Smalltalk
Stand & Perspektiven
aus der Sicht eines
Software-Ingenieurs

Hans-Jürgen Hoffmann

PU1R9/90

Oktober 1990

Fachgebiet Programmiersprachen und Übersetzer
Fachbereich Informatik / Department of Computer Science
Technische Hochschule Darmstadt / University at Darmstadt

Alexanderstr. 10, D-6100 Darmstadt, Germany
E-Mail, BITNET/EARN: XIP1HJHO@DDATHD21

Es handelt sich bei diesem Bericht um einen Beitrag zu *State of the Art* des Oldenbourg-Verlags, P. Schnupp (Hrsg.): Programmiersprachen jenseits von Algol und Cobol, Band 9. Diese Fassung ist an einigen Stellen ausführlicher als der Beitrag.

Smalltalk
Stand & Perspektiven aus der Sicht eines Software-Ingenieurs

Hans-Jürgen Hoffmann, Techn. Hochschule Darmstadt, FG Programmiersprachen und Übersetzer im FB Informatik, Darmstadt

---

Univ.-Prof. Dr.-Ing. Hans-Jürgen Hoffmann ist seit 1971 an der Techn. Hochschule Darmstadt im Fachbereich Informatik tätig und leitet dort das Fachgebiet Programmiersprachen und Übersetzer. Der Schwerpunkt seiner Interessen liegt in Fragen des Entwurfs und des Benutzens interaktionsfähiger Programme, bei der wissensgestützten Systemprogrammierung und beim objektorientierten Programmieransatz. Vor der Universitätstätigkeit war Prof. Hoffmann nach einem Studium der elektr. Nachrichtentechnik an der TH Stuttgart mehrere Jahre bei IBM im Böblinger Entwicklungslabor und im Forschungslabor in Zürich mit Systemprogrammieraufgaben beschäftigt.

---

1. Smalltalk, ein kurzer Blick in die Geschichte und das Umfeld

Als Wissenschaftler, der sich mit Smalltalk schon des längeren beschäftigt, bekommt man immer wieder den Eindruck, daß die Fachöffentlichkeit mit diesem Sprach- und Systemansatz nicht so ganz zurechtkommt.

Als Indiz seien zwei Aussagen zur Einordnung von Smalltalk angegeben, sollte doch eine saubere, begründbare Einordnung eines Artefakts der Ausgangspunkt eines wissenschaftlichen Verständnisses sein: In Verlagsprogrammen (u. ä.) wird Smalltalk immer wieder zu den KI-Sprachen geschlagen (z.B. geschehen mit dem Smalltalk-Buch des Verfassers /HOF87/) und taucht dann in Verbindung mit dem Begriff Sprachen der sog. fünften Generation auf. Oder: Kürzlich fand ich (in /CHA90/) als Einleitung zu einem Aufsatz als erstes den Satz "*Smalltalk ... ist gleichzeitig eine der bekanntesten und der unbekanntesten, eine der am meisten genutzten*

und am wenigsten angewandten Sprachen der 4. Generation". Vielleicht sollte man solche Einordnungsversuche nicht zu ernst nehmen, da ja die Generationenbildung bei den Programmiersprachen wohl mehr von Vertriebsstrategen als von Fachleuten vorgenommen wird. M.E. bildet Smalltalk eine eigenständige Kategorie in der Programmiersprachenlandschaft, mit verschiedenen Wurzeln sicherlich und mit vielen Einflußlinien zu anderen Sprachen. Smalltalk ist weder eine Sprache der sog. vierten noch der sog. fünften Generation. Smalltalk entzieht sich derartigen Einordnungsversuchen. Die knappste, aber doch sehr aussagekräftige Darstellung dieses Sachverhalts fand ich in einer Zeichnung in einem Aufsatz von Kreutzer /KRE90/, die ich (mit freundlicher Zustimmung des Vieweg-Verlags) hier als Abb. 1 übernehmen möchte. Und, provozierend, möchte ich sagen, Smalltalk ist so einzigartig, so sonderbar, aber in einem gewissen Sinne auch so leistungsfähig, daß es müßig ist, nach so einer Einordnung überhaupt zu fragen.

So ziemlich unbestritten ist, daß Smalltalk zu den sog. objektorientierten Sprachen und Systemen gehört. Nur, wer hat schon den Begriff *Objektorientiertheit* abschließend definiert. Man frage mich auch nicht danach, ich würde nur auf andere verweisen: Auf einen der Aufsätze von P. Wegner, z.B. /WEG86/, /WEG87/, /WEG89/, auch /WEG90/, auf das Heft 145 des Handbuchs der modernen Datenverarbeitung -- *Objektorientierte Systementwicklung* -- /HMD89/, zur Abgrenzung z.B. gegenüber Ada auf /SCHa88/, auf das Heft 3/90 der Zeitschrift Wirtschaftsinformatik -- *Objektorientierte Anwendungssysteme und Systemsoftware für die 90er Jahre* -- /WIRa90/ oder auf das Januar/Februar 1990-Heft von *iX* -- *OOPS unter UNIX, objektorientierte Programmiersysteme* -- /iX90/; ganz neu schließlich auf das Themenheft *Object-oriented Design* der Comm. ACM /ACM90/. Und wer es dann noch nicht weiß, der mußte die Lebenserfahrung machen, daß man Schlagworte der Informatik nicht hinterfragen soll (zumindest nicht, wenn man eine kurze Antwort erwartet); symptoma-

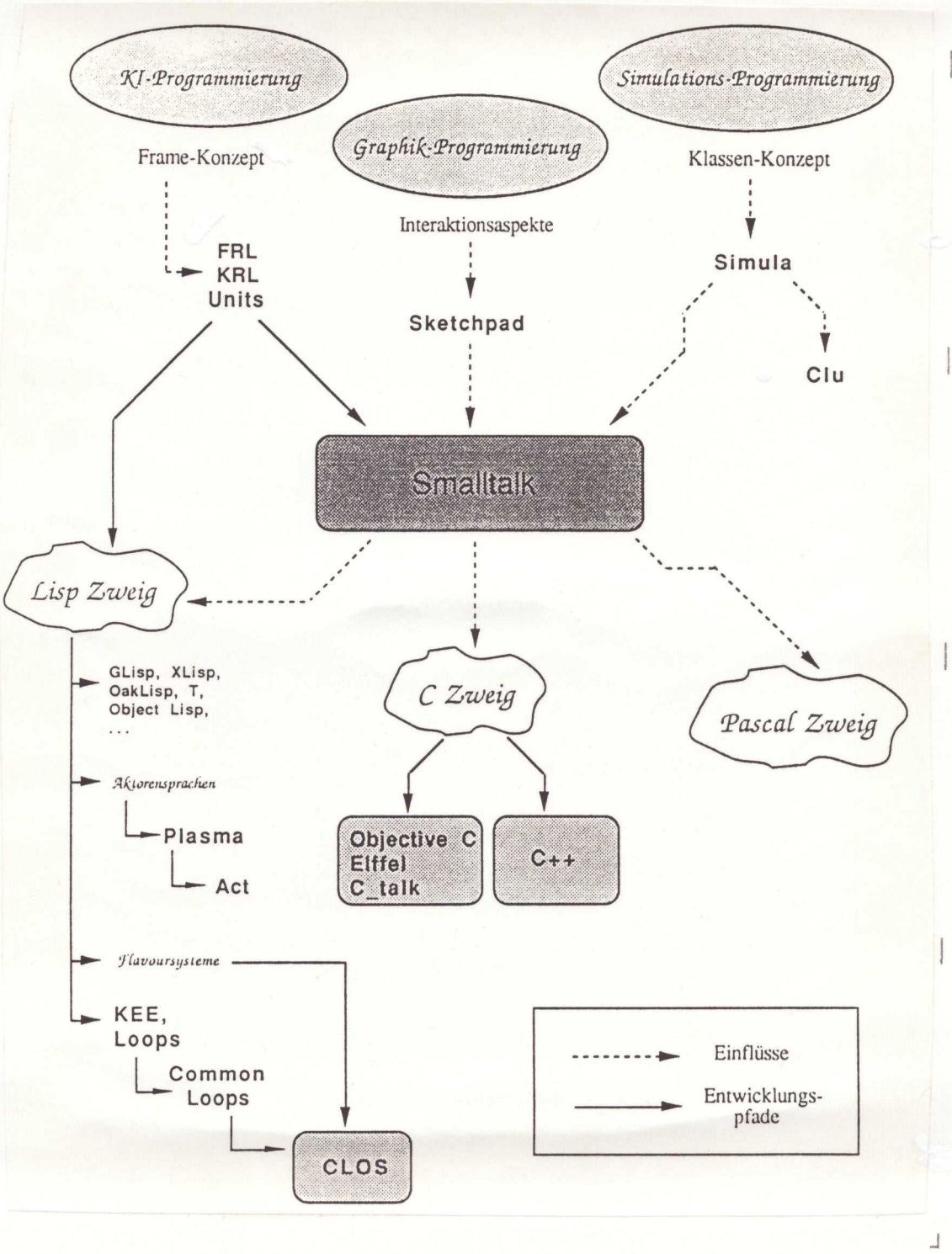


Abb. 1 Historische Wurzeln einiger objektorientierter Programmiersprachen und -systeme (aus /KRE90/)

tisch dafür ist der prägnante Satz "*Object-oriented remains a term which is interpreted differently by different people*" im Leitaufaufsatz /KOR90/ des gerade genannten Hefts der Comm. ACM.

Wedekind, der tiefschürfend denkende Informatikprofessor und Kenner der Logik, der Philosophie und der Antike weist im übrigen in /WED90/ darauf hin, daß bei den alten Griechen *"Objektorientierung"* nicht von *"Problemorientierung"* zu unterscheiden gewesen wäre. Auch in anderer Hinsicht ist der Aufsatz sehr lesenswert. Nur noch zwei Kostproben: *"Vererbung"* ist gleichzusetzen mit *"Gemeinsamkeit"*; Bezug auf die, wie er sagt, Konstitutionsrelation *"Art-Gattung"* (*"Oberklasse-Unterklasse"*) und die *"benutzt"*-Relation (*"Aussenden einer Botschaft"*) wäre angemessener als das Einführen neuer Begriffe.

Smalltalk ist etwas Greifbares (wir werden weiter unten davon hören), man kann diese Sprache und das untrennbar dazu gehörige Programmier- und Benutzungssystem bei seiner täglichen Arbeit einsetzen. Smalltalk entstand in den 70-er Jahren im Xerox Palo Alto Research Center (Xerox PARC). Den Anstoß zu dieser Entwicklungsarbeit gab Alan Kay, der Leiter der dortigen Learning Research Group; er realisierte damit eine Vision, die er in seiner Doktorarbeit *The reactive engine* /KAY69/ entwarf (auch diejenigen Rechnerbenutzer, die sich nicht eines der Smalltalk-Systeme bedienen, ziehen ihren Nutzen aus seiner Arbeit, entstanden daraus doch auch wesentliche Architekturkonzepte hochinteraktiver persönlicher Rechner und der Fenstersysteme, die wir alle heute täglich einsetzen). Weitere wichtige Beiträge zur Entwicklung von Smalltalk zur Produktreife kamen von Adele Goldberg (hier sei nur an die drei Smalltalk-Bücher /GOL83/, /GOL84/, /KRA83/ erinnert, die sie -- teilweise mit Koautoren -- schrieb, und an ihre derzeitige Arbeit als Präsidentin der Firma, die jetzt Smalltalk-80 vermarktet -- ParcPlace Systems in Mountain View, Kalifornien); Frau Goldberg war während des entscheidenden Entwicklungszeitraums von Smalltalk,

der zum Produkt *Smalltalk-80* führte, Leiterin des System Concepts Laboratory von Xerox PARC. Die überzeugenden Eigenschaften von Smalltalk-80 haben auch andere Firmen veranlaßt, eigene Entwicklungen voranzutreiben und auf den Markt zu bringen (ich gehe auf einige davon im 3. Kapitel ein).

2. Ein sehr, sehr kurzes Kennenlernen von Smalltalk

Nun, was macht das besondere an Smalltalk aus, was ist die "Sprachidee" von Smalltalk, wie ist die Vision von Kay zu einer "reagierenden Maschinerie" umgesetzt?

2.1. Die *"Sprachidee"* von Smalltalk

Wie würden Sie mit einer reagierenden Maschinerie umgehen wollen? Wenn Sie einem bestimmten Teil dieser Maschinerie -- wir wollen ein solches Teil *Objekt* nennen -- einen Anstoß oder Auftrag geben könnten -- oder, wie wir sagen wollen, eine *Botschaft* zusenden könnten --, würden Sie erwarten, daß das Objekt seinen inneren Zustand (die *Objekt-*, *Exemplar-* oder *Ausprägungsvariablen* -- ein einheitlicher Begriff hat sich da im Deutschen noch nicht durchgesetzt, den Amerikanismus *Instanzvariablen* möchte ich nicht verwenden bzw. verwendet wissen --) entsprechend verändert und eine geeignete Rückmeldung, wieder in Form eines Objekts, an Sie zurück gibt. Natürlich müßten Sie nun einige Notationen erlernen, wie Botschaften zu formulieren sind, doch das möchte ich Ihnen ersparen, nicht weil es schwierig ist, sondern weil ich in diesem Aufsatz nicht den Platz dazu habe; einige Beipiele, wie solche Botschaften zu formulieren sind, sollen genügen:

$a asUpperCase .
Ich erwarte, daß ich ein großes *A* zurückerhalte. Sie auch? Das vorangestellte Dollarzeichen kennzeichnet Alphabetzeichen.

-5 + 17 .
Ich lese dies als: Objekt *minus fünf*, schaffe als Ergebnis ein Objekt, das Du selbst, erhöht um *siebzehn*, bist, und gebe mir das Ergebnisobjekt zurück -- natürlich erwarte ich dabei das Objekt *zwölf*! --.

17388 gcd: 1251 .
Hier wird deutlicher, daß dem *Adressaten* der Botschaft, d.h. der Zahl *17388*, in der Botschaft ein durch einen *Selektor*, wie man dazu sagt, identifizierter Anstoß, nämlich *gcd -- greatest common divisor --*, zusammen mit einem *als Parameter dienenden Objekt*, der Zahl *1251*, zugesandt wird; die Antwort ist, wie man nicht anders erwartet, das Objekt *neun*.

5 between: 11 and: 111 .
Hier handelt es sich um den Selektor *between: and:*, dem man zwei Objekte als Parameter mitgeben muß; die Antwort ist das Objekt *false*.

myCalendar insert: date for: reason .
Stellen Sie sich nun vor, daß es sich beim Adressaten nicht um ein so übliches Teil eines Rechners wie eine Zahl -- die sich irgendwo im Speicher Ihres Rechners befinden kann --, sondern um ein komplex zusammengesetztes Teil handelt, das dazuhin noch einen Bezug zu einer Gegebenheit Ihrer Umwelt aufweist, die Sie auf Ihrem Rechner modellieren wollen, z.B. Ihren Taschenkalender. Und stellen Sie sich vor -- ich habe dies bei den vorhergehenden Beispielen unterschlagen --, daß das Adressatenobjekt auf der Bildschirmfläche Ihres Rechners visualisiert ist, wie eben so, um beim gewählten Beispiel zu bleiben, ein Kalender aussieht, ein Rechteck mit eingeschlossenen Teilrechtecken und einigen Beschriftungen. Und stellen Sie sich weiter vor, daß Sie durch Manipulationen der Maus an Ihrem Rechner ein Teilrechteck ausgewählt und dann ein Aufblendmenü gezeigt bekommen haben, das u.a. einen Menüpunkt *neuer Termin* enthielt. Ein eingetippter Text würde den Zeitpunkt und Grund der neuen Verabredung festgehalten haben. *Direktmanipulativ*, wie man sagt, an der *visualisierten Quasirealität objektorientiert*, haben Sie eine Botschaft an Ihr Kalenderobjekt aufgebaut und weitergegeben; die Antwort, wieder visualisiert, wäre Ihr Kalender mit den ergänzten Angaben.

Um so etwas zu erreichen, müßten Sie -- ein Smalltalk-System gibt da viele Hilfestellungen -- die Eigenschaften aller Objekte, mit denen Sie umgehen wollen, in einer geeigneten Weise beschreiben. Dazu gehören würde der *Datenaspekt* der Objekte, d.h. die Exemplarvariablen, die ihren Zustand festhalten; und dazu gehören würde der *Handlungsaspekt*, wie ich es nennen möchte, der aussagt, wie Ihre Objekte auf zu-

gesandte Botschaften reagieren sollen. Die Reaktion eines Objekts auf eine ihm zugesandte Botschaft, d.h. die Vorgehensweise bei ihrer Bearbeitung, hätte man als *Methode* (wie das bei Smalltalk heißt) festzulegen. Beim Bearbeiten einer Botschaft könnten, und das ist der wesentliche Mechanismus, weitere Botschaften gebildet, an andere Objekte (oder auch an das gleiche) gesandt und die jeweiligen Ergebnisobjekte dann passend eingebaut werden.

Wollten Sie das tatsächlich für jedes Ihrer Objekte mit all diesen Eigenschaften einzeln und möglicherweise bei Gleichartigkeit unter den Objekten wiederholt beschreiben? Ich vermute, nicht. Hier greift der Gedanke (im objektorientierten Ansatz und so selbstverständlich auch bei Smalltalk), nicht einzelne Objekte sondern *Klassen* von Objekten zu beschreiben; auch unter Klassen Beziehungen herzustellen -- man sagt dazu *Ererbungsbeziehungen* (ich halte diese Sprechweise sachlich und im Deutschen für zutreffender als der Bezug auf *Vererbung*), die eine *Klassenhierarchie* bilden --, wenn die Objekte mehrerer Klassen einige Gemeinsamkeiten aufweisen, die man gemeinsam durch die Zugehörigkeit zu einer Oberklasse erfaßt. Erwähnen muß ich es, wenn ich es auch nicht im Detail ausführen kann, *Klassen bilden selbst Objekte*, auch ihnen kann man Botschaften zuschicken (am einleuchtendsten ist diese Aussage vielleicht, wenn ich sage, daß ich den im obigen Beispiel als *myCalendar* angesprochenen Adressaten dadurch erhalten habe, daß ich einer Klasse *Calendar* eine Botschaft mit Selektor *new* zugeschickt habe -- ich habe das Objekt mit Etikett *myCalendar* aus der Klasse *Calendar "ausgeprägt"* --).

Den Begriff *Methode* und die Vorgehensweise bei der Programmierung einer Methode muß ich noch etwas detaillierter erläutern: Der Kopf einer Methodenbeschreibung legt das *Botschaftenmuster* (in Form der auftretenden Selektoren) der Botschaft fest, für deren Bearbeitung die Methode vorgesehen ist. Die Gesamtheit der Botschaftenmuster, die ein Objekt *"versteht"*, nennt man sein *Protokoll* (ich habe hier ererbte

Methoden nicht berücksichtigt). Der Rumpf einer Methodenbeschreibung ist eine *Ablaufbeschreibung* für das Aussenden von weiteren Botschaften und das Einbauen der Ergebnisobjekte; hier ist die Vorgehensweise der traditionellen, algorithmischen Programmierung am ähnlichsten.

Nun sollte verständlich sein, was in einer Klassenbeschreibung zu stehen hat (ich habe auch hier vereinfacht):

- Die Bezeichnung der Klasse.
- Wie die Klasse in der Klassenhierarchie einzuordnen ist.
- Was der Datenaspekt der aus der Klasse ausprägbaren Objekte ist, d.h. was die Exemplarvariablen jedes ihrer Objekte sind.
- Was der Handlungsaspekt der aus der Klasse ausprägbaren Objekte ist, d.h. was die Methoden im Protokoll jedes ihrer Objekte sind.

Ein Beispiel einer Klassenbeschreibung soll diese Betrachtung abschließen, siehe Abb. 2. Verwendet habe ich dabei die Klasse *Integer* aus Smalltalk/V; eine ihrer Methoden, nämlich die mit dem Botschaftenmuster ***gcd:*** (siehe oben eine damit formulierte Botschaft) habe ich aus dem Quellcode der Implementierung, die mir vorliegt, entnommen. Das ist für Smalltalk-Implementierungen üblich; man kann im Quellcode der gesamten Klassenhierarchie herumstöbern und (z.B. zum Wiederverwenden in einer Methode, die man gerade programmiert) sich etwas Interessantes herausholen.

2.2. Die *"reagierende Maschinerie"* von Smalltalk

Bis jetzt stand die "Sprachidee" im Vordergrund meiner kurzen Vorstellung von Smalltalk. Die "reagierende Maschinerie", das zweite Charakteristikum, abgeleitet aus der Vision von Alan Kay, ist einmal zu verbinden mit dem durchgehend möglichen Visualisieren aller Objekte, sei es systemtechnisch mit einem *Inspektor* oder sei es anwendungsbezogen mit eigens dafür programmierten *graphisch/bildlichen Mitteln* (oben sagte ich dazu *"visualisierte Quasirealität"*), und dem direktmanipulativen Umgehen mit ihnen an der Bildschirmfläche; wie auch andererseits mit der Transparenz des gesamten

```
Number subclass: #Integer          <===== Kopf der Klassenbe-
  instanceVariableNames: ''               schreibung
  classVariableNames: ''
  poolDictionaries: '' !

!Integer class methods ! !         <===== Es gibt keine Klassen-
                                           methoden

!Integer methods !                 <===== Die Methoden der Objek-
                                           te der Klasse Integer
:
:                                          herausgegriffen
                                   <===== ist die Methode gcd:
gcd: anInteger                     <===== Botschaftenmuster
    "Answer the greatest           <==== Kommentar
     common divisor between
     the receiver and
     anInteger."
    | u v r |                      <===== lokale Variable des
    u := self abs.                         Methodencode
    v := anInteger abs.            <===== Methodencode, man er-
    [v = 0]                                kenne die einzelnen
        whileFalse: [                      Botschaften
            r := u \\ v.
            u := v.                > <===== Zwischen [ und ] eine
            v := r ].                       Programmschleife; sie
                                            wird solange ausge-
    ^u !                 <=== das           führt, als die Ausfüh-
                             Ergeb-        führung von [v = 0] den
:                            nisob-        Wert false liefert
:                            jekt
```

Abb. 2 Ausschnitt aus der Beschreibung der Klasse *Integer* und Quellcode ihrer Methode mit Botschaftenmuster *gcd:* (Smalltalk/V).

Systemzustands, integriert für Programm-(d.h. Klassen-)entwurf wie für Programmbenutzung, die das *"Herumstöbern"* (engl. *browsing*) im Klassensystem erlauben, den Programmierer/Benutzer sogar fortwährend dazu ermuntern. Verpackt ist die "reagierende Maschinerie", die werkzeugartige Programmier- und Benutzungsfunktionalität (und anderes, was ich hier nicht erwähnen konnte) unter dem einheitlichen Denkmodell, daß man mit visualisierten, direkt manipulierbaren Objekten der Klassen *Inspector* bzw. *Browser* (oder wie sie im Einzelfall einer Implementierung genau heißen mögen) umgeht, denen Botschaften in der gleichen Weise (z.B. durch Auswahl

in einem Aufblendmenü) zugeschickt werden können, wie wir das schon bei den "Anwendungsobjekten" gesehen haben; die auch -- wenn sich der Programmierer/Benutzer daran wagt -- in der gleichen Weise (um-)programmiert werden können, wie alle anderen (Anwendungs-)objekte bzw. -objektklassen. Der visualisierte Kalender des obigen Beispiels und ein Inspektor, der dazu dient, einen Kurvenverlauf anzusehen, unterscheiden sich in der direktmanipulativen Handhabbarkeit nicht. Man kann die "reagierende Maschinerie" von Smalltalk in allen ihren Teilen in gleicher Weise bedienen und steuern.

Die Eigenschaften der "reagierenden Maschinerie" kann man am besten durch ein Arbeiten mit ihr, also dem Arbeiten mit einem Smalltalk-System, erfahren; ein Notbehelf, da nur ein ruhender Eindruck, ist ein "Schnappschuß" einer visualisierten Benutzungssituation. Ich möchte dies an zwei Beispielen der beiden wichtigsten Implementierungen zeigen. Zunächst in Abb. 3 ein Schnappschuß von der Benutzung von Smalltalk-80, entnommen (mit freundlicher Zustimmung des Carl Hanser Verlags) aus /HOF87/ und enthalten in einer längeren Serie solcher Schnappschüsse einer Bibliotheksanwendung /BUDb87/, deren Programmierung dort im Aufsatz /BUDc87/ dargelegt ist. Das zweite Beispiel zeigt mehr die Programmentwicklungssituation, wie sie sich anhand zweier charakteristischer Fälle (Stöbern in der Klassenhierarchie und Ansehen einer Methode -- hier für die Berechnung des absoluten Betrags einer Zahl -- bzw. Inspizieren und eventl. Ändern eines Objekts -- hier einer Zeichenreihe --) in Abb. 4a bzw. Abb. 4b herausstellen läßt (mit freundlicher Zustimmung des Vogel Buchverlags aus /KAS89/ entnommen).

2.3. Smalltalk als objektorientierte Sprache

Smalltalk ist eine objektorientierte Sprache. Kürzlich las ich von Smalltalk als dem "Paradebeispiel" /CLA90/ dafür. Die sehr rein gehaltene Vorstellung des Botschaftenaustauschs, um einen Effekt zu erreichen, wie auch die uneinge-

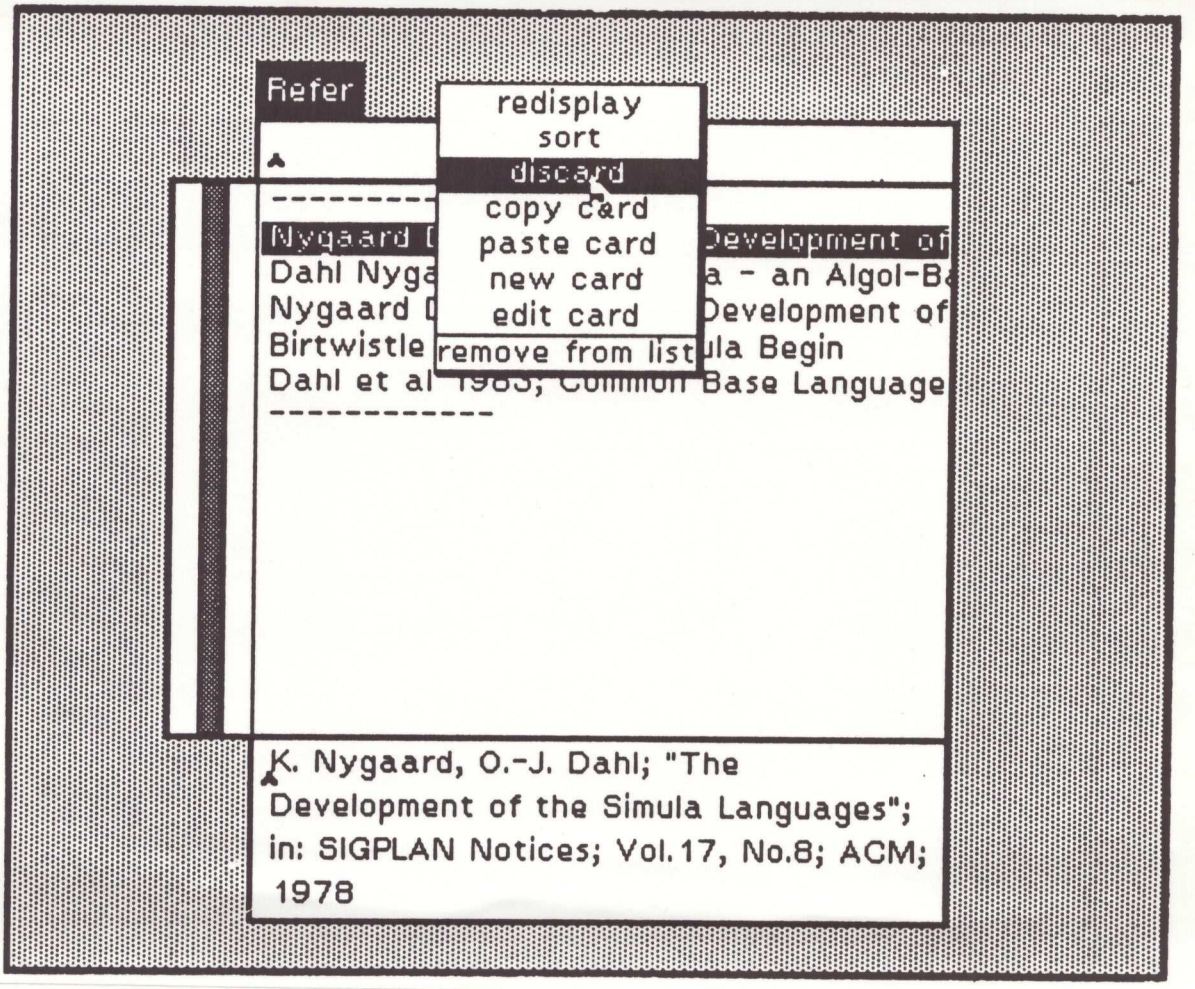


Abb. 3 Arbeitsfenster der in /BUDb87/ entwickelten Bibliotheksanwendung *Bib* (entnommen aus /HOF87/).

schränkte Gleichartigkeit im Aufbau (Datenaspekt) und im Verhalten (Handlungsaspekt) der programmierer-/benutzerdefinierten Anwendungs- und der Systemobjekte (Anwendungs- und Systemobjektklassen), das direktmanipulative, an der visualisierten Quasirealität objektorientierte Umgehen mit den Objekten an der Bildschirmfläche zeichnen Smalltalk gegenüber den anderen objektorientierten Programmiersprachen aus, machen es einzigartig und machen es, in diesem Sinne, bei modernen, interaktiven Anwendungen besonders leistungsfähig (siehe auch Abschn. 4.3.).

Muß man hier auf die anderen wichtigen, modernen objekt-

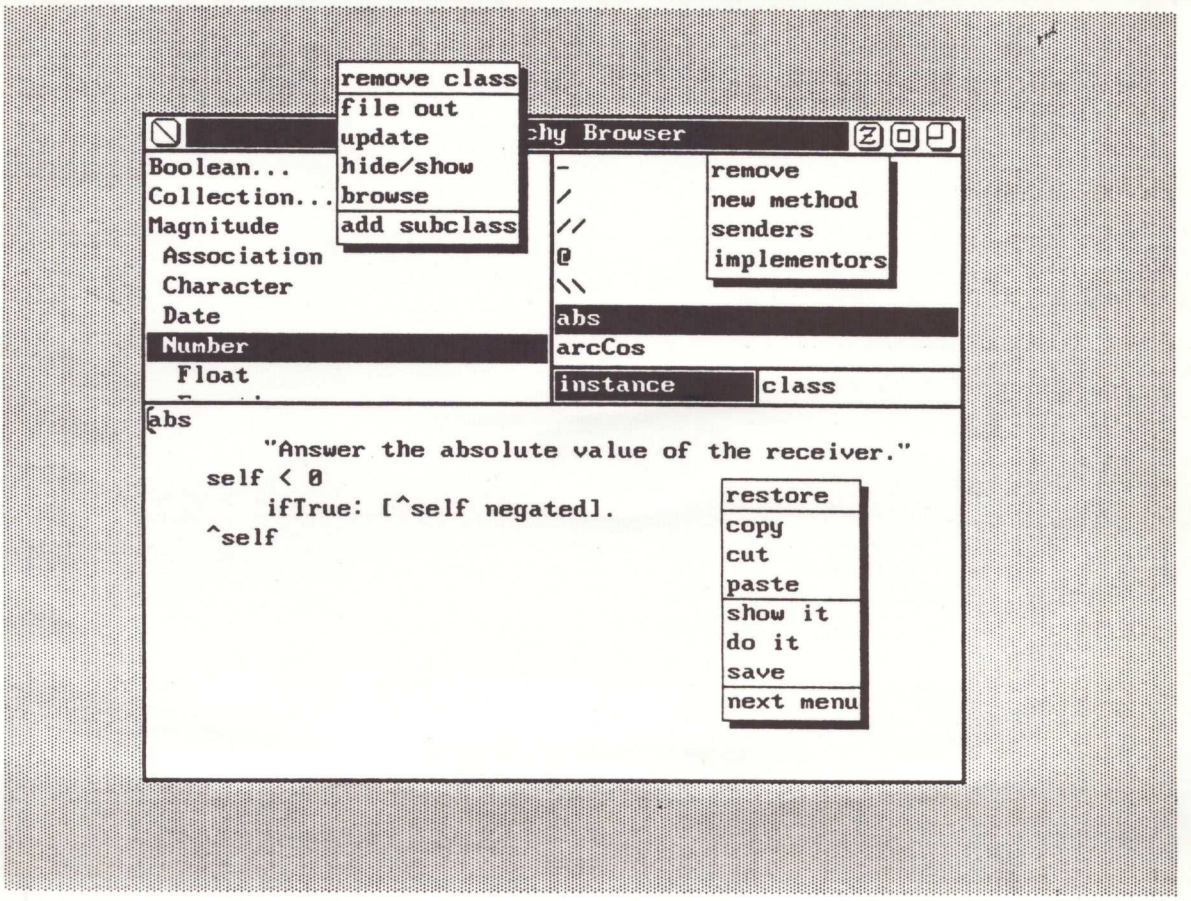


Abb. 4a *ClassHierarchyBrowser*, Standardfenster zur Programmentwicklung (entnommen aus /KAS89/)

orientierten Sprachen eingehen, auf C++ (siehe /STR86/), auf Eiffel (siehe /MEYa88/), auf CLOS (siehe /KEE88/)? Wer unbedingt einen Vergleich sehen möchte (allerdings ohne CLOS), lese die Aufsätze von Blaschek, Pomberger und Stritzinger /BLAa89/ und /BLAb89/. In Weiterentwicklung einer Gegenüberstellung aus /BLAb89/ möchte ich weitere Kriterien heranziehen (sie werden, zumindest zum Teil, weiter unten noch angesprochen) und auch CLOS (und eine neuere Version von C++) mit einschließen (siehe Abb. 5). Ich meine, daß da Smalltalk nicht so schlecht wegkommt, wenn man gewisse Abstriche bei den Ererbungsstrategien (siehe Abschn. 4.7.) und der Effizienz (siehe Abschn. 4.3.) toleriert; was software-

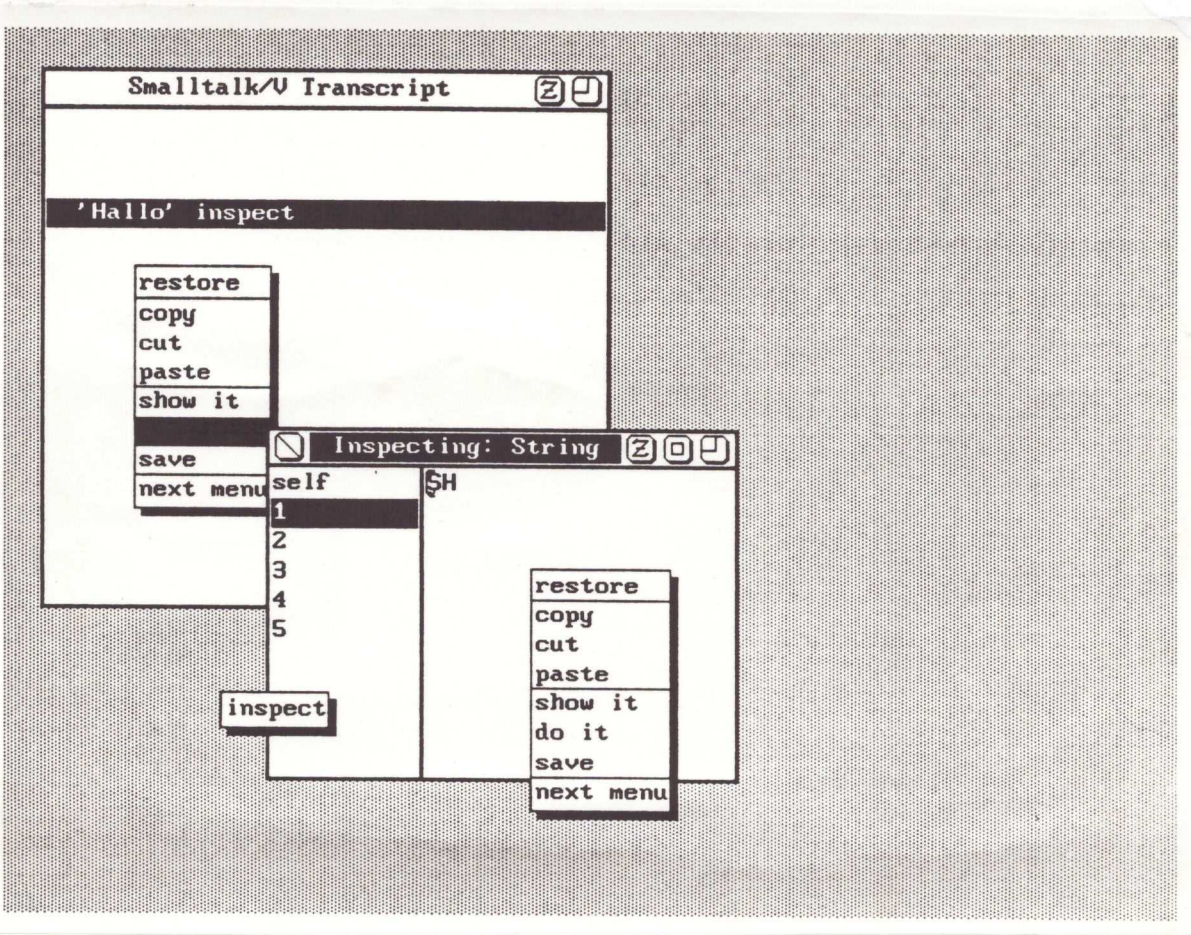


Abb. 4b *Inspector*, Standardfenster zum Inspizieren und Verändern von Objekten (entnommen aus /KAS89/)

technische Robustheit (siehe Abschn. 4.2.), Vorkehrungen zum Wiederverwenden von Programmstücken (hierzu im Zusammenhang mit Smalltalk ein Verweis auf /SCHb90/) und die Arbeitsmöglichkeiten in einem Team anbelangt, da ist m.E. Eiffel sowieso eine Klasse für sich selbst (siehe /MEY88/) -- aber das wäre das Thema eines anderen Aufsatzes! --.

2.4. Das besondere Leistungskriterium von Smalltalk

Das Zusammenkommen der vorgestellten Sprachidee, die besonders klare Modellierungskonzepte für eine Anwendungsaufgabe bei der Programmierung bietet, und der Eigenschaften der "reagierenden Maschinerie", dabei besonders die uneinge-

Sprache / Kriterium	C++	CLOS	Eiffel	Small-talk
mehrf. Ererbg	ja	ja	ja	nein
eingeschränk. Ererbung	ja	ja	ja	möglich
geschützte Operatnen	ja	nein	nein	nein
abstr. Klassn	möglich	ja	ja	ja
Speicherbe-reinigung	nein	ja	ja	ja
Zuverlässigkt Robustht	gering	gering	sehr hoch	mittel-mäßig
Laufzeiteffi-ziens	hoch	mittel-mäßig	einiger-maßen hoch	gering
Einheitlichkt d. Daten.	gering	gering	mittel-mäßig	sehr hoch
Dokumentatns-wert	gering	gering	hoch	mittel-mäßig
Sprachkomple-xität	hoch	hoch	mittel-mäßig	sehr gering
Persistenz	nein	nein	möglich	ja
Arbeitsumge-bung	nicht in-tegriert	ja	in Vorbe-reitung	ja, musterhaft
Programmsamm-lungen	einge-schränkt	ja	ja	ja, musterhaft
Wiederver-wendbark.	fraglich	möglich	ja, musterhaft	ja,

Abb. 5 Ein Vergleich unter modernen, objektorientierten Programmiersprachen

schränkte Integration von Anwendungs- und Systemobjekten (bzw. -objektklassen) und die Art des objektorientierten, direktmanipulativen Umgangs mit ihnen auf der Bildschirmfläche, machen m.E. das besondere Leistungsverhalten von Smalltalk aus. Wenn es mit einer einigermaßen hohen Implementierungsleistung realisiert ist, kommt man zu einem herausragenden interaktiven Programmier- und Benutzungssystem.

Smalltalk empfehle ich (wegen der doch bei einigen Rechnermodellen nicht gerade überwältigenden Implementierungsleistung) uneingeschränkt für den motivierten Anwender, dem es auf klare Modellbildung seiner Anwendung in den software-

technischen Gebilden eines Programmsystems und auf besondere Flexibilität in der Weiterentwicklung ankommt, der einen interaktionsfähigen Prototyp geradlinig zum Vollausbau führen will, ohne dazwischen Modell und softwaretechnischen Rahmen zu wechseln.

Wer sich mit Smalltalk etwas gründlicher auseinandersetzen möchte, findet Lesenswertes in /SCHb88/.

3. Der heutige Stand

Man kann heutzutage mit Smalltalk arbeiten, selbst mit verhältnismäßig bescheiden ausgerüstetem Rechner. Ein großer Arbeitsplatzrechner (von DEC, von SUN oder von HP) mit hochauflösendem Bildschirm und Smalltalk-80 von ParcPlace Systems ist selbstverständlich von Vorteil. Es muß aber nicht sein. Mit der Implementierung von G. Heeg (Dortmund) arbeitet schon ein Atari Mega ST beeindruckend. Die Implementierungen Smalltalk/V und Smalltalk/V 286 von Digitalk auf IBM PC DOS-Rechnern (oder kompatiblen), Smalltalk/V PM für den Presentation Manager auf IBM OS/2-Rechnern bzw. Smalltalk/V Mac auf dem Apple Macintosh zeigen auch, was sich da machen läßt, objektorientiertes Programmieren und Arbeiten, klein aber fein. Schließlich ist zum Hineinschnuppern das *Little Smalltalk* von T. Budd, eine Implementierung in C unter UNIX, zu nennen.

Ich möchte im folgenden nähere Angaben zu einigen dieser Implementierungen machen (eine ausführlichere Darstellung, Stand Anfang 1989, findet sich in /MIT89/). In einigen Fällen gibt es interessante Zusätze oder Weiterentwicklungen, auf die ich auch hinweisen möchte. Ein kurzer Abschnitt mit einer Anleitung, wie man sich in das Programmieren und Arbeiten mit Smalltalk einarbeiten kann, soll dieses Kapitel beschließen.

3.1. *Smalltalk-80*, die Implementierung von XEROX PARC bzw. ParcPlace Systems

Das Smalltalk-80-System ist 1981 vorgestellt /BYT81/ und verfügbar gemacht worden. Seitdem gab es eine Reihe von weiteren Implementierungen in Lizenz von XEROX PARC bzw. später dann von ParcPlace Systems. Die Systeme sind heutzutage völlig ausgereift und bieten, neben dem von mir oben herausgestellten Kriterium der besonderen Leistungsfähigkeit für interaktive Anwendungen, auch eine ausreichende Implementierungsleistungsfähigkeit (die selbstverständlich durch die inzwischen gebotene hohe Prozessorleistungsfähigkeit unterstützt ist). Die Implementierungskonzepte, die Ingalls für den Vorläufer Smalltalk-76 ausgearbeitet hatte /ING76/, haben so glänzend die Bewährungsprobe bestanden (und Implementierungen vieler anderer objektorientierter Sprachen und Systeme beeinflußt oder gar erst ermöglicht). Wer sich für die geschichtliche Entwicklung und Implementierungsdetails (des ursprünglichen Smalltalk-80-Systems) interessiert sei auf /BAU87/, /SCHa90/ und besonders auf /KRA83/ verwiesen (über neuere Implementierungen gibt es nicht so detaillierte Angaben, einiges ist in den /OOPxx/-, /ECOxx/- bzw. /TOOxx/-Bänden zu finden). Vergessen sollte man nicht, daß Smalltalk-80 (und Nachkommen bzw. die dann weiter unten erwähnten, anderen Smalltalk-Nachbauten) weitgehend in Smalltalk programmiert sind und so dem Stöbern -- und wer es nicht sein lassen will, auch dem Ändern -- offenstehen; die Systemprimitive sind meist ordentlich dokumentiert.

Ein Smalltalk-80-System (auch die Nachkommen und die meisten Nachbauten) besteht im wesentlichen aus zwei Teilen: der *virtuellen Welt* (*virtual image*) und der *virtuellen Maschine* (*virtual machine*). Letztere realisiert die Abbildung auf einen Rechner (bzw. ein Betriebssystem auf einem Rechner), die Abbildung ist (regelmäßig) interpretierend (ich sage dazu noch etwas in Abschn. 4.3.), die Speicherverwaltung und die Systemprimitive. Die virtuelle Welt umfaßt insbesondere das gesamte Klassensystem, bei Smalltalk-80 von

der Größenordnung 250 Klassen mit über 4500 Methoden.

Nun, was gibt es heute als Smalltalk-80-Systeme, auf welchen Rechnern? Systeme für die UNIX-Umgebung mit 680x0-Prozessoren bei (alphabetisch) Apollo, Cadmus, Hewlett-Packard, Sun; für Atari und für MacIntosh; und andere. Die Lizenzpreise liegen bei einigen tausend Mark. Diejenigen, die so eine Implementierung kennen, sind mit dieser Aufzählung zufrieden; diejenigen, die sich für eine Neuinstallation interessieren, mögen auch noch den nächsten Abschnitt 3.2 beachten. Der Vollständigkeit halber möchte ich das Smalltalk-80-System von Tektronix nicht unerwähnt lassen (das allerdings wohl nicht mehr vertrieben wird), eine Lizenzentwicklung.

Smalltalk-80 **ist eine Handelsmarke (*trade mark*) von ParcPlace Systems, Inc., Mountain View, CA, USA.**

3.2. *Objectworks* von ParcPlace Systems

Smalltalk-80 wird heute als Bestandteil von *Objectworks* angeboten, *Objectworks for Smalltalk-80* (es gibt auch ein *Objectworks for C++*). Mit der anderen Bezeichnung wollte man von dem Sprachkonzept von Smalltalk weg wohl die Betonung auf das Umgebungskonzept, d.h. das Smalltalk-80-System, legen; hier liegt ja, wie ich schon mehrfach ausführte, die besondere Leistungsfähigkeit von Smalltalk für die interaktive Benutzung wesentlich mitbegründet. Selbstverständlich ist beim Übergang dann auch noch einiges hinzugekommen (auf das ich aber hier aus Platzgründen nicht näher eingehen möchte); nicht weglassen kann man m.E. den Hinweis auf die vorgenommene Integration in einige Fenstersysteme (das Smalltalk-Fenstersystem als Untersystem!), selbst bei Vernetzung von Rechnern; und den Hinweis auf einen Zusatz, die *Schere* -- wie ich ihn nennen möchte, um das engl. Wort *stripper* zu vermeiden --, mit ihr kann man nach Fertigstellen eines Anwendungsprogramms alles heraustrennen, was zum Programmablauf nicht benötigt wird, Speicherplatz wird gespart, es kann ein zurechtgelegtes, allerdings nicht mehr

veränderbares Anwendungsprogramm ausgeliefert werden.

Dieses Produkt, *Objectworks for Smalltalk-80*, gibt es (fast) für jeden Rechner bzw. jedes Betriebssystem (die *Plattform*, wie man das jetzt nennt), der bzw. das heutzutage aktuell ist: 80386-Prozessoren und MS-DOS (etwas, was bei Smalltalk-80 lange gefehlt hat!); Atari-Rechner; Apple Macintosh; UNIX-Maschinen wie DECstation 3100, HP-9000 Serie 300, PCS Cadmus und Cadmus RCU, Sun-1, Sun-4; Sun386i, Apollo 3000/4000/DNxxx. /RÜG90/ ist ein Testbericht.

Objectworks **ist eine Handelsmarke (trade mark) von ParcPlace Systems, Inc.**

3.3. Implementierungen von Smalltalk-80 von G. Heeg

An diesem Entwicklungsstand von Smalltalk-80 und von Objectworks war Georg Heeg, der rührige deutsche Smalltalk-Promoter in Dortmund, nicht unbeteiligt. Von ihm stammen (meines Wissens) die Cadmus- und die Atari-Implementierung (und einige weitere. hier nicht im Detail ansprechbare Zusätze), was in diesem Aufsatz nicht unerwähnt bleiben soll.

3.4. *Smalltalk/V* von Digitalk

Wie oben gezeigt wurde, verlief die Smalltalk-80-Entwicklung sozusagen am oberen Ende der persönlichen Rechner. Die Firma Digitalk nahm sich mit der Implementierungsfamilie Smalltalk/V früh (etwa ab 1985, *"methods"*, /WEB85/) der auf dem Markt verbreiteten Rechnern mit 8086/8088- und 80286-Prozessoren an, die unter MS-DOS bzw. PC-DOS betrieben werden. Es wurden (und mußten wohl) Abstriche bei der Implementierungsleistung in Kauf genommen (werden), die bei den weniger leistungsfähigen Prozessoren und Betriebssystemen unvermeidlich waren. Trotzdem meine ich, daß diese Implementierungen hinsichtlich der interaktiven Benutzungsleistung bei einigermaßen ausgebauten Rechnern schon Beachtung verdienen. Es gibt Unverträglichkeiten zu Smalltalk-80, auch geringe Unverträglichkeiten innerhalb der Smalltalk/V-Familie, die aber über das sonst von Software gewohnte Maß an

Unverträglichkeit nicht hinausgehen, so daß man von echten Smalltalk-Implementierungen sprechen kann. Und vom Preis her (einige 100 Mark) gesehen ist Smalltalk/V auch interessant (/CRA89/ mit Angaben zu den Verhältnissen in den USA für Implementierungen für den Apple Macintosh-Rechner).

Smalltalk/V wird angeboten als Smalltalk/V für die Plattform (IBM PC, PC/XT, PC/AT, PS und kompatible; PC-DOS bzw. MS-DOS), als Smalltalk/V286 entsprechend für Rechner mit (mindestens) 80286-Prozessor, als Smalltalk/V PM bei diesen Rechnern mit dem OS/2 Presentation Manager, als Smalltalk/V Window für das MS Windows 3.0-System und als Smalltalk/V Mac für Apple Macintosh II, SE und Plus. /SCH89/ kann als weiterführende Literatur empfohlen werden; hier wird der Umgang mit dem Smalltalk/V-System gut herausgearbeitet.

Um dem an Smalltalk Interessierten einen Eindruck zu geben, was sich in einem Smalltalk-System an wiederverwendbaren und weiterentwickelbaren Klassen befindet, möchte ich in Abb. 6 (mit freundlicher Genehmigung von Digitalk Inc.) die Klassensammlung von Smalltalk/V vorstellen. Ich hätte auch diejenige von Smalltalk-80 wählen können, nur es hätte mehr Raum beansprucht (Smalltalk/V hat um 100 Klassen und rund 2000 Methoden, doch deutlich weniger als vergleichsweise das volle System Smalltalk-80 mit über der doppelten Anzahl an Klassen); der Leser entschuldige dieses Argument.

Smalltalk/V enthält übrigens einen Prolog-Interpreter, Kombination objektorientierter und regelgesteuerter Programmierung ist möglich.

Für das Zurechtlegen eines fertiggestellten Anwendungsprogramms zur Auslieferung gibt es auch hier einen Zusatz, genannt *Application Manager*; er wird von einer Firma Soft-Pert Systems, Ltd., Nashua, NH, USA abgeboten (direkt bei Digitalk gibt es für diesen Zweck auch eine sog. *.EXE*-Lizenz).

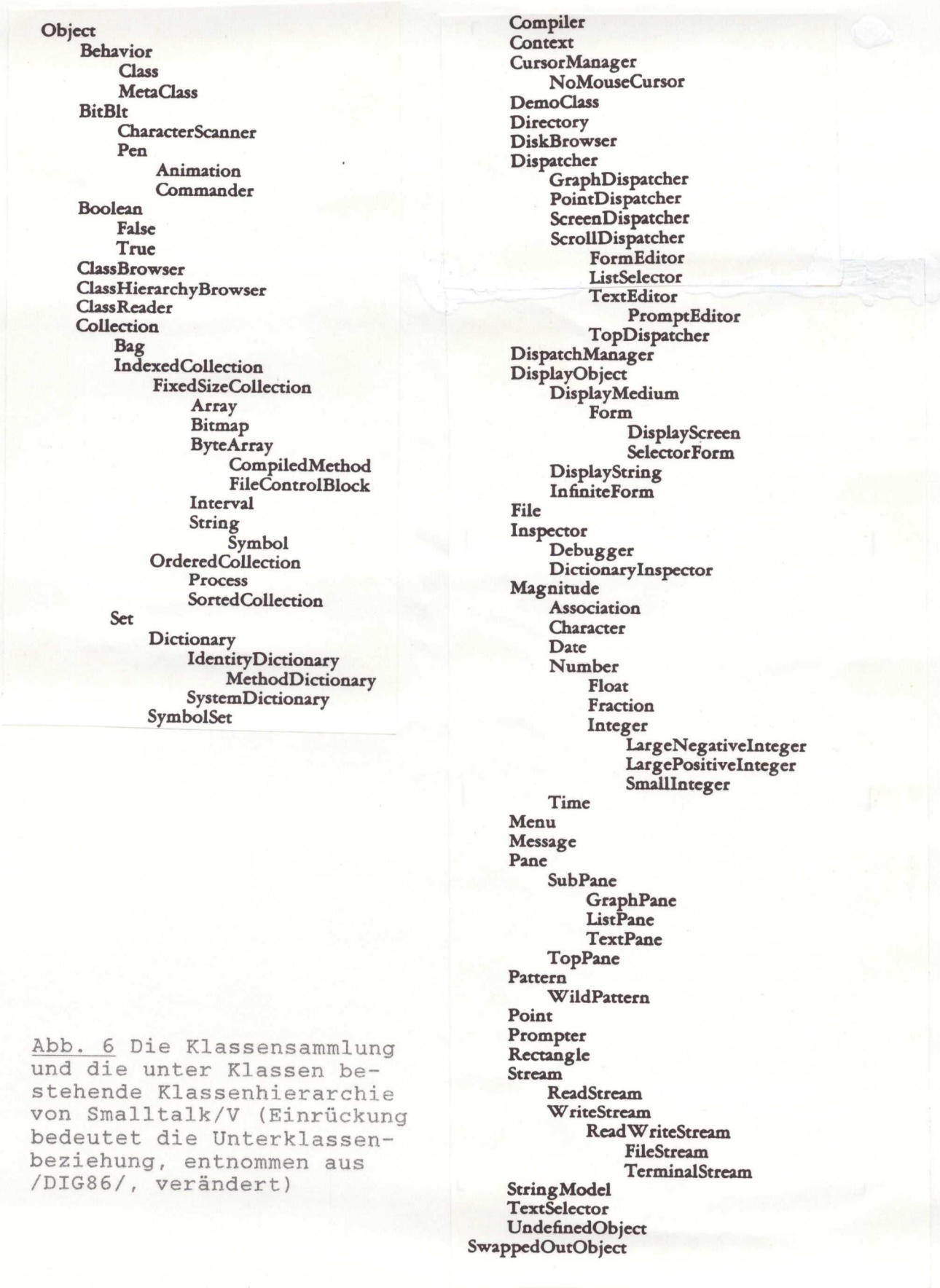


Abb. 6 Die Klassensammlung und die unter Klassen bestehende Klassenhierarchie von Smalltalk/V (Einrückung bedeutet die Unterklassenbeziehung, entnommen aus /DIG86/, verändert)

3.5. "*Goodies*" von Digitalk (ursprünglich von Carleton University, Ottawa), *Widgets/V 286* von Acumen Software u.ä.

Diesen Abschnitt habe ich nicht aufgenommen, um auf eine spezielle Weiterentwicklung eines der Smalltalk-Systeme (hier der Smalltalk/V-Familie) zu sprechen zu kommen, sondern um dem Leser einen Eindruck über Art und Vorgehensweise zum Wiederverwenden und Weiterentwickeln in einem Anwendungsgebiet zu geben. Allgemeinere Aussagen über verfügbare Anwendungsgebiete und wie man daran kommt, finden sich in Abschnitt 3.6.

Seit der Einführung von Smalltalk/V hat sich das Angebot an hochauflösenden Graphikbildschirmen erhöht. Das Arbeiten mit einem Smalltalk-System ist angenehmer, je bessere Graphikfähigkeiten man ausnutzen kann. In anderen Systemen braucht man da neue *"Treiber"*. Bei Smalltalk wird an den Klassen, die die Bildschirmdarstellung bestimmen, eine entsprechende Änderung bzw. Weiterentwicklung vorgenommen (selbstverständlich auch bei den Systemprimitiven): *EGA/VGA-Extension Kit*.

Auch mit dem Smalltalk-System will ein Benutzer andere Rechner erreichen können, Vernetzung ist gefragt. Klassen, die die zum Aufbau der Verbindungen nötigen Bildschirmfenster behandeln, sind hinzuzufügen: *Communications Application Pack*.

Aus Systemprogrammierungssicht erweisen sich einige Zusätze als wünschenswert: Mehrfachverarbeitung, um gewisse Simulationsanwendungen realisieren zu können; Zusammenarbeit mit PC-DOS/MS-DOS; Weitergabe ausgeprägter Objektstrukturen auf den peripheren Speicher und über Kommunikationsnetze; Ergänzung des (rückwärts-verkettenden) Prolog-Inferenzvorgehens um Vorwärtsverkettung; (u.a.): *Goodies#1 Application Pack*.

Programmier- und Benutzungshilfen wie Buchstabierprüfung beim Schreiben des Programmcodes in Methoden; programmtechnische Simulation der Gleitpunktrechnung, wenn kein Koprozessor eingebaut ist; Ausgabe für Laserdrucker in Post-

Script; formatiertes Ausdrucken (*pretty printing*) von Klassen; einige Benutzungshilfen für das Arbeiten mit der Maus; verbesserte Menüstrukturen; weitere Arten von Scheiben (Scheiben, man erinnere sich, sind das, was in Fenstern eingesetzt wird) für einige Bildschirmfenster; Zufallszahlengenerator; (u.a.): *Goodies#2 Carleton Tools.*

Vorkehrungen zur anwendungsbezogenen Kontrolle über Klassen und Versionen von Klassen; Unterstützen optisch lesbarer Schrift bei der Eingabe; 3-D graphische Anwendungen; Visualisieren und Handhaben gerichteter, azyklischer Graphen; Matrixoperationen; (u.a.): *Goodies#3 Carleton Projects.*

Weitere Bausteine für die Entwicklung von interaktiven Anwendungen, die auch die aus anderen Systemen bekannt gewordenen *"Widgets"* (ein Kunstwort) anbieten: *Widgets/V 286* (von Acumen Software, Palo Alto, CA, USA).

Wesentlich bei dieser Vorgehensweise, typisch für den Umgang mit Smalltalk-Systemen, ist mir die problemlose, saubere Integration solcher Änderungen bzw. Weiterentwicklungen in das Klassensystem des Ausgangsprodukts. Ohne Zutun kann sofort mit dem *Stöberer* auch hier Einsicht gewonnen, können individuelle Anpassungen vorgenommen werden; andere Hilfsmittel (wie der *Inspektor*, die -- in Abschn. 4.2. noch einzuführende -- *Testhilfe*) erfassen sofort auch die neuen Klassen und deren Ausprägungen. Die Komplexität in der Handhabung durch den Programmierer und/oder Benutzer steigt nicht an.

3.6. Anwendungsprogramme für Smalltalk

Die gerade besprochenen "*Goodies*" stellen Programme dar, die einem Smalltalk-Anwender zur Wiederverwendung vorprogrammierte und getestete Klassen (bestimmter -- zugegebenermaßen systemnaher -- Anwendungsgebiete) bereitstellen. Dies ist das typische Vorgehen, um für Smalltalk Anwendungsprogramme zu entwickeln und zur Weiterverbreitung/Wiederverwenung anzubieten. In /BAU87/ wird das (in Bezug auf Smalltalk-

80, gültig selbstverständlich aber auch für die anderen Smalltalk-Systeme) mit einem ganz kurzen, treffenden Satz charakterisiert: "*Smalltalk-80-Programmierung ist stets Weiterentwicklung des bestehenden Smalltalk-80-Systems*". Inzwischen gibt es viele solcher Weiterentwicklungen, für viele Anwendungsgebiete, man muß sich im Einzelfall nach dem umsehen, was man benötigt. Erst nach erfolgloser Suche sollte man an eine Eigenentwicklung herangehen. In der (frühen) Smalltalk-Literatur (wie /ALE85/, /BAU87/, /PUG88/) wurde noch besonders darauf hingewiesen, was es gibt; heute liest man darüber nur noch in den Kundenzeitschriften der Anbieter von Smalltalk-Implementierungen (z.B. im *Smalltalk-80 Newsletter / ParcPlace Newsletter* von Xerox PARC / ParkPlace Systems; in der Kundenzeitschrift *Scoop* von Digitalk; in derjenigen von Tektronix -- genannt *theActiveView*, inzwischen eingestellt --; den Ankündigungen (*Products and ...*) von Georg Heeg Smalltalk-80-Systeme u.a.), dem Newsletter of OOPSTAD (*object-oriented programming for Smalltalk Applications Development Association*), selbstverständlich auch in Konferenzbänden (wie /OOPxx/, /ECOxx/ bzw. /TOOxx/) und Zeitschriften wie dem *Journal of Object-oriented Programming*, oder wie *BYTE* (u.a.).

3.7. *The Little Smalltalk* von T. Budd

Prof. Budd hatte 1984 begonnen, eine Untermenge von Smalltalk unter UNIX zu implementieren. Eine gute Portabilität wurde für das um 1986 fertiggestellte, öffentlich verfügbare (*"public domain"*) Produkt *The Little Smalltalk* dadurch erreicht, daß es in der Programmiersprache C programmiert wurde; unterstützt durch das Buch /BUDa87/ fand es schnell Verbreitung. Wenn auch einige Abänderungen / Auslassungen gegenüber der vollen Sprache bzw. dem vollen Benutzungssystem (z.B. dem Stöberer u.ä.) bedauerlicherweise in Kauf zu nehmen sind, ist diese Implementierung zum Kennenlernen von Smalltalk und dem Vorgehen bei der objektorientierten Programmierung mit Hilfe von Smalltalk durchaus

brauchbar.

3.8. Smalltalk-Standard

Im Jahr 1988 begann ein Komitee unter dem Dach von IEEE die Arbeit an einem Smalltalk-Standard. Die Rückfrage beim damaligen Vorsitzenden des Komitees, Dr. Peter Deutsch, Mitarbeiter bei ParcPlace Systems, während der Arbeit am Manuskripts dieses Aufsatzes fiel knapp aus: Kein Bedarf, keine Aktivitäten mehr, keine Ergebnisse.

3.9. Einarbeiten in Smalltalk

Einige Firmen, die Smalltalk in ihrem Verkaufsprogramm haben, bieten Kurse an. Ich möchte niemanden davon abhalten, an einem derartigen Kurs teilzunehmen. Auch im Seminargeschäft ist Smalltalk zu finden, oft unter dem Zugtitel *Objektorientierung* verpackt, wo es dann als erstes und als "Paradepferd" behandelt wird.

Aber auch im Selbststudium läßt sich der Einstieg in Smalltalk finden. Dies besonders deswegen, weil die mit den Produkten mitgelieferten Manuale im allgemeinen sehr ordentlich sind, auch durchzuarbeitende Beispielprogramme mitgeliefert werden (wie es bei der PC-Programmierung ja oft geschieht); und deswegen, weil der Interessierte sich systematisch unter Ausnutzen der direktmanipulativen Handhabbarkeit visualisierter Objekte, wie sie die mitgelieferten Systemkomponenten erlauben, insbesondere der *Stöberer*, der *Inspektor* und die *Testhilfe*, Schritt für Schritt an die wiederverwendbaren Programme in den als Klassenhierarchien angelegten Programmsammlungen heranführen lassen kann. Bei Smalltalk ist, um es lässig zu sagen, dieses Kennenlernen der Klassenhierarchie das A und O des (Selbst-)studiums, alles andere ist (für jemanden, der Rechnerumgang und objektorientierte Programmierung -- dazu siehe eine Bemerkung weiter unten -- gewöhnt ist) ein Katzenspiel.

Und wer es nicht durch Probieren erlernen möchte, wer zunächst Grundlagen erarbeiten möchte; hier sind drei Lite-

raturzitate: /LIN88/ und /LIEb89/, in gewissem Umfang auch /BEC89/.

4. Perspektiven

In diesem letzten Kapitel möchte ich einige Punkte ansprechen, an denen Kritik an Smalltalk geübt werden könnte bzw. an denen Vorstellungen zum Weiterführen der mit objektorientierten Sprachen im allgemeinen und mit Smalltalk im besonderen eingeschlagenen Wege ansetzen.

Zuvor eine Bemerkung: Im vorgegebenen Umfang dieses "*state-of-the-art*"-Artikels läßt sich nicht auf jeden Punkt und jede Perspektive eingehen; viele weitere Anregungen finden sich in den Büchern und Aufsätzen, die als weitere Literaturempfehlungen aufgeführt sind, besonders auch in den Tagungsbänden (/OOPa86/, /OOPb86/, /OOPa87/, /OOPb87/, /OOP88/, /OOP89/, /OOP90/, /ECO87/, /ECO88/, /ECO89/, /TOO89/, /TOO90/ -- das mir noch nicht vorliegt --) und in dem *Journal of Object-oriented Programming*, generell hinsichtlich objektorientierter Vorgehensweise und den damit zusammenhängenden Fragen wie auch speziell hinsichtlich Smalltalk.

4.1. "*Programmieren ohne Netz*"

In Anlehnung an das Umschlagsbild des ersten Themenhefts über Smalltalk aus dem Jahre 1981 (/BYT81/) wird gerne ein Heißluftballon mit Smalltalk assoziert, das "steuerlose" Sichdahintreibenlassen oder Dahingetriebenwerden (das aber auf dem erwähnten Titelbild doch die "Ziellandung" auf dem steilen Fels mit der trutzigen Burg im stürmischen Ozean erlaubt). Ist dieses Bild so gerechtfertigt? Ich meine nicht. Unterstellt, daß der Programmierer/Benutzer mit objektorientiertem Vorgehen an sich etwas anfangen kann, läßt sich mit Smalltalk das Ziel erreichen, über mehrere Etappen ("*Prototypen*", wie man das durchaus positiv gemeint gerne nennt), explorativ /ALE85/, *"ohne Netz"*. Letzteres Assoziation halte

ich für zutreffender, *"Programmieren ohne Netz"*, hat man doch, wie in anderen interpretierenden Ansätzen (z.B. bei den Sprachen des "LISP-Zweigs", vgl. Abb. 1), keine (formale) Hilfestellung durch ein Übersetzungsprogramm, keine Begrenzung der Gültigkeit von Bezeichnern und damit eine Eingrenzung von Zugriffsrechten in einer Blockstruktur (eine Diskussion dieser Frage findet sich bei /LEH86/), keine statische Typbestimmung und -überprüfung, kein (die Implementierungsleistung steigerndes) Umsetzen des abstrakten Zwischencodes (bei Smalltalk aufbauend auf der virtuellen Maschine) in den Befehlssatz der real eingesetzten Maschine, keine Kontrolle über die Speicherbelegung u.ä. Ist dies ein Mangel? Selbst wenn man dies meint, es (wurde bzw.) wird an Smalltalk-Übersetzern gearbeitet (z.B. /ATK86/, /SAM86/, /BUS87/, /JOH88/, /WAG90/)!

4.2. Robustheit von Smalltalk-Programmen

Man erwartet, daß ein Programm sich robust gegen allfällige Fehler oder Sondersituationen verhält. Das ist bei Programmen, die in einer objektorientierten Weise aufgebaut sind, nicht anders als sonst. Welche Vorkehrungen finden sich bei Smalltalk, wie wirkt sich ein Fehler bzw. eine Sondersituation aus? Ein Objekt kann mit einer Botschaft, die es empfangen hat, nichts anfangen. Aufgrund des Vererbungsmodells wird die Kette (Smalltalk hat nur Einfachererbung, siehe Abschnitt 4.7) der Oberklassen durchsucht; gelangt man dabei bis zur Wurzel der Vererbungshierarchie, d.h. der Klasse *Object*, finden sich dort eine Reihe von Methoden (bei Smalltalk/V z.B. ***invalidMessage***), die jedenfalls ansprechen; typischerweise wird dadurch die *Testhilfe* (*debugger*) mit einem sog. Untersuchungsfenster (*walkback window*) aktiviert. Und nun obliegt es dem Programmierer/Benutzer, etwas Sinnvolles zu unternehmen, um des Fehlers bzw. der Sondersituation Herr zu werden. Selbstverständlich kann ein Programmierer, wenn er dies für erforderlich hält, eine derartige Methode in einer seiner (Unter-)Klassen umdefinie-

ren. So einfach ist das. Erreicht man aber damit das, was landläufig bei robusten Programmen als Systemreaktion erwartet wird?

In der Literatur fand ich zu dieser Frage (bei Smalltalk im speziellen und bei objektorientierten Ansätzen im allgemeinen) wenig -- vielleicht von EIFFEL /MEYb88/ abgesehen --. Symptomatisch für die z.Zt. kaum befriedigende Situation bei der praktischen Behandlung kann man m.E. das neue Buch /FED90/ herausstellen, in dem erst wichtige Grundlagen für die Fehlerbehandlung bei objektorientiert programmierten Programmen gelegt werden, die eben noch nicht in Sprachen wie Smalltalk umgesetzt sind.

4.3. Implementierungsfragen

Zur Implementierung (bei interpretierendem Vorgehen) noch eine Bemerkung: Smalltalk-80 (auch Smalltalk/V und die Varianten davon) sprechen, wie in Abschnitt 3.1 schon erwähnt, von der *virtuellen Maschine* und von der *virtuellen Welt*. Letzterer Begriff hat etwas mit der Persistenz zu tun; darauf komme ich in Abschn. 4.8. zurück. Die virtuelle Maschine ist gut dokumentiert (/GOL83/); sie ist vergleichbar einer "abstrakten Maschine", die "Zwischencode" (den sog. *bytecode*) interpretiert, wie man das für andere Programmiersprachen auch kennt. Aus Platzgründen kann ich darauf -- wie auf andere Lösungen -- nicht detailliert eingehen. Interessant ist vielleicht, daß es auch Ansätze für eine Hardware-Unterstützung einer Implementierung gibt (ähnlich dem Vorgehen zur Implementierung von Lisp auf einer Lisp-Maschine); hier verschwindet natürlich der Unterschied zwischen interpretierendem und kompilierendem Vorgehen. Die im Literaturverzeichnis angegebenen Tagungsbände (/OOPxx/ usw.) enthalten einige Aufsätze darüber (auch /FOR88/ und /RIC89/ -- siehe in diesem Abschnitt weiter unten -- machen davon Gebrauch). Einen speziellen Literaturhinweis möchte ich noch machen auf /LUT87/.

Interpretierendes Arbeiten eines Sprachsystems ist bei

bester Implementierung kaum in der Effizienz mit einem kompilierenden Vorgehen zu vergleichen, zumindest wenn man Effizienz in der traditionellen Weise *Rechenschritte pro Zeiteinheit* betrachtet. Man bezog früher (zuletzt sah ich das bei /BAU87/ bzw. /MIT89/) jede Implementierung hinsichtlich dieses Leistungsmaßes auf den *Dorado*-Rechner von XEROX, auf dem eine der ersten Smalltalkimplementierungen vorgenommen wurde (übrigens ebenfalls mit Mikrocode-/Hardwareunterstützung, siehe /DEU83/); eine Implementierung, die diesen Leistungsstand erreicht, gilt als brauchbar. Nun, inzwischen sind Rechner mit höherer Prozessorleistung und bessere Implementierungen auf dem Markt (siehe oben).

Man sieht, betrachtet man die Bedeutung der interaktiven Arbeitsweise, aber eigentlich ein anderes Maß als aussagekräftiger an. Ich möchte es als *Arbeitsschritte pro Zeiteinheit* charakterisieren, Arbeitsschritte, die der Programmierer/Benutzer zum Erreichen seines Arbeitsziels an der visualisierten Quasirealität direktmanipulativ anzustoßen hat. Die Flexibilität, die Smalltalk infolge seiner interpretierenden Vorgehensweise als "Dynamik" in den Daten- und Programmstrukturen anbietet, schlägt dann auch zu Buche. Aber auch bei dieser Betrachtungsweise sollte man gesteigerte Leistung im traditionellen Sinn bei Prozessor und Implementierung nicht hintanstellen, da ist schon noch eine Verbesserung wünschenswert.

Man erlaube mir noch (nochmals) einen Hinweis auf mir bekannt gewordene Smalltalk-Implementierungen der letzten Jahre an deutschen Universitäten, nämlich auf /FOR88/, /HÜT85/ und /RIC89/; sollte es weitere Implementierungen geben, wäre ich für einen Hinweis dankbar (auf die Implementierungen von G. Heeg, die auf professionellem Standard kommerziell angebotene Produkte hervorbrachten, bin ich schon in Abschnitt 3.3 eingegangen).

4.4. Gibt es ein Modell für die Entwicklungsarbeit mit Smalltalk?

"*Ohne Netz*", wie ich das in Abschn. 4.1. sagte, meint aber auch, daß inzwischen doch einigermaßen gebräuchlich gewordene Vorgehensweisen, die das Software-Ingenieurwesen schon lange propagiert, Spezifikation in einer Entwurfsphase vor einer Implementierungsphase, Verifikation der entstandenen (besser wäre, der entstehenden) Programme, defensives Programmieren, Vorkehrungen zum Programmieren im Team, auch Hinzuziehen weitreichender, rechnergestützter Entwurfswerkzeuge (unter dem Schlagwort "CASE" bekannt), strikte Dokumentation (siehe dazu aber /REE89/) u.ä., nicht zur Verfügung stehen. Dieser Einwand gilt aber nicht nur für Smalltalk, auch andere objektorientierte Sprachen (Eiffel wohl am ehesten ausgenommen /MEY88/) leiden darunter. Es fehlt ein fundiertes, griffiges Modell zur Vorgehensweise bei objektorientierter Programmentwicklung. Das ist aber weniger eine Schwäche einer Sprache sondern vielmehr des objektorientierten Ansatzes an sich (bzw. des Verständnisses des Umgangs mit diesem). Das Problem ist erkannt; eine schöne Aussage dazu las ich kürzlich in /BER90/:

> "Imagine a 'playing field' with well-defined object-oriented software engineering in the middle. We see two teams approaching the middle, from two different directions:
> - o One team is the 'object-oriented programming crowd' .. The players on this team, despite their internal disagreement, have very few problems identifying objects and classes. However, when asked about well-defined approaches to things like 'object-oriented design', they tend either say things like 'I know how to do it, but I am not sure I can tell you', or 'huh?'.
> - o The other team very much resembles Hannibal crossing the Alps. They bring a great deal of useless 'baggage' with them. You guessed it, they are the 'structured crowd', and their 'useless baggage' consists of things like data-flow diagrams, structure charts, entity-relationship diagrams, and relational databases. When asked why they want to use these items for object-oriented software engineering, they respond with answers like: 'We know how to use these, and we don't know how to solve problems without them', or 'Hey, man! We just sunk a

> lot of money into CASE tools which support these things, and we gotta use'em'."

Diese Szene bessert sich aber. Gerade in der neueren Literatur gibt es da einiges zum Nachlesen; Beipiele sind /KOR90/, /WIRb90/, /HEN90/, /STO89/.

4.5. Wie beurteilt man die Güte eines Smalltalk-Programms?

Ingenieurmäßiges Entwickeln von Programmen braucht auch Kriterien zum Beurteilen der Güte einer Entwicklung, ein Komplexitätsmaß. Bei Programmen, die in einer objektorientierten Sprache wie Smalltalk formuliert sind, ist man da noch nicht so weit wie vergleichsweise bei traditionellen Sprachansätzen. Einen interessanten Weg verfolgt Lieberherr /LIEa89/, in dem er ein *Maß der Nähe* einführt, um die in der Klassenhierarchie beim Versenden bzw. Empfangen von Botschaften zu überbrückende Distanz zu kontrollieren. Für Smalltalk heißt das dann (in Klammern Einfügungen von mir):

> *In all message expressions inside method M* (of a class C) *the receiver must be one of the following objects:*
> - *an argument object of M, including objects in pseudo-variables self and super* (self ist das betroffene Objekt mit allen Methoden im Protokoll, super nur mit den ererbten),
> - *an immediate part of self* (also eine Exemplarvariable),
> - *an object that is either an object created directly by M or an object in a global variable* (ja, ja, das gibt's).

Wird es das sein? Ich möchte noch auf Erfahrungsberichte anderer als nur der Autoren warten.

4.6. Portabilität und ähnliches

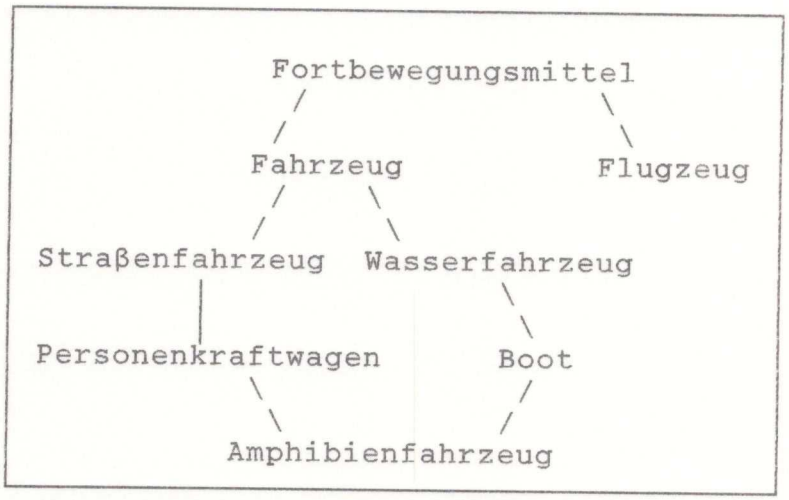
Nun muß ich etwas in die Niederungen der Programmierwissenschaft steigen. Wie steht es bei Smalltalk mit der Portabilität, mit dem Mitverwenden von Programmen, die in anderen Programmiersprachen abgefaßt sind, mit dem Anschluß von Betriebssystemdiensten u.ä.? Der Herausgeber dieses Heftes hat explizit danach gefragt, so möchte ich nicht dadurch kneifen, daß ich keine Antwort gebe. Die Antwort ist, nicht jeder wird voll befriedigt sein: Smalltalk ist ein in sich ge-

schlossenes Programmsystem; natürlich kann man auf der Ebene primitiver Methoden etwas für Portabilität, Fremdprogrammanschluß oder Zugriff auf Hardware und Basissoftware tun (man lese nach, wie es geht in /GOL83/, Teil 4; oder in /DIG86/ mit Implementierungsbeispiel für PC DOS-Rechner), aber braucht man so etwas?

4.7. Zur Ererbungsstrategie in Smalltalk

Smalltalk kennt (nur) den baumförmigen Aufbau in der Ererbungshierarchie. Dies ist eine (u.U. nicht unerhebliche) Einschränkung. Wie soll man beispielsweise die in der Abb. 7 skizzierte Situation von Beziehungen unter Objektklassen erfassen? Andere objektorientierte Sprachen und Pro-

Abb. 7
Klassenbeziehung unter *Fortbewegungsmitteln*



grammiersysteme (z.B. C++ -- ab Version 2.0 -- bzw. Eiffel) sind da flexibler. Sie kennen die *Mehrfachererbung*, mit deren Hilfe die in der Abbildung skizzierte Situation leicht modelliert werden kann.

Betrachten wir die Situation bei Smalltalk nochmals genauer:

- Ererben des Datenaspekts ist *unbeschränkt kumulierend*. Man kann sich vorstellen, daß alle bei der Klassenbeschreibung des betrachteten Objekts und in den Klassenbeschreibungen eventueller weiterer Oberklassen vermerkten Exemplarvariablen den kumulierten Satz von Exemplarvariablen des Ob-

jekts bilden (Namenskonflikte wie üblich von unten nach oben verdeckend aufgelöst); es gibt (ohne Anwenden von Tricks) keine Möglichkeit, den Zugriff auf ererbte Exemplarvariable zu verhindern (Verstoß gegen das Prinzip der Datenkapselung, ein anerkanntes Prinzip des Software-Ingenieurwesens, das objektorientierte Ansätze eigentlich hoch halten wollten, vgl. /SNY86/); daher als *unbeschränkt* charakterisiert. Diese Vorstellung könnte man auch bei einer Mehrfachererbung beibehalten, wenn event. Namenskonflikte in einer zu definierenden Weise auf einer Verdeckungsschicht behandelt wären.

- Der Handlungsaspekt wird ebenfalls *unbeschränkt kumulierend* behandelt. In das Protokoll einer Klasse sind alle Methoden aufgenommen, die bei Oberklassen festgelegt sind; Namenskonflikte (die regelmäßig und beabsichtigterweise auftreten und zwar beim Spezialisieren des Handlungsaspekts für die Objekte einer Unterklasse) werden entsprechend aufgelöst (eventl. mit dem *super*-Zugriff transparent gemacht). Soweit könnte man sich auch eine Erweiterung mit Mehrfachererbung vorstellen, Behandeln von Namenskonflikten auf einer Schicht wie beim Datenaspekt. Aber, zwischen Methoden bestehen zusätzliche Bekanntsein-Beziehungen, die bei den Exemplarvariablen nicht vorliegen, wenn beim Abarbeiten einer Methode eine Botschaft an eine andere ererbte Klasse (die sich plötzlich als verdeckt herausstellt) ausgeschickt wird; eine zunächst brauchbar erscheinende Verdeckungsstrategie kann sich so schnell als unbrauchbar herausstellen.

Diese Argumentation soll auftretende Schwierigkeiten plausibel machen. Eine genauere Betrachtung der Probleme, auch der bei der in Smalltalk vorgenommenen Einschränkung auf nur einfache Vererbung nicht erreichten Ziele des Software-Ingenieurwesens, findet sich (z.B.) in der schon zitierten Arbeit von Snyder /SNY86/; /HAL87/, /BRE89/ (u.a.) diskutieren die Probleme aus theoretischer Sicht, /WED90 ist in diesem Zusammenhang auch sehr lesenswert. Der scharfsinnige Vorschlag von Borning und Ingalls /BOR82/ fand bei den verfügbaren Smalltalk-Implementierungen keine Verwendung.

4.8. Objektpersistenz

Eine Anwendung kann man nicht beliebig an- und ausschalten. Wenn für mich als Benutzer die auf einem Rechner eingerichtete Anwendung von Nutzen sein soll, möchte ich nicht bei einer Unterbrechung mir alle möglichen Dinge notieren und bei der Fortsetzung der Arbeit am Rechner neu eingeben

müssen. Bei Wiederaufnahme des Umgangs mit dem System soll dieses so weiterarbeiten, als ob es die durch den Sitzungsabbruch eingeleitete Unterbrechung nicht gegeben hätte. Hinsichtlich der Programme (d.h. in unserem Fall der Klassenbeschreibungen) sind wir dieses Verhalten gewohnt (man bedenke aber, daß bei Smalltalk auch während einer Sitzung, bei der es primär um Benutzung geht, durchaus an einer Klassenbeschreibung etwas herumprogrammiert worden sein kann, was man selbstverständlich automatisch vermerkt haben möchte). Daten überdauern in herkömmlicher Technik nur, wenn sie in Dateien abgelegt sind, was eine vorgängig, aktiv veranlaßte Abspeicherung voraussetzt; und sie sind nur dann wieder beim Neubeginn des Programmablaufs greifbar, wenn sie durch eine aktive Maßnahme wieder eingelesen sind. In Smalltalk ist dies anders: Alle ausgeprägten Objekte überdauern, ich muß da keine besondere Vorkehrung machen (wenn ich das Überdauern nicht haben möchte, dann ist das auch möglich, erfordert aber von mir eine aktive Entscheidung dazu). Diese Eigenschaft des Smalltalk-Systems nennt man *Objektpersistenz* (auch nur *Datenpersistenz*, was aber m.E. bei Smalltalk an der Sache vorbeigeht). Man sei sich bewußt, daß wegen der Smalltalk-Eigenschaft, *Klassen sind Objekte*, damit auch das Überdauern der Programme bzw. von Änderungen an ihnen ohne weiteres Zutun erreicht ist.

Objektpersistenz findet man in anderen interaktiven Systemen kaum. Eine mit einem der gebräuchlichen Fenstersysteme aufgebaute Anwendung sieht möglicherweise während der Nutzung ähnlich aus und läßt sich in ähnlicher Weise handhaben wie ein Smalltalk-System, was man aber nicht findet ist dieses automatische Erhalten des gesamten Systemzustands bei einer Programmunterbrechung. Für Smalltalk ist nichts natürlicher als diese Eigenschaft, in meinen Augen ein entscheidender Vorteil in der Benutzung (auch ein Beitrag zu dem von mir herausgestellten Leistungsmaß für interaktives Arbeiten).

4.9. Zum Programmieren interaktiver Anwendungen unter Smalltalk

In einem ganz anderen Bereich hat Smalltalk auch einen wesentlichen Beitrag geleistet: Lange Zeit war das sog. *Seeheim*-Modell (/PFA85/) der Ausgangspunkt für Überlegungen zur Architektur der Benutzungsoberflächen-Steuerungssysteme (besser bekannt unter dem Begriff UIMS, *user interface management system*). Es erwies sich als schwierig, hier tatsächlich die angestrebte Trennung zwischen Interaktionssteuerung, die getrennt entworfen sein sollte, und Anwendung zu realisieren. Man mußte eine umständliche, wie man sagt, semantische Rückführung von der Anwendung zur Steuerung vorsehen, um z.B. solche Effekte wie das Hellerwerden eines bei einer Maus- bzw. Kursorbewegung überstrichenen Ausschnitts der Bildschirmfläche (einem "Darstellungselement") mit einem visualisierten Objekt zu erreichen; aber nur in dem Fall, daß nicht ein anderweitig bereits "ergriffenes" Objekt nur mehr oder minder beiläufig das betrachtete Darstellungselement berührt hat /HAR89/.

Smalltalk stellt demgegenüber das sog. MVC-Modell (*model-view-control*), in dem der Anwendung (*model*) u.U. mehrere Paare von Sichtsteuerungs- (*view*) und Sensor- (*control*) Komponenten zugeordnet sind, Komponenten, über die, nach Aufgaben getrennt, die Steuerung und im Bedarfsfall die Rückführung bewerkstelligt werden. Dieser Ansatz hat sich als wesentlich flexibler erwiesen (und bildet die Grundlage auch anderer direktmanipulativer Benutzungsoberflächen, die eine Trennung zwischen Anwendung und Steuerung angestrebt haben). Niemand erwartet nun etwas anderes, als daß bei Smalltalk die Sichtsteuerungskomponenten und die Sensorkomponenten Ausprägungen entsprechender Klassen sind. Sie sind als eine Art Dienstleistungsobjekte in der Lage, Objekte des Modells, denen sie zugeordnet sind, objektorientiert zu visualisieren bzw. Manipulationen des Benutzers an den Visualisierungen zu erkennen und in Einwirkungen auf die Objekte des Modells umzusetzen. Das alles geschieht, wie sollte es anders sein,

durch Zusenden dafür vorgesehener Botschaften. Smalltalk-Implementierungen bieten dem Programmierer, je nach Zweck und gewünschter Visualisierungsform bzw. Sensorfunktionalität unterschiedliche Klassen von *view-control*-Objekten an, die er zum Programmieren eines interaktionsfähigen Programms nach seinen Vorstellungen einsetzen, auch anpassen kann.

Obwohl von der architektonischen Struktur her säuberlich getrennt, läßt sich damit das Benutzungsoberflächen-Steuerungssystem und das Anwendungssystem in gleicher Weise programmieren. Das ist ein Impuls für die Softwaretechnologie, der noch gar nicht in all seinen Auswirkungen abgeschätzt werden kann, fangen doch Anwendungsentwickler erst allmählich an, diesen Ansatz aufzugreifen (in der professionellen Entwicklung von Fenstersystemen hat der Ansatz schon früher Einzug gehalten). Man darf mich allerdings nicht falsch verstehen: Es ist sehr wohl bekannt, daß man eine so gestaltete Benutzungsoberfläche nicht so nebenbei erhält; der damit verbundene Programmieraufwand ist nicht gering. Anleitungen können dabei helfen, so beispielsweise das "Kochbuch" von Krasner und Pope /KRA88/ (das man leider nicht als besonders geglückt formuliert einstufen kann) oder die kommentierte Beispielentwicklung in /BUDc87/; das letzte Wort zur "Programmiererfreundlichkeit" ist da m.E. aber noch nicht gesprochen. Das Zurverfügunghaben von Bausteinen für Darstellungselemente, den sog. "Widgets" (vgl. Abschnitt 3.5), zur Programmierung dann auch der Objektvisualisierungen wird eine Hilfe sein.

4.10 Visuelle Unterstützung bei der Smalltalk-Programmierung

Hier möchte ich nicht über das Smalltalk-Fenstersystem und die verschiedenen Entwurfshilfsmittel etwas sagen, die, wie wir ja in Abbildungen gesehen haben, durchaus dem Programmierer eine visuelle Unterstützung bieten. Hier würde ich gerne eine m.E. der Smalltalk-Welt angemessene Form der Unterstützung des Programmierers in Form der Visualisierung der auftretenden "Konstrukte", von Klassen, von Klassenhie-

rarchien, von Bestandteilen von Klassen und ihren Ausprägungen als Objekte, von Methoden und den in ihnen auftretenden Botschaften also (letzteres sozusagen das *Flußdiagramm* bzw. *Struktogramm* der Smalltalk-Programmierung) vorstellen -- und nicht nur die allein verfügbare Form als textuelle Beschreibung der Konstrukte --; für das Programmieren, wie auch dann, animiert, für das Beobachten des Programmablaufs (und damit der Vorgänge in der simulierten Welt). Das wäre m.E. diesem durch und durch objektorientierten System und der dabei zugrundeliegenden Idee, mit visualisierten Objekten der quasirealen Umwelt direktmanipulativ umzugehen, natürlicherweise angemessen. Leider kann ich nur über Ansätze berichten, Ansätze, die nicht einmal speziell auf Smalltalk ausgerichtet sind (oder könnte man da ähnlich wie bei Flußdiagrammen/Struktogrammen -- auf der unteren Programmebene, wo wir uns da ja bewegen würden, Programmieren im Kleinen -- auf die Ausrichtung auf eine bestimmte Programmiersprache verzichten?).

Ausgehend von Arbeiten von Booch /BOO86/ haben Wasserman et al. /WAS90/ die OOSD-Notation (*object-oriented structured design*) in Weiterführung seines Ansatzes *Software through pictures* /WAS87/ entwickelt (stark von den Gegebenheiten bei Ada beeinflußt). Wasserman sieht diese Notation mehr in den Vorbereitungsphasen, d.h. für das Festlegen einer Spezifikation, weniger für die Implementierung (Phasen, die in Smalltalk, wie wir gesehen haben, kaum trennbar sind), deutet allerdings Kombinationsmöglichkeiten mit dem Programmiersystem an. Es fällt der Begriff CASE in diesem Zusammenhang. Wer macht das zuerst für Smalltalk?

Den Übergang von der textuellen Beschreibung der beim Programmieren angesprochenen Konstrukte (gerade in Smalltalk sind das ja alles Objekte!) zu einer visuellen Darstellung, mit der direktmanipulativ umgegangen werden kann, kann man (im Generellen) in Aufsätzen wie /MYE86/ bzw. /AMB89/ studieren. Doch finden sich unter den dort besprochenen (bzw. sonst irgendwie bekannt gewordenen) Programmiersystemen zum

visuellen Programmieren keine Lösungen für den speziellen Fall der objektorientierten Programmierung bzw. noch spezieller der Smalltalk-Programmierung (als einen ersten, denkbaren Ansatz möchte ich an /CUN86/ erinnern). Nur eine Graph-Darstellung der Klassenhierarchiebeziehung, der *Klassen-Objekt*-Beziehung usw. sehe ich bei Smalltalk als nicht natürlich und als nicht angemessen an.

4.11. In der realen Welt läuft alles simultan und verteilt ab; man kann sich mit sich selbst beschäftigen (Selbstreflexion). In Smalltalk auch?

In einem objektorientierten System möchte man mit den Gegenständen und Gegebenheiten der realen Welt modellierend umgehen, die zugehörige Programmiersprache muß gestatten, dies auszudrücken. Diese Überlegungen waren den Smalltalk-Entwicklern sicher geläufig, gibt es doch in Smalltalk-80 einige (wenige) Vorkehrungen, Simultanität (,Verteilung) und Selbstreflexion /FOO89/ zu behandeln. Sehr pragmatisch gibt es Erweiterungen, die in diese Richtung gehen (siehe Abschnitt 3.5). Letztlich ist dies aber keine zufriedenstellende Lösung.

Sieht man den Tagungsband eines *Workshop on object-based concurrent programming* durch, der aus Anlaß der OOPSLA-Konferenz 1988 stattfand /ACM88/, muß man zur Meinung kommen, daß hier noch einige wissenschaftliche Vorarbeit zu leisten ist, bis eine für den Alltagseinsatz brauchbare Lösung gefunden ist. Der schon erwähnte Prof. Wegner habe, so wird berichtet, bei der Eröffnungssitzung gesagt: "*Object-oriented programming was concerned primarily with encapsulation and inheritance, while concurrency was concerned primarily with threads and synchronization*". Diese Gebiete klaffen (noch) auseinander. Und das gilt auch für die engere Betrachtung von Smalltalk (vgl. aber /YOK87). Aus meiner Literaturkenntnis heraus, kann ich hier nicht einmal auf eine Besserung hinweisen.

4.12. Smalltalk-Programmierung im Team

Soweit *Programmieren im Team* nur als Weitergeben von Klassenbeschreibungen zum Wiederverwenden verstanden wird, ist alles in Ordnung.

Meint man darunter mehr, sieht es anders aus: Fehlanzeige wäre nicht die voll zutreffende Aussage. Teammitglieder können schon in der Klassenhierarchie unabhängig an verschiedenen Stellen arbeiten und nötige Ergänzungen bzw. Änderungen vornehmen, um eine gestellte Aufgabe zu erfüllen. Auch das wechselseitige Einbringen von Klassenbeschreibungen in die virtuellen Welten jedes Beteiligten ist unterstützt. Nur, gewisse Verwaltungsvorgänge, die sonst eine Entwicklungsumgebung für Teamarbeit als Selbstverständlichkeit abhandeln kann, diese müssen manuell und bei jedem einzeln erledigt werden (wenn man sich dafür nicht ein eigenes Entwicklungssystem vorab in Smalltalk zurechtgelegt hat, eine Zusatzentwicklung, selbstverständlich machbar, aber nicht im System enthalten). Versionenkontrolle, der nächste Schritt, wäre in ähnlicher Weise anzugehen; was noch; ebenso. Smalltalk ist seiner Architektur nach ein integriertes Entwicklungs- und Benutzungssystem eines Einzelnen, das muß man in Erinnerung behalten.

Literatur zum Beitrag

/ACM88/ Proc. ACM SIGPLAN Workshop on object-based concurrent programming; ACM SIGPLAN Notices, vol. 24, April 1989.

/ACM90/ Themenheft *Object-oriented Design*; Comm. ACM, vol. 33, September 1990.

/ALE85/ J. H. Alexander: Exploratory application development using Smalltalk; Notizen zu Interaktiven Systemen, Heft 14, Mai 1985, 17 - 27.

/AMB89/ A. L. Ambler, M. M. Burnett: Influence of visual technology on the evolution of language environments, IEEE Computer, vol. 22, October 1989, 9 - 22.

/ATK86/ R. G. Atkinson: Hurricane: An optimizing compiler for Smalltalk. In /OOPb86/, 151 - 158.

/BAU87/ H. Baumeister, H. Ganzinger, G. Heeg, M. Rüger: Smalltalk-80; Informationstechnik it, 29. Jahrg., August 1987, 241 - 251.

/BEC89/ K. Beck: A laboratory for teaching object-oriented thinking. In /OOP89/, 1 - 6.

/BER90/ E. V. Berard: Object-oriented methodologies; S.S.E. Journal, a Newsletter of the Society for Software Engineering, vol. 2, no. 4, Fall 1990, 4 - 8.

/BLAa89/ G. Blaschek, G. Pomberger, A. Stritzinger: Programmiersprachen zur objektorientierten Systementwicklung -- ein Vergleich. In /HMD89/, 80 - 93.

/BLAb89/ G. Blaschek, G. Pomberger, A. Stritzinger: A comparison of object-oriented programming languages; Structured Programming, vol. 4, October 1989, 187 - 197.

/BOO86/ G. Booch: Object-oriented development; IEEE Trans. Software Eng., vol. SE-12, Dezember 1986, 211 - 221.

/BOR82/ A. Borning, D. Ingalls: Multiple inheritance in Smalltalk-80; Proc. AAAI, 1982, 234 - 237.

/BRE89/ H. Bretthauser, T. Christaller, J. Koop: Multiple vs. single inheritance in object-oriented programming languages. What do we really want?; Tagungsband Alternative Konzepte für Sprachen und Rechner, EWH Rheinland-Pfalz, Abtlg. Koblenz, Bericht 6/89, 1989, 218 - 228.

/BUDa87/ siehe allgemeine Literatur zu Smalltalk

/BUDb87/ R. Budde, K. Kuhlenkamp, K.-H. Sylla, H. Züllighoven: Bib -- ein Bibliographie-System. In /HOF87/, 59 - 88.

/BUDc87/ R. Budde, K. Kuhlenkamp, K.-H. Sylla, H. Züllighoven: Programmentwicklung mit Smalltalk. In /HOF87/, 89 - 121.

/BUS87/ W. R. Bush, A. D. Samples, D. Ungar, P. N. Hilfinger: Compiling Smalltalk-80 to a RISC; ACM SIGPLAN Notices, vol. 22 / ACM Operating Systems Review, vol. 21 / ACM Computer Architecture News, vol. 15, October 1987, 112 - 116.

/BYT81/ Themenheft *Smalltalk*; BYTE, vol. 6, August 1981.

/BYT86/ Themenheft *Object-oriented Languages*; BYTE, vol. 11, August 1986.

/BYT89/ Themenheft *Object-oriented Programming*; BYTE, vol. 14, März 1989.

/CHA90/ M. Charlier: Smalltalk, ein Fabelwesen zum Anfassen; Computer & Software Magazin, 19. Jahrg., Heft 1/2, März 1990, Seiten 60 - 61.

/CLA90/ H. Clausen, A. Hofmann in einer Lehrgangsankündigung der Carl-Cranz-Gesellschaft e.V., Nov. 1990.

/CRA89/ D. Crabb: Smalltalk can be cheap; BYTE, vol. 14, April 1989, 141 - 146.

/CUN86/ W. Cunningham, K. Beck: A diagram for object-oriented programs. In /OOPb86/, 361 - 367.

/DEU83/ L. P. Deutsch: The Dorado Smalltalk-80 Implementation: Hardware architecture's impact on software architecture. In /KRA83/, 113 - 126.

/DIG86/ siehe allgemeine Literatur zu Smalltalk

/ECO87/ Proc. European Conf. on Object-oriented Programming, Springer-Verlag, LNCS 276, 1987.

/ECO88/ Proc. European Conf. on Object-oriented Programming, Springer-Verlag, LNCS 322, 1988.

/ECO89/ Proc. European Conf. on Object-oriented Programming, Cambridge University Press, 1989.

/FED90/ C. Feder: Ausnahmebehandlung in objektorientierten Programmiersprachen; Springer-Verlag, IFB 235, 1990.

/FOO89/ B. Foote, R. E. Johnson: Reflective facilities in

Smalltalk-80. In /OOP89/, 327 - 335.

/FOR88/ M. Forkel: Portierung des Smalltalk-80 Systems auf eine Symbolics Lisp-Maschine; Diplomarbeit, TU Berlin, Inst. f. Angew. Informatik, 1988.

/GOL83/ siehe allgemeine Literatur zu Smalltalk

/GOL84/ siehe allgemeine Literatur zu Smalltalk

/HAL87/ D. C. Halbert, P. D. O'Brien: Using types and inheritance in object-oriented languages. In /ECO87/, 20 - 31.

/HAR89/ R. Hartson: User-interface management control and communication; IEEE Software, vol. 6, January 1989, 62 - 70.

/HEN90/ B. Henderson-Sellers, J. M. Edwards: The object-oriented systems life cycle. In /ACM90/, 142 - 159.

/HMD89/ Themenheft *Objektorientierte Systementwicklung*; HMD, Handbuch der modernen Datenverarbeitung, 26. Jahrg., Heft 145, 1989.

/HOF87/ siehe allgemeine Literatur zu Smalltalk

/HÜT85/ S. Hütte, H. Streich: Systementwurf und Implementierung der virtuellen Maschine für Smalltalk-80; Diplomarbeit, Rheinische Friedrich-Wilhelms-Univ. Bonn, 1985.

/ING76/ D. Ingalls: The Smalltalk-76 programming system design and implementation; Proc. Principles of Programming Languages, 1978, 9 - 15.

/iX90/ Themenheft *OOPS unter UNIX, objektorientierte Programmiersysteme*; *iX* Multiuser-Multitasking-Magazin, Heft Januar/Februar 1990.

/JOH88/ R. Johnson, J. Graver, L. Zurawski; TS: An optimizing compiler for Smalltalk. In /OOP88/, 18 - 26.

/KAS89/ siehe allgemeine Literatur zu Smalltalk

/KAY69/ A. C. Kay: The reactive engine; Doctoral dissertation, Univ. of Utah, 1969.

/KEE88/ S. E. Keene: Object-oriented programming in COMMON LISP, a programmer's guide to CLOS; Addison-Wesley, 1988.

/KOR90/ T. Korson, J. D. McGregor: Understanding object-oriented: A unifying paradigm. In /ACM90/, 40 - 60.

/KRA83/ siehe allgemeine Literatur zu Smalltalk

/KRA88/ G. E. Krasner, S. T. Pope: A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80; J. Object-oriented Programming, vol. 1, August 1988, 26 - 49.

/KRE90/ W. Kreutzer: Grundkonzepte und Werkzeugsysteme objektorientierter Systementwicklung -- Stand der Forschung und Anwendung --. In /WIR90/, Seiten 211 - 227.

/LEH86/ O. Lehrmann Madsen: Block structure and object oriented languages. In /OOPa86/, 133 - 142.

/LIEa89/ K. J. Lieberherr, I. M. Holland: Assuring good style for object-oriented programs; IEEE Software, vol. 6, September 1989, 38 -48.

/LIEb89/ K. J. Lieberherr, A. J. Riel: Contributions to teaching object-oriented design and programming. In /OOP89/, 11 - 22.

/LIN88/ J. Lindskov Knudsen, O. Lehrmann Madsen: Teaching object-oriented programming is more than teaching object-oriented programming languages. In /ECO88/, 21 - 40.

/LUT87/ R. Lutze: Die Implementierung von Smalltalk. In /HOF87/, 129 - 188.

/MEYa88/ B. Meyer: Object-oriented software construction; Prentice Hall, 1988, ISBN 0-13-629049-3.

/MEYb88/ B. Meyer: Programming as contracting; Interactive Software Engineering, Inc., Goleta, TR-EI-12/CO, version 1, 1988.

/MIT89/ siehe allgemeine Literatur zu Smalltalk

/MYE86/ B. A. Myers: Visual programming, programming by example, and program visualization, a taxonomy; Proc. CHI-Conference, 1986, 59 - 66.

/OOPa86/ Proc. Object-oriented Programming Workshop; ACM SIGPLAN Notices, vol. 21, October 1986.

/OOPb86/ Proc. Object-oriented Programming Systems, Languages and Applications; ACM SIGPLAN Notices, vol. 21, November 1986.

/OOPa87/ Proc. Object-oriented Programming Systems, Languages and Applications; ACM SIGPLAN Notices, vol. 22, December 1987.

/OOPb87/ Addendum to Proc. Object-oriented Programming Systems, Languages and Applications; ACM SIGPLAN Notices, vol. 23, May 1988.

/OOP88/ Proc. Object-oriented Programming Systems, Languages and Applications; ACM SIGPLAN Notices, vol. 23, November 1988.

/OOP89/*1) Proc. Object-oriented Programming: Systems, Languages and Applications; ACM SIGPLAN Notices, vol. 24, October 1989.

/OOP90/ Proc. Object-oriented Programming: Systems, Languages and Applications bzw. Proc. European Conf. on Object-oriented Programming; ACM SIGPLAN Notices, vol. 25, October 1990.

/PFA85/ G. Pfaff (ed.): User interface management systems; Springer Verlag, 1985.

/PIN88/ siehe allgemeine Literatur zu Smalltalk

/PUG88/ J. R. Pugh: The future of Smalltalk; J. Object-oriented Programming, vol. 1, April/May 1988, 58 - 63.

/REE89/ T. Reenskaug, A. L. Skaar: An environment for literate Smalltalk programming. In /OOP89/, 337 - 345.

/RIC89/ J. Richter: Über eine Implementierung von Smalltalk auf einer Lisp-Maschine; Studienarbeit, Techn. Hochschule Darmstadt, FG Programmiersprachen und Übersetzer, 1989.

/RÜG90/ M. Rüger, A. Nyhuis, H. Guhl: Objects at work, Objectworks für Smalltalk-80 und C++. In /iX90/, 30 - 34.

/SAM86/ A. D. Samples, D. Ungar, P. Hilfinger: SOAR: Smalltalk without bytecodes. In /OOPb86/, 107 - 118.

/SCHa88/ L. Schmitz: Objektorientiert = gut?; PC-Magazin, 1988, Heft 51, Seiten 46 - 51 und Seite 101.

/SCHb88/ L. Schmitz: Ein bißchen Smalltalk; PC-Magazin, 1988, Heft 52, Seiten 52 - 58.

/SCH89/ L. Schmitz: Smalltalk für den PC; PC-Magazin, 1989, Heft 1, Seiten 51 - 57.

*1) Die Bedeutung der Abkürzung OOPSLA hat sich ab 1989 geändert.

/SCHa90/ R. Schulz: Das Obskure, objektorientierte Programmierung mit der Smalltalk-80-'Entwicklungsumgebung'. In /iX90/, 42 - 45.

/SCHb90/ L. Schmitz: Wiederverwendbarkeit von Software -- eine Fallstudie anhand von Ada und Smalltalk; Informatik-Spektrum, Band 13, Heft 2, 1990, 71 - 85.

/SNY86/ A. Snyder: Encapsulation and inheritance in object-oriented programming languages. In /OOPb86/, 38 - 45.

/STO89/ H. Stoyan: Objektorientierte Systementwicklung. In /HMD89/, 3 - 12.

/STR86/ B. Stroustrup: The C++ programming language; Addison-Wesley, 1986, ISBN 0-201-12078-X.

/TOO89/ Proc. Technology of Object-oriented Languages and Systems, 1989.

/TOO90/ Proc. Technology of Object-oriented Languages and Systems, 1990, im Druck.

/WAG90/ W. Wagemann: A fast code generator for typed Smalltalk; Diplomarbeit, Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign und Techn. Hochschule Darmstadt, FG Programmiersprachen und Übersetzer, 1990.

/WAS87/ A. I. Wasserman, P. A. Pircher: A graphical, extensible integrated environment for software development; ACM SIGPLAN Notices, vol. 22, Januar 1987, 131 - 142.

/WAS90/ A. I. Wasserman, P. A. Pircher, R. J. Muller: The object-oriented structured design notation for software design representation; IEEE Computer, vol. 23, März 1990, 50 - 63.

/WEB85/ B. Webster, T. Yonkman: Methods, a preliminary look; BYTE, vol. 10, March 1985, 152 - 154.

/WED90/ H. Wedekind: Objektorientierung und Vererbung; Informationstechnik it, 32. Jahrg., April 1990, 79 - 86.

/WEG86/ P. Wegner: Classification in object-oriented systems. In /ACMa86/, 173 - 182.

/WEG87/ P. Wegner: Dimensions of object-based language design. In /ACMa87/, 168 - 182.

/WEG89/ P. Wegner: Learning the language. In /BYT89/, 245 - 253.

/WEG90/ P. Wegner: Concepts and paradigms of object-oriented programming; ACM OOPS Messenger, vol. 1, no. 1, August 1990, 7 - 87.

/WIRa90/ Themenheft *Objektorientierte Anwendungssysteme und Systemsoftware für die 90er Jahre*; Wirtschaftsinformatik, 32. Jahrg., Heft 3, 1990.

/WIRb90/ R. J. Wirfs-Brock, R. E. Johnson: Surveying current research in object-oriented design. In /ACM90/, 104 - 124.

/YOK87/ Y. Yokote, M. Tokoro: Experience and evaluation of concurrent Smalltalk. In /OOPa87/, 406 - 415.

Allgemeine Literatur über Smalltalk

Bei den ersten drei Büchern handelt es sich um die "Klassiker". In ihnen wurde, abgesehen von /BYT81/, zum ersten Mal Smalltalk und das Produkt *Smalltalk-80* einer breiten Öffentlichkeit vorgestellt. Die Bücher sind sehr ausführlich. Sie gelten in Anbetracht der komplexen, technisch orientierten Materie als sehr gut geschrieben.

/GOL83/ A. Goldberg, D. Robson: Smalltalk-80: The language and its implementation; Addison-Wesley, 1983, ISBN 0-201-11371-6.

/GOL84/ A. Goldberg: Smalltalk-80: The interactive programming environment; Addison-Wesley, 1984, ISBN 0-201-11372-4.

/KRA83/ G. Krasner (ed.): Smalltalk-80: Bits of history, words of advice; Addison-Wesley, 1983, ISBN 0-201-11669-3.

Die Implementierung von Smalltalk durch die Firma *Digitalk*, als *Smalltalk/V* bezeichnet, führte zur Verbreitung von Smalltalk auch auf den IBM und IBM-kompatiblen PC/DOS-Rechnern. Auch dieses Buch (wie die entsprechenden Bücher für die anderen Implementierungen) ist eingängig geschrieben, manualartig.

/DIG86/ NN (Digitalk, Inc.): Smalltalk/V, Tutorial and programming handbook; Digitalk, Inc., 1986 (Firmenschrift ohne ISBN-Nummer).

Das Buch zu der "kleinen" Implementierung; für denjenigen, der sich nur über das Wesentliche informieren möchte, sehr zu empfehlen.

/BUD87/ T. Budd: A little Smalltalk; Addison-Wesley, 1987, ISBN 0-201-10698-1.

Nun kommen einige nicht auf eine spezielle Implementierung, sondern mehr auf Sprache und Programmierung bezogene Bücher. /PIN88/ hat sehr gut durchgearbeitete Beispiele. Bei /MIT89/ soll man sich an dem Hinweis auf C++ nicht allzu sehr stören, etwa 175 Seiten über Smalltalk stehen 35 Seiten über C++ gegenüber. Ich mußte hier eine Auswahl treffen, es gibt noch weitere Bücher dieses Genre.

/PIN88/ L. J. Pinson, R. S. Wiener: An introduction to objectoriented programming and Smalltalk; Addison-Wesley, 1988, ISBN 0-201-19127.

/MIT89/ J. Mittendorfer: Objektorientierte Programmierung mit C++ und Smalltalk; Addison-Wesley, 1989, ISBN 3-89319-165-8.

/KAS89/ E. Kasper, R. Kasper: Objektorientiertes Programmieren in Smalltalk; Vogel Buchverlag, 1989, ISBN 3-8023-0237-0.

Schließlich noch der Hinweis auf ein Buch, das anläßlich des vom *German Chapter of the ACM* im Jahr 1985 in Zusammenarbeit mit der Fachgruppe *Interaktive Systeme* der GI in

Darmstadt veranstalteten *Tutoriums über Smalltalk* entstand.

/HOF87/ H.-J. Hoffmann: Smalltalk verstehen und anwenden;
Hanser-Verlag, 1987, ISBN 3-446-14556-7.

Glossareinträge

Adressat
Adressat, ein --> *Objekt*, ist der Empfänger einer --> *Botschaft*, die er aufgrund seines --> *Protokolls* annehmen und bearbeiten kann. Der Begriff *"Adressat"* ist verwandt mit dem Begriff *"Prozedur-/Funktionsname"* einer algorithmischen, höheren Programmiersprache.

als Parameter dienendes Objekt
einer --> *Botschaft* mitgegebenes --> *Objekt*, durch das die Botschaft näher charakterisiert wird. Der --> *Adressat* der Botschaft wird (im Regelfall) das Objekt als Adressat einer von ihm während der Ausführung der --> *Methode*, die die betrachtete Botschaft bearbeitet, ausgesandten neuen Botschaft (oder dort wieder als Parameter) heranziehen.

Ausprägen eines Objekts
Zum --> *Handlungsaspekt* eines --> *Klassenobjekts* gehörende operative Aufgabe, ein neues --> *Objekt* entsprechend der --> *Klassenbeschreibung* (technisch) für den Programmablauf herzustellen.

Botschaft
Programmtechnisches Mittel zur Weitergabe einer operativen Aufgabe während der Ausführung einer --> *Methode* an ein anderes --> *Objekt*, den --> *Adressaten*; der Botschaft können --> *als Parameter dienende Objekte* mitgegeben werden. Der Adressat muß in seinem --> *Protokoll* eine --> *Methode* haben, die die Botschaft annehmen und bearbeiten kann. Der Begriff *"Botschaft"* ist verwandt mit dem Begriff *"Funktions-/Prozeduraufruf"* algorithmischer, höherer Programmiersprachen.

Botschaftenmuster
strukturelle und notationelle Festlegung, wie eine --> *Botschaft* aufzubauen bzw. zu formulieren ist, vergleichbar dem *"Funktions-/Prozedurkopf"* (bzw. der *"Signatur"*) in algorithmischen, höheren Programmiersprachen. Das Botschaftenmuster legt insbesondere den --> *Selektor* fest, der im --> *Protokoll* des --> *Adressaten* die zugehörige --> *Methode* identifiziert.

browser (--> *Stöberer*)

Bytecode
Befehl der --> *virtuellen Maschine*

Datenaspekt
Zusammenfassung der technischen Eigenschaften, die als Datenstruktur den Zustand eines --> *Objekts* ausmachen, bzw. der beschreibenden Eigenschaften (*"Attribute"*) des Objekts aus Sicht der Anwendung/Benutzung. Infolge von

--> *Ererbungsbeziehungen* kann sich der Datenaspekt aus verschiedenen --> *Klassenbeschreibungen* ergeben.

debugger (--> *Testhilfe*)

direktmanipulativ
Umschreibung der (für Smalltalk typischen) Vorgehensweise bei (der Programmierung und) der Benutzung durch Ausdeuten und Einwirken auf die am Bildschirm visualisierten --> *Objekte* der --> *Quasirealität*.

einfache Ererbung
--> *Ererbungsbeziehung* unter --> *Klassen*, bei der eine "Unter"-klasse nur genau eine "Ober"-klasse hat. Es bildet sich eine baumartige --> *Klassenhierarchie* aus.

Ererbungsbeziehung
Beziehung unter --> *Klassen von Objekten*, mit der gemeinsame Objekteigenschaften (und zwar bezüglich des --> *Daten-* wie des --> *Handlungsaspekts*) zusammengefaßt und von einer "Unter-"klasse aus einer "Ober"-klasse (bzw. deren Beschreibungen) entnommen werden können.

Exemplar (--> *Objekt*)

Exemplarvariable
Mittel zur Repräsentation und Speicherung des Zustands eines --> *Objekts* (oder eines Teils davon, --> *Datenaspekt*), vergleichbar einer lokalen Variablen bzw. eines Felds in einem Datensatz bei algorithmischen, höheren Programmiersprachen. Exemplarvariable können (kumulierend) --> *ererbt* sein.

Handlungsaspekt
Zusammenfassung des operativen Verhaltens eines --> *Objekts*, realisiert in den --> *Methoden*, aufgrund derer es --> *Botschaften* annimmt und bearbeitet, bei denen es --> *Adressat* ist; aus Benutzungssicht die --> *direktmanipulativen* Einwirkungsmöglichkeiten an der --> *visualisierten Quasirealität*. Methoden können (kumulierend) --> *ererbt* sein.

inspector (--> *Inspektor*)

Inspektor (engl. *inspector*)
zum Programmier- und Benutzungssystem (der *"reagierenden Maschinerie"* von Smalltalk) gehörende Systemkomponente, mit der der Programmierer/Benutzer jedes --> *Objekt* hinsichtlich seines --> *Daten-* und seines --> *Handlungsaspekts* untersuchen kann. Diese Komponente dient insbesondere als Werkzeug beim Testen eines Programmablaufs.

Instanz (--> *Objekt*)

Klasse von Objekten
Zusammenfassung von --> *Objekten*, deren Eigenschaften in gleicher Weise in einer --> *Klassenbeschreibung* festgelegt sind.

Klassenbeschreibung
Verfahren der Beschreibung einer --> *Klasse von Objekten*. Zur Beschreibung gehört die Einordnung in die --> *Klassenhierarchie*, die Angabe der --> *Exemplarvariablen* der --> *Objekte*, die aus der Klasse --> *ausgeprägt* werden können (--> *Datenaspekt*), und die Angabe der --> *Methoden*, die zur Verfügung stehen (--> *Handlungsaspekt*), wenn ein zur Klasse gehörendes Objekt --> *Adressat* einer --> *Botschaft* ist. In Smalltalk bildet die Beschreibung der zur Klasse gehörenden Objekte und die Beschreibung des --> *Klassenobjekts* (zugehörig zu der entsprechenden --> *Metaklasse*) eine textuelle Einheit. Es gibt Zusammenhänge zwischen einer Klassenbeschreibung und einer Typdefinition einer algorithmischen, höheren Programmiersprache.

Klassenhierarchie
Einordnung von --> *Klassen von Objekten* aufgrund der --> *Ererbungsbeziehung* (genaugenommen liegt in Smalltalk, trotz der allein vorgesehenen --> *einfachen Ererbung*, aus bestimmten, hier nicht erklärten Umständen keine vollständige Hierarchie vor).

Klassenobjekt
das --> *Objekt*, das --> *Adressat* von --> *Botschaften* ist, die an eine --> *Klasse von Objekten* gerichtet sind, insbesondere der Botschaft, mit der ein neues Objekt in die Klasse aufgenommen wird (--> *Ausprägen eines Objekts*). In vielen Fällen wird Klassenobjekt und --> *Klassenbeschreibung* synonym gebraucht.

Klassensystem
Gesamtheit der --> *Klassen von Objekten*, die eine --> *virtuelle Welt* bilden, auch die Beschreibung aller derartigen Klassen.

Little Smalltalk
eine (vergleichsweise) einfach gehaltene Smalltalk-Implementierung von T. Budd /BUDa87/.

Mehrfachererbung
--> *Ererbungsbeziehung* unter --> *Klassen von Objekten*, bei der eine "Unter"-klasse mehrere "Ober"-klassen haben kann. Es bildet sich eine netzartige --> *Klassenhierarchie* aus.

Metaklasse
--> *Klassenobjekt* (mit zugehöriger --> *Klassenbeschreibung*) von --> *Klassen von Objekten* bzw. von dem diese er-

fassenden --> *Klassenobjekt*.

Methode
einen Teil des --> *Handlungsaspekts* eines --> *Objekts* ausmachende operative Einheit, vergleichbar einer Funktion bzw. Prozedur in algorithmischen, höheren Programmiersprachen. Zum Annehmen und Bearbeiten einer --> *Botschaft*, die dem Objekt als --> *Adressat* zugeht, muß es in seinem --> *Protokoll* eine entsprechende Methode aufweisen. Die Gesamtheit der durch das Protokoll erfaßten Methoden des Objekts bestimmt seinen Handlungsaspekt ausschließend und vollständig. Bei der Programmierung einer Methode (in der --> *Klassenbeschreibung*) wird im Methodenkopf das --> *Botschaftenmuster* und die --> *als Parameter dienenden Objekte* sowie der Methodenrumpf festgelegt, der aussagt, in welcher Weise operativ (durch Aussenden neuer, weiterer Botschaften) in der betrachteten Methode vorgegangen werden muß.

Methodensuche
Verfahren, mit dem für eine --> *Botschaft* (der --> *Adressat* und) die --> *Methode* zu ihrer Bearbeitung gesucht wird. Methodensuche berücksichtigt --> *Ererbungsbeziehungen* (kumulierende Behandlung des --> *Handlungsaspekts*).

MVC-Modell
Entwurfsvorstellung, nach der --> *Objekten* (der --> *visualisierten Quasirealität*, *"model"*) eine Sichtsteuerungskomponente (*"view"*) und eine Sensorkomponente (*"control"*) technisch zugeordnet wird (letztere immer paarweise auftretend).

Objectworks
Weiterentwicklung von --> *Smalltalk-80*.

Objekt (aus Sicht der Anwendung/Benutzung)
Objekt ist ein Gegenstand / eine Gegebenheit der --> *virtuellen Welt*, in der die betrachtete Anwendungsaufgabe zur Bearbeitung durch den Rechner modelliert wird, und der/die in Gesamtheit das einem Benutzer individuell zur Verfügung stehende Programmier- und Benutzungssystem ausmachen. Bzw. (in der --> *visualisierten Quasirealität*) ein Darstellungselement, an dem der Benutzer die von ihm beabsichtigte Einwirkung in den Ablauf des Modells --> *direktmanipulativ* manifestieren kann. Ein Objekt ist durch beschreibende Eigenschaften (*"Attribute"*, --> *Datenaspekt*) und durch die Einwirkungsmöglichkeiten (--> *Handlungsaspekt*) entsprechend seinem --> *Protokoll* charakterisiert.

Objekt (aus technischer Sicht)
Objekt ist die der Realisierung von operativen Einheiten zugrundeliegende Datenstruktur, vergleichbar einer *Variablen* einer algorithmischen, höheren Programiersprache.

Ein Objekt hat einen internen Zustand (--> *Datenaspekt*) und verfügt über Methoden (--> *Handlungsaspekt*), die sein operatives Verhalten ausmachen.

Objektpersistenz
die Eigenschaft einer --> *virtuellen Welt*, daß die in ihr --> *ausgeprägten Objekte* ihren Zustand (--> *Datenaspekt*) beibehalten, auch wenn der Programmierer / Benutzer vorübergehend den Umgang mit dem Programmier- und Benutzungssystem einstellt (d.h. z.B. seinen Rechner für eine andere Arbeit verwendet).

Objektvariable (--> Exemplarvariable)

Protokoll
den --> *Handlungsaspekt* eines --> *Objekts* beschreibende Zusammenfassung aller --> *Methoden* (auch der --> *ererbten*), die bei der --> *Methodensuche* zur Verfügung stehen, wenn das Objekt als --> *Adressat* in einer --> *Botschaft* auftritt. Es sind (im wesentlichen) Angaben der Methodenköpfe (--> *Methode*, --> *Botschaftenmuster*) aufgenommen.

(visualisierte) Quasirealität
die am Bildschirm dargestellten (d.h. visualisierten) --> *Objekte* (Ausschnitt der --> *virtuellen Welt*) als Gegenstände / Gegebenheiten, an denen zur --> *direktmanipulativen* Einwirkung durch den Benutzer die betrachtete Anwendungsaufgabe und Fortschritte bei ihrer Behandlung zur Bearbeitung durch den Rechner modelliert werden.

Schere (engl. *stripper*)
zu Zusätzen von --> *Smalltalk-80* bzw. --> *Smalltalk/V* gehörendes Hilfsmittel zum Zurechtlegen eines fertiggestellten Anwendungsprogramms zur Auslieferung (ohne dem Benutzer die Systemkomponenten zum Weiterentwickeln des Programms mitzugeben).

Selektor
im --> *Botschaftenmuster* getroffene Festlegung (z.B. durch ein Operat*ionszeiche*n oder ei*n Wortsymbo*l), mit der im --> *Protokoll* eines --> *Adressaten* die zugehörige *Methode* bei der --> *Methodensuche* identifiziert wird.

Smalltalk
(objektorientierte) Programmiersprache und Programmier-/Benutzungssystem, definiert in /GOL83/ bzw. /GOL84/.

Smalltalk-80
Implementierung von --> *Smalltalk* (durch XEROX bzw. ParcPlace Systems).

Smalltalk/V
Implementierung von --> *Smalltalk* (durch Digitalk, /DIG86/).

Stöberer (engl. *browser*)
zum Programmier- und Benutzungssystem (der *"reagierenden Maschinerie"* von Smalltalk) gehörende Systemkomponente, mit der der Programmierer/Benutzer im --> *Klassensystem* (u.ä., z.B. einem Dateikatalog) herumsuchen (*"herumstöbern"*) kann.

stripper (--> *Schere*)

Systemprimitive
Menge von --> *Methoden*, die zwar im --> *Klassensystem* eingeordnet, aber in den --> *Klassenbeschreibungen* nicht programmiert sind. Sie werden durch die --> *virtuelle Maschine* realisiert.

Testhilfe (engl. *debugger*)
zum Programmier- und Benutzungssystem (der *"reagierenden Maschinerie"* von Smalltalk) gehörende Systemkomponente, mit der der Programmierer/Benutzer nach Auftreten eines Fehlers oder einer Sondersituation die Ursache (z.B. auch durch Heranziehen eines --> *Inspektors*) ermitteln kann. Die Testhilfe wird (beispielsweise) automatisch bei der --> *Methodensuche* angesprochen, wenn für eine --> *Botschaft* sich kein --> *Adressat* findet, der sie aufgrund seines --> *Protokolls* annehmen und bearbeiten kann.

Vererbungsbeziehung (--> *Ererbungsbeziehung*)

virtual image (--> *virtuelle Welt*)

virtual machine (--> *virtuelle Maschine*)

virtuelle Maschine
(abstrakte, d.h. programmierte) Maschine, die die Speicherung der --> *Exemplarvariablen* aller --> *ausgeprägten Objekte* organisiert und --> *Systemprimitive* als Befehle annimmt; sie realisiert auch das Zusenden von Botschaften an ihre --> *Adressaten* und die --> *Methodensuche*.

virtuelle Welt
Dem Benutzer sich (infolge der --> *Objektpersistenz*) überdauernd darbietende Gesamtheit der --> *Objekte*, die die betrachtete Anwendungsaufgabe modellieren und die das dabei dem Benutzer individuell zur Verfügung stehende Programmier- bzw. Benutzungssystem ausmachen.