.

5 Mögliche Erweiterungen

Die Evaluierung der Lispfunktionen von Verarbeitungsobjekten kann in Anlehnung an die Netztheorie auch als Schalten (im Sinne einer Transition) aufgefaßt werden. Daraus ergibt sich, daß VIPEX gefärbte Netze [*Reisig 85*] realisiert. Da die Aktivierungsbedingung für alle Objekttypen gleich ist, stellen Zustandsdarstellungen von Kompositionsobjekten Vergröberungen und ihre Strukturdarstellungen Verfeinerungen von Netzen dar. Die Kombination "Datenausgang – Leitung – Dateneingang" kann als *Kanal* angesehen werden. Somit beschreiben zyklenfreie Systeme in VIPEX *kausale Netze mit individuellen Marken*, die als *Auftragssysteme* [*Jessen & Valk 87*] gedeutet werden können.

Komplexere Objekte wie beispielsweise ein Multiplexer lassen sich mit der oben genannten Aktivierungsbedingung nicht realisieren [*Möller 88, Haarslev & Möller 89*]. Deshalb erscheint es sinnvoll, in VIPEX ein weiteres Aktivierungsschema anzubieten, bei dem ein Objekt bei jedem Eintreffen (oder auch Verlassen) eines Datenelementes aktiviert wird. Kompositionsobjekte

stellen dann allerdings keine Vergröberungen bzw. Verfeinerungen im Sinne der Netztheorie dar.

Eine andere Erweiterung wäre, für einen Dateneingang bzw. -ausgang den Anschluß von mehreren Leitungen zuzulassen. Dabei ist der Fall unkritisch, daß zwei oder mehr Leitungen mit einem Dateneingang verbunden sind. Werden allerdings mehrere Leitungen an einem Datenausgang zugelassen, so müssen Entscheidungskriterien existieren, um eine Leitung auszuwählen. Eine nicht-deterministische Leitungsauswahl wird in den meisten Anwendungsfällen ungeeignet sein. Als Lösung ist denkbar, daß die nachgeschalteten Objekte Prädikate definieren, die dann zur Auswahl einer Datenleitung (und damit auch eines Empfängers) benutzt werden.

6 Zusammenfassung

Dieser Beitrag hat Visualisierungen für den Entwicklungsprozeß experimenteller Programmsysteme diskutiert. Am Beispiel der Umgebung VIPEX wurde gezeigt, wie sich Visualisierungen beim Programmentwurf einsetzen lassen, wenn als Architektur ein Datenflußmodell zugrunde gelegt wird. Hervorzuhebende Merkmale sind die Strukturierung und Komposition von Programmen mithilfe von Kompositions- und Gruppenobjekten sowie Vererbungsgraphen für Objektparameter.

Literaturverzeichnis

Benzon 85: The Visual Mind and the MacIntosh, B. Benzon, *Byte* **10**, 1 (Jan. 1985), 113–130.

Brown 88: Exploring Algorithms Using Balsa-II, M.H. Brown, *IEEE Computer* **21**, 5 (May 1988), 14–36.

Chang 87: Visual Languages: A Tutorial and Survey, S.K. Chang, In: *Gorny & Tauber 87*, pp. 1–23.

Chang et al. 86: Visual Languages, S.K. Chang, T. Ichikawa, P.A. Ligomenides (eds.), Plenum Press, New York and London, 1986.

Faasch 87: Konzeption und Implementation einer objektorientierten Experimentierumgebung für die Bildfolgenauswertung in Ada, H. Faasch, *Dissertation*, Universität Hamburg, Fachbereich Informatik, Nov. 1987.

Faasch & Haarslev 88: Eine graphische ADA-Programmierumgebung für modulare Experimentalsysteme, H. Faasch, V. Haarslev, In: Proceedings, GI-Workshop *Sprachspezifische Programmierumgebungen*, G. Snelting (Hrsg.), TH Darmstadt, 6.–8. April 1988, pp. 196–205.

Fisher 88: An Overview of a Graphical Multilanguage Applications Environment, G. Fisher, *IEEE Transactions on Software Engineering* **14**, 6 (June 1988), 774–786.

Foley & McMath 86: Dynamic Process Visualization, J.D. Foley, C.F. McMath, *IEEE Computer Graphics and Applications* **6**, 3 (March 1986), 16–25.

Glinert & Tanimoto 84: PICT: An Interactive Graphical Programming Environment, E.P. Glinert, S.L. Tanimoto, *IEEE Computer* **17**, 11 (Nov. 1984), 7–25.

Gorny & Tauber 87: Visualization in Programming, P. Gorny, M.J. Tauber (eds.), 5th Interdisciplinary Workshop in Informatics and Psychology, Schärding, Austria, May 1986, Lecture Notes in Computer Science, Vol. 282, Springer Verlag, Berlin, 1987.

Haarslev 87a: Eine ergonomische Benutzerschnittstelle für den Anwendungsbereich der Bildfolgenauswertung, V. Haarslev, Software-Ergonomie '87, Berlin, 27.-29. Apr. 1987, Berichte des German Chapter of the ACM, W. Schönpflug, M. Wittstock (Hrsg.), Teubner-Verlag, Stuttgart, 1987, pp. 176–186.

Haarslev 87b: Human Factors in Computer Vision Systems: Design of an Interactive User Interface, V. Haarslev, In: *Second IFIP Conference on Human-Computer Interaction* – INTERACT '87, Stuttgart, F.R. Germany, 1-4 September, 1987, Proceedings, H.-J. Bullinger, B. Shackel (eds.), North-Holland, Amsterdam, 1987, pp. 1021–1026.

Haarslev & Möller 88a: Visualisierung und Animation in der experimentellen Bildauswertung, V. Haarslev, R. Möller, In: Proceedings, GI/ÖCG-Fachgespräch *Visualisierungstechniken und Algorithmen,* Wien, 26.–27. Sep. 1988, W. Barth (Hrsg.), Informatik-Fachberichte Nr. 182, Springer Verlag, Berlin, 1988, pp. 213–223.

Haarslev & Möller 88b: Eine graphische Umgebung zur experimentellen Bildverarbeitung, V. Haarslev, R. Möller, In: Proceedings, 10. DAGM-Symposium Zürich, 27.–29. Sep. 1988, H. Bunke, O. Kübler, P. Stucki (Hrsg.), Mustererkennung, Informatik-Fachberichte Nr. 180, Springer Verlag, Berlin, 1988, pp. 319–325.

Haarslev & Möller 88c: Visualization of Experimental Systems, V. Haarslev, R. Möller, In: Proceedings, *1988 IEEE Workshop on Visual Languages,* Pittsburgh/PA, Oct. 10–12, 1988, IEEE Computer Society Press, 1988, pp. 175–182.

Haarslev & Möller 89: VIPEX: Visual Programming of Experimental Systems, V. Haarslev, R. Möller, erscheint in: *Visual Languages and Visual Programming,* S.K. Chang (ed.), Plenum Press, New York and London, 1989.

Hirakawa et al. 86: HI-VISUAL: A Language Supporting Visual Interaction in Programming, M. Hirakawa, N. Monden, I. Yoshimoto, M. Tanaka, T. Ichikawa, In: *Chang et al. 86*, pp. 233–259.

Jessen & Valk 87: Rechensysteme: Grundlagen der Modellbildung, E. Jessen, R. Valk, Studienreihe Informatik, W. Brauer, G. Goos (eds.), Springer Verlag, Berlin, 1987.

Lieberman 84: Seeing what your programs are doing, H. Lieberman, *International Journal of Man-Machine Studies* **21**, 4 (Oct. 1984), 311–331.

McCormick et al. 87: Visualization in Scientific Computing, B.H. McCormick, T.A. DeFanti, M.D. Brown, *ACM Computer Graphics* **21**, 6 (Nov. 1987).

Möller 88: Gestaltung und Implementierung einer graphischen Dialogschnittstelle auf einer Lisp-Maschine nach dem Vorbild eines datenfluß- und objektorientierten Bildfolgenanalysesystems, R. Möller, *Studienarbeit*, Universität Hamburg, Fachbereich Informatik, Apr. 1988. Erschienen als Mitteilung Nr. FBI-HH-M-163/88.

Reisig 85: Petri Nets, W. Reisig, EATCS – Monographs on Theoretical Computer Science, Vol. 4, W. Brauer, G. Rozenberg, A. Salomaa (eds.), Springer Verlag, Berlin, 1985.

Reiss 87: Visual Languages and the GARDEN System, S.P. Reiss, In: *Gorny & Tauber 87*, pp. 178–198.

Shu 86: Visual Programming Languages: A Perspective and a Dimensional Analysis, N.C. Shu, In: *Chang et al. 86*, pp. 11–34.

Tauber 87: On visual interfaces and their conceptual analysis, M.J. Tauber, In: *Gorny & Tauber 87*, pp. 106–123.

Dr. Volker Haarslev
Ralf Möller
Universität Hamburg, Fachbereich Informatik
Bodenstedtstraße 16, D-2000 Hamburg 50
E-mail: haarslev@rz.informatik.uni-hamburg.dbp.de