Joachim Tuchel

Lebendes Objekt

Smalltalk: ein aktueller Klassiker

Sprachen wie Java und C++ haben die Objektorientierung zum derzeit wohl populärsten Programmieransatz aufsteigen lassen. Von der 'reinen Lehre' sind sie aber ein ganzes Stück entfernt. Wer die erleben will, sollte mal einen Blick auf die Sprache Smalltalk werfen: Aktuelle Versionen machen sie zu einem mächtigen Werkzeug für Projekte aller Art, mit dem man nebenbei auch noch eine Menge Spaß haben kann.

In der modernen Softwareentwicklung hat sich der objektorientierte Ansatz auf breiter Front durchgesetzt. Entwicklern verspricht er vor allem die Chance, einmal implementierten Code problemlos wiederzuverwenden. Auch Anwender ziehen einen Nutzen aus der Objektorientierung, denn sie unterstützt besser als andere Ansätze einen iterativen Entwicklungsprozess, der den Endanwender eng in die Arbeit einbindet. Die Produktivitätsvorteile objektorientierter Techniken im Vergleich zur klassischen strukturierten Analyse und Programmierung sind inzwischen unbestritten.

Großen Wirbel hat im Siegeszug der Objektorientierung die Sprache Java verursacht, die sich – glaubt man dem ersten Augenschein – als Standardsprache für Projekte jeder Art und Größe etabliert hat. Dabei entwickelt sich gerade diese Plattform inzwischen zu einer sehr komplexen und schwer beherrschbaren Umgebung. Spätestens seit Einführung der J2EE und Enterprise Java Beans gehen viele Vorteile der objektorientierten Konzepte verloren, und die Komplexität relativ einfacher Projekte steigt unnötig stark an.

Dabei nahm es mit der Objektorientierung einen ganz anderen Anfang: Anfang der siebziger Jahre startete ein Team im Xerox PARC (Palo Alto Research Center) mit dem Ansatz, die Handhabung und Programmierung von Computern so weit zu vereinfachen, dass selbst Kinder sie bewerkstelligen können. Im Zuge dieses Projektes entstanden einige revolutionäre Konzepte: der erste tragbare Tablet PC (Dynabook), das erste grafische Benutzersystem mit der Idee sich überlappender Fenster, die Maus als Bediengerät – und die Objektorientierung samt einer Implementierungssprache: Smalltalk.

Die wichtigste Idee hinter Smalltalk war es von Anfang an, Komplexität zu vermeiden. Der Entwickler sollte sich stets mit dem fachlichen Inhalt seines Problems beschäftigen, anstatt sich mit technischen Details herumschlagen zu müssen. Als tragisches Moment in der Geschichte von Smalltalk kann wohl die Tatsache gelten, dass kein Geringerer als Steve Jobs von der jungen Firma Apple für ein neues Projekt namens Lisa großes Interesse daran hatte, das Smalltalk-System zu lizenzieren, aber Xerox nicht bereit war, an Jobs zu verkaufen. Zweifelsohne würde die EDV-Welt heute ganz anders aussehen, wenn er damals Erfolg gehabt hätte.

Gegen einen breiten Einsatz der Sprache Smalltalk sprachen lange Zeit einige ihrer zentralen und damals bahnbrechenden Konzepte: Smalltalk bringt eine grafische Oberfläche mit, die für die damalige Zeit sehr hohe Anforderungen an die Hardware stellte. Dazu kam die völlig neue Idee, eine so genannte virtuelle Maschine einzusetzen, um die technischen Details der Hardwareplattform vor dem Entwickler zu verbergen. Auch eine Speicherverwaltung, die sich automatisch um die Freigabe nicht mehr benötigter Speicherbereiche kümmert (Garbage Collection), belastet den Rechner stärker als die herkömmlichen Techniken. Angesichts der Tatsache, dass selbst zehn Jahre nach Entwicklung der Sprache Smalltalk so genannte Mini- und Personal Computer mit Taktraten von unter 3 Megahertz und mit gerade mal 64 Kilobyte Speicher gang und gäbe waren, ist es durchaus verständlich, dass Smalltalk auch heute noch mit dem Vorurteil zu kämpfen hat, es sei ein Speicher- und Performance-Fresser.

In Zeiten Gigahertz-getakteter Multimedia-PCs mit grafischer Bedienoberfläche und mehreren hundert Megabyte Speicher hat sich diese Einschätzung allerdings überholt. Im Gegenteil wird heutzutage der Einsatz virtueller Maschinen als großer Vorteil zum Beispiel der Java-Plattform oder von Microsofts .NET-Umgebung gepriesen – nicht zuletzt aufgrund der Tatsache, dass sich dadurch die Unterschiede zwischen verschiedenen Hard- und Softwareplattformen ausgleichen lassen, ohne das Anwenderprogramm an die je-

weilige Umgebung anpassen zu müssen.

Nostalgie oder Kostenvorteile?

Wozu sollte man sich heutzutage noch mit einer fast dreißig Jahre alten Programmiersprache beschäftigen? Die Antwort auf diese Frage liegt eigentlich auf der Hand: Es gibt nur wenige Techniken auf dem IT-Markt, die so viel Zeit hatten zu reifen.

In verschiedenen Untersuchungen wurde gezeigt, dass sich mit Smalltalk deutlich effizienter entwickeln lässt als mit anderen Programmiersprachen. Ein Entwickler kann ein gegebenes Problem schneller und mit weniger Quellcode lösen. Typischerweise ist eine Aufgabenstellung in Smalltalk mit etwa der Hälfte oder gar weniger der Code-Menge zu lösen, die in C++ oder Java nötig wäre. Außerdem gibt es Studien, die Smalltalk-Projekten eine wesentlich kürzere Time-to-Market bescheinigen [1].

Eine andere Studie zeigt auf, dass in Smalltalk-Programmen weniger Laufzeitfehler auftreten: Ein Smalltalk-Entwickler erzeugt bei der Implementierung derselben Funktionalität nur ein Drittel der Fehler, die einem Java-Entwickler unterlaufen, und nur ein Sechstel der Fehler eines C++-Entwicklers. Die Autoren der Untersuchung verglichen verschiedene Programmiersprachen nach Kosten pro Function Point und nach der Anzahl von Defekten. Dabei schnitt Smalltalk um den Faktor zwei bis drei besser als Java oder C++ ab. (Die Ergebnisse dieser Studie waren lange Zeit online verfügbar, können aber inzwischen leider nur noch gegen Kostenbeitrag bei der Firma SPR (www.spr.com/products/programming.htm) bestellt werden.) Dahinter steckt eine sehr einfache Rechnung: Weniger Code enthält weniger Fehler und verlangt weniger Pflegeaufwand.

Gerade die Pflege einer Anwendung sollte nicht außer Acht gelassen werden. Schon Fred Brooks [2] erkannte 1974, dass der Großteil der Kosten eines Projekts in der Pflegephase, also in der Zeit nach der Entwicklung, entsteht. Die eigentlichen Entwicklungskosten betragen nur etwa 20 Prozent der Gesamtkosten, die ein System über seine Lebenszeit hinweg verursacht. Angesichts dieser Erkenntnis zeigt sich, dass sich die Wahl einer Implementierungssprache sehr schnell bezahlt macht, die insbesondere bei der Fehlerträchtigkeit entscheidende Vorteile bietet.

Der Schlüssel zum Geheimnis

Ein Mensch kann sich maximal fünf bis sieben Dinge gleichzeitig merken, ohne durcheinander zu kommen, und folgert aus bestimmten Mustern auf neue Situationen. Mit nur fünf reservierten Schlüsselwörtern ist Smalltalk sehr schnell zu erlernen. Zudem ist die Sprache völlig konsistent, das heißt, es gibt keine Ausnahmen zu ihren Grundregeln.

C++ und Java sind im Vergleich dazu geradezu gespickt von solchen Ausnahmen. C++ kennt beispielsweise 50 reservierte Wörter, und beide Sprachen unterscheiden zwischen Objekten und Basistypen (zum Beispiel Integer vs. int). Obgleich solche Unterschiede durchaus beherrschbar sind, machen sie die Softwareentwicklung unnötig komplex. Die Folge ist, dass ein Java-Programmierer nie weit entfernt von einer Online- oder Offline-Dokumentation anzutreffen sein wird.

In Smalltalk ist alles ein Objekt. Selbst Klassen sind Objekte, mit denen zur Laufzeit interagiert werden kann. Sämtliche Operationen bestehen aus einer Meldung an ein Objekt. Es gibt keine syntaktischen Schleifen- oder Bedingungskonstrukte, sondern sie werden durch entsprechende Meldungen abgebildet. Für erfahrene C++- oder Java-Entwickler ist dies zunächst ungewohnt, einem Einsteiger in objektorientierte Programmierung erleichtert es den Anfang jedoch sehr.

Die Konsistenz der Sprachkonzepte in Smalltalk und die sehr einfache Syntax verkürzen die Lernkurve der Sprache enorm und ermöglichen es, sehr schnell effektiv an fachlich orientierten Lösungen zu arbeiten, anstatt sich mit verschiedenen Paradigmen herumzuschlagen.

In Java, C++ und allen anderen statisch typisierten Sprachen muss der Typ eines Objektes bereits zur Kompilierzeit bekannt sein. Dadurch stellt der Compiler schon beim Übersetzen sicher, dass es zur Laufzeit nicht zu Typfehlern ('Type Mismatch') kommt. Somit sind zur Laufzeit keine Überprüfungen von Objekttypen mehr notwendig. Das bringt allerdings auch Probleme mit sich. So finden sich in typischen C++- und Java-Anwendungen zahlreiche Typecasting-Operationen, die Typprüfungen zur Kompilierzeit

Die Smalltalk-Syntax auf einen Blick

Die Syntax der Sprache Smalltalk ist extrem einfach, und ihre Definition passt tatsächlich in diesen kleinen Kasten. Es gibt keine speziellen Konstrukte etwa zur Klassendefinition oder zur Erzeugung neuer Instanzen. Diese sind einfache Methoden einer Klasse oder ihrer Oberklassen und stellen deshalb keine Syntax-Elemente der Sprache dar.

Reservierte Schlüsselwörter

Grundsätzlich kann ein Methodenname aus jeder beliebigen Folge alphanumerischer Zeichen ohne Sonderzeichen bestehen. Es existieren gerade mal fünf reservierte Schlüsselwörter, deren Bedeutung systemweit eindeutig und nicht zu verändern ist:

self: Schlüsselwort, um sich auf das aktuelle ausführende Objekt zu beziehen und Methoden daran aufzurufen (vergleichbar this in Java oder C++).

super: Wie self, nur soll dabei mit der Suche nach der auszuführenden Methode in der Superklasse des Objekts begonnen werden. Damit kann eine Methode der Oberklasse unverändert beibehalten und um spezifische Erweiterungen ergänzt werden, ohne die Methode der Oberklasse zu kopieren.

true: Repräsentiert die einzige Instanz der Klasse True, die einen wahren Boolean-Wert darstellt.

false: Ist die einzige Instanz (beziehungsweise deren systemweiter Name) der Klasse False, die im Gegensatz zu true einen unwahren Wert darstellt.

nil: Steht für die einzige Instanz der Klasse UndefinedObject und damit für einen undefinierten Wert (vergleichbar null in Java). Variablen enthalten vor ihrer Initialisierung stets den Wert nil.

Unveränderliche Operatoren

Smalltalk kennt grundsätzlich keine Operatoren, wie dies in anderen Sprachen der Fall ist. So sind mathematische Operatoren wie +, –, * oder / ganz normale Methoden, die ein Entwickler jederzeit überschreiben kann (man sollte sich der Konsequenzen bewusst sein …). Lediglich zwei Operatoren sind in Smalltalk fix und können nicht verändert werden:

:= dient der Zuweisung von Werten an eine Variable (a := 2).

^ dient der Rückgabe eines Wertes aus einer Methode (vgl. return in Java oder C++). Gibt der Entwickler nicht explizit einen Wert zurück, ist das Ergebnis einer Methode stets das Empfängerobjekt.

Sonstige Syntaxelemente

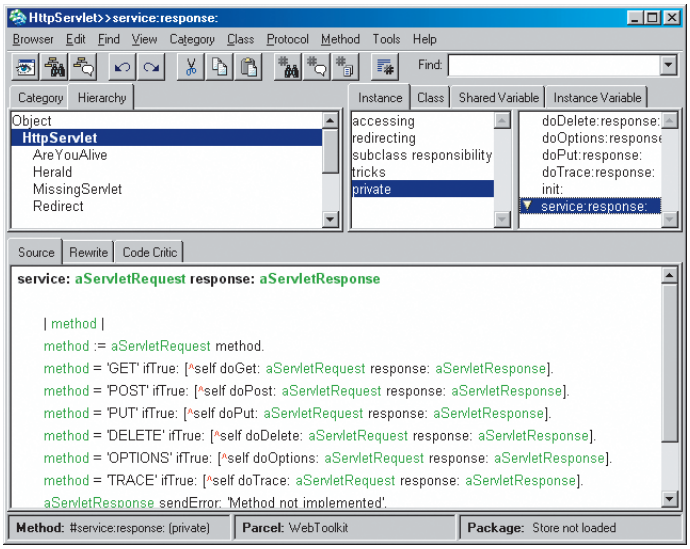
. Der Punkt beendet eine Anweisung. Das in anderen Sprachen für diesen Zweck verwendete Semikolon kennzeichnet bei Smalltalk so genannte kaskadierte Operationen, bei denen dasselbe Objekt nacheinander mehrere Meldungen geschickt bekommt.

" " Kommentare werden in doppelte Hochkommata gestellt und können überall im Quelltext eingestreut werden.

| | Zwischen Pipe-Symbolen werden temporäre Variablen definiert. Zum Beispiel definiert |zaehler| eine temporäre Variable namens zaehler.

aushebeln. Im Laufe eines Projekts treten in der Regel allerlei neue und geänderte Anforderungen auf, die dazu führen können, dass sich der Typ eines bestimmten Objekts ändert. Wenn so etwas passiert, muss der Entwickler – oder im schlimmsten Fall jedes Mitglied eines Programmiererteams – seinen Code nach Casts auf diesen Typ durchflöhen und die entsprechenden Stellen anpassen. Fehler bei derartigen Umbauten können fatale Folgen für das System haben, denn nicht selten treten sie erst zur Laufzeit zutage.

In Smalltalk ist der Typ eines Objekts nicht entscheidend. Im Mittelpunkt steht das Protokoll, das ein Objekt unterstützt, also die Menge von Meldungen, die es versteht. Somit ist es möglich, über eine Collection heterogener Objekte zu iterieren und ein und dieselbe Meldung an all diese Objekte zu schicken, solange sie nur alle dieselbe Methode implementieren. In C++ oder Java müssten all diese Objekte vom selben Typ abstammen oder auf denselben Typ gecastet werden, um dieselbe Funktion zu implementieren. Alleine dieses 'Verbiegen' von Objekten auf einen bestimmten Typ bedarf einer Erfolgskontrolle sowie entsprechender Maßnahmen (etwa eines try-catch-Konstrukts oder einer if-Abfrage) für den Fall, dass ein Objekt nicht erfolgreich uminterpretiert werden kann. Dies an sich stellt schon eine potenzielle Fehlerquelle dar, die ein Smalltalker gar nicht kennt.



Das zentrale Werkzeug einer Smalltalk-Entwicklungsumgebung ist der Class Browser zum Bearbeiten der Quelltexte.

Entwicklungswerkzeuge

Ein grundsätzlicher Unterschied zwischen Smalltalk und anderen Entwicklungswerkzeugen ist das Konzept des so genannten Image. Dieser abgeschlossene Raum enthält zur Entwicklungs- wie zur Laufzeit sämtliche Klassen und Objekte, die im Gesamtsystem verfügbar sind. Dadurch kann man auf jedes Objekt jederzeit zugreifen, kann es modifizieren und Methoden daran aufrufen. Der Vergleich mag brutal klingen, aber ein Smalltalk-Entwickler ist wie ein Chirurg, der direkt am offenen Herzen operiert – ohne den Patienten zu betäuben. Klassische Compile- und Link-Zyklen existieren nicht, das System kompiliert inkrementell (just in time), wenn der Entwickler eine Methode speichert.

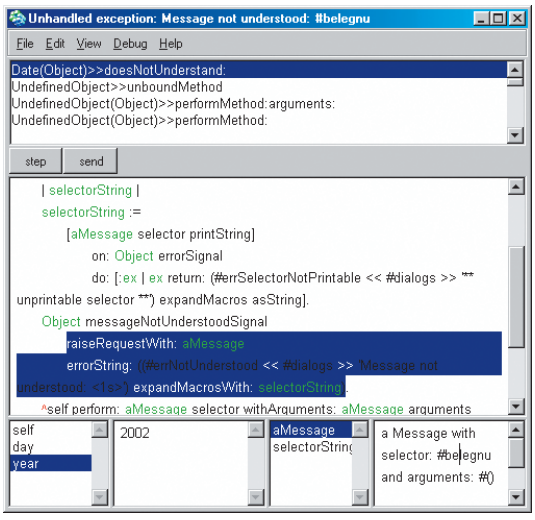
Um die Anzahl zunächst verwirrender Eigenschaften noch ein wenig zu erhöhen, sei hier noch erwähnt, dass die meisten Smalltalk-Systeme komplett in Smalltalk implementiert sind. Das bedeutet zum Beispiel, dass es möglich ist, die Klassenbibliothek und sogar die Entwicklungsumgebung selbst jederzeit anzupassen und zu erweitern – entsprechende Erfahrung und Courage vorausgesetzt. Zudem ist die gesamte Klassenbibliothek im Quelltext verfügbar und kann so durchforscht und gegebenenfalls angepasst werden. Es ist also typischerweise nicht nötig, dicke Bücher auf dem Schreibtisch zu haben, die die grundlegenden Funktionen der Basisklassen erklären: Ist die Arbeitsweise einer Methode unklar, sieht man sie sich an.

Mit den in Smalltalk-Umgebungen existierenden Werkzeugen zum Debuggen kann man die Ausführung des Codes an jeder Stelle unterbrechen und Schritt für Schritt anhand des Quelltextes analysieren. Dabei ist der komplette Zustand sämtlicher Objekte verfügbar – das System läuft ja gerade. Es ist durchaus gängige Praxis, eine Methode während ihrer Ausführung anzuhalten, zu modifizieren und sofort die neue Version weiter auszuführen. Im Unterschied zu anderen Entwicklungssprachen bietet Smalltalk hier auch den Vorteil, dass man sich bei der Fehlersuche jedes andere Objekt im System zu jeder Zeit ansehen und es verändern kann, man ist nicht auf den Inhalt des Execution Stack begrenzt. Auch lässt sich sehr schnell und effektiv nach Implementierern ('Implementors') oder Aufrufern ('Senders') einer Methode suchen, ohne dass dabei hunderte oder tausende Dateien durchsucht und analysiert werden müssten.

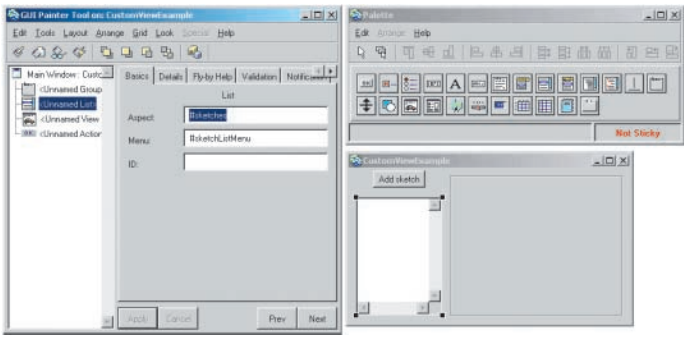
Genau genommen besteht das Programmieren einer Anwendung in Smalltalk darin, das vom Hersteller der Entwicklungsumgebung mitgelieferte Image mit den enthaltenen Werkzeugen so lange zu verändern und um neue Klassen, Objekte und Methoden zu ergänzen, bis die 'Objektwolke' das tut, was sie soll. Zum Starten einer Anwendung lädt man das gespeicherte Image einfach wieder in die virtuelle Maschine, die im Idealfall sogar auf einer anderen Hardware- oder Betriebssystemumgebung laufen kann. Um Platz zu sparen und unerwünschte Einblicke in den Code zu verhindern, kann man vor dem endgültigen Speichern noch die nur zum Programmieren benötigten Teile des Systems wie den Compiler oder den Debugger aus dem Image entfernen.

Smalltalk-Entwicklungsumgebungen stellen sehr mächtige Debugger und Inspektoren zur Verfügung. Ein Inspektor ist ein Werkzeug, mit dem sich ein einzelnes Objekt ansehen und modifizieren lässt. Der umfassende Einblick in das System zu jedem Zeitpunkt ist eine enorme Hilfe beim Aufspüren von Fehlern. Viele Java- und C++-Werkzeuge bieten bereits Ansätze in diese Richtung, sind aber noch weit von dem entfernt, was in Smalltalk alltäglich ist.

Diese dynamischen Eigenschaften von Smalltalk prädestinieren es für iterative Entwicklungsansätze. Der Smalltalk-Entwickler arbeitet oft direkt im Debugger, um im Rahmen kleiner Unit-Tests Fehler zu bereinigen. Die Grenzen zwischen Test und Entwicklung werden fließend oder verschwinden völlig. Damit sind natürlich auch die besten Voraussetzungen geschaffen, sehr früh mit Endanwendern über die Anforderungen und Eigenschaften eines Produkts zu sprechen und so zyklische Vorgehensweisen zu etablieren. Beides – ständiges Testen während der Entwicklung und enge Rückkopplung mit dem späteren Endanwender – führt zu einer hohen Produktqualität, ohne den Entwicklungsprozess merkbar zu beeinflussen. Angesichts dieser Tatsachen vermag es nicht zu verwundern, dass Ansätze wie



Smalltalkers Lieblingsplätzchen: der Debugger, in dem jeglicher Quelltext jederzeit bearbeitet und der korrigierte Code sofort ausgeführt werden kann.



Auch Werkzeuge zum GUI-Design sind in Smalltalk-Umgebungen gang und gäbe. Hier der 'GUI Painter' von Cincoms VisualWorks.

eXtreme Programming oder Refactoring ihren Ursprung in Smalltalk-Projekten haben, wo sie auch umfangreich praktiziert werden.

Was fehlt?

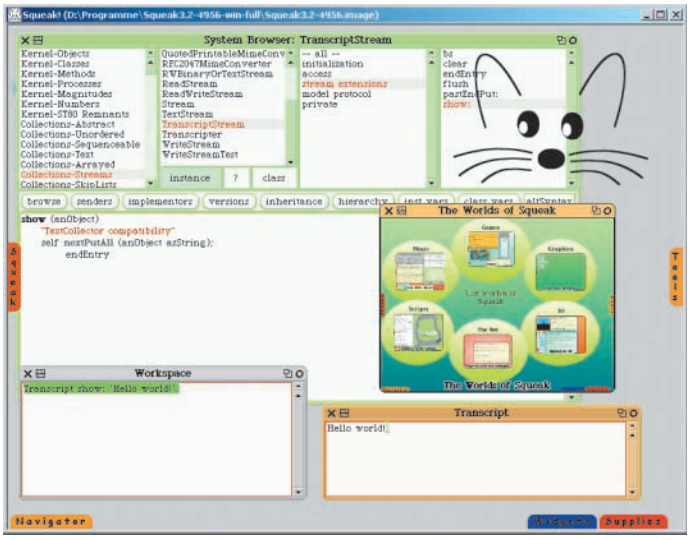
Folgt man dieser Aufzählung der Vorteile von Smalltalk, so drängt sich die Frage auf, warum diese Sprache vergleichsweise wenig Popularität genießt. Eine der einfachsten Antworten darauf ist, dass auch Smalltalk nicht die allumfassende Lösung aller Probleme ist, genauso wenig wie C++, Java oder die Objektorientierung an sich.

Ein weiterer Grund mag sein, dass mit der Firma ObjectShare einer der bekanntesten Anbieter von Smalltalk-Entwicklungswerkzeugen Mitte der neunziger Jahre einige Management-Fehler mit finanziellen Problemen paarte. Der noch immer zu spürende Hype um Java gibt weitere Ansätze zur Erklärung der Marktsituation.

Dabei ist es nicht so, dass Smalltalk wesentliche Schwächen gegenüber Java oder C++ aufweisen würde. Die wichtigsten Smalltalk-Umgebungen bringen allesamt eine komplette Palette unterstützter Schnittstellen mit. Angefangen bei direkten Schnittstellen zu DLLs und Integration mit Middleware-Technologien wie Datenbanken, Messaging-Systemen und Transaktionsmonitoren bis hin zu Komponententechnologien wie CORBA, Java RMI und EJBs eignen sie sich für eine große Bandbreite von Integrationsszenarien. Mit der Unterstützung aller gängigen Internet-Protokolle und Standards wie HTTP, SMTP, POP3 oder SNMP sind sie gerüstet für moderne Anforderungen im E-Business-Umfeld. Selbst XML, SOAP und die Implementierung von Web Services sind in Smalltalk möglich.

Smalltalk ist also entgegen der Meinung mancher Unkenrufer eine durchaus lebendige Technologie, die einiges an Innovationspotenzial mitbringt. So ist der erste verfügbare Refactoring-Browser eine Geburt der Smalltalk-Community, die nach einem kleinen Durchhänger in den Jahren 1998 bis 2000 inzwischen wieder kräftig wächst – nicht zuletzt durch die steigende Zahl verfügbarer Entwicklungsumgebungen.

Smalltalk wird in Deutschland in einer Vielzahl von Unternehmen eingesetzt. Dabei zeigen sich große Smalltalk-Projekte in fast jeder Branche: Banken und Versicherer sind unter den Smalltalk-Anwendern ebenso vertreten wie fast jeder bedeutende Automobil- oder Chemiehersteller. Dienstleistung, Handel und Logistik sind ebenfalls Felder, in denen Smalltalk-Systeme bereits seit Jahren erfolgreich ihren Dienst versehen.



Bei Squeak versteckt sich ein vollwertiges Smalltalk-System hinter einer gewöhnungsbedürftigen Bonbon-Optik.

Smalltalk ist aber nicht nur ein Werkzeug für Mega-Projekte. Gerade der Spaßfaktor, den die Arbeit mit Smalltalk mit sich bringt, prädestiniert die Sprache für kleinere Projekte im kommerziellen, aber auch im privaten Umfeld. Gerade für Letzteres ist sicherlich interessant, dass fast jeder Hersteller für den privaten Gebrauch eine Version seines Tools zur freien Verwendung bereitstellt, sicherlich mit dem Hintergedanken, die Entwicklungsumgebung über das private Umfeld in Unternehmen hinein zu etablieren – ein Ansatz, der in der Vergangenheit mehrfach funktioniert hat.

Woher?

Prominentester Smalltalk-Anbieter ist die Firma IBM, die ihre VisualAge-Produktfamilie mit einem Smalltalk-Produkt begründete. *VisualAge Smalltalk* zeichnet sich vor allem durch die Vielzahl von Integrationsmöglichkeiten mit im kommerziellen Bereich weit verbreiteten IBM-Produkten wie CICS, MQSeries, DB2 Universal Database und WebSphere aus. Mit VisualAge Smalltalk lassen sich Anwendungen für alle gängigen Plattformen entwickeln: Zu OS/390, AIX, Solaris, HP-UX, OS/2 und Windows 3.1 bis XP ist seit der Version 6.0 auch Linux hinzugekommen. Mehr Informationen finden sich auf der Produktseite der IBM unter www.software.ibm.com/ad/smalltalk. IBM bietet eine funktional und zeitlich unlimitierte Evaluierungsversion zum Download an.

Der wichtigste Konkurrent von IBM ist die Firma Cincom, die derzeit drei Smalltalk-Entwicklungsumgebungen im Portfolio hat. Was sich auf den ersten Blick seltsam anhört, war vor zwei Jahren eine sehr gute Nachricht für alle Smalltalk-Fans. Da kaufte Cincom nämlich die finanziell zusammengebrochene Firma ObjectShare und gab damit deren beiden Smalltalk-Produkten wieder eine Zukunft. Dabei hatte Cincom mit ObjectStudio bereits ein eigenes Smalltalk-Produkt im Portfolio.

Während man sich bei Cincom entschlossen hat, das Produkt *VisualSmalltalk Enterprise* nur noch für eine gewisse Zeit zu unterstützen, werden die anderen beiden IDEs kontinuierlich weiterentwickelt. *ObjectStudio* ist nur auf Windows verfügbar und zeichnet sich durch eine hervorragende Integration mit diesem Betriebssystem aus. *VisualWorks* hingegen läuft auf einer Vielzahl von Systemplattformen (Windows-Familie, einige Unix-Derivate inklusive Linux sowie Mac OS 9 und X) und ist zwischen diesen binärkompatibel, das heißt, ein Image kann ohne jegliche Änderung von einer Plattform auf eine andere übertragen werden, bei null Portierungsaufwand. Cincom bietet sowohl VisualWorks als auch ObjectStudio zur nichtkommerziellen Nutzung bei vollem Leistungsumfang zum Download unter www.cincom.com/smalltalk an.

Ein Smalltalk mit etwas anderem Fokus ist *Gemstone/S* von der gleichnamigen Firma Gemstone Systems Inc. Dieses System ist ein persistenter Smalltalk-Objektserver, der als Backend zu Smalltalk-Umgebungen wie VisualAge oder VisualWorks dient. Eine nichtkommerzielle Version für Linux kann auf der Website unter www.gemstone.com heruntergeladen werden. Gemstone hat besonders viel Erfahrung im Bereich der persistenten Speicherung von Objekten und deren Bereitstellung für verteilte Applikationen.

Die Firma Object Arts in Großbritannien ist eher ein Neuling auf dem Markt, gemessen an 30 Jahren Smalltalk-Entwicklung. Das Produkt *Dolphin*

Smalltalk aus diesem Hause zeichnet sich durch eine sehr schnell wachsende Anhängerschaft, attraktive Preise und eine hervorragende Integration mit Windows aus. Dolphin Smalltalk sieht sich als direkter Konkurrent zu VisualBasic oder Delphi und zielt nicht so sehr auf den Enterprise-Markt, sondern eher auf kleinere Softwareprojekte und Entwickler, die ausschließlich für Windows-Plattformen entwickeln. Dolphin Smalltalk wird in mehreren Editionen angeboten, von denen es unter www.object-arts.com auch Evaluierungsversionen gibt.

Eine beliebte Smalltalk-Umgebung ist *Smalltalk/X* aus dem Hause eXept Software AG im süddeutschen Bönnigheim. Diese Umgebung ist die auf den meisten Plattformen verfügbare Smalltalk-Implementierung überhaupt. Der Hersteller nutzt dieses Tool vor allem für eigene Projekte im industriellen Umfeld und hat sich interessanterweise Ende des letzten Jahres dazu entschlossen, Smalltalk/X sowohl zur kommerziellen als auch nichtkommerziellen Nutzung völlig freizugeben. Es steht auf der Firmen-Homepage unter www.exept.de zum Download bereit.

Mit *SmallScript* (www.smallscript.org) entsteht derzeit eine sehr interessante Smalltalk-Umgebung, die auf die .NET-Plattform von Microsoft hin ausgerichtet ist. SmallScript versteht sich als eine Skript-Umgebung für die .NET-Plattform, die die Sprache Smalltalk um einige hilfreiche Konstrukte erweitert. Neueste Verlautbarungen der Entwickler von SmallScript deuten darauf hin, dass das Produkt, das derzeit im Beta-Stadium ist, unter dem Namen S# auf den Markt kommen wird.

Eine sehr interessante, freie Alternative ist *Squeak*. Dieses Open-Source-Projekt ist unter www.squeak.org zu finden. Interessant an Squeak ist, dass mit Alan Kay einer der Smalltalk-Erfinder maßgeblich daran beteiligt ist. Das Projekt Squeak war nach seinem Auftakt bei Apple einige Zeit in Händen des Disney-Konzerns und ist vor allem im Hinblick auf die Integration verschiedenster Medien zu multimedialen Anwendungen interessant. Die Mickymaus-Company selbst nutzt Squeak für ihre bekannten Themenparks. Squeak ist in erster Linie eine Plattform zum Entwickeln von Multimedia-Applikationen und stützt sich auf eine große Gemeinde von Entwicklern rund um den Globus, vergleichbar etwa mit der Linux-Community.

Fazit

Die Sprache Smalltalk ist eine ausgereifte und sehr interessante Basis für kleine, mittlere und vor allem große Projekte, die zum einen schnelle Reaktionen bei sich ändernden Geschäftsprozessen ermöglicht und zum anderen durch ihren sehr lebendigen Charakter großen Spaß an der Entwicklungsarbeit vermittelt. Als Alternative zu eher schwergewichtigen hybriden Sprachen wie C++ oder Java mit ihren umfangreichen Regelwerken bietet sie sich an, einmal fremdzugehen und sich jenseits des Tellerrands umzusehen. Mit den inzwischen sehr ausgereiften Internet-Features ist in Smalltalk praktisch jede Art von moderner Anwendung, ob Client/Server- oder Web-Applikation, realisierbar. Die seit ein bis zwei Jahren wieder wachsende Zahl von Smalltalk-Nutzern wird ihren Teil dazu beitragen, dass Smalltalk weiterhin eine interessante Alternative zu anderen – weniger flexiblen – Entwicklungssprachen bleibt.

Wer sich näher mit Smalltalk beschäftigen möchte, dem sei geraten, eine der hier genannten Entwicklungsumgebungen herunterzuladen, ein Tutorial (Adressen siehe Kasten) oder ein Buch zur Hand zu nehmen und einfach mal loszulegen. Es lohnt sich auf alle Fälle, sich mit einer vollständig objektorientierten Programmiersprache zu beschäftigen, selbst wenn man sich dann am Arbeitsplatz wieder mit halbherzigen Lösungen plagen muss. (hos)

Literatur

[1] Ed Klimas, Getting The Biggest Bang For Your Buck, VisualAge Magazine, Mai 1998, www.lineaengineering.com/Resources/Productivity_/productivity_.html

[2] Frederick P. Brooks, The Mythical Man-Month, Essays on Software Engineering, Anniversary Edition (2nd Edition), Addison-Wesley, 1995, erstmals erschienen 1974, inzwischen ein Klassiker im Bereich des Software-Programmmanagements

Soft-Link **0302188**

Interessante Links zum Thema Smalltalk

Einen Abriss der Geschichte von Smalltalk und grafischer Benutzeroberflächen aus Sicht einer der Hauptakteure, Alan Kay, findet sich auf www.artmuseum.net/w2vr/timeline/Kay.html. Kays Ziel war und ist es, durch das Verstehen menschlicher Denk- und Begreifweisen Wege zu finden, Computer für den Menschen begreifbar und damit beherrschbar zu machen. Nicht zuletzt darauf lässt sich die Klarheit und Eleganz der Sprache Smalltalk zurückführen.

Im Web gibt es viele Fundstellen für Tutorials und sonstige Materialien rund um Smalltalk. Für eine erleichterte Suche existieren zwei sehr gut gepflegte Portale, www.whysmalltalk.com und www.goodstart.com.

Stéphane Ducasse von der Universität Bern hat eine Sammlung von Büchern rund um Smalltalk ins Web gestellt, die entweder nur als Online-Literatur verfügbar sind oder nicht mehr gedruckt werden. Ziel ist es, die Liste noch zu erweitern. Ein Besuch unter www.iam.unibe.ch/~ducasse/WebPages/FreeBooks.html lohnt sich also ganz bestimmt.

Eine der ersten Adressen für OO-Entwickler überhaupt ist die Linksammlung 'Cetus Links'. Unter www.cetus-links.org/oo_smalltalk.html finden hier auch Smalltalker unzählige Verweise auf WWW-Angebote, die sich mit ihrer Lieblingssprache beschäftigen.

Eine wichtige Informationsquelle für alle Probleme rund um Smalltalk sind Usenet-Newsgroups. Dabei gibt es Gruppen, die sich mit Smalltalk ganz allgemein auseinander setzen, etwa comp.lang.smalltalk oder comp.lang.smalltalk.advocacy, und solche, in denen nur eine spezielle Entwicklungsumgebung im Mittelpunkt steht, etwa comp.lang.smalltalk.dolphin oder ibm.software.vasmalltalk.

Weltweit gibt es einige User-Groups, in denen sich Smalltalker virtuell oder auch persönlich zu interessanten Veranstaltungen rund um das Thema Smalltalk, eXtreme Programming et cetera treffen. Deutschlands noch sehr junge German Smalltalk Users' Group (GSUG) präsentiert sich mitsamt einem Wiki-Server (ein interaktiver Webserver, der jedem Besucher erlaubt, Seiten anzulegen und bestehende zu ändern, siehe auch S. 176; eine Abwandlung davon sind Swikis, die eben in Smalltalk implementiert sind) unter www.gsug.org. Die GSUG stellt Interessierten gegen Kostenbeteiligung auch eine CD mit den wichtigsten frei verfügbaren Smalltalk-Versionen zur Verfügung. Die GSUG ist eine Fraktion der ESUG, der European Smalltalk Users' Group, deren Webpräsenz sich unter www.esug.org findet.

Eine erwähnenswerte Initiative, an der sich einerseits praktisch alle Anbieter von Smalltalk-Entwicklungsumgebungen beteiligen, die aber andererseits jedem offen steht, ist das so genannte Camp Smalltalk, eine lockere Folge von zwanglosen Treffen einiger engagierter Smalltalker. Im Rahmen dieser Camps werden interessante Projekte (etwa ein freies Persistenzmodell oder ein XML-Austauschformat für Smalltalk-Quellcode etc.) realisiert. Das nächste Camp Smalltalk wird im Juni 2003 in Deutschland, genauer gesagt in Gronau bei Stuttgart stattfinden. Für interessierte Beobachter, aber auch für Leute, die aktiv an den Projekten des Camp Smalltalk mitarbeiten wollen, ist das zweifellos ein Termin, den man sich merken sollte. Mehr Informationen zum Camp Smalltalk finden sich unter www.exept.de/sites/exept/deutsch/Events/cs.html.

c't