

# The Smalltalk Report

The International Newsletter for Smalltalk Programmers

November/December 1991

Volume 1 Number 3

## RESPONSIBILITY- DRIVEN DESIGN

By Rebecca Wirfs-Brock

### Contents:

#### Features/Articles

- 1 Responsibility-driven design  
by Rebecca Wirfs-Brock
- 11 Smalltalk comes to the mainframe, part 1  
by Glenn J. Reid

#### Columns

- 6 *Getting Real: Should classes be initialized?*  
by Juanita Ewing
- 9 *GUIs: ObjectWorks/Smalltalk Release 4  
for MS-Windows 3.0: a look at the lower  
levels* by Greg Hendley and Eric Smith

#### Departments

- 15 *Conference Report: Digitalk's Smalltalk/V  
Developers Conference '91*  
by Paul White
- 17 *Lab Report: Using and studying Smalltalk  
in the User Interface Institute*  
by Mary Beth Rosson
- 19 *What They're Saying About Smalltalk*
- 20 *Product Announcements*

**O**bject-oriented design is a process that creates a model of interacting objects. Models leave out trivial details and focus on the essential aspects of the thing they represent. A model isn't supposed to be an exact replica of the original!

Simply using an object-oriented programming language or environment does not, in itself, guarantee miraculous results. Like any human endeavor, software design requires discipline, hard work, inspiration, and sound technique. Some people have touted object-oriented design as the next great revolution in software simply because it allows a designer to capture "real world" objects and, with just a little bit of transformation, create a working program. It isn't that easy. There's a lot of judgement, abstraction skills, and lessons learned from previous experiences that go into creating an object-oriented model. And the approach taken to define and describe objects will have profound impacts on the resulting design.

This article describes an object-oriented modeling technique called responsibility-driven design.<sup>1,2</sup> This approach draws upon the experiences of a number of very successful and productive Smalltalk designers. The concepts and motivations behind responsibility-driven design were initially formulated when Brian Wilkerson, myself, and others developed and taught a course on object-oriented concepts and design to Tektronix engineers. These engineers were working on object-oriented projects that would be implemented in Smalltalk and C++, as well as conventional languages.

### GOALS FOR THE DESIGN PROCESS

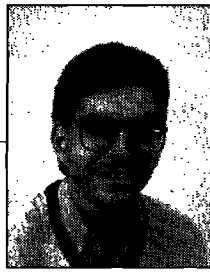
We had several goals for presenting an object-oriented design method. First, we wanted to encourage exploration of alternatives early in the design, yet provide criteria and a structure for analyzing and improving upon initial design decisions. We wanted designers to first develop the "big picture" of how key objects in their application interact before filling out precise details of individual classes. It is far, far easier to consider, refine, and even discard ideas up front before major investments of time and money have been made. Decisions made about the internal structure and algorithmic details of an object before there is a clear understanding of its role and purpose are often flawed. These decisions, if made too early, can require major revision. We wanted to present a process that helped designers avoid some costly rework. No process can ever eliminate it. Design, by nature, is an incremental journey of discovery and refinement. With each new refinement, comes further insights and changes. It is important, however, not to force design decisions before there is enough knowledge to make intelligent tradeoffs.

We wanted designers to initially focus on building a model of *interacting* objects. Each object's role or purpose as well as its interactions with other objects is important to understand. Finally, we wanted to present modeling techniques and guidelines that were language independent. Principles and design practices followed by experienced Smalltalk developers are applicable to other object-oriented designs. Given these fundamental objectives, let's examine the process of responsibility-driven design.

*continued on page 4...*



John Pugh



Paul White

## EDITORS' CORNER

Welcome again! Well, we've finally convinced you that *The Smalltalk Report* is for real. It has been gratifying to see the terrific response we have received to the first two issues. Meeting many of you at OOPSLA and the Digitalk Developers Conference has allowed us to get your feedback directly. Most of it has been positive — the only comment uttered enough times to warrant a response from us has been the charge that what is being stated by some of our columnists is, to paraphrase, "not the right way Smalltalk should be used." To that comment, we say "Let's hear your rebuttal!" Our role as editors is not to decide what is "right" and "wrong" — only to ensure that arguments get presented in a professional and informative manner. We welcome your responses — either in the form of an article or through our Messages: soapbox. Along these lines, one worthwhile suggestion received was to solicit two members of the Smalltalk community to debate a topic of current interest. We would be delighted to offer such a feature. However, as usual, pinning people down to contribute is never easy. Any suggestions you may have for suitable candidates and topics for such a feature would be welcome.

In this issue, we welcome Rebecca Wirfs-Brock as a new columnist to the *The Smalltalk Report*. As author and speaker, Rebecca's opinions are widely respected within the Smalltalk community. Her approach to object-oriented design is well known and is well suited to the Smalltalk community. In upcoming issues, Rebecca promises to go beyond her current writings to provide greater detail and insight into the issues of object-oriented design. In this issue, she begins by reviewing the responsibility-driven approach to design. Also in this issue, Glenn Reid discusses the Smalltalk/370 project — an ongoing effort to bring Smalltalk to the world of mainframes.

Two of our regular columns are again featured in this issue. Greg Handley and Eric Smith continue their look at GUIs by describing the low-level interactions occurring in ParcPlace's Release 4, while Juanita Ewing continues her Getting Real column by discussing a problem well known to Smalltalkers — how to systematically arrange for the initialization of class variables. Finally, in our Lab Report section, Mary Beth Rosson describes the considerable amount of Smalltalk research taking place at the IBM T. J. Watson Research Laboratories.

Listening to the debates at the Digitalk Developers Conference or looking around the exhibits floor at OOPSLA, it was hard to come to any conclusion other than that interest and development in Smalltalk is thriving as never before. Moreover, real examples of the use of Smalltalk in commercial projects are now plentiful. We hope to report on some of these "success stories" in upcoming issues. To begin this process, we present an overview of the activities of the Digitalk Developers Conference.

*John Pugh* *Paul White*

## The Smalltalk Report

### Editors

John Pugh and Paul White  
Carleton University & The Object People

### SIGS PUBLICATIONS

#### Advisory Board

Tom Atwood, Object Technology  
Grady Booch, Rational  
George Bosworth, Digitalk  
Brad Cox, Information Age Consulting  
Chuck Duff, The Whitewater Group  
Adele Goldberg, ParcPlace Systems  
Tom Love, Consultant  
Bertrand Meyer, ISE  
Meilir Page-Jones, Wayland Systems  
Shesa Pratap, CenterLine Software  
P. Michael Seashols, Versant  
Bjarne Stroustrup, AT&T Bell Labs  
Dave Thomas, Object Technology

### THE SMALLTALK REPORT

#### Editorial Board

Jim Anderson, Digitalk  
Adele Goldberg, ParcPlace Systems  
Reed Phillips, Knowledge Systems Corp.  
Mike Taylor, Instantiations  
Dave Thomas, Object Technology International

#### Columnists

Juanita Ewing, Instantiations  
Greg Handley, Knowledge Systems Corp.  
Ed Klimas  
Suzanne Skubics, Object Technology  
Eric Smith, Knowledge Systems Corp.  
Allen Wirfs-Brock, Instantiations  
Rebecca Wirfs-Brock, Tektronix

### SIGS Publications Group, Inc.

Richard P. Friedman  
Group Publisher

#### Art/Production

Elisa Varian, Production Manager  
Susan Culligan, Creative Director  
Kristin R. Juba, Production Editor  
Caren Polner, Desktop Designer

#### Circulation

Diane Badway, Circulation Business Manager  
Kathleen Canning, Fulfillment Manager  
John Schreiber, Circulation Assistant

#### Marketing/Advertising

James Kavetas, Advertising Director  
Diane Morancie, Account Executive  
Geraldine Schafraan, Advertising Sales Assistant

#### Administration

David Chatterpaul, Accounting  
Suzanne W. Dinnerstein, Conference Manager  
Jennifer Fischer, Assistant to the Publisher  
Laura Lea Taylor, Administrative Assistant  
Margherita R. Monck  
General Manager



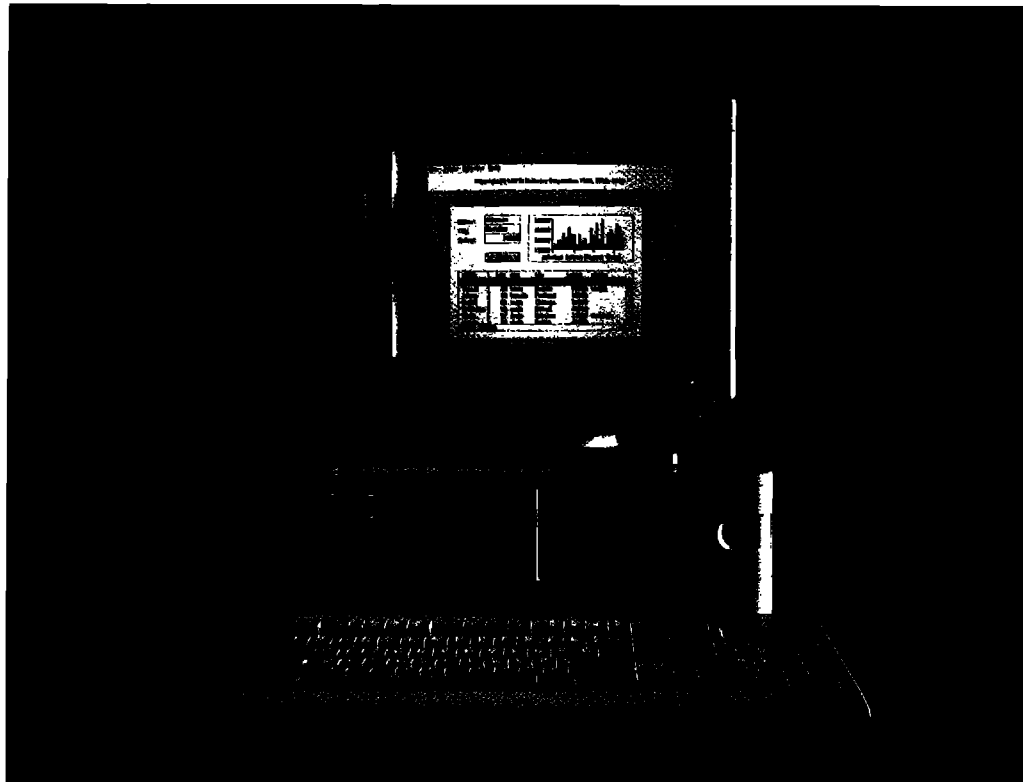
Publishers of *Journal of Object-Oriented Programming*, *Object Magazine*, *Hotline on Object-Oriented Technology*, *The C++ Report*, *The Smalltalk Report*, *The International OOP Directory*, and *The X Journal*.

The Smalltalk Report (ISSN# 1056-7976) is published 9 times a year, every month except for the Mar/Apr, July/Aug, and Nov/Dec combined issues. Published by COOT, Inc., a member of the SIGS Publications Group, 588 Broadway, New York, NY 10012 (212)274-0640. © Copyright 1991 by COOT, Inc. All rights reserved. Reproduction of this material by electronic transmission, Xerox or any other method will be treated as a willful violation of the US Copyright Law and is flatly prohibited. Material may be reproduced with express permission from the publishers. Mailed First Class. Subscription rates 1 year, (9 issues) domestic, \$65, Foreign and Canada, \$90. Single copy price, \$8.00. POSTMASTER: Send address changes and subscription orders to: THE SMALLTALK REPORT, Subscriber Services, Dept. SML, P.O. Box 3000, Denville, NJ 07834. Submit articles to the Editors at 91 Second Avenue, Ottawa, Ontario K1S 2H4, Canada.

# Unleash the True Power of Client/Server Computing...



...fast!



ENFIN/2 offers a unique integration of 4GL visual development tools within an object oriented development environment which enables application developers to achieve a quantum leap in productivity.

#### Graphical User Interfaces:

- All CUA display objects predefined
- Workplace support for pick, drag and drop of icons
- Additional display objects such as business graphics, tables, bitmaps and more
- Portability of source code between Windows and OS/2 PM

#### Client Server Architecture:

- Point and click linking of SQL tables and queries to display objects and report objects
- Support for most popular SQL engines and DBase
- Support for cursors, scrollable cursors, and embedded SQL

#### Open Architecture:

- APPC LU6.2 and EHLAPPI host communications
- DDE, DDL's, clipboard and OLE

#### Object Oriented Development Environment:

- Enfin's Smalltalk object oriented programming language
- Object oriented development and debugging tools
- Extensive library of reusable and extendable classes
- Run Time Generator includes unlimited distribution license

#### 4GL Visual Development Tools:

- Graphical User Interface Builder
- Graphical Report Builder
- Data Entry Definition Tool
- SQL workbench
- Financial modeling facility

Point and click simplifies the creation of SQL queries to support client/server architecture.

Supports pick, drag and drop workplace applications.

Includes a professional set of object-oriented development tools.

Visually develop user interfaces.

The image shows two screenshots of the ENFIN/2 software. The top screenshot displays a graphical query builder interface with a menu bar (File, Query, Table, Window, Help) and a workspace containing several tables: ALLTYPER, CUSTOMER, EMPLOYEE, PRODUCT, and SALES. A query is being constructed with the following SQL code:

```

SELECT * FROM EMPLOYEE
ORDER BY NAME
GROUP BY
HAVING
AND
OR
IN
NOT BETWEEN
SALES
WHERE AGE <= 25 AND SEX = 'F'
    
```

The bottom screenshot shows a graphical report interface with a menu bar (File, View, Options, Help) and a data table. The table has columns for NAME, AGE, SEX, CITY, SALARY, and ST. The data is as follows:

NAME	AGE	SEX	CITY	SALARY	ST
Jackson	40	male	Seattle	25000	WA
Turner	40	male	Seattle	25000	WA
Jones	34	female	Portland	35000	OR
Smith	21	male	Chicago	21000	IL
Williams	55	male	Detroit	40000	MI

Next to the table is a bar chart showing salary distribution. The chart has a y-axis labeled 'SALARY' with values 0, 20000, and 50000. The bars represent the salary for each employee listed in the table.

ENFIN/2 is being used by the world's biggest companies in Windows and/or OS/2 PM: Mercedes Benz, Deutsche Bank, DuPont, IBM, Southern California Edison, Bayer, Security Pacific Bank, Ceiba Geigy, Bosch, Southern California Gas, and many more.

"We have decided to market ENFIN/2 because of the unique and rich OS/2 support for Corporate application development. Some of our largest customers have selected ENFIN/2 to develop truly object oriented OS/2 Presentation Manager applications because of ENFIN/2's superb support of the OS/2 Database Manager and its APPC LU6.2 host communications support."

Ludwig von Reiche, IBM Germany

# ENFIN

SOFTWARE CORPORATION

1(800) 922-4372 U.S.

1(800) 722-4372 CA

6920 Miramar Road, Suite 307  
San Diego, California 92121  
(619) 549-6606 ■ FAX: 549-6798

*continued from page 1...*

## RESPONSIBILITIES AND COLLABORATIONS

To start, object-oriented design typically is quite exploratory and iterative. Unless designers are building (or rebuilding) an application with which they are intimately familiar, a clear vision of the key classes does not exist. Creating a model requires understanding system requirements as well as skill in identifying and creating objects. Building consensus and developing a common vocabulary among team members is important. Initially, designers look for classes of key objects, trying out a variety of schemes to discover the most natural and reasonable way to abstract the system into objects.

Responsibility-driven design focuses on what *actions* must get accomplished and which objects will accomplish them. How each action is accomplished is deferred. A good starting point for defining an object is describing its role and purpose in the application. Details of internal structure and specific algorithms can be worked out once roles and responsibilities are better understood.

A responsibility is a cohesive subset of the behavior defined by an object. An object's responsibilities are high-level statements about both the knowledge it maintains and the operations it supports. An analogy between designing objects and writing a report can clarify the intent of listing each object's responsibilities. An object's responsibilities are analogous to major topic headings in an outline for a report. The purpose of developing an outline (and then a detailed outline) before writing a report is to map out the topics to be covered in the report and their order of presentation. Similarly, the purpose of outlining an object's responsibilities is to understand its role in the application before fleshing out the details. A good way to determine an object's responsibilities is to answer these questions: What does this object need to know to accomplish each goal it is involved with? What steps toward accomplishing each goal should this object be responsible for?

Objects do not exist in isolation. Object-oriented applications of even moderate size can consist of hundreds if not thousands of cooperating objects. A *collaboration* is a request made by one object to another. An object will fulfill some responsibilities itself. Fulfilling other responsibilities likely requires collaboration with a number of other objects. Object collaborations can be determined by examining each responsibility and answering the questions: What other objects need this result or knowledge? Is this object capable of fulfilling this responsibility itself? If not, from what other objects can or should it acquire what it needs?

## THE CLIENT/SERVER MODEL

Collaborations can be modeled as client/server interactions. A client makes a request of a server to perform operations or acquire knowledge. A server provides information or performs an operation upon request. Clients and servers are

roles objects assume during a collaboration. Modeling client/server interactions can help to reinforce information hiding. A client shouldn't care *how* a server performs its duties, only that it responds appropriately. On the other hand, a server is obligated to respond appropriately to any such request.

The relationship between client and server can be formalized in a contract. A contract is a set of related responsibilities defined by a class. It also describes the ways in which a given client can interact with a server. It lists requests that a client can make of a server. Both client and server must uphold the terms of their contract. The client fulfills its obligation by only making those requests specified in the contract. The server must respond appropriately to those requests. Later, with a more complete understanding of our design, we can fill in the fine print of each contract. This can involve specifying details of client requests (including message names and arguments, preconditions that must be met before making a request, and postconditions that will be true after the server has performed the requested operation<sup>3</sup>). In early stages of design, however, it is enough to understand contracts stated in general terms.

The design process outlined thus far consists of finding key objects, defining their roles and responsibilities, and understanding their patterns of collaborations. Responsibility-driven design initially focuses on *what* should be accomplished, not *how*. Using a client/server model of object collaboration, we identify each object's public interfaces by answering: What actions is each object responsible for performing? What information is each object responsible for providing?

## A SIMPLE TOOL

Given an initial set of key objects, the designer can evaluate object responsibilities and collaborations by testing how the model responds to a variety of requests. Scenarios that the application must handle can be described, and requests or events can be fed into the model. Patterns of collaboration required to handle each situation can be traced. Running through a number of typical scenarios rapidly points out gaps in understanding. It is not uncommon to find new key objects and discard ill-conceived ones, to evaluate and reassign responsibilities, or to fabricate new mechanisms as part of this process. It is also a time when missing or conflicting system requirements surface and require clarification before completing the initial model.

Kent Beck and Ward Cunningham<sup>4</sup> initially developed the concept of using index cards to teach object-oriented concepts. The idea behind CRC cards (for class-responsibility-collaboration) was to provide a quick, effective way to capture the initial design of an object (see Figure 1).

The name of each class is written on an index card. Each identified responsibility is succinctly written on the left side of the card. If collaborations are required to fulfill a responsi-



## How should classes be initialized?

Class initialization should be systematic and predictable. Have you ever sent messages to a class and received strange errors related to uninitialized class data? Uninitialized class data is a result of Smalltalk programming conventions and lack of support by the programming environment. Some minor changes in the Smalltalk programming environment could greatly improve this situation.

### WHAT NEEDS TO BE INITIALIZED?

Smalltalk classes have two or three different kinds of class data that must be initialized:

- class variables
- class instance variables
- pool dictionaries

Smalltalk-80 derived dialects have class instance variables, but Smalltalk/V dialects do not. Each kind of class data has semantic differences.

Classes can have class variables, which are shared between all instances and the class. Class variables can be referenced from both instance and class methods simply by referring to the name of the class variable.

Class instance variables are storage for the class and can be referenced only from class methods. Instance methods that need the information stored in a class instance variable must

send a message to a class method, which can return the requested information.

Pool dictionaries are shared between several classes. The keys in a pool dictionary can be directly referenced from both instance and class methods.

### WHICH OF THESE IS INHERITED?

All Smalltalk programmers are familiar with the inheritance semantics of instance variables. The variable is inherited, but not its value.

Unlike instance variables, a class variable and its value are inherited. That means that the value of a class variable is shared with subclasses and all of their instances. The diagram in Figure 1 shows a class variable defined by class A. The subclass B inherits the class variable. Instance methods from the subclass and superclass are able to reference class variables.

Class instance variables are much like the instance variables we use all the time in Smalltalk programming, except that class instance variables are instance variables for a class instead of for instances of a class. The semantics of class instance variables are similar to those of instance variables. The variable is inherited, but not its value. Each class must fill in its own value. These kind of variables are handy because subclasses can easily override values defined in a superclass.

In Figure 2, class A defines a class instance variable, i. Subclass B inherits the class instance variable i and defines another class instance variable. Only class methods can refer to class instance variables. The receiver of the message deter-

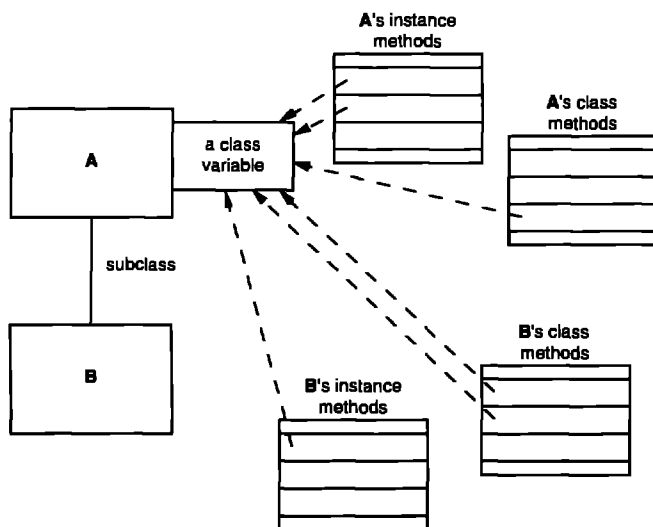


Figure 1. Class variables.

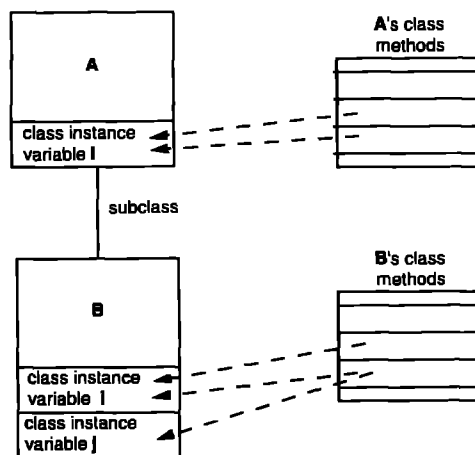


Figure 2. Class instance variables.

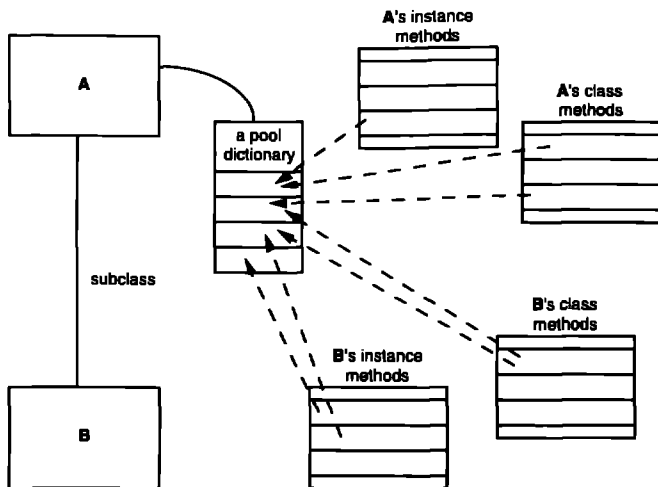


Figure 3. Pool dictionaries.

mines which storage slot will be referenced. For example, suppose the method `initialize` references the class instance variable `i`. If the receiver of the message `initialize` is A, then the storage slot in class A will be referenced. If the receiver is B, then the storage slot in class B will be referenced.

Pool dictionaries are similar to class variables. Pool dictionaries and their values are inherited by subclasses. In Figure 3, class A defines a pool dictionary. Both class and instance methods can reference keys in the pool dictionary. Subclass B inherits the pool dictionary, including its values. Instance and class methods from subclass B can reference keys in the pool dictionary.

Let's examine the inheritance consequences.

### WHAT HAPPENS IF CLASS METHODS ARE USED TO INITIALIZE?

Class data needs to be initialized. The most common practice is to use a class method called `initialize`. This method typically is used to initialize all class data, no matter what kind of class data it is.

Assume an initialization method initializes a class variable. The class that defines the class variable needs to execute the initialization method. Since the value of a class variable is inherited, subclasses don't need to execute this method. In fact, it may be an error to do so because some valuable data may have accumulated in a class variable. Subclasses inherit the `initialize` method, but should not execute this method. This situation is a violation of good object-oriented programming.

Assume the `initialize` method initializes a class instance variable. The value of this variable is not inherited, so subclasses *must* execute the `initialize` method to initialize the variable. Either an inherited method or a local initialization

# E3

## Editor Enhancements for Smalltalk/V Windows and Smalltalk/V 286

**Multi-function editing for Smalltalk/V, compatible with the standard editor and adding over 200 user accessible commands, including :**

- Text Status Pane
- Online Help
- Key Customization
- Command Lookup/Completion
- Enhanced Cut/Paste (with multiple copies viewable in place)
- Copy Ring Processing
- Place Marking
- Macro Facility
- Text Transposition
- Easy-to-use Search and Replace
- Case Alteration
- Text Fill and Margin Settings
- Abbreviation Facility
- Non-printing Character insertion and value report
- Programming Support
- User Preferences
- Miscellaneous Goodies

E3 / W : \$80.00                      E3 / 286 : \$75.00  
 ( plus \$10.00 shipping / handling per item ).  
 Refund if not satisfied.              VISA / MasterCard Accepted.

**Object Orchard Ltd.**  
 9 Fettes Row, Edinburgh, Scotland, UK.  
 PHONE: +44 31 558 1815    FAX: +44 31 556 2718

method can be used. The initialization method must be executed by each class.

Pool dictionaries are shared between several classes. The current Smalltalk convention is for one of the classes to provide an initialization method. Should the initialization method be executed by subclasses? No. It may wipe out valuable accumulated data. This case is analogous to the situation with class variables.

### DOES SMALLTALK HAVE INITIALIZATION CONVENTIONS?

Smalltalk has a convention for the initialization of instances: it is to invoke the superclasses' initialization method if subclasses must override it. The convention arose because the superclass can initialize variables. Subclasses avoid duplicating the inherited code.

Initialization methods typically look like this:

```
initialize
  "Invoke the receiver's inherited method. Initialize my variable to the integer 2."
  super initialize.
  myVariable := 2
```

### CAN WE APPLY THIS CONVENTION TO CLASS INITIALIZATION?

The `super initialize` convention doesn't work well with class initialization methods. A Smalltalk programmer can't tell if the initialization method should be executed without examin-

ing the code. If only class instance variables are initialized, this convention works. If class variables or pool dictionaries are initialized, then this convention doesn't work since the values of these variables are inherited.

Smalltalk programmers don't restrict their initialization methods. They use initialize methods for all kinds of class data. Therefore, the existence of an initialize method in a hierarchy is not a good indication of initialization requirements.

“

The super initialize convention doesn't work well with class initialization methods.

”

#### DO CLASS INITIALIZATION METHODS WORK?

Even if you ignore the inheritance issues and the super initialize convention problems, there are still flaws in class initializations contained in class methods. Execution of the initialize method is an action that is separate from the compilation of the initialize method. This separation leads to another problem. How many times have you edited an initialize method but forgotten to execute it?

If you file in someone else's class, it may or may not have a do-it to perform an initialize. Dialects of Smalltalk-80 try to get around this problem by automatically filing out an initialize do-it when a class containing an initialize method is filed out. When the class is filed back in, the do-it is executed. This heuristic fails in two common cases. When a developer creates a method for initialization and calls it something other than initialize, such as `initializeVariables`, then the programming environment fails to detect the purpose of the method and does not treat it specially. Because this heuristic only examines behavior in a single class, it also fails when an inherited initialization method needs to be executed by a subclass.

#### SHOULD INHERITED METHODS EXECUTE WITHOUT ERROR?

Any inherited method should be able to execute without error. Dialects of Smalltalk-80 override inappropriate inherited methods with an implementation consisting of `shouldNotImplement`. High-quality class hierarchies should never allow users to execute methods that create errors. High-quality programming environments should not encourage the construction of code that creates these errors. Unfortunately, Smalltalk class initialization conventions promote the inheritance of inappropriate methods.

The Smalltalk programming environment has a quality problem.

#### SHOULD CLASS INITIALIZATION BE INHERITED?

No. Since it is impossible to tell if a class initialization method should be inherited without examining the code, class initialization methods should never be inherited. Initialization methods too frequently contain references to inherited class variables and pool dictionaries. It is too easy to make a mistake and execute an inherited initialization method that is inappropriate.

The code that performs class initialization should be compiled in the scope of the class to reference class data, but it doesn't have to be a class initialization method. The Smalltalk programming language has enough power and flexibility to provide another mechanism in the programming environment for class initialization.

#### HOW SHOULD CLASSES BE INITIALIZED?

Class initialization should not be a class method. Instead, classes should have a separate component that contains the class initialization code. We will call this the *class initialization*.

The class initialization should be bundled with the class and supported by the programming environment as a part of the class. The code that performs the initialization should be able to reference all class data: class variables, class instance variables, and pool dictionaries. The initialization code should be executed so that `self` is bound to the class. These characteristics are also characteristics of class methods, so programmers don't have to change the way they write initialization code. They just have to designate it as class initialization code.

Separate class initialization code is beneficial in several ways. Inheritance of inappropriate class methods avoids costly errors. Functionality can be added to the programming environment to improve productivity. When the code to initialize a class is identified, the programming environment can take special action to support its intended functionality. For example, when the class initialization is redefined, it could be automatically executed by the environment. The class initialization could automatically be filed out when a class is filed out and executed when the class is filed back in. If just these actions are supported by the programming environment, then much time would be saved by Smalltalk programmers.

All of this can easily be implemented in a Smalltalk programming environment. The result is separate class initialization whose purpose is known by the programming environment. This kind of class initialization is not inherited and thus avoids errors. A class initialization can send messages to the class and, in doing so, may execute class methods. ■

*Juanita Ewing is a senior staff member of Instantiations, Inc., a software engineering and consulting firm that specializes in developing and applying object-oriented software projects, and is an expert in the design and implementation of object-oriented applications, frameworks, and systems. In her previous position at Tektronix Inc., she was responsible for the development of class libraries for the first commercial quality Smalltalk-80 system. Her professional activities include Workshop and Panel Charis for the OOPSLA conference.*



# ObjectWorks\Smalltalk Release 4 for MS-Windows 3.0: a look at the lower levels

Not having had a chance to look over ParcPlace's new version of Smalltalk since Version 2.5, it was with some interest that we pulled open the package containing ParcPlace's most recent effort, Release 4 (R4). Previous versions of ParcPlace's Smalltalk would either take over the display or provide the entire Smalltalk environment within a single host window. The present version departs somewhat from this mold. Now each tool, such as a browser, workspace, file list, etc., opens up in its own host window. The Smalltalk windows now coexist on the screen with other host windowing programs such as word processors, file managers, games, etc.

None of this sounds particularly new of course. The twist is that Release 4 does all this with a virtual image that is the same on all of its platforms. The base image shipped with the release looks the same whether you are on X, MS-Windows, or Mac. This column is the result of several days of concerted digging to find out how they do it.

## TERMINOLOGY

But first! Discussing windows in the context of a system called Windows using an environment with a class named Window without first agreeing on some terms is a quick path to confusion. So, we'll agree on the following. When referring to the host windowing environment, we'll use "Windows." When discussing a particular Windows window, we'll say "window." In the case of a Smalltalk object that represents a window, an expression like "Smalltalk Window" will be used to note the fact that it is a Smalltalk object and that it is of class Window. Smalltalk objects representing visual areas within a Smalltalk Window that are not themselves associated with a window will be called views.

## WINDOWS, WINDOWS, AND VIEWS

When a browser is opened in R4, what you see is the usual type of window for whatever environment you might happen to be using — in this case Windows, which contains a bunch of panes. Only the outermost window, with its resize borders, title bar, and menu bar, etc., is a Windows window. The rest of the contents are represented by Smalltalk views. They are drawn according to instructions in the virtual image and so will look the same regardless of which environment you run Release 4 in.

Inside Smalltalk, there are three basic kinds of things that are fronting for the things you see on the display: Windows,

Wrappers, and Views. A Smalltalk Window holds everything else. This is the Smalltalk object that fronts for the actual window. Wrappers can hold views and provide things like bordering and scroll bars. Last are the views that display information like text and graphics.

For those familiar with the old model-view-controller paradigm, the comforting news is that it is still here but with a cleaner implementation. Each of the views and Smalltalk Windows have their own controller. Here we will be primarily concerned with ScheduledWindow and its controller. This will most often be an instance of StandardSystemController.

## HANDLING EVENTS

At least two things must happen to allow Smalltalk to work with Windows. First, Windows messages must make it into Smalltalk so input may be handled and display requests routed to the method that knows how to display. Second, graphics operations performed using the Release 4 portable graphics model must be converted to Windows calls. In this column, we'll look at what happens to Windows messages that get sent to a Smalltalk window. At a later date, we can look at the equally interesting process of mapping the portable graphics operations to calls to the host graphics environments.

In Windows, messages are sent to windows for many reasons. Some inform the window of input such as mouse movement or keyboard activity. Others make requests of the window such as redisplaying a damaged area. Release 4 treats each of these categories of message a bit differently, but the two major categories of interest are messages pertaining to input and messages requesting display.

## INPUT EVENTS

When a key is pressed, an input message is sent to a window, a WM\_CHAR message for example. It begins a long and winding road at the end of which the appropriate Sensor and Controller objects will become aware that an input event has occurred. A fair amount of what happens during this process happens in the virtual machine (VM), out of sight of the Smalltalk programmer, so what follows is partially surmise.

The first step is for the WndProc of the window receiving the input to handle the WM\_CHAR message. The handler for this message will package the information about the event into a sixteen-byte event structure. With the exception of the part of the structure that identifies the window that received the

message, these sixteen bytes will contain the same data under any host windowing system given an equivalent event. That is to say, an "A" typed in Windows will yield the same sixteen bytes as an "A" typed in X, with the exception of the window identifier that necessarily varies in each instance. This is the point, very early on, where we stop talking Windows and start talking Smalltalk. Before we get the information to the virtual image, it is already in a system-independent format. This is one way Release 4 maintains its extreme portability.

After the event structure is filled in, the VM presumably arranges for the InputSemaphore, a class variable of InputState, to receive the message #signal. The details of just how exactly the VM calls GetMessage() and returns from message handlers is unclear. One would assume that ParcPlace has implemented their own light-weight processes in the VM.

Meanwhile, there is a process that has been waiting on the InputSemaphore in the method InputState>>run. This process wakes up and reads the sixteen-byte event into an Array via a primitive. The event is then passed on to InputState>>process. Here, the Smalltalk Window with the handle identified in the event structure is located. This will most often be an instance of ScheduledWindow. Input events are then translated into messages, like #eventButtonPress:, that are sent to the WindowSensor of the Smalltalk Window.

The WindowSensor will pass along any critical information, such as it's time to become active, to its Smalltalk Window. Any important mouse and keyboard state information is now maintained in the InputState, of which there is usually only one global instance. Keystrokes will be queued up in the WindowSensors of the Smalltalk Windows to whom they were sent. The Controllers will be able to get at this information by polling their Sensors in the time-honored fashion.

When a Smalltalk Window becomes active, it tells its controller to start up a control loop very similar to that used in Smalltalk-80's past. The control loop of the Smalltalk Window will notice the fact that a keyboard event has been left in its Sensor's queue. The event is then handled as usual, for example, with a #keyboardActivity message.

## DISPLAY EVENTS

The steps for processing display messages such as WM\_PAINT are the same up to the point where they get to InputState>>process:. This method will translate the paint message into a Smalltalk message such as #eventDamage:. Then, as with input events, this message is passed along to the Smalltalk Window's WindowSensor. The fact that the Window has been damaged is recorded, along with what part of the Smalltalk Window needs repair, in the WindowSensor. The global list ScheduledControllers is also informed that a Smalltalk Window has been damaged.

ScheduledControllers having been duly informed of the need for a redisplay, the ControlManager, in its next pass through its control loop, will notice the fact and ask all scheduledControllers to check for events. These Controllers then

in turn ask their views to do the same. In the case of a WM\_PAINT, eventually some Smalltalk ScheduledWindow, for example, will notice that it has a pending damage event and process it. This results in a #displayOn: message being sent to the Smalltalk Window with the end result being that only the damaged areas are redisplayed.

## OVERALL

This is quite a lot of work to go through to take a Windows message and convert it into something Smalltalk can understand. One reason things are this complicated is that, in addition to converting Windows messages to Smalltalk messages, the input handler is also converting an event-driven system into a polling system. The Controller objects in Smalltalk still poll their Sensors just as they always have. The fact that the world outside is now event driven is hidden from them.

The upside to this is that at least part of the input-output model of the new release is familiar to developers. There are several aspects to the downside. First, this whole process is implemented by fairly complex code that is distributed among several classes. It is not to be casually modified. Second, either it's not fast enough or gets out of sync sometimes. On occasion, events, such as quick mouse clicks, may get lost in the shuffle. When dealing with display events, the programmer must realize that the redisplay is not processed immediately upon receipt of the WM\_PAINT message as it is in most Windows applications. The request for redisplay is effectively queued and performed when the relevant Controller gets around to it. In most cases, this is not a problem but, if the author of a new Controller is not careful how he requests and handles redispays, there may be a perceptible delay in the screen update.

In any case, Release 4 provides the expected ParcPlace level of cross-platform portability along with a whole new level of integration with the host environment. No easy task. Those working with Release 4 might want to start looking around down at this level of the system. An interesting exercise would be to eliminate the conversion to a polling interface and keep the event-driven nature of the system all the way to the Controllers. This can be done without sacrificing portability since the events are normalized before they even see Smalltalk. Some folks around the office have done this and are reported to be happy with the results. There are several other interesting ideas to try. Release 4 provides a whole new world of Smalltalk to explore. ■

---

*Greg Hendley is a member of the technical staff at Knowledge Systems Corporation. His OOP experience is in Smalltalk/V(DOS), Smalltalk-80 2.5, Objectworks/Smalltalk Release 4, and Smalltalk/V PM.*

*Eric Smith is a member of the technical staff at Knowledge Systems Corporation. His specialty is custom graphical user interfaces using Smalltalk (various dialects) and C.*

*They may be contacted at Knowledge Systems Corporation, 114 MacKenan Dr., Cary, NC 27511, or by phone at (919) 481-4000.*

---

# SMALLTALK

---

## COMES TO THE

---

## MAINFRAME,

---

## PART I

*Glenn J. Reid*

**R**ecently, our consulting firm has created an implementation of Smalltalk on a new platform. Our version of Smalltalk, syntactically equivalent to the PC versions, was implemented on an IBM System/370 mainframe. Throughout this document, we shall refer to our Smalltalk implementation as Smalltalk/370.

The development of Smalltalk/370 is the second phase in our ongoing effort to introduce object-oriented (OO) technology in a mainframe environment. Implementing a mainframe version of Smalltalk has been a challenging task, where the nature of the environment has presented special problems. In part 1 of this article, we present a description of some of those challenges with our own solutions, a discussion of some of the important aspects of the Smalltalk language, and our plans for future enhancement of this environment. In part 2, we will elaborate on those future plans, with a more technical discussion of our proposal to introduce variable typing within Smalltalk/370.

### HISTORICAL BACKGROUND

Prior to beginning the Smalltalk/370 project, we implemented several mainframe applications using a primitive form of OO technology, employing data abstraction and code encapsulation without the benefit of an OO programming language. Inheritance was not available, and programming and debugging tools often associated with OO languages were, of course, absent. Although, in each of these cases, the projects were considered successful (meaning that they were within budget and, in some cases, under budget), it was apparent that we would certainly not achieve order of magnitude reductions in delivery of software using this approach.

Reuse of code, for instance, was not achieved easily, although some small gains were made in this area. We felt that

to further improve productivity an OO language was required. At the time (and still to our knowledge), no OO languages were available for our environment. Some research pointed out that, while there were several OO languages in existence (for the PC), some had gained in popularity more than others. C++ was a candidate, as was Smalltalk. We noted that the Smalltalk syntax had been chosen as a base for at least one commercial object database management system (ODBMS) called GemStone by its creators at Servio Corp. Smalltalk seemed to offer a more "pure" OO environment, whereas C++ allowed the programmer to get "outside" the OO environment and program in the traditional manner. While some consider this a benefit of C++ (mainly for performance reasons that will be discussed later), the author feels that this type of activity should be allowed only under conditions strictly governed by the system, and as little as possible. For these reasons, and others, we felt that Smalltalk was the language of choice for our environment.

In the next section, we discuss some of the implementation aspects of Smalltalk/370 as well as some of the more important features of Smalltalk. Throughout, an attempt will be made to highlight areas that concern the larger system environment. To begin, some discussion is included on object persistence, which, while not included in the current version of Smalltalk/370, deserves some special mention due to its importance.

### OBJECT PERSISTENCE

Object persistence may be loosely defined as taking place when an object is still in existence, accessible as an object, or after a transaction or unit of work has been completed. The key phrase "accessible as an object," means that the object exhibits its normal behavior, reacting to messages and so on. Persistent objects reside on disk (although at times they may be in main memory), giving the user the appearance of an infinite virtual memory. A programmer no longer has to go outside the OO paradigm to use long-term data, meaning that the benefits of behavior captured with data are retained. Equally important, is the concept of concurrency, where the integrity of an object's data is maintained under simultaneous access by several users.

While the concepts and power of the Smalltalk language are immediately impressive, they do not currently extend into the realm of database management. Smalltalk comes down near (but not quite) to the level of a traditional programming language when it comes to dealing with file I/O. To preserve the behavioral aspects of data, it is required that data be stored in the Smalltalk "image" (i.e., a copy of RAM). Obviously, this approach is limited when the volume of data is too large. However, interfacing to traditional databases requires that objects be taken out of the OO environment. Such objects not only lose their behavioral aspect, they are also not directly accessible by an OO programming language. Objects are referenced in memory essentially by address, and to move an ob-

ject to disk and still treat it as an object is not a simple task. The object becomes a persistent object, something object database management systems (ODBMS) revolve around.

Current ODBMSs accomplish object persistence in two different ways:

1. Superimpose an object management layer over a traditional (e.g., relational) DBMS. The object management layer can automatically export/import objects through the underlying DBMS, making it appear to the user that objects are still in virtual memory.
2. Read/write the object intact using some new access method that accesses objects stored on disk in a format that supports OO behavior.

Both approaches can preserve the behavior of objects, as well as concurrent access by multiple users.

Both approaches require a significant effort to implement. Several ODBMSs are in existence today and these have been well documented elsewhere.

Building an ODBMS is certainly beyond the scope of our current efforts. We make mention of this technology here because we feel very strongly that persistent objects will be required in the future to fully exploit the potential of OO technology. This is holds doubly true in the mainframe environment, where databases often occupy gigabytes of disk space. Our inability to deal with this data as objects, unless using expensive and complex import/export processes, will place a limit on the gains we can make over traditional technology. It will probably be several years (at least) before ODBMS technology is proven reliable and performs well enough for large mainstream applications. For now, we have dealt with file I/O in a simple manner, supplying classes and methods to deal with traditional databases. Obviously, the less often objects have to be exported from the image, the better. It should be mentioned that recent mainframe operating system offerings such as IBM's MVS/ESA offer the potential for very large images. While this does not remove the necessity for an ODBMS, it may help to minimize the number of times objects have to be exported out of the OO environment.

## METHODS

In the PC version of Smalltalk, methods reside in the image with application data. Recognizing that we had to deal with a potentially large number of methods, without the benefit of an ODBMS, we opted to store our methods in a disk library outside of the image. An instance of the CompiledMethod class keeps track of the location of the method, including whether the method is loaded or on disk. At first reference, a method is loaded and kept in storage. In this way, only the methods actually used by an application are loaded, and we have virtually no limit on the number of methods we can create. Pointers to all loaded methods are flushed when the image is saved to prevent an application from loading an image in a subsequent session that contains an invalid method pointer.

## INTERPRETATION VS. COMPILATION

The PC version of Smalltalk is interpretive in nature, the "compiler" producing intermediate byte codes that are read in by a virtual machine at runtime resulting in execution of code in the virtual machine. While this process has been optimized to the point where it performs in an acceptable manner on a dedicated processor such as a PC, we felt that the virtual machine overhead might be too much in a mainframe where a large number of time-sharing applications are using a single processor.

“

It will probably be several years  
(at least) before ODBMS technology is  
proven reliable and performs well  
enough for large mainstream  
applications.

”

We have chosen to compile our Smalltalk code eliminating any overhead of a virtual machine. Note that some meanings of "compiled" in the OO world imply static binding of methods to messages. This is not the case with Smalltalk/370, as is discussed in the next section.

## DYNAMIC BINDING AND TYPING

If any area of OO technology should cause concern regarding performance, it is dynamic or late binding. Very simply, dynamic binding associates a method with a message at runtime, whereas static binding associates a method with a message at compile time. For illustration purposes, let us use the familiar example:

Snoopy bark

where Snoopy is an instance of class SmallDog, a subclass of Dog. Assume that class Dog contains the method bark, while class SmallDog contains a different implementation of the same method.

At compile time, static binding would change the message Snoopy bark to the equivalent of a direct function call to the bark method in class SmallDog.

Dynamic binding would route the message bark to the class SmallDog at runtime and select the bark method to be performed. The runtime difference between the two approaches is the processor time it takes to route the message, that is, find the correct method to be performed. This performance differ-

---

ence can sometimes be significant. It is interesting to note that in some critical cases, such as iteration (e.g., `whileTrue:`), conditional expressions (e.g., `ifTrue:`), and others, Smalltalk uses a `fastpath` static binding approach which does not, in fact, execute the published source code that is sent with the product.

If the `bark` method in the `Snoopybark` example was removed from the system, static binding would require that the method referencing it be recompiled as it now contains an invalid reference. Dynamic binding would require no such re-compilation, for at runtime the method would simply be inherited from class `Dog`. Change management systems are required to deal with the dependencies introduced by static binding.

Static binding can only occur if the compiler "knows" what class an object belongs to at compilation time. Smalltalk is a "typeless" language, meaning that in traditional Smalltalk, such associations of object to class are not possible at compile time. Recognizing that there are times when dynamic binding may produce unacceptable performance in a production system, we intend to explore extending Smalltalk with the concept of constraints on types (this extension would be part of a future version). We have confined ourselves at this time to investigating typing of named variables, excluding such things as intermediate results generated during expression evaluation. Potential candidates for typing are:

- dictionary variables (i.e., class, pool, and global variables)
- instance variables
- arguments
- named temporaries
- receivers

In all cases, the affected variable would be constrained to belong to a particular class or one of its subclasses (i.e., the variable has been "typed"). As was mentioned earlier, an in-depth discussion of our implementation of typing will be deferred to part 2 of this article.

Dynamic binding would remain the primary and preferred way of associating messages with methods. Typing would be used in situations that caused performance degradation or as a data validation tool. Intuitively, the best use of typing applies in high-use areas where typed languages can typically produce very efficient code. Coincidentally, these areas correspond to functions in Smalltalk that undergo few changes since they are integral to the basic functioning of the system. Some example preliminary candidates for typing might be `Arrays`, which are frequently used in the `at` and `atput` messages, and array indices that participate in `Integer` operations. In some actual program samples we have studied, up to 40% of message routing would be removed by typing these examples.

## OBJECT IDENTITY

Object identity refers to the faculty of the system to distinguish between two physically distinct objects whose contents are the same. Conversely, the system can also determine if two

objects, called by different names, are in fact the same object. This parallels our view of the world, in which physically distinct objects with identical characteristics are recognized as separate entities. Similarly, calling a real world object by two different names does not alter the fact that it is the same object. As this form of object identity is supported by Smalltalk, it is included in our implementation. In Smalltalk, object identity is supported by methods such as `==`, which returns true if two references point to the same object.

A desirable extension of the concept of object identity is the ability to recognize an object after it has undergone some change in its attributes. As an example, a child grown to adulthood should be recognized as the same object with some different attributes and behavior, not an entirely different object. This type of recognition is made possible through the use of versions. A versionable object may have more than one copy of itself, each copy with different attributes and/or different behavior. Different versions of an object may be kept in a "version set" to maintain the relationship between them.



A desirable extension of the concept of object identity is the ability to recognize an object after it has undergone some change in its attributes.



---

Simple version control might allow creation of multiple versions of attributes (i.e., of class and instance variables) as well as creation of multiple versions of behavior (i.e., methods). Under this scheme, an object would be linked to the version of the class under which it was created and would behave accordingly. This would allow the programmer to explore "what if" scenarios to explore alternative approaches to problem solving in a more controlled manner.

A considerably more complex version control might allow instances of an old version to be used as if they were instances of a new version and vice versa. This would require object behavior to simulate presence/absence of variables, etc.

It is our intent to introduce a simple version control mechanism in Smalltalk/370 that will allow the creation of multiple versions of classes and methods. This will serve to further enhance the exploratory nature of programming within the Smalltalk environment. One way this might be effected would be to store each class in its own `Array` of `n` elements, where each element would point to a different version of the class (a

version set). `CompiledMethods` would be contained in a version set pointed to by a `MethodDictionary`. Conventions could be set up such that the last element of a version set is the current version, the first element is the production version, etc.

The idea of version sets for classes and methods would allow the programmer to experiment with code implementations without using the unwieldy process of saving/restoring the image and overwriting good code simply to test out an idea. Additionally, the promotion process, where code is moved from a test level to a production level, might be simplified. Code could be moved from one level up to the next by adding a new version. Combined with some change management procedures, this might make the process of "backing out" a change simpler, if required.

“

In the mainframe environment, files are potentially accessible to all users of the system.

”

#### ENVIRONMENT

Smalltalk/370 is currently implemented on an IBM 3090 mainframe running IBM's MVS/ESA operating system.

Smalltalk/370 runs in two modes: under the control of IBM's time sharing option (TSO) and interactive system productivity facility (ISPF) using text-based monochrome IBM 3270 terminals or in batch mode without connection to a terminal. When connected to a terminal, cursor keys and tab keys are used to position the cursor.

ISPF is used as the terminal communications interface. Window (or panel) formats may be prepared in two ways: using an ISPF window definition facility (not in Smalltalk/370) that creates fixed format windows which Smalltalk/370 can interface with or through the use of ISPF dynamic areas where window format is dynamic and under control of Smalltalk/370. The second mode of window control is still under development. In either mode, windows can contain multiple panes that operate independently (scrolling, selection, etc.) as in the PC environment.

Editing of methods may be performed in two ways, both available through our online browser. The first is our own editing facility, with limited functionality at this time; the second, the editing facility of ISPF. Similarly, disk files are browsed or edited through a Smalltalk/370 browser that in turn uses ISPF edit or browse.

Smalltalk/370 is placed in batch or interactive mode by an application start-up message, which is passed to it when it is loaded. In both environments, it operates as a single-user ma-

chine. Intercommunication between users is probably possible through the operating system, but is currently not in place.

As mentioned earlier, file access is performed through supplied classes and methods that deal with the access protocol of each file system. Currently supported file access methods are queued sequential access method (QSAM) and virtual storage access method (VSAM), to supply both sequential and keyed file mechanisms, respectively. In the mainframe environment, files are potentially accessible to all users of the system. Access is controlled by the operating system (and other subsystems). Normally, users (i.e., programs) request access to files through the use of an external language call job control language (JCL). In the Smalltalk/370 environment, file access is performed through Smalltalk code, removing the requirement for coding JCL.

#### SUMMARY

Smalltalk/370 appears to be a viable programming environment. We have not yet encountered any unacceptable implementation concerns or design flaws during testing of the product. However, more work needs to be done before it can be turned over to the general programming population. Currently, Smalltalk/370 is deficient in interactive debugging tools and some other environment support applications.

As was mentioned at the outset of this document, Smalltalk/370 is phase 2 of a larger effort. Later phases necessarily will include work in the areas of training, integration into the existing production environment, and performance measurements. We hope that this experience will provide some insight into the integration of this important new technology in a corporate environment. ■

---

*Glenn J. Reid is president and founder of QSYS Systems Consultants Inc., a consulting and software development company whose main area of expertise is in the application of object-oriented technology. Architect of Smalltalk/370, Mr. Reid is currently involved in the development and application of a complete project life cycle approach to developing object-oriented systems in a mainframe environment.*

# Digitalk's Smalltalk/V Developers Conference '91

Digitalk held their inaugural Smalltalk/V developers conference at Universal Studios in Los Angeles on September 10th through 12th. The conference itinerary was packed with information concerning Digitalk's plans for the various dialects of Smalltalk, panel sessions, announcements, demonstrations, and interesting "sidelines" (thanks to Jim Anderson for the glasses, my kids loved them). Trying to keep things running on schedule proved to be a real problem. Each session seemed to have so much information to provide — no one would leave one session to make way for the next. This report presents an overview of many of these activities.

In all, the conference had over 200 attendees over the two and one-half days. The conference included sessions on Digitalk strategies, Smalltalk internals, and future directions of Smalltalk. There were sessions on topics such as front ending mainframe environments, changes required in methodologies, the use of CRC cards, and "Smalltalk and multimedia." There were also panels on training and education, team programming, and persistent objects. The keynote address was given by Dan Ingalls, who gave a superb talk on the history of Smalltalk. There were also (brief) time slots for product exhibitions, with a dozen or so companies demonstrating their wares.

## DIGITALK NAMED IBM PARTNER IN AD/CYCLE

The biggest piece of news (apart, of course, from the debut of *The Smalltalk Report*) actually occurred in New York on the opening day of the conference when IBM announced that Digitalk was the latest corporation to join the family of companies in the AD/Cycle partnership. It was never made clear whether the timing of this announcement was planned or whether Digitalk was just incredibly lucky. In any event, it did have a significant impact on the conference as it will, I'm sure, on Digitalk's (and Smalltalk's) future. One had to wonder, however, what effect this enhanced partnership will have on the relationship between Digitalk and its smaller customers.

IBM had a definite presence at the conference, with two sessions having IBM people describing the place Smalltalk has in IBM's future plans. For example, it was made clear that Smalltalk is "the language of choice" for development for OS/2 2.0. In fact, Digitalk stated their intention to release their version of V PM 2.0 almost immediately after IBM's release of 2.0. There was some discussion throughout the conference of whether Digitalk should wait until they can handle multiple process threads or leave this facility for version 2.1.

Also, Smalltalk/V PM 1.3 will be fully compliant with IBM's CUA '91 architecture.

## THE LAFKIT

Although the IBM AD/Cycle announcement was billed as the most significant development of the conference, the event that generated the most excitement by far was the demonstration of Digitalk's forthcoming LAFKit (Look and Feel Kit). This is Digitalk's first foray into the end user computing marketplace and is being positioned as a competitor to the Visual Basic and ObjectVision products. It was inspired by such systems as Interface Builder on the NeXT machine, the Fabrik system by Dan Ingalls, David Smith's InterCons, and Apple's HyperCard. Although it is possible to use Smalltalk as a scripting language, LAFKit applications are intended to be constructed with no knowledge of Smalltalk and no traditional programming skills. LAFKit offers a paradigm for describing computation based on connecting high-level "components" together using "wires." Components, such as a mail server or database interface, for example, are "black box" type objects that have a well-defined public protocol. The wires connecting them together represent the messages being sent. Unlike data flow diagrams, what is being described is truly the message flow between objects, not simply data flow within a program.

Digitalk has been successful in seamlessly integrating displayable components, such as windows and panes, with nondisplayable components, such as sorters and mailers. New applications developed using the LAFKit can be packaged as new components to be subsequently reused in other applications, can be packaged as an application, or can be converted to a dynamic link library (DLL). In fact, Digitalk touts that the LAFKit will allow integration with components developed using other languages, although no demonstration of this was given. There are few written references for the LAFKit — see the Byte article<sup>1</sup> for a brief introduction.

The demo given had Dan Goldman and Mike Teng each developing online their own version of a personalized mail system, using the existing "mailer" component to service all the low-level mail functions. In a matter of minutes, both were able to construct (almost) fully functional mail systems that had features such as send, receive, and edit messages and management of their own mailing lists. Although this was extremely impressive, it was made possible because the necessary

components for its construction already existed. It is not clear exactly how much work is involved in developing new components, but one can only conjecture that to get them right can't be easy. It is like generating code to be reused in ways never thought of before — it's a nice idea, but it comes only through sweat and trial and error!

Despite the obvious excitement, and there was much of it (Dan Shafer, for one, could hardly contain himself), many important questions remained unanswered. The most obvious was the availability of the LAFKit. Digitalk made no commitment as to its release date, but it seems clear that it won't be within "the next short while." Also, this new computational metaphor will undoubtedly require a new mechanism for debugging. It doesn't seem appropriate to get a walkback stack when you're in the middle of programming using an iconic visual tool. Digitalk gave no indication how they plan to address this issue. Finally, as Digitalk acknowledged, the success of LAFKit depends on the availability of good components. It is unclear at this stage what the base "palette," or library of components, will be. Presumably, there will be opportunities for third parties to supply new components.

#### WINDOWS VS. PM VS. UNIX VS. MAC

It seemed Digitalk received a bit of a surprise from the makeup of the attendees. As it turned out, the majority were V PM users rather than the expected V Windows users. This proved interesting as Digitalk seemed to be prepared to demonstrate that much of their new development, such as the LAFKit, is being done for V Windows first. When they pointed this out, they received howls from the audience. Another occurrence, which probably was not such a surprise, was the complaints from the small, but very vocal, V Mac community about the lack of effort being put into improving V Mac. In fact, when Digitalk took a straw vote as to whether users would prefer Digitalk to focus on adding more features to V Windows and V PM or to focus on compatibility across a greater range of platforms, including the Mac and UNIX, the vote was significantly in favor of the latter. This seemed to take the Digitalk people completely by surprise. My guess is that you can expect to see Digitalk giving greater priority to the UNIX world (the IBM RS-6000 and SPARCStation are the expected first target platforms) at the expense of V Mac users. Barbara Noparstak promised to let attendees know about any decisions Digitalk makes as a result of this vote through their own Scoop newsletter.

#### PLUGGABLE OBJECT LIBRARIES

In versions of Smalltalk prior to Smalltalk/V Windows and V PM, the Smalltalk image consisted of one large monolithic file. In Windows and PM, the image was split into runtime and development libraries with a user's application residing in an executable file. Small applications resulted in very small executables. V PM version 1.2 extended this idea to the notion of pluggable object libraries for packaging specific func-

tionality and provided tools for building them. Digitalk announced that they would introduce a hardware-independent representation of code in their 32-bit implementations that would allow pluggable object libraries to be portable across platforms. They also claimed their 32-bit architecture would offer performance improvements, where, for example, V PM 2.0 will operate more than twice as fast as V PM 1.3, while requiring significantly less space.

#### MOMENTA DEMONSTRATION

Momenta Corporation gave a surprise demonstration of their new "pen-based" computer. Without going into too much detail here, the box is a completely new design. Based on the 386SX chip, the machine offers a sleek design that uses a pen as its primary input device (although it does have a detachable keyboard). The machine is billed as the executive's notebook computer and contains such features as built-in fax functionality and power technology that allows the machine to operate for up to six hours at a time.

Why is Momenta of interest to the Smalltalk community? The application development environment is Smalltalk V/286 enhanced with Object Technology International's Envy/Developer and Acumen's Widgets/286 products. Independent software vendors wishing to develop applications for Momenta can purchase a developers package that includes software and training for a reasonable price, which is actually below the cost of the machine itself. Their intention seems to be to encourage as many developers to enter the market as possible.

Smalltalkers will also be interested in the enhanced object metaphor used throughout all applications. The idea is that instead of having behavior available only within a particular application (through the menus presented for the application), behavior is maintained by the objects generated by the application. For example, consider what happens when you move a section of data from your spreadsheet application to your word processor. Typically, the functionality of the spreadsheet is lost once it is pasted into the new application — it simply becomes an inert piece of your document. In Momenta, each object retains its own functionality, wherever it is located. So, in this example, selecting the spreadsheet within the word processor would give you access to all of the behavior applicable to spreadsheet objects. Developers of new software components will be expected to maintain this same functionality.

#### A NOTABLE EVENT

The most amazing feat of the entire conference actually had nothing to do with Digitalk and occurred at the strangest time. Everything had been running smoothly at the conference until Tim Andrews from Ontos Corporation, during the persistent objects session, stood up and began a sentence with "What's so much better about C++ rather than Smalltalk is

*continued on page 18...*



# Using and studying Smalltalk in the User Interface Institute

**Reports of current work in Smalltalk taking place in leading university and research laboratories.**

**W**e have been using Digitalk's Smalltalk/V in the User Interface Institute of the IBM T.J. Watson Research Center for a number of years. It has become our standard environment for prototyping user interface techniques and for the iterative development of applications. At the same time, Smalltalk has been an object of study: several researchers are examining different approaches to Smalltalk instruction; others are developing and testing tools intended to support experts' design and implementation of Smalltalk applications.

## APPLICATION AND USER INTERFACE PROTOTYPING

Our efforts in this area have focussed on two sorts of projects, the building and testing of a variety of user interface components in Smalltalk and the exemplification of these components in realistic applications.

So, for example, several researchers have created multiple versions of possible extensions to IBM's Common User Access (CUA), the specification of software look and feel for the Systems Application Architecture (contact is John Richards, jtr@watson.ibm.com). Through iterative evaluation and refinement, these researchers have explored how to support pervasive direct manipulation across a number of application types.

Addressing a more specific user interface arena, the Interactive Media Project (contact is Mark Laff, mrl@watson.ibm.com) is conducting research in tools, applications, and usability of object-oriented multimedia technology. Their ImpBuilder tool presents a graphical, direct-manipulation interface to interactive multimedia presentations. Media objects such as video, audio, graphics, animation, and text can be easily arranged in both space and time without requiring programming. Using ImpBuilder, these researchers have created several applications, including a biographical "scrapbook" and an informational "electronic magazine." Studies have investigated issues in design and navigation of such interactive multimedia applications.

## SMALLTALK INSTRUCTION

Smalltalk, although recognized as a good platform for rapid prototyping and software reuse, is widely regarded as difficult

to learn. Unlike learning a procedural language like Pascal or C, learning Smalltalk is dominated by browsing and code comprehension. Learners of Smalltalk typically experience a long, slow start-up phase in which they become familiar with the class hierarchy and object-oriented computational model but do little meaningful work (our colleague Dave Smith calls this "climbing the Smalltalk mountain").

The Minimalist Tutorial and Tool Set project (MiTTS; contact is Mary Beth Rosson, rosson@watson.ibm.com) focusses on users' earliest experiences with Smalltalk. The project's goal is to provide a short (four- to six-hour) introduction to Smalltalk/V that offers programmers experienced with other languages a taste of what programming Smalltalk applications is all about, hopefully leaving them motivated and confident enough to begin climbing the mountain. A short manual guides new users through explorations and modifications of a simple example application (a blackjack game). The learning activities are supported by a Bittalk Browser (which filters out classes not relevant to the application) and by a View Matcher (which coordinates multiple views of the example application, views containing information normally accessed through independent tools).

The Molehill project (contact is Kevin Singley, singley@watson.ibm.com) combines elements of intelligent tutoring technology with hypermedia and key word searches to create an instructional environment more structured than exploratory learning, yet less constraining than traditional tutoring. The goal of Molehill is to provide learners with online tutoring support through a semistructured curriculum of simple but meaningful projects. Molehill supports code comprehension and browsing as well as code generation, something that has not been attempted in other programming tutors. Because Smalltalk learners are often experienced procedural programmers, the Molehill project also addresses the transfer from procedural to object-oriented programming, exploiting learner's prior knowledge where appropriate while at the same time highlighting differences.

## SMALLTALK TOOLS

We have added tools to the environment to make our programming activities more effective. As researchers, we have been also been experimenting with new ways of working with the Smalltalk language and environment.

The Change Manager (contact is Jerry Archibald, arch@watson.ibm.com) is intended to make our Smalltalk development work more efficient and reliable. The tool's main interface is a browser that can be used in conjunction with or in place of a class hierarchy browser. The programmer creates and then chooses among "change groups;" these groupings correspond to code development categories. Any work accomplished while a change group is active is associated with that group and can be manipulated (e.g., filed out) as a unit. The level of granularity is the individual Smalltalk method, which means that a single class can be associated with multiple change groups, with only the relevant methods included. This has proven to be especially useful when development work includes additions to preexisting classes (e.g., collections, magnitudes).

Another project is developing and testing an example-based reuse documentation tool, the reuse View Matcher (contact is Mary Beth Rosson). This tool coordinates different views of an example application that uses a target class being considered for reuse. These views include a concrete depiction of the class' usage protocol, organized by typical usage episodes (e.g., starting up the application); the associated code browser is a "senders" browser, so that programmers are immediately directed to an example usage of a method defined for the tar-

get class, rather than the method's implementation. Another view of the example application is an "object map" that schematizes instance variable relationships among the important objects in the example application.

The work on the Portia environment (contact is Eric Gold, egold@watson.ibm.com) focusses on facilitating "instance-centered" (as contrasted with code-centered) approaches to application design and development. The hope is to make the creation, location, examination, manipulation, and reuse of instances a more common and natural component of Smalltalk development activities. Support for this consists of broad-based modifications to existing system tools (e.g., the workspace, the class hierarchy browser) as well the provision of new tools (e.g., object repository, an object locator). ■

---

*Mary Beth Rosson is a research staff member at IBM's T.J. Watson Research Center in Yorktown Heights, NY, where she has been since 1982. Her current research centers on the cognitive processes underlying the design and implementation of software, especially the consequences of the object-oriented programming paradigm for these activities. She can be reached at the User Interface Institute, IBM Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, (914) 784-7738, rosson@watson.ibm.com.*

*continued from page 16...*

that..." He didn't get to finish it — it literally set the fire alarms in the hotel ringing. Amazing, these Smalltalk people can pull anything off!

**SUMMARY**

This article provided just a sampling of the information that was shared at the developers conference. Almost all of the sessions were recorded on audio cassette, although I am not sure whether or not they are being made available to the general public. If you would like more information concerning the

conference, I suggest you contact Digitalk directly (electronically or by phone) or watch for information in their Scoop newsletter.

Overall, the feeling with which I think most people left the conference was a sense that Smalltalk is moving forward and that Digitalk has a reasonably good sense of where it would like to go. They have the problem of being pulled in many directions and of trying to manage their growth to meet those demands. For a first attempt at a developers conference, I think Digitalk has much of which to be proud! ■

**REFERENCES**


[1] Ullman, E. OOP made visual: Digitalk's LAFKit, *Byte Magazine*, vol. 16:8 August 1991.

---

*Paul White is an Editor of The Smalltalk Report. Currently, he is a professor of computer science at Carleton University. He is also a founding member of The Object People, a firm specializing in object-oriented technology. He can be reached on CompuServe at 70524,3613 for comments.*

**Universal Database**  
**OBJECT BRIDGE™**

This developer's tool allows Smalltalk to read and write to:  
**ORACLE, INGRES, SYBASE, SQL/DS, DB2, RDB, RDBCDD, dBASEIII, Lotus, and Excel.**



**Intelligent Systems, Inc.**

506 N. State Street, Ann Arbor, MI 48104 (313) 996-4238 (313) 996-4241 fax

## WHAT THEY'RE SAYING ABOUT SMALLTALK

### Excerpts from industry publications

... With respect to ... the inclusion of system-defined types — Smalltalk, C++, and CLOS nearly run the entire gamut. In Smalltalk types and classes are identified: There is no semantic difference between those supplied by the system and those defined by the user. In C++, the only methodical value types — those for which the user can declare methods — are user-defined classes. In CLOS, types and classes are separate concepts; every class corresponds to a unique type, but not every type has a class behind it. However, in order to cover all system-defined data types, CLOS defines a set of classes which span the preexisting Common Lisp types. One set of types spans a second set of types if every object in the second set is a member of at least one type in the spanning set. This set is large enough to encompass most representational distinctions within Common Lisp implementations, but small enough that each system-defined data type is directly covered by a single spanning type. This allows implementors to retain their historic, low-level optimizations based on representational types ...

... With respect to...extensibility for new representational types — neither Smalltalk, C++, nor Lisp allows the user to define representational types. For Lisp, there is a defined way to recognize a representational type ...

... With respect to ... the use of declarative types — C++ uses strong, static typing as much as possible, and Smalltalk uses only runtime typing. Common Lisp has a rich, extensible, but optional declaration syntax. Although implementations are not required to do any static type-checking based on a program's use of type declarations, a number of compilers take full advantage of this feature, especially for optimization ...

... Smalltalk — which supports only single inheritance — signals an error if the same-named instance variable is specified in a subclass. With disjoint unions, there are as many copies of the conflicting item as there are superclasses with an item by that name. C++ uses this mechanism for member data elements. Only some of the named elements are visible in the C++ derived class, because C++ supports name encapsulation between subclasses and super classes. CLOS creates a single composite description for a slot from all the inherited slots of the same name ...

... Different languages provide a variety of tools positioned along a spectrum for aiding software development. At one end of the spectrum are languages like C++, in which the only tools are external development environments. In the middle are languages like Smalltalk that provide residential development environments having access to every part of the language and its implementation. At the other end of the spectrum are languages like CLOS that provide linguistic mechanisms to support development ...

LISP, John K. Foderaro, *Communications of the ACM*, 9/91

... The Momenta [pen-based operating environment] will be bundled with a generous selection of MSE starter applications, including an address book, appointment calendar and fax manager. The MSE starter applications are state-of-the-art programs. They are object-oriented, which simplifies sharing of information by several applications, and allow you to create compound docu-

ments that contain a variety of embedded data objects. However, the MSE applications can only be created on Momenta's proprietary development system, which is an advanced derivative of the Smalltalk programming language. Although the Momenta development system provides an innovative and advanced language for object-oriented programming, it is considered an idiosyncratic tongue by some of the third-party software developers that have been approached by Momenta. Considering the heavy lobbying by Go and Microsoft, Momenta may find it tough to lure enough developers. Without sufficient third-party applications, it could be forced to rely on selling hardware — which, ironically, might be used primarily with pen-based operating systems developed by its rivals.

Momenta poised to penetrate pen computer market,  
Frederic E. Davis, *PC Week*, 9/2/91

... From their demonstrations, I have no doubt that the proprietary Windows development offerings such as Realizer, GFA Basic, VZ Programmer, and, to a lesser degree, Actor and the two Smalltalks, can do great things in the hands of their developers and those who invest heavily in learning and understanding the programs' various strengths and weaknesses. However, I didn't find myself as instantly hooked by any of them as I was by Visual Basic and QuickC for Windows ...

Windows developers finally have the right tools for the job,  
Steve Gibson, *InfoWorld*, 9/16/91

... Improved access to standard software development and modeling tools should be a major by-product of Sematech's move to open CIM. Because the first approach relied on proprietary data structures and programming interfaces, it would have required development tools dedicated to the Sematech CIM architecture. Sematech would have had to create such tools itself or convince ISVs to do so. Since it uses more standard, open interfaces, the new Sematech CIM architecture will be able to use less expensive, commercially available software development tools. In fact, a major piece of the new architecture is an expanded, object-oriented modeling and computer-aided software engineering component. Based on commercial tools including the Smalltalk object-oriented development language and the Object Management Group's evolving object services, the Sematech CIM development architecture should make it easier for users to model, prototype, code and make changes to CIM systems. Sematech officials admit their decision to go to open CIM carries some risks. For one thing, neither the OSF Distributed Computing Environment nor the Object Management Group object infrastructure is complete. Also, the decision to base the architecture exclusively on OSF/DCE necessarily excludes members' existing proprietary-based CIM architectures. Although existing applications built on the proprietary architectures can be included in the object-oriented models of the new architecture, they won't share interfaces or data structures ...

Sematech crafts an open CIM strategy,  
Jeff Moad, *Datamation*, 9/1/91

## PRODUCT ANNOUNCEMENTS

*Product Announcements are not reviews. They are abstracted from press releases provided by vendors, and no endorsement is implied. Vendors interested in being included in this feature should send press releases to our editorial offices, Product Announcements Dept., 91 Second Ave., Ottawa, Ontario K1S 2H4, Canada.*

### Instantiations introduces a team development environment for Smalltalk

Instantiations has announced its new Convergence/Team Engineering Environment, the first multiuser, multirepository team programming environment for Smalltalk. Convergence/Team enables programming teams to effectively develop commercial and industrial applications using ParcPlace Systems' Objectworks\Smalltalk.

Convergence/Team combines the productivity of Smalltalk with a powerful team development environment, giving groups of programmers the power to create large-scale, production-quality Smalltalk applications.

Convergence/Team's innovative shared repository environment allows team members on different types of computers to simultaneously create, browse, access, and share code. The size and number of repositories is limited only by available disk space.

Instantiations has also announced Convergence/Team Services, a full range of support services designed to rapidly move development teams up the learning curve and to accelerate the design and implementation of commercial and industrial applications using Convergence and Smalltalk.

*For more information, contact Instantiations, Inc., 921 S.W. Washington, Ste. 312, Portland, OR 97205, (503) 242-0725.*

### Smalltalk bundled with Tigre

At OOPSLA '91, Tigre Object Systems, Inc., of Santa Cruz, CA announced a new agreement with ParcPlace Systems of Mountain View, CA. Tigre now bundles the Tigre Programming Environment with ParcPlace's Objectworks\Smalltalk object-oriented language, making it easier and less expensive for new users to get started with object-oriented technology. The Tigre Programming Environment, which uses Objectworks\Smalltalk as its scripting language, lets developers create state-of-the-art graphical user interface (GUI) programs that run, without parting, on Macintosh II, Microsoft Windows 3.0, and all popular UNIX workstations. With Tigre, GUI creation is significantly faster than with only Smalltalk or C++.

*For more information, contact Tigre Object Systems, 3004 Mission St., Santa Cruz, CA 95060, (408) 427-4900, or fax (408) 457-1015.*

### Servio announced GeODE—the first code-free development environment for building object database applications

Servio Corporation has announced the GemStone Object Database Development Environment, GeODE, a suite of development tools for visually and graphically designing and building ODBMS applications. GeODE is a code-free environment targeted at commercial application developers building systems such as order entry, inventory control, and general ledgers as well as more complex systems such as biomedical, financial analysis, and manufacturing.

GeODE is platform- and GUI-independent, allowing developers to build applications that run on any platform and native windowing system.

GeODE was designed for the applications developer whose ex-

perience is based on traditional programming methods and languages. GeODE provides a tightly integrated set of tools that allows the developer to visually create screens, graphically direct application flow, and customize the application through graphical point-and-click interaction. Other GeODE tools include debuggers, cross-reference tools, and graphical browsers. Since GeODE is an interactive, graphical environment, it gives developers immediate feedback on application design and implementation decisions.

*For more information, contact Servio Corporation, 1420 Harbor Bay Pkwy., Alameda, CA 94501, (415) 748-6200, or fax (415) 748-6227.*

### Momenta chooses Smalltalk/V language for its new pentop computer

Momenta International has announced that its pentop computer uses Digitalk Corporation's Smalltalk/V language as the operating environment for its pen-based interface.

Momenta's pentop systems offer a comprehensive set of integrated applications which were developed in Smalltalk/V. Momenta's development environment allows for independent software developers to easily develop applications for the pentop system that are consistent with the look and feel of Momenta's applications. This integrated approach to program development provides Momenta computer users with consistent applications based on a common graphical user interface, dramatically decreasing learning time.

The new computer will be shipped with a number of essential programs, among them a spreadsheet, memo maker, presentation maker, appointment calendar, faxer, personal journal, chart maker, and hand printing recognition trainer. All the programs are pen-optimized (completely controlled by the pen) and take full advantage of the Momenta user interface. Momenta offers a special ISV program to encourage future development on its system, and ISVs have already begun developing other Smalltalk/V-based applications.

*For more information, contact Digitalk, Inc., 9841 Airport Blvd., Los Angeles, CA 90045, (213) 645-1082, or fax (213) 645-1306.*

### Synergistic Solutions enhances Smalltalk\SQL

Synergistic Solutions Inc. has announced additional platform support for Smalltalk\SQL, the portable database interface for Smalltalk. The product works in conjunction with ParcPlace Systems Objectworks\Smalltalk to enable graphical user interface (GUI) applications to access information stored in relational databases. Corporations can now leverage their existing data with the benefits of object-oriented development.

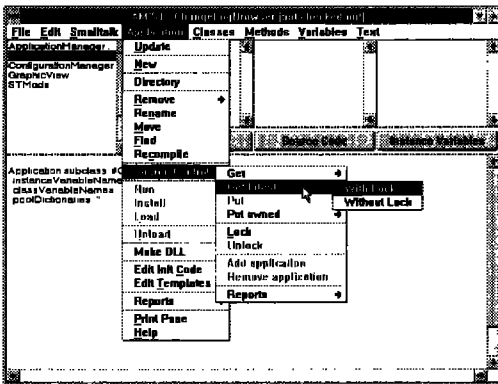
Direct database support is currently available for the Sybase and Oracle databases. DB2, Informix, Ingres, Rdb, and other databases may be accessed through a variety of gateway products. Several other direct interfaces will be announced later this year. The combination of direct and gateway support for popular databases provides flexibility for developers. Organizations can now rapidly develop portable object-oriented GUI applications and access data in both relational and object-oriented databases.

Smalltalk\SQL is a set of portable classes that encapsulate

# Take Control of Your

# Applications with

Bring your large, complex object-oriented applications under control with AM/ST, the Application Manager for Smalltalk/V. The AM/ST Application Browser helps both individuals and development teams to create, integrate, maintain, document, and manage Smalltalk/V application projects.



### Price List

DOS V	\$150
DOS V/286	\$395
Macintosh V/Mac	\$395
OS/2 V/PM	\$475
Site Licenses	CALL

### New Productivity Tools I

Windows 3.0	
V/Windows	\$475
Change Browser*	\$195
Source Control** PM or Windows	
first copy	\$1,595
subsequent	\$595



**Coopers  
& Lybrand**

SoftPert Systems Division  
One Main Street  
Cambridge, MA 02142  
(617) 621 3670 or (617) 621 3671 Fax

*"With AM/ST, Smalltalk/V is a leader in serious multi-person development."*

David Orstein, Sage Software

*"Gave me a real edge in Design and Analysis"*

Hal Hildebrand, Anamet Labs

- **Applications Hierarchy**  
Every class has an owner.  
Functional view across classes and related methods within classes.  
Applications port easily across platforms.

- **Automatic Documentation**  
Revision history for each method.  
Analysis and design reports.  
Customizeable documentation templates.

- **Source Control**  
Integrate work of several users.  
\* Save and browse multiple revisions easily.  
\*\* Check-in, check-out, and lock source code.  
Customize code templates.  
Develop in a LAN environment.  
Deliver applications without AM/ST.

- **Static Analysis Tools**  
Application consistency reports.  
Graphical views of hierarchies.  
Cross-reference of variable and method usage.  
Up-to-date method index.

- **Dynamic Analysis Tools**

database access through an object-oriented metaphor called SQL Agent. Developers simply "plug in" the appropriate agent for their target database. There are no modifications necessary to applications using ANSI SQL. Pluggable SQL agents are capable of instantiating objects based on arbitrary queries including relational joins. Smalltalk/SQL allows organizations to quickly develop object-oriented applications using their existing database investment and provides a migration path toward object-oriented databases. The resulting applications are flexible enough to respond to the rapidly changing business environment and competitive pressures.

For more information, contact Synergistic Solutions Inc., 63 Joyner Dr., Lawrenceville, NJ 08648, (908) 855-7634.

### ParcPlace Systems announces FACETS\4GL 2.0

ParcPlace Systems has announced that the FACETS\4GL fourth generation language (4GL) application development tool for Object-works\Smalltalk now includes an interface builder. FACETS\4GL 2.0 enhances the capabilities of traditional 4GLs with a graphical user interface (GUI) builder and provides a migration path from 4GLs to object-oriented technology. FACETS\4GL is developed by Reusable Solutions and marketed by ParcPlace Systems.

The new FACETS\4GL interface builder includes several styles of event-driven buttons, default styles to easily change the look and feel of an application, a choice of fonts, and a color palette.

With the FACETS interface builder, applications can be enhanced easily with different kinds of buttons, including bordered, shadowed rectangular, transparent, opaque, raised, rounded, shadowed rounded, radio, and check box. And event-driven architecture ensures that actions and scripts are attached to the buttons. In addition, color selection has been simplified so users can point and click to select from a color palette. Users can also change the default styles to alter the look and feel of an entire application in one step.

For more information, contact ParcPlace Systems, 1550 Plymouth St., Mountain View, CA 94042, (415) 691-6700, or fax (415) 691-6715.

### Object Migrator for Smalltalk/V Win and Smalltalk/V PM

Hierarchical Applications Limited has announced the availability of its Object Migrator product for Smalltalk/V Windows and PM. Object Migrator allows the Smalltalk developer to segregate modified classes from the base or unmodified methods. It aids the developer in identifying those methods that have been changed since the last Smalltalk release or since the last time a Smalltalk image was modified.

Object Migrator also aids the Smalltalk developer in migrating modified classes from one Smalltalk release to another or from one Smalltalk image to another. It allows Smalltalk developers to share classes more freely since Smalltalk can be made to FILE-OUT either all modified classes or only selected classes from the hierarchy of modified classes.

The modified class hierarchy is maintained in a completely transparent manner which frees the Smalltalk/V developer to concentrate on the software problem at hand rather than on the maintenance of the Smalltalk class hierarchy across Smalltalk releases.

For more information, contact Hierarchical Applications Limited, 7491 N. Federal Hwy., Ste. 277C5, Boca Raton, FL 33487, (512) 838-1234.

### Digitalk announces Smalltalk/V PM 1.3 and V PM Database Interface

Digitalk announced Smalltalk/V PM Release 1.3 and the Database Interface for Smalltalk/V PM. Smalltalk/V PM Release 1.3 fully supports the new CUA architecture, known as CUA '91, which includes the new advanced controls IBM intends to ship with OS/2 2.0. The Database Interface provides simplified access to IBM's OS/2 Extended Edition Database Manager and the Microsoft SQL Server.

For more information, contact Digitalk, Inc., 9841 Airport Blvd., Los Angeles, CA 90045, (213) 645-1082, or (800) 922-8255.

# You've heard all the buzzwords surrounding object databases...

**inheritance**  
*versioning*  
**encapsulation**  
**methods**  
*messages*

## Let GemStone® introduce you to a few more...

*available today*  
**stable releases**  
*mission critical*  
**dependable support**

After introducing the first commercial object database in 1987, Servio has continually set the standard for stability and reliability. Our object database system, GemStone, has undergone years of refinement by incorporating the feedback of the world's largest installed base of object database customers.

With our second generation release, GemStone sets the new standard for performance in multi-user, mission critical environments. Only GemStone supports both C++ and Smalltalk, the most popular object programming languages. In addition, GemStone offers integration of corporate information via gateways into SQL database systems such as Sybase as well as non-SQL database systems.

**Call Servio to find out more about  
the latest buzzwords in object databases.**

**(800)243-9369**



**SERVIO**

**Servio Corporation**

1420 Harbor Bay Parkway, Alameda, CA 94501, (415)748-6200, Fax (415)748-6227

Copyright 1991, Servio Corporation. GemStone is a registered trademark of Servio Corporation.  
All other product and company names may be trademarks of the respective companies with which they are associated.

# Putting Smalltalk To Work!

1980	Smalltalk Leaves The Lab.	We were there.
1984	First Commercial Version Of Smalltalk.	We were there.
1985	First Industrial Quality Smalltalk Training Course.	We were there.
1987	First Fully Integrated Color Smalltalk System.	We were there.
1988	Responsibility-Driven Design Approach Developed.	We were there.
1991	Smalltalk Mainstreamed in Fortune 100 Applications.	WE ARE THERE.
<b>NEW!</b>	First multi-repository, group programming environment.	<b>NEW!</b>

## Smalltalk Technology Adoption Services

Technology Fit Assessment  
Expert Technical Consulting  
Object-Oriented System Design/Review  
Proof-of-Concept Prototypes  
Custom Engineering Services & Support

## Smalltalk Training & Team Building

Smalltalk Programming Classes:

Objectworks Smalltalk Release 4  
Smalltalk V/Windows      V/PM      V/Mac

Building Applications Using Smalltalk

Object-Oriented Design Classes:

Designing Object-Oriented Software: An Introduction  
Designing Object-Oriented Systems Using Smalltalk

Mentoring:

Project-focused team and individual learning experiences.

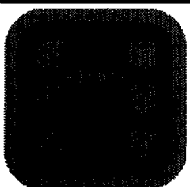
## Smalltalk Development Tools

**NEW!** Convergence/Team Engineering Environment™

Multi-user/shared repository development environment for teams creating production-quality Smalltalk applications.

Convergence/Application Organizer Plus™

Version management, development tools, and improved code modularity for individual Smalltalk developers.



# Instantiations, Inc.

1.800.888.6892



# WINDOWS AND OS/2: PROTOTYPE TO DELIVERY. NO WAITING.

In Windows and OS/2, you need prototypes. You have to get a sense for what an application is going to look like, and feel like, before you can write it. And you can't afford to throw the prototype away when you're done.

With Smalltalk/V, you don't.

Start with the prototype. There's no development system you can buy that lets you get a working model working faster than Smalltalk/V.

Then, incrementally, grow the prototype into a finished application. Try out new ideas. Get input from your users. Make more changes. Be creative.

Smalltalk/V gives you the freedom to experiment without risk. It's made for trial. And error. You make changes, and test them, one at a time. Safely. You get immediate feedback when you make a change. And you can't make changes that break the system. It's that safe.

And when you're done, whether you're writing applications for Windows or OS/2, you'll have a standalone application that runs on both. Smalltalk/V code is portable between the Windows and the OS/2 versions. And the resulting application carries no runtime charges. All for just \$499.95.

So take a look at Smalltalk/V today. It's time to make that prototyping time productive.

## Smalltalk/V

Smalltalk/V is a registered trademark of Digitalk, Inc. Other product names are trademarks or registered trademarks of their respective holders.

Digitalk, Inc., 9841 Airport Blvd., Los Angeles, CA 90045  
(800) 922-8255; (213) 645-1082; Fax (213) 645-1306

### LOOK WHO'S TALKING

#### HEWLETT-PACKARD

*HP has developed a network troubleshooting tool called the Network Advisor. The Network Advisor offers a comprehensive set of tools including an expert system, statistics, and protocol decodes to speed problem isolation. The NA user interface is built on a windowing system which allows multiple applications to be executed simultaneously.*

#### NCR

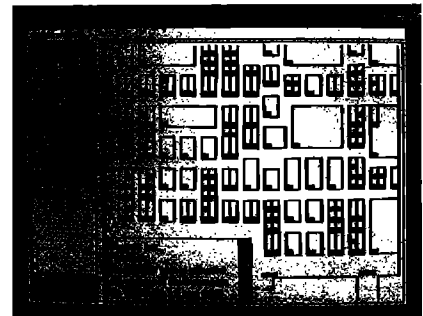
*NCR has an integrated test program development environment for digital, analog and mixed mode printed circuit board testing.*

#### MIDLAND BANK

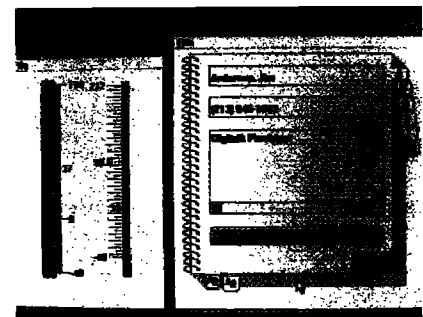
*Midland Bank built a Windowed Technical Trading Environment for currency, futures and stock traders using Smalltalk V.*

## KEY FEATURES

- World's leading, award-winning object-oriented programming system
- Complete prototype-to-delivery system
- Zero-cost runtime
- Simplified application delivery for creating standalone executable (.EXE) applications
- Code portability between Smalltalk/V Windows and Smalltalk/V PM
- Wrappers for all Windows and OS/2 controls
- Support for new CUA '91 controls for OS/2, including drag and drop, booktab, container, value set, slider and more
- Transparent support for Dynamic Data Exchange (DDE) and Dynamic Link Library (DLL) calls
- Fully integrated programming environment, including interactive debugger, source code browsers (all source code included), world's most extensive Windows and OS/2 class libraries, tutorial (printed and on disk), extensive samples
- Extensive developer support, including technical support, training, electronic developer forums, free user newsletter
- Broad base of third-party support, including add-on Smalltalk/V products, consulting services, books, user groups



This Smalltalk/V Windows application captured the PC Week Shootout award—and it was completed in 6 hours.



Smalltalk/V PM applications are used to develop state-of-the-art CUA-compliant applications—and they're portable to Smalltalk/V Windows.