The Smalltalk Report

The International Newsletter for Smalltalk Programmers

September 1991 **Volume 1 Number 1**

THE COMMERCIAL EVOLUTION OF SMALLTALK

By Abdul K. Nabi

Contents:

Over the last decade, Smalltalk has made a dramatic evolution from a visionary software research project into a commercial environment that is spearheading object-oriented programming, the next step in software technology.

The focus of this article is the evolution of commercial Smalltalk from the early market to today's commercial success and what the future may hold. In addition to the evolution of Smalltalk, the changes in the needs and demands of software development that contributed to the spread of Smalltalk in commercial application development will be discussed.

THE EARLY YEARS OF SMALLTALK

Xerox, in the interest of broadening the Smalltalk base, licensed Smalltalk to several hardware vendors (Apple, Hewlett-Packard, and Tektronix) and one startup software vendor (Softsmarts). This groundwork led to the creation of a Smalltalk marketplace.

Xerox sold the Smalltalk environment bundled with its proprietary graphics workstations, which were quite expensive. These workstations pioneered the idea of high-performance, graphical, interactive desktop computers. Xerox used Smalltalk internally to develop custom document and information management applications. One major application created in 1982, was a desktop publishing system for *The New York Times*. The next year, Xerox developed Analyst, one of the first and best known commercial Smalltalk applications.

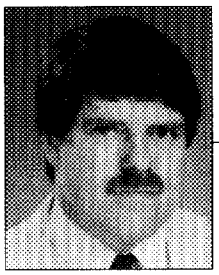
The Analyst is an integrated information management system incorporating document processing and layout, a spreadsheet, charting, database, hypertext, links, and a world map. The embedded object architecture, level of integration, and seamlessness of the applications is outstanding even by today's standards. Analyst's capabilities are unmatched by any other software package, partly because Analyst is written in Smalltalk. Analyst is still being sold as a commercial, end-user application by Xerox Special Information Systems.

To put the early years of Smalltalk at Xerox in some perspective, Smalltalk-80 was created about the same time the IBM-PC was introduced. The Analyst was introduced two years before the Apple Macintosh.

One of the early users of the exotic Xerox workstation and the Analyst were US intelligence agencies. These agencies required the horsepower, graphics, and high-performance development environment that the Xerox workstation and Smalltalk provided to create interactive analysis workstations. This helped Smalltalk emerge from the lab into the marketplace. However, Xerox did not market these workstations and Smalltalk-80 to the general software development market.

Tektronix was also an early pioneer of commercial Smalltalk, delivering its first Smalltalk in 1985 (which, like Xerox's, ran on a proprietary workstation). Unlike Xerox, Tektronix was actively marketing the environment as a development tool, and at a considerably lower cost (since by 1985 processors like the 68000 that Tektronix used made low-cost workstations possible). Tektronix also used Smalltalk to develop custom software for their workstations (such as a front-end for VLSI test equipment). Although successful

continued on page <None>...



John Pugh

Paul White

EDITORS' CORNER

Welcome to the first issue of *The Smalltalk Report*! The Smalltalk community has long yearned for its own publication. With your help, *The Smalltalk Report* will fill the void.

The use of Smalltalk within industry is expanding rapidly. Only industry insiders know many of the exciting developments that are taking place and, as Abdul Nabi points out in his lead article, they are not permitted to share them. Many companies believe it to be a strategic advantage to be using Smalltalk and don't wish their competitors to be aware they are using it. So, the language that started it all (with apologies to Simula) is now seen by many as the development system of choice for the 1990s. As our good friend Dave Thomas (paraphrasing Winston Churchill) is quoted as saying, "Smalltalk is the worst possible programming environment — until compared with all other programming environments."

Our aim at the *The Smalltalk Report* is to support the growth of Smalltalk as a development vehicle for object-oriented systems and to serve as a focal point for sharing ideas and experiences gained from the employment of Smalltalk technology in areas as diverse as real-time embedded systems and financial systems. By publishing nine times a year, we will be able to bring you timely information on new software releases of all dialects of Smalltalk, third-party products, class libraries, books, industry news, etc. We will address all aspects of application development with Smalltalk, e.g., project management, analysis and design, development tools, language issues, metrics, performance issues, and education and training.

In the lead article of our premiere issue, Abdul Nabi takes us on a tour through time — from the early days of Smalltalk at Xerox PARC to current implementations and applications. He discusses the issues faced by Smalltalkers in the past, problems that have been solved, and the challenges that lie ahead. He explains how, and why, the commercial evolution of Smalltalk has unfolded in the manner it has and speculates where this evolution will lead.

In this first issue, we also introduce two of our regularly appearing columns. In "Getting Real," Juanita Ewing discusses the issues of developing industrial strength applications with Smalltalk. In her first column, she addresses the pros and cons of employing class ownership as a vehicle for code management within a programming team. In their GUI column, Greg Hendley and Eric Smith tackle the topic of graphical user interfaces. In the first installment of a two-part column, they discuss giving application windows dialog box functionality in Smalltalk/V PM. In future issues, watch out for other columns; on design from Rebecca Wirfs-Brock and on "Smalltalk with Style" from Ed Klimas and Suzanne Skublics.

Our authors and columnists are willing to stand up and be counted, expressing their own personal, sometimes controversial, opinions. To make this forum truly effective, we encourage you to "jump into the foray" and let your ideas be heard. Use our "Messages" corner as a soapbox to vent your own opinions. In this issue, Allen Wirfs-Brock laments the absence of a conference where Smalltalk users and developers can get together and share their work.

Also in this issue: Charles Rovira suggests enhancements to the compress changes facility in Smalltalk/V Windows, Ralph Johnson describes the Typed Smalltalk project at the University of Illinois, and Jim Salmons reviews the WindowBuilder/V product from Acumen.

The Smalltalk Report is written by Smalltalkers for Smalltalkers; we encourage you to contribute.

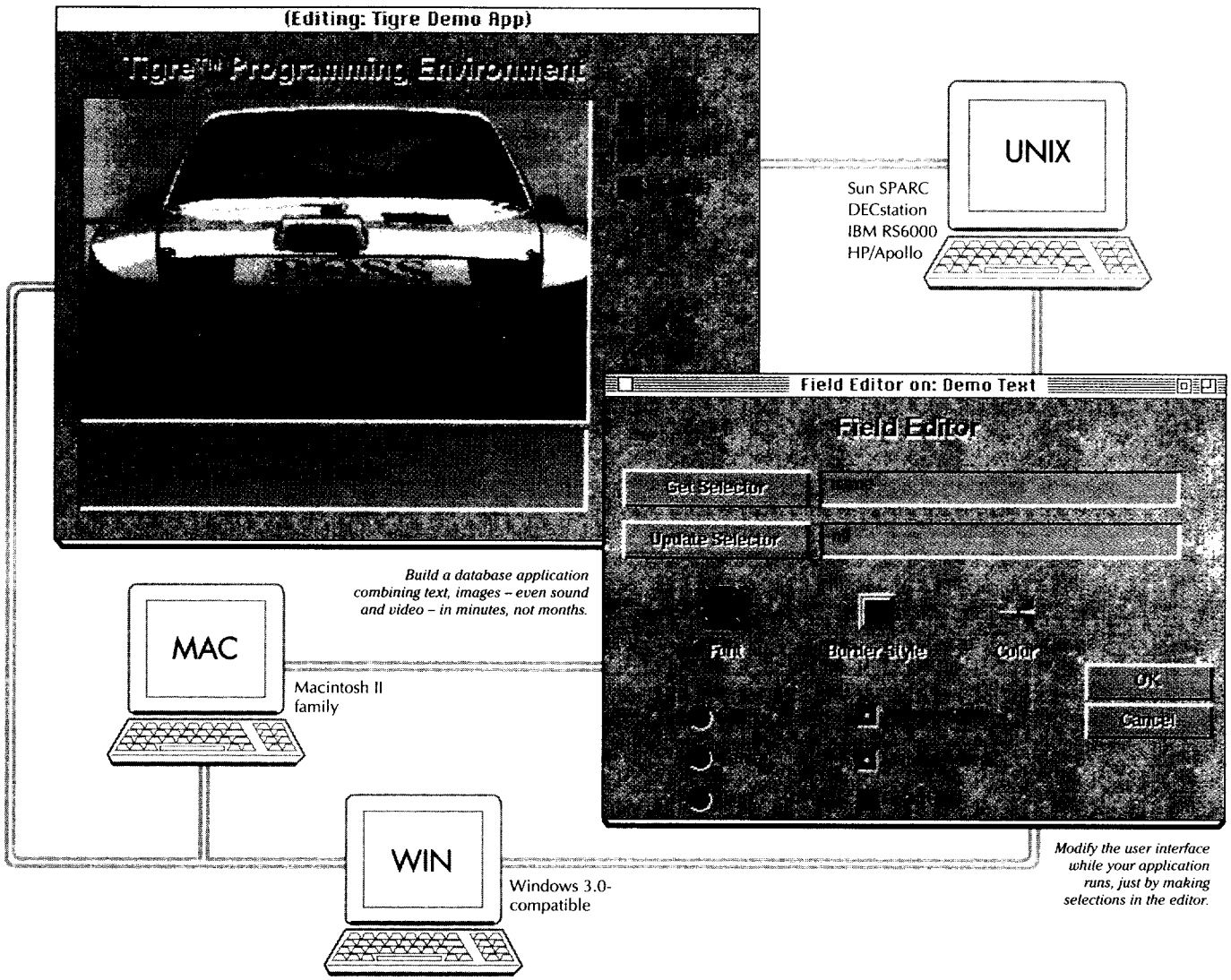
Enjoy the first issue!

John Pugh and Paul White
Editors

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.

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.**

Fastest Path to Platform Independence.



Leap free of platform limitations and deliver full-color GUI applications in half the time...with Tigre™.

Introducing an incredible OOP breakthrough: A complete development environment that lets you create object-oriented, multi-user applications that run across all major platforms and networks. *And* lets you deliver them up to 80% faster than ever before.

Tigre™ Programming Environment, running with Objectworks® \ Smalltalk Release 4, offers a set of tools that turns a major hassle into a quick drag. Literally. Because it lets you build customized, color GUIs just by dragging and dropping.

You'll choose from a large library of user interface components. Objects like scrolling text fields, check boxes, radio buttons and more.

Drag them from the palette onto your application screen. Move and resize them as often as necessary. No recompiling needed. And virtually no code to write. Tigre's Interface Designer automatically creates the Smalltalk GUI for you.

Give the interface your unique imprint by clicking selections to change color, font, borders, icons, etc. And you can add your own custom GUI creations to the library for reuse.

Use Tigre's multi-user, object-oriented database manager, to provide network-compatible access to text, images, icons, sounds – any type of stored data.

Phone now for a complete package of information on Tigre. There's never been a faster track to freedom.

TIGRE OBJECT SYSTEMS, INC.

Call: (408) 427-4900, Fax: (408) 457-1015
3004 Mission Street, Santa Cruz, CA 95060

PUBLISHER'S NOTE

"Isn't it time for an independent publication devoted exclusively to Smalltalk users" is a question I'm frequently asked at conferences. Even though Smalltalk is celebrating its tenth anniversary this fall (since *Byte*'s landmark issue lauding the language) there's been a surprising paucity of editorial coverage devoted to Smalltalk in any publication since.

We at SIGS Publications have seen a recent resurgence in the interest in and usage of Smalltalk. It remains the archetype of a pure and fully integrated O-O development environment as OOP explodes in the 1990s. According to Ovum's *Object Technology Sourcebook*, Smalltalk sales (in the US and Europe) are currently $21 million and are expected to double to $40 million by 1993 — making it one of the fastest-growing languages.

The time has come for an independent forum devoted exclusively to Smalltalk users' informational needs. *The Smalltalk Report* will publish over 200 pages of need-to-know information on Smalltalk during its first volume year. Our editorial mission, simply stated, is to stimulate, track, and evaluate Smalltalk usage on a worldwide basis.

Welcome to the premiere issue. It represents hundreds of hours of thinking, writing, and research. By reading *The Smalltalk Report*, you can quickly benefit by gaining access to nowhere-else-found techniques, advice, ideas, source code, and "insider news" — a veritable goldmine of consolidated information. You can rely on what you read in *The Smalltalk Report* to be timely and accurate. We publish it with the same editorial integrity as we do our sibling publications, the *Journal of Object-Oriented Programming*, *Object Magazine*, *The C++ Report*, the *Hotline on Object-Oriented Technology*, and *The International OOP Directory*.

I encourage you to contact us regarding your opinion of this issue and what you'd like to see in upcoming issues. Your feedback is valuable to us as the newsletter evolves.

I invite you to plug into the insiders network of Smalltalk developers by becoming a Charter Subscriber. Join our family of well-informed readers. We look forward to serving your informational needs. Enjoy the premiere issue!

Sincerely,

Richard Friedman

Richard P. Friedman
Group Publisher

continued from page 1...

in-house, Tektronix, primarily a test equipment manufacturer, had difficulty marketing their Smalltalk workstation. By 1985, developers and organizations were standardizing on mainstream personal computers and workstations; thus, the appeal of a custom workstation was limited. However, Tektronix developed a large group of people experienced in developing and marketing Smalltalk and object-oriented development that would later seed several successful companies in the Smalltalk and OOP market.

Digitalk introduced Methods, a text-based Smalltalk, in 1985, and then Smalltalk/V, a graphics version, in 1986. The most significant feature was that Digitalk's Smalltalk ran on the popular IBM-PC. By providing a low-cost Smalltalk for the IBM-PC, Digitalk expanded the base of Smalltalk users, many of whom were building prototypes or custom applications. The success of these early developers spread the use of Smalltalk as a commercial development environment.

In early 1986, a company by the name of Softsmarts brought the first Smalltalk-80 for the IBM PC/AT to the market. Like Digitalk, the Softsmarts version proved that Smalltalk could run on low-cost personal computers. Softsmarts was also the first Smalltalk-80 to incorporate color graphics and external language support. However, the PC marketplace for Smalltalk became dominated by Digitalk with lower-cost versions of their Smalltalk/V product (which included the same feature set and ran on eight-bit PCs).

About the same time, the group within Xerox PARC that had created Smalltalk wanted to spin off a company that would focus on marketing Smalltalk. From that drive, ParcPlace Systems was born. The first few years were spent creating the infrastructure of the company and creating portable commercial versions of Smalltalk for PCs, Macintoshes, and UNIX workstations.

Although the early and mid-1980s laid the groundwork for the future growth of Smalltalk, both the state of the computer industry and Smalltalk itself prevented the widespread acceptance and use of the language.

One perception of Smalltalk that remains to this day is that it performs poorly compared to standard languages. Much of the perception is based on the fact that early versions of Smalltalk were interpreted and included automatic storage reclamation (garbage collection). What is interesting to note is that even early versions of Smalltalk performed quite well (most people made performance statements without direct experience). Much of this performance is based on the fact that Smalltalk is best suited for complex, interactive information analysis and management applications. In simpler applications, the overhead of Smalltalk becomes a factor that makes it uneconomical for those applications. As the application becomes more interactive or complex, the power of Smalltalk becomes a key benefit in both development time and cost. Also, performance can be improved since the lower-complexity code that is created by using Smalltalk can be optimized

more easily than the large, complex amount of code created in traditional languages.

Another major obstacle to the widespread use of Smalltalk was a lack of acceptance of both graphical user interfaces and the object-oriented paradigm. Most computing was being done on either terminals or text-based PCs. Graphics were reserved for video games and exotic applications. Much of this was due to the fact that high-performance hardware and high-resolution graphics displays were not commonly available. PCs were being used in data entry, or simple analysis, and not to provide highly interactive interfaces or to solve complex problems. Thus, the range of applications that Smalltalk is ideally suited for were not being widely developed.

Many early projects done in Smalltalk were either prototypes or systems that evolved through many iterations. O-O analysis and design methodologies along with good implementation strategies were still forming (many of these early projects contributed to this process). However, in comparison to traditional development languages, Smalltalk appeared cavalier, undisciplined, and immature. The speed perception and lack of object-oriented analysis and design methodologies created the perception that Smalltalk, although good for rapid prototyping, could not be used in a disciplined way to create robust, high-quality, commercial applications.

During the middle to late 1980s, desktop computing and GUI-based interfaces became accepted. Smalltalk was behind in integrating with the standard GUIs that were emerging, continuing to provide its own nonstandard GUI. Also, Smalltalk was a closed language, not allowing interfaces to other languages or libraries. However, now all versions of Smalltalk from both ParcPlace and Digitalk integrate with the standard windowing systems and external languages. As a result, Smalltalk provides one of the best environments for development of host-based applications (given the complexity of GUI programming interfaces).

Another major block was the lack of Smalltalk developers, tools, training, and support services. These were areas of the market that had to grow to make Smalltalk a viable commercial development environment.

SMALLTALK TODAY

Over the last five years, the Smalltalk industry has grown and most of the hurdles have been cleared. Smalltalk has become more widely used in commercial software development. This is due not only to changes in the Smalltalk environment itself, but also to the software development marketplace as a whole.

Changes in the overall marketplace have played a key role in the success of Smalltalk. Time-to-market, adaptability, and cost control have become increasingly crucial factors in overall business success. It is this business environment that has accelerated the acceptance of object-oriented technologies and Smalltalk. Information systems managers today are more interested in solutions than in the technology employed. Many clients we work with that would have never considered Smalltalk as a development and delivery language a few years ago are now pursuing aggressive Smalltalk strategies.

The personal, highly interactive, graphical environment that Smalltalk pioneered is now accepted and several GUIs are widely in use. One result of the acceptance of GUIs is a further increase in software complexity and development costs. Developers now have to deal with the large library of APIs that GUI and operating system have. Actually, the Smalltalk windowing system, which has been considered difficult to use, looks absurdly simple when compared to standard windowing system libraries.

“Many clients ... that would have never considered Smalltalk as a development and delivery language a few years ago are now pursuing aggressive Smalltalk strategies.”

Both ParcPlace and Digitalk have introduced versions of Smalltalk that access and use the host GUI and APIs. When there were no standard windowing systems, Smalltalk provided its own. This, of course, was unacceptable once standards for windowing systems were established. Both Digitalk and ParcPlace versions of Smalltalk run using the host windowing system and allow access to the host API and external language functions (i.e.,Windows and Presentation Manager DLLs).

One of the main differences between Digitalk and ParcPlace's versions of Smalltalk is how they provide host integration. Digitalk's Smalltalk V Mac, V Windows, and V PM provide tight integration with their host and use the host environment controls and libraries (windows, menus, buttons, and so on). ParcPlace's Objectworks/Smalltalk Release 4 uses only the higher-level and portable host services (windows, fonts, and graphics) to provide image portability across platforms and operating systems. Objectworks/Smalltalk does allow the developer to access the host's controls and libraries at the expense of portability.

The amount of computing horsepower that sits on the average desktop today exceeds the horsepower that came with the original $100,000 Smalltalk machines from Xerox. This allows the user to use the system in an interactive graphical environment to solve increasingly complex problems. However, the cost and development time of software using traditional methods has not kept pace, leaving idle MIPS on the desktop. Smalltalk allows an effective, efficient, and cost-effective way to develop interactive applications that solve complex problems.

Both Digitalk and ParcPlace have also improved performance of the system by switching from a purely interpreted environment to executing compiled code. The performance of garbage collection has also been increased. In a variety of applications, particularly highly interactive and complex analysis, Smalltalk actually performs as well as or better than systems developed with traditional languages.

An example of high-performance Smalltalk in a commercial application is HPMS, a system developed for Hewlett-Packard by Knowledge Systems Corporation. The HPMS system is a complex process modeling tool primarily designed for manufacturing. It includes heavy computation and graphics for flow autorouting and diagramming. Most who see the system believe that it actually was written partly or entirely in C. However, HPMS is implemented entirely in Smalltalk without the use of C or assembly code. More information on the HPMS system can be found in Robert Whitefield and Ken Auers' article "You can't do that with Smalltalk! Or can you?" in the May/June 1991 premiere issue of *Object Magazine*.

Besides the Smalltalk language vendors, several other companies have formed to provide tools, training, consulting, and support for Smalltalk. Without this framework of companies providing the supporting products and services, corporations could not make the commitment and investment in Smalltalk.

One company, Object Technology International (OTI), provides team development and source code control tools for Smalltalk (essential for large-scale commercial development). OTI has also used Smalltalk successfully in ROM-based embedded controller applications, where typically low-level languages are used.

TODAY'S OBSTACLES FOR SMALLTALK

Many of the companies that are using Smalltalk are very secretive about their use (to the point of not allowing any Smalltalk books to be visible in offices). These companies view the use of Smalltalk as a strategic competitive advantage. Unfortunately for those in the Smalltalk industry, this reluctance to share success stories makes it difficult to promote wider use of the language through examples. Often people in the Smalltalk industry, when talking about Smalltalk's success, must be vague with lines like, "All sorts of companies are having tremendous success with Smalltalk, but we can't tell you about any of them."

As larger projects are being developed with Smalltalk (by companies we can't talk about), more time is being spent on analysis, design, and software quality. When used for prototypes, analysis and design are not significant issues. Still, for high-quality production software, Smalltalk requires design, testing, and iteration. Even today, many users first developing with Smalltalk get enamored of the enormous productivity gains of Smalltalk and try to turn functional prototypes into commercial software (which ends up being low in quality, difficult to maintain, or taking longer than expected). The process of managing the Smalltalk software lifecycle and then reuse of code are still issues. As more experience in managing the high productivity of Smalltalk is compiled, issues such as reuse and quality will be better understood.

Companies now making the investment in Smalltalk development face the difficulty of finding resources and education. The number of experienced Smalltalk programmers is limited, and competition for those developers is heavy. In addition, training in-house developers in object-oriented technology and Smalltalk takes approximately two months. After the initial training period, six months of use is required before enough experience is developed to create quality commercial software. Managers have difficulty accepting these time frames, given the pressure to deliver. Often this pressure is the reason for using Smalltalk.

SMALLTALK TOMORROW

Over the next few years, several significant products will come out using Smalltalk. Smalltalk development and product success stories will be published (many in The Smalltalk Report). The base of users and projects will expand both in organizations already using Smalltalk and in new ones.

The Smalltalk industry will expand with more companies being formed to provide products and services, particularly developer training, analysis and design tools, code generation, application frameworks, and tools to manage large-scale reuse of code.

Several companies will deliver integrated analysis, design, development, and lifecycle management tools developed in Smalltalk. These tools will push object-oriented application development into a more disciplined and efficient level, particularly in large organizations.

For application developers in both large and small organizations, more development tools will be delivered. Interface builders and application frameworks such as Acumen's Widgets and Tigre Object Systems' Tigre Programming Environment are already in use building successful commercial applications.

In the software development community, there is always the tendency to find the best technology. Currently, in the object-oriented arena, many are looking for a winner, be it C++, Smalltalk, Eiffel, and so on. The history of software shows us that there isn't a winner, just as there isn't any best automobile. There will be a variety of languages and tools to support various types of development. Smalltalk will find success in commercial applications, particularly in interactive desktop analysis applications, where the power of Smalltalk is best applied. ❁

REFERENCE

[1] Whitefield, R. and K. Auer. You can't do that with Smalltalk! Or can you? *Object Magazine*, **1**(1), 64–69, 1991.

Abdul K. Nabi can be reached at Knowledge Systems Corporation, 114 MacKenan Dr., Ste. 100, Cary, NC 27511.

Juanita Ewing

Should classes have owners?

As Smalltalk engineering projects grow larger, the need for reusable code increases. Developers need to build larger applications even faster. The easiest way to increase the capabilities and scope of an application is to reuse more classes. Large applications require teams of Smalltalk programmers to glue these reusable classes together and write some application-specific code, too.

WHAT IS A REUSABLE CLASS?

Classes are reusable in two ways: as a client making instances or as the basis for new subclasses. The characteristics of these two kinds of reusable classes are different. For client use, you want a fleshed-out and general class. For subclassing, you want a minimal and flexible class. Beyond these characteristics, how do you tell if a class is reusable? To paraphrase Ralph Johnson, a class isn't reusable until proven reusable. That means it has been used in more than one application.

It takes extra time and effort to write classes that are reusable. This extra effort is a separate programming activity. Developers caught up in deadlines for delivering an application often don't have the time necessary to flesh out and polish their classes. For example, developers will initially create a single class that should be refactored into a combination of an abstract class and a concrete class. The concrete class can be reused by making instances of it and the abstract class can be reused by making new subclasses derived from it. The reusability of classes written with the goal of multiple uses is much greater than those written for specific roles in an application.

In conjunction with supporting teams of developers, some Smalltalk environments actively promote the creation of reusable classes. One of the goals of these environments is to separate application engineering from the creation of reusable units of code.

IS CLASS OWNERSHIP A GOOD BASIS FOR PROMOTING THE CREATION OF REUSABLE CLASSES?

Suppose each class is owned by a single developer. The theory is that an owner feels responsible for and will take the extra effort to make a class truly reusable. The creation of reusable classes is important to the entire organization as well as the developers. Programming environment capability by itself is not enough. To back up this capability in the programming environment, the developers' organization must reward the production of reusable code. Responsibility and ownership are established management techniques for motivating employees. It's become common practice in manufacturing environments to give employees more responsibility and have them provide input about the manufacturing process. Employees don't just screw on lug nuts anymore.

Let's assume the owner of a class is rewarded for producing a reusable class. What if another developer finds a bug in that class, or thinks of a useful extension? In a system with class ownership, the owner writes the code to fix the bug or writes a new method. He is the one who is motivated to make the class more reusable.

WHO IS BEST QUALIFIED TO FIX THE BUG OR WRITE THE NEW METHOD?

In the case of the bug, the best qualified person may be the developer who detected the symptom of a problem and isolated the error. After the detective work, fixing the bug may be simple. And, sometimes it is difficult to reproduce a bug. In the case of the new method, maybe the person who thought of the extension knows best how to implement it. Maybe in both cases the owner and the person suggesting the change need to work together to come up with the best solution. The best qualified person depends on the situation. Flexibility in the programming environment is critical.

Systems with class ownership are not flexible. Even the motivational aspects are wrong for flexibility. What is the motivation for developers who are not owners?

DO CLASSES EXIST IN ISOLATION?

When a class is part of an application, it interacts, or collaborates, with other classes. Sometimes the collaboration is part of a framework. For example, a view and a controller collaborate as part of the MVC framework. An instance of view is never used alone. It is always paired with a controller. Because of the relationship between these two classes, coupled with the fact that modifications in one class will probably require corresponding modifications in the other class, there is a strong reason for the same developer to own both of these classes. It makes sense that any related classes should also be owned by the same developer. Evidently all parts of a framework should be owned by the same developer.

Continuing this example, what about the view's relationship with its model? Some views have a close connection with their

models. This argues that the model should be owned by the same developer that the view and controller are owned by. And yet in different applications the same view may collaborate with different models. Are all of those models owned by the developer that owns the view? Class ownership doesn't take into account the flexibility required by multiple applications.

A subclass is closely related to its superclass. If the behavior of a class changes, there may be ramifications in the subclass, requiring corresponding changes in subclasses. This implies that the same developer should own classes that are hierarchically related. Obviously, if one developer owns the entire image, we aren't talking about teams of Smalltalk programmers anymore.

If classes have owners and related classes are owned by the same developer to improve the efficiency of the team, how do you devise a reasonable partitioning if the ownership is restricted to a single developer per class? The answer is, you can't. The goal of grouping related classes conflicts with the goal of distributing classes to individual owners.

> "The advantage of multiple developers is to allow multiple perspectives and therefore create more general classes."

DO MULTIPLE DEVELOPERS AFFECT THE QUALITY OF CLASSES?

Close collaboration between developers is important in the production of reusable classes. People who are working together tend to be more creative. Multiple perspectives increase the likelihood of more general abstractions. Multiple developers are an advantage. The result of multiple developers is classes that are well fleshed out and suitable for client use and classes that are general abstractions suitable for subclassing.

SHOULD CLASSES BE ACCESSIBLE TO MULTIPLE DEVELOPERS?

The programming environment needs to promote developers working together. One way to do this is to make classes accessible to multiple developers. That way, each developer could make changes when most appropriate. If you have one owner, what do you do when that owner goes on vacation? What if the owner is ill at a critical time in the project? The programming environment should make it easy to implement contingency plans to keep a project going.

Since a reusable class is produced by a team of people, the entire team should be rewarded. Team programming environments usually have author designations for accountability. Outstanding efforts will continue to be noticed in these environments because of accountability features.

HOW DOES THE PROGRAMMING ENVIRONMENT KEEP THINGS FROM FALLING BETWEEN THE CRACKS?

How do you ensure that the entire class hangs together? You don't want to end up with classes that are a hodgepodge of functionality. Some automatic checks could be installed to produce warnings if, e.g., a method contains no references to self or instance variables.

Most of the consistency checks for a class cannot be automated at this time. A human still needs to browse and understand a class to see if it follows basic design principles. In a cooperative team, this responsibility can be shared. Peer reviews, or more formal code reviews, are an essential part of team efforts.

The programming environment should be able to restrict the set of developers for a class to avoid unauthorized modifications. Many operating systems offer these kinds of limitations. A team programming environment could be even more selective. Also, it is a good idea to place at least one experienced person with a group of inexperienced people. People who have good rapport generally program together well.

A programming environment for teams of Smalltalk developers should promote the creation of reusable classes by rewarding all developers. The advantage of multiple developers is to allow multiple perspectives and therefore create more general classes. Another benefit of multiple developers is more apparent in the final stage of the software lifecycle. Classes that are developed by multiple programmers are therefore understood by multiple programmers. It is easier for the organization to maintain classes because more than one person has the knowledge and understanding required for the job. ●

Juanita Ewing is a senior staff member of Instantiations, Inc., a software engineering and consulting firm that specializes in developing and applying object-oriented technologies. She has been a project leader for commercial 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 Chairs for the OOPSLA conference.

Greg Hendley and Eric Smith

Giving application windows dialog box functionality in Smalltalk/V PM, part 1

Welcome to the first installment of what we hope will be long-running column! Smalltalk has been around for some years now. When Smalltalk was young, the idea of applications having windows the way cats have kittens was a new one. Smalltalk environments of yore preceded the proliferation of standardized window environments. Therefore, they tended to carry their own windowing system with them. These old clunkers would grab the whole machine (keyboard, screen, and mouse) and have their own way with them.

Of late, however, the world has been changing. For nearly every kind of desktop workstation, from the PC-clone to the top-of-the-line UNIX workstation, there is a standard windowing system available. Applications that run on these machines are increasingly expected to conform to the interface standards of the host windowing system. Further, they are expected to work with other applications running under the same windowing system.

Fortunately, Smalltalk has kept up. Both of the major vendors are beginning to support "host windows." In this column, we'll be providing information on the nuts and bolts of getting applications going in Smalltalk while working with the facilities provided by the host windowing system. To begin this issue, we'll dive right in to a two-part examination of how to build dialog boxes wholly within Smalltalk/V PM.

Dialog boxes are useful for displaying messages and gathering input from the user. In Smalltalk/V PM, there are two subclasses of DialogBox to handle simple cases. MessageBox is useful for getting quick yes/no or confirm/cancel information from the user. Prompter is useful for posing a question and soliciting an answer. There are other DialogBox subclasses for find and replace, choosing fonts, and defining new subclasses. In each case, a specific Presentation Manager (PM) dialog resource is used.

The resource defines the types and locations of the dialog's controls. To define a dialog with a different layout of controls, a new PM dialog resource must also be defined. This can be done with the dialog box editor or the linker and resource compiler that come with the Presentation Manager development kit.

If you have been using Smalltalk for a while you may ask, "Why can't all the work be done in Smalltalk?" This column proposes one approach to building custom dialogs wholly within Smalltalk. This approach creates a subclass of ApplicationWindow and gives it some useful behavior currently found only in DialogBox. In addition to convenience, there are two advantages to building dialogs wholly within Smalltalk. One advantage is you can use your own custom panes in addition to control panes. (DialogBox is restricted to holding control panes.) The other advantage is that once you know how, you can add the behavior to any application window.

ESSENTIAL BEHAVIORS OF DIALOGBOX

Since we will be taking the essential behavior of DialogBox and adding it to ApplicationWindow, let's identify what that behavior is. Under DialogBox in the encyclopedia of classes[1] is the comment:

> *"A DialogBox is a popup window used to display messages and gather input from the user. A dialog box can be modal or modeless. A modal dialog box requires that the user terminates that dialog box before using the window that opened the dialog. A modeless dialog box allows the user to continue to use the window without terminating the dialog box."*

So, an optional behavior of dialogs is being modal. (Application windows are modeless.)

You may have noticed another behavior: dialogs seem to stick with the application window that created them. If an application window and it's dialog are partially obscured by other windows and either the application or its dialog is selected, the application and the dialog window come to the front together. This sticking together is one of a set of behaviors dialogs have because of the ownership relationship between a dialog and its application window. The application window is said to own the dialog.

The option of being modal and the ownership relationship are considered to be essential behaviors of DialogBox. Other behaviors such as displaying messages, gathering user input, opening, closing, and passing messages are already part of being an application window. The rest of this column will discuss modality and ownership, where they are documented, what they mean, how Smalltalk/V PM uses them, some ways for you to use them, and finally, how to put it all together to make your own dialogs wholly within Smalltalk/V PM. The remainder of part 1 will cover modality. Part 2 will cover ownership and putting it all together.

MODALITY

Chapter 19, Dialog Windows, of ref. 2 (pp. 247–262) describes two kinds of modality dialogs may have in PM. A dia-

log may be system modal or application modal. When a dialog is system modal, the dialog takes control from all other windows in the system. When a dialog is application modal, it takes control from all other windows in the application. (As an aside, any window may be created system modal. Further discussion on this is deferred to a later issue.) Dialogs in Smalltalk/V PM are neither.

Dialogs in Smalltalk/V PM are modal only to the window that was active when the dialog was opened. As a result, modality for Smalltalk/V PM dialogs is handled within Smalltalk. This makes it fairly easy to move the modality behavior to ApplicationWindow.

The mechanism for making dialogs modal is documented on page 468 of the *Smalltalk/V PM Handbook*.[1]

> *"Dialog boxes can be made modal to the currently active window by putting self processInput as the last line in your dialog box's open method. processInput will not return until the user closes the dialog box (actually, until another method in your dialog box class sends self close). Again, see NewSubclassDialog for an example."*

> "... dialogs seem to stick with the application window that created them. If an application window and its dialog are partially obscured by other windows and either the application or its dialog is selected, the application and the dialog window come to the front together."

To understand what goes on when dialogs are made modal, let's look at the method processInput in DialogBox. This method is inherited by NewSubclassDialog:

```
processInput
        "Make the receiver modal to its owner window.
        This method doesn't return until close has been
        sent to the receiver."
    | cursor |
    Processor currentProcessIsRecursive ifTrue: [
        self error: 'Cannot do modal dialog during recursion.'].
    owner disable.
    cursor := Cursor.
    CursorManager normal change.
    sem := Semaphore new.
    [CurrentProcess makeUserIF. Notifier run] fork.
    sem wait.
    CurrentProcess makeUserIF.
    cursor change.
```

Two actions are taken to make the dialog modal: the owner is disabled and processing in the method is blocked. Disabling the owner is easiest, so let's look at it first.

DISABLING

Disabling the owner means the dialog's application window is prevented from receiving any more keyboard or mouse input. Conceptually, the dialog's owner is the application window that created the dialog. In implementation, the owner is set to the frame window of the active window when the dialog receives the message fromModule:id: or fromResFile:. The code that finds and sets the owner is the same in both methods. Examining the code confirms that the owner is a window handle.

```
owner isNil ifTrue: [
owner := Notifier activeMainWindow.
owner notNil ifTrue: [owner := owner frameWindow]].
owner isNil ifTrue: [owner := WindowHandle queryActive].
```

The method for disable in WindowHandle sends the method enableWindow:fEnable: to PMWindowLibrary, the sole instance of PMWindowLibraryDLL. In response, PMWindowLibrary calls the MS OS/2 function WINENABLEWINDOW. MS OS/2 responds by disabling the frame window and all its child windows (see pp. 260–261 of ref. 3). So, the owner ignores all future keyboard and mouse input until it is enabled. The dialog enables its owner in the method close.

BLOCKING

Blocking means that the Smalltalk process executing the method stops. The method does not return (and so the calling method does not continue) until the process is unblocked. The blocking is done in three lines:

```
sem := Semaphore new.
[CurrentProcess makeUserIF. Notifier run] fork.
sem wait.
```

The first line is simple. It initializes the semaphore. The second line makes a new user interface process and starts it processing events. The third line actually blocks the process the method is executing in. The process is blocked and the method does not return until the semaphore is signaled. The dialog signals its semaphore in its method close. Once the semaphore is signaled, the method processInput is resumed, the user interface process is restored, and the method returns. The method close for DialogBox closes the dialog and undoes both actions taken in processInput:

```
close
        "Close the receiver."
    owner enable.
```

```
    owner makeActive.
    Notifier remove: self.
    PMWindowLibrary destroyWindow: handle.
    sem notNil ifTrue: [
        sem signal.
        Processor terminateActive.]
```

The first two lines enable the owner (the frame window of the dialog's application window) and restore it as the active window. The next two lines do some clean-up necessitated by the dialog having been created differently than a normal window. The last three lines signal the semaphore and kill the active process. Signaling the semaphore allows the process that was waiting on the semaphore to resume. This lets the method processInput resume and return.

ADDING MODALITY TO APPLICATIONWINDOWS

This is where you try out what you just read. Start by creating a subclass of ApplicationWindow. Give it an instance variable to hold the semaphore:

```
ApplicationWindow subclass: #DialogApplicationWindow
    instanceVariableNames: 'sem '
    classVariableNames: ''
    poolDictionaries: 'PMConstants '
```

Copy the method processInput from DialogBox. At the end of processInput add the line:

```
    super close. "<<Added because of difference between close in
DialogBox and ApplicationWindow."
```

Copy most of the method close from DialogBox. Modify it to look like this:

```
close
        "Close the receiver."
        "Most of this is copied from
        DialogBox. GLH 18 July 1991."
    owner enable.
    owner makeActive.
    sem notNil
        ifTrue: [
            sem signal.
            Processor terminateActive.]
        ifFalse: [super close.] "<<Added in case I am not modal. "
```

The differences between the close methods in DialogBox and ApplicationWindow will be covered in the next installment of this column. For now, just trust that this is necessary. The class will also need a method for finding and setting the dialog's owner. Make the three lines from from Module:id: into a method:

```
findAndSetOwner
        "Find the active window and make it my owner.
        This code is copied from DialogBox>>fromModule:id:.
        GLH 18 June 1991."
    owner isNil ifTrue: [
        owner := Notifier activeMainWindow.
        owner notNil ifTrue: [owner := owner frameWindow]].
    owner isNil ifTrue: [owner := WindowHandle queryActive].
```

The last method is for convenience:

```
openModal
        "Open and become modal the way
        most dialogs do. GLH 18 June 1991."
    self findAndSetOwner.
    super open.
    self processInput.
Now, test the class. Open a workspace. This will be the application
window for the dialog. From the workspace type, select, and do
together:
Temp := DialogApplicationWindow new.
Temp openModal.
Terminal bell.
```

Notice you can no longer type or use the mouse with the workspace. Also, note the last line did not execute. The dialog is modal to the workspace. The workspace is disabled and the process is blocked. Now, close the dialog. The bell will sound. Closing the dialog unblocks the process as expected. Also, note the workspace is enabled so the mouse and keyboard work with it.

Note: if you completely covered the dialog with the workspace, all is not lost. Simply type and do

```
Temp close
```

from the Transcript. This is why the global variable Temp was used.

In part 2, we will add ownership to DialogApplicationWindow and tie everything together. If there is space, you'll be shown some other ways to use these dialog behaviors. ❁

REFERENCES

[1] *Smalltalk/V®PM Tutorial and Programming Handbook*, Digitalk Inc., Los Angeles, CA, January 1989.

[2] *Microsoft Operating System/2 Programmer's Reference*, Vol. 1, Microsoft Press, Redmond, WA, 1989.

[3] *Microsoft Operating System/2 Programmer's Reference*, Vol. 2, Microsoft Press, Redmond, WA, 1989.

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.

COMPRESSING CHANGES IN /V WINDOWS

Charles-A. Rovira

After suffering through a series of embarrassing crashes, I came to the conclusion that the SystemDictionary>>compressChanges method for Digitalk's Smalltalk/V Windows and Smalltalk /V Mac lacked a little something in the robustness column. Here's what I did to remedy the situation.

SMALLTALK WILL CRASH IF ABUSED

As loath as anyone might be to admit it, Smalltalk does suffer from certain problems when dealing with resources that are not its own. The Mac and Windows are great environments when it comes to providing objects and functionality, but the integration with Smalltalk is not as complete as it needs to be. The holes are quite deep enough to break an ankle when you stumble into them.

A HANDLE IS NOT A MONIKER

It is possible to leave handles or pointers to objects lying around and to trip over these objects, handles, or pointers when saving the image or during the course of execution. Leaving things unreclaimed is easy to do during the heat of a debugging session ... I've done it often enough. Saving the image after every victory, no matter how minor it might seem, is an essential component of debugging, but save your image with unreleased handles and life will rapidly become unpleasant.

After a while, an image becomes slow, bloated, and unreliable. What could be better than starting fresh with a new copy, straight from the shrink-wrapped diskettes, and filing in all of your work. It's all been preserved in the change.log or the /V Mac image data fork. Unfortunately, the change.log contains a record of **everything** that you've done while developing your system of application. Every successful *doIt* execution, two copies of every class definition, every method you've defined, as many times as you've defined it, every selector you've gotten rid of. Everything!

Filing in your change.log will take the maximum amount of time and it's not likely to work. The *doIts* are the show stoppers. Any *doIt* that brought up a window or otherwise interfered with the scheduling process is likely to stop the filing in of the log.

CLEAN UP YOUR ROOM

There is a way to shrink the change.log, remove all of the miscellany, folderol, and failure, and leave only the shiny new code: Smalltalk compressChanges. As it comes shipped by Digitalk, this copies the code to a new change.log, adjusting all pointers as it does so, saving the image when it's finished, and throwing away the old change.log. Try it ... you'll like it. Except that we're attempting to dispose of a flabby, flatulent, or faltering image, so we're going to try filingIn the change.log, appropriately renamed, into a brand spanking new copy of /V ... ***Wrong!*** CompressChanges cleans up a little too much. All the class definitions are now missing. Filing in the change.log will halt at every class asking if you want to declare <yourClassNameHere>. Then it will merrily reject all the code because, since <yourClassNameHere> is not a class, it does not understand *methods* (don't ask ...). Also, classes that need initialization will once again need initialization and there will be nothing to tell you which these are. The last straw is that any global variables you might have used in your application are now in lost in dataspace.

WHAT K-TEL HAS TO TEACH US

Get one now! It's new! Improved! Get one NOW! It won't rust, rot, or testify in court! Get one now! It's tanker bilge. It's a dessert topping! It's both! Rated X, the unknown. Positively no one admitted. Consult your local listings.

GET A LOAD OF THIS

In keeping with the tradition of writing frightfully incomplete articles, there is one minor component missing from the listing included, mostly because it deserves a separate article in its own right. To move global variables and their values out of the image and onto a file from which they can be recovered requires something called a Loader. Due to idiosyncrasies peculiar to each implementation of Smalltalk, Loaders tend to be as individual as the system in which they reside. Also, since this facility is needed in a development environment, loaders, at least as I implement them, tend to use the compiler because operation is faster than using becomes:.

SON OF COMPRESSCHANGES

The code in Listing 1 is capable of compressing the change.log into a form that can be filedIn into a new image. After a short initialization sequence to record what-classes and globals are in the new system, all of the classes and global variables are

defined, the methods are saved, and the required class initialization is performed and global values are loaded in.

To keep track of what additions or changes have been made to the original image it is necessary to determine what classes and globals are present in the image. This is the function of the initialization sequence. The code not directly related to *SystemDictionary>>compressChanges* is there to keep track of the system as it changes.

- *Class>>comment* is necessary because I use class comments for a class hinting mechanism that allows me to verify methods to ensure that I won't get **'does not understand:'** walk-backs. It can also ensure that cloned images don't contain more objects, classes, or methods than is absolutely necessary. Like the Loader, this deserves its own article.
- *Class>>removeFromSystem* was modified to add the very last line. It also checks if the class has instances and prompts the operator through a ConfirmDialog. This is a standard Widgets/286, Widgets/Mac dialog which I implemented in /V Windows to maintain compatibility. The method can be changed to simply abort if there are any instances of the class.
- *Class>>removeInstances* just does what it says it does.
- The following methods have been modified to keep comments around across recompilation: *Class>>subclass:instanceVariableNames:classVariableNames:poolDictionaries:*, *Class>>variableByteSubclass:classVariableNames:poolDictionaries:*, *Class>>variableSubclass:instanceVariableNames:classVariableNames:poolDictionaries:*
- *MetaClass>>name:environment:subclassOf:instanceVariableNames:variable: words:pointers:classVariableNames:poolDictionaries:comment:changed:* was modified to remove redefined classes from a list of the classes that were present in the original image.
- *SystemDictionary>>compressChanges* is where the metaphorical rubber hits the yellow brick road. It has been modified to add a preamble to the image that gathers all classes and all global variables defined in the original image and to do messages sends of the following selectors:
    - *SystemDictionary>>compressClassDefsOf:into:* places all new or changed class definitions into the change.log. This method is implemented recursively to ensure that the class hierarchy is respected. To make it easier to relate class definitions with their order in the CHB, the subclasses are sorted alphabetically. This message is sent immediately after the preamble.
    - *SystemDictionary>>compressClassInitsInto:* finds all user-defined classes that implement an initialize selector and places the appropriate message send so that the class will be initialized automatically on filing in the log.
    - *SystemDictionary>>compressGlobalDefsInto:* reserves name space for all new globals used in any methods in the change.log. This message is sent immediately after having saved all of the global values.
    - *SystemDictionary>>compressGlobalInitsInto:* loads the globals from a 'recovery.dat' file. This will be the last message in the change.log
    - *SystemDictionary>>compressGlobalValuesInto:* saves all new globals used in the image in a 'recovery.dat' file. This is sent immediately after saving all class definitions.
    - *SystemDictionary>>removeKey:ifAbsent:* keeps track of deleted globals. If if it necessary to modify globals that come with the system, remove them from the system before replacing them with their new value. This ensures that they are unloaded.

"Due to idiosyncrasies peculiar to each implementation of Smalltalk, Loaders tend to be as individual as the system in which they reside."

PITFALLS

There are none. I have used this method to successfully save change.logs that contained all information necessary to recover my system after some real doozies. I sometimes refresh the image and fileIn the change.log to ensure that I have no obscure semicircular references or other uncollected garbage.

Since implementing these changes, I am much more comfortable about experimenting with objects and resources outside Smalltalk's control. When I'm trying a triple somersault from the flying trapeze, it's always nice to have a safety net. ❁

Now based in Ottawa, Canada, Charles-A. Rovira has been involved with data processing since 1975 and with Smalltalk and other object-oriented technologies since 1987. His CompuServe ID is: [71230,1217]. He'll admit to some unusual literary influences, such as Douglas Adams, Terry Pratchett, and D.H. Lawrence. Also Kierkegard, but why bring him up.

Listing 1.

```
Class methods
comment
    "answer the class comment"
    ^comment
removeFromSystem
    "Remove the receiver from Smalltalk. Report an error if there are any sub-
    classes or instances of the receiver."
... added line of code
    OriginalClasses remove: myName asSymbol ifAbsent: []
removeInstances
    self withAllSubclasses do: [:aClass |
        aClass allInstances do: [:anInstance |
                anInstance become: String new] ].
subclass: classSymbol
    instanceVariableNames: instanceVariables
    classVariableNames: classVariables
    poolDictionaries: poolDictNames
    "Create or modify the class classSymbol to be a subclass of the receiver with
    the specified instance variables, class variables, and pool dictionaries."
... inserted lines of code
    | aComment originalClass |
    comment := String new.
    originalClass := Smalltalk at: classSymbol ifAbsent: [].
    originalClass notNil ifTrue: [
        aComment := originalClass comment].
... modified line of code
        comment: aComment
        changed: nil
variableByteSubclass: classSymbol
    classVariableNames: classVariables
    poolDictionaries: poolDictNames
    "Create or modify the class classSymbol to be a variable byte subclass of the
    receiver with the specified class variables and pool dictionaries."
... inserted lines of code
    | aMetaClass aComment originalClass |
    aComment := String new.
    originalClass:= Smalltalk at: classSymbol ifAbsent: [].
    originalClass notNil ifTrue:
        [aComment := originalClass comment].
... modified line of code
        comment: aComment
        changed: nil

variableSubclass: classSymbol
    instanceVariableNames: instanceVariables
    classVariableNames: classVariables
    poolDictionaries: poolDictNames
    "Create or modify the class classSymbol to be a variable subclass of the re-
    ceiver with the specified instance variables, class variables, and pool dictio-
    naries."
... inserted lines of code
    | aMetaClass aComment originalClass |
    aComment := String new.
    originalClass := Smalltalk at: classSymbol ifAbsent: [].
        originalClass notNil ifTrue: [
        aComment := originalClass comment].
... modified line of code
        changed: nil
MetaClass methods
name: newName
    environment: aSystemDictionary
    subclassOf: superclass
    instanceVariableNames: stringOfInstVarNames
    variable: variableBoolean
    words: wordBoolean
    pointers: pointerBoolean
    classVariableNames: stringOfClassVarNames
    poolDictionaries: stringOfPoolNames
    comment: commentString
    changed: changed
        "Private – Create or modify the class and the metaclass of name new-
        Name to be as defined by the arguments. Check if an OriginalClass is
        being redefined"
... added line of code
    OriginalClasses remove: newName asSymbol ifAbsent: [].
    ^answer
SystemDictionary methods
compressChanges
    "Build a new change log file retaining only the latest version of changed
     methods in the current change log. Save the image to the image file."
    | logDirectory stream tempLogName dialog aFileStream |
    dialog := DialogBox new
        fromDLLFile: 'vwdlgs.dll'
        templateName: 'CompressingChange'.
    dialog showWindow.
    logDirectory := (Sources at: 2) file directory.
    stream := logDirectory newFile: 'ChangLog.tmp'.
    stream lineDelimiter: Cr.
    tempLogName := stream pathName.
... added lines of code
    stream nextPutAll:
' "evaluate"
    | originalClasses originalGlobals |
    originalClasses := SortedCollection new.
    originalGlobals := SortedCollection new.
    Smalltalk associationsDo: [:each |
        (each value isKindOf: Class)
                ifTrue: [originalClasses add: each key ]
                ifFalse: [originalGlobals add: each key]].
    Smalltalk at: #OriginalClasses put: originalClasses.
    Smalltalk at: #OriginalGlobals put: originalGlobals.!!';
    cr.
... added lines of code
    self compressClassDefsOf: Object into: stream.
    self compressGlobalValuesInto:
        (aFileStream := File pathName: 'recover.dat').
    aFileStream close.
    self compressGlobalDefsInto: stream.
```

```
... added lines of code
    self getSourceClasses do: [ :class |
        self compressChangesOf: class class into: stream.
        self compressChangesOf: class into: stream].
... added lines of code
    self compressClassInitsInto: stream.
    self compressGlobalInitsInto: stream.
... added line of code
    stream close.
    (Sources at: 2) close.
    File remove: (Sources at: 2) pathName.
    File rename: tempLogName to: (Sources at: 2) pathName.
    Sources at: 2 put:
        (logDirectory file: (Sources at: 2) file name).
    (Sources at: 2) lineDelimiter: Cr.
    ApplicationWindow new saveImageNoConfirm.
    dialog close
        compressClassDefsOf: aClass into: aStream
    "Write into a stream all of the hierarchy of class definitions that are new to
    the image."
    | classes |
    (OriginalClasses includes: aClass name asSymbol) ifFalse: [
        aClass fileOutOn: aStream.
        aStream nextPut: $!!; cr ".
        Transcript cr; show: aClass name"].
        classes := aClass subclasses asSortedCollection:
        [ :first :second | first name < second name].
    classes do: [ :aSubclass |
        self compressClassDefsOf: aSubclass into: aStream]
compressClassInitsInto: aStream
    "Write initialization code for all classes that have it"
    | initializedClasses |
    aStream nextPutAll: ' "evaluate" '; cr.
    initializedClasses := self select: [ :anEntry |
        (anEntry isKindOf: Class) &
          (anEntry class selectors includes: #initialize) &
          (OriginalClasses includes: anEntry) not].
    initializedClasses do: [ :aClass |
        aStream nextPutAll: self name , ' initialize.'; cr].
    aStream nextPutAll: '!!'; cr
compressGlobalDefsInto: aStream
    "Set up all global names"
    | globalsDictionary |
    (OriginalGlobals includes: #OriginalClasses)
        ifFalse: [OriginalGlobals add: #OriginalClasses].
    (OriginalGlobals includes: #OriginalGlobals)
        ifFalse: [OriginalGlobals add: #OriginalGlobals].
    aStream nextPutAll: ' "evaluate" '; cr.
    globalsDictionary := self reject: [ :anEntry |
        anEntry isKindOf: Class].
    globalsDictionary keysDo: [ :aSymbol |
        (OriginalGlobals includes: aSymbol) ifFalse: [
                aStream nextPutAll: ' Smalltalk at: #',
                  aSymbol printString, ' put: nil.'; cr]].
    aStream nextPutAll: '!!'; cr.
compressGlobalInitsInto: aStream
        "Get globals if we can load them."
    aStream nextPutAll: ' "evaluate"
    | collectionOfAssociations aFileStream |
    Smalltalk at: #Loader ifAbsent: [
        Transcript cr; show:
        "Loader not available. Globals not loaded.".
        ^nil].
    aFileStream := Disk file: "recover.dat".
    aFileStream size > 0 ifFalse: [
        aFileStream close.
        File remove: aFileStream pathName.
        Transcript cr; show:
        "Recover.dat not available. Globals not loaded ".
    ^nil].
  collectionOfAssociations := Loader new readFrom: f.
  aFileStream close.
  collectionOfAssociations do: [:aPair |
  Smalltalk add: aPair]. !!!!'
compressGlobalValuesInto: aStream
    "Save (unload) all of the globals into aStream"
    | aCollection classList |
    Smalltalk at: #Loader ifAbsent: [
        Transcript cr; show:
;                   'Loader not available. Globals not saved.'.
        ^nil].
        aCollectionOfAssociations := OrderedCollection new.
        classList := #(Behavior Persistent ClassReader
                ClipboardManager Compiler Context CursorManager
                DelayedEvent DeletedfClass Directory Dos
                DynamicDataExchange EmptySlot File Font
                GraphicsMedium "GraphicsTool" InputEvent Loader
                Menu MenuItem Message NotificationManager
                ProcessScheduler ViewManager Window WinHandle
                WinInfo WinLogicalObject WinStructure).
        (OriginalGlobals includes: #OriginalClasses)
                ifFalse: [OriginalGlobals add: #OriginalClasses].
        (OriginalGlobals includes: #OriginalGlobals)
                ifFalse: [OriginalGlobals add: #OriginalGlobals].
        self associationsDo: [:aPair |
                (aPair value isKindOf: Class) ifFalse: [
                  (OriginalGlobals includes: aPair key) ifFalse: [
                        (classList includes: aPair value) ifFalse: [
                          aCollectionOfAssociations add: ea ] ] ] ].
        Loader new write: aCollectionOfAssociations to: aStream.

removeKey: aKey ifAbsent: aBlock
        "We're getting rid of something in Smalltalk. Check if it's an
        OriginalGlobal."
        OriginalGlobals remove: aKey ifAbsent: [].
        ^super removeKey: aKey ifAbsent: aBlock
```

LAB REPORT

Ralph Johnson

The Typed Smalltalk project at the University of Illinois

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

The Typed Smalltalk project is one of several object-oriented projects at the University of Illinois at Urbana-Champaign, and the largest that uses Smalltalk. The goal of the Typed Smalltalk project is to make Smalltalk as fast as any other language by using optimizing compiler technology. We want to make Smalltalk fast without losing any of its advantages or changing the way it is used. We want to hide the compiler from the programmer and keep the programming environment just as interactive and useful for prototyping as Smalltalk has always been.

Typed Smalltalk is a large project with many components. These components fall into two categories, language changes and the compiler. The major language change is a type system that was designed to fit the way Smalltalk programmers program, not to force programmers to use a particular style. Type information does not change the meaning of a program but is just an annotation on an untyped program. Although the original motivation for the type system was to provide information that the compiler could use to make programs faster, it is also useful documentation.

One important part of the type system is a type inference system that automatically finds the types in a program. The compiler can infer a type for a method (the types of all the variables used in the method and its return type), but a programmer can refine these types to make the type more precise. The goal is for the programmer to rely on type inference when a program is being written and is changing a lot, and then to narrow down the types as the program moves from development to production use.

The compiler (TS) uses type information to convert Smalltalk into efficient machine code. TS is entirely written in Smalltalk. It has been designed to be portable and has a table-driven code generator. We currently have code generators for the M68020, the NS32032, and the SPARC and are working on one for the i80386.

The biggest problem with the back-end is that it is slow. The best way to solve this is to compile it. Unfortunately, TS does not work well enough yet to compile itself.

The project has had two major problems. We are using a single technique to attack both problems. The first problem is endemic to building large software systems: making the system reliable. The second is endemic to academic projects: building a large system on a shoestring budget with high attrition rates of workers. Although we have had some funding from NSF and a little from Tektronix and BNR, a lot of the work on the compiler has been done by unsupported students working on thesis projects. These volunteers work for the fun of it, so the work must be fun, and they tend to leave just about the time they have mastered the system.

The computer center of my alma mater had a sign that gave the "ten laws of computing." I don't remember most of them, but one of them was that "All nontrivial programs have bugs in them." A corollary was "If your program has no bugs then it is trivial."

Since we are trying to make a reliable optimizing compiler, this implies that we must build a trivial optimizing compiler. Unfortunately, optimizing compilers are big and complicated,and tend to be buggy. In spite of this, we have tried to make TS as simple as possible by rewriting parts that are complex. We have rewritten some of the parts at least a half dozen times. This has greatly improved the reliability of TS, though there are many parts that are still complex and TS is still unreliable. Part of the Smalltalk culture is rewriting code until it is elegant, easy to understand, and reusable. Our strategy fits into this culture perfectly.

Another reason for reliability problems is improper testing. Most people do not think that testing is fun, so volunteers are unlikely to develop and implement thorough test plans. Also, exhaustive tests of optimizing compilers are very difficult. Finally, the Smalltalk culture does not recognize the value and difficulty of testing and there are few tools to support it. Although the first two problems are peculiar to us, the third is widespread in the Smalltalk community and needs to be fixed.

One of the keys to having thesis projects produce useful software is to limit the scope of each project and to give the student time to rewrite the software several times. This not only produces better software, but the students are happier because they know they have done a good job. A good MS thesis project is to rewrite an overly complex part of the compiler, so this approach helps make the compiler more reliable.

MS students tend to spend a semester learning TS and Smalltalk, a semester doing useful work, and a couple of months writing a thesis. The high attrition rate has made painfully obvious the need for high-quality documentation.

This has led us to concentrate on developing documentation and figuring out how best to describe the system. One thing that we have learned is to concentrate on the big picture and ignore information that can be learned just as easily with the browser. Thus, pictures that list the entire class hierarchy are not important, but descriptions of the meaning of the hierarchy are. Lists of all the methods in a protocol are not important, but descriptions of what each method does are.

Although most of the work on TS has been done at the University of Illinois, Justin Graver, who did the original work on type inference, is now at the University of Florida and has several students working on projects related to the compiler. Thus, TS is a multiinstitution project. We hope that it will become reliable enough to be useful in the near future and that many more people will start to use it.

SMALLTALK CODE ARCHIVE

The University of Illinois has an archive of Smalltalk software and of papers on object-oriented programming. TS is not in the archive yet. However, the archive contains a lot of software that was developed at Illinois including Foible, a framework for visual programming environments that was written in Smalltalk-80. It also contains the archive of Smalltalk-80 developed by Manchester University software and the archive of Smalltalk-V software developed by the International Smalltalk Association.

You can access the archive by anonymous ftp to st.cs.uiuc.edu (which is currently an alias for speedy.cs.uiuc.edu at [128.174.241.10]) or by sending e-mail to archive-server@st.cs.uiuc.edu of the form

```
To: archive-server@st.cs.uiuc.edu
Subject:
path yourname@your.internet.address
archiver shar
encoder uuencode
help
encodedsend ls-lR.Z
```

which will cause the archive server to e-mail instructions to you. Report problems with the archive to archive-manager@st.cs.uiuc.edu.

As a last resort, you can get the entire contents of the archive on an Exobyte tape or 1/4" QIC-24 (DC600A cartridges) in tar format, on Macintosh disks, or on DOS 3 1/2" inch disks by sending $200 to William Voss at Department of Computer Science, 1304 W. Springfield, Urbana, IL 61801. ●

Ralph Johnson is in the Department of Computer Science at the University of Illinois at Urbana-Champaign. He can be reached there at 1304 W. Springfield, Urbana, IL 61801, or by phone at (217) 244-0098, or via e-mail at johnson@cs.uiuc.edu.

MESSAGES

Allen Wirfs-Brock

Smalltalk, organization, and you

A forum for sharing ideas, tips, and experiences or just a place to have your say ...

This morning, I was reading through the papers in the conference proceedings of the 1991 USENIX C++ Conference. This is a collection of eighteen papers relating to the application, implementation, and possible extension of C++. In general, the papers are well written, most are informative, and some are controversial. Collectively, they show that there is a a large and thriving C++ community that is not only actively applying and evolving their language but also communicating, sharing, and recording their shared experiences. They clearly show that C++ is a living, dynamic language.

As a Smalltalk user and implementor, my immediate reaction to this collection was a sense of longing for a similar set of papers concerning Smalltalk. I know there is comparable work being done in the Smalltalk community. If this wasn't the case, Smalltalk would be a dead language. The problem is that there is currently no forum for Smalltalk users and developers to get together and share this work. Why not?

For several years, OOPSLA provided such a forum. If you look at the proceedings of the first two or three OOPSLA conferences, you will find a large number of papers that directly relate to Smalltalk. This is not the case today. Why? Because OOPSLA is now a large, formal, scientific conference that addresses all aspects of object-oriented technology. To be accepted at OOPSLA, a paper must address original ideas that are broadly applicable to object-oriented technology. A paper that is of utility only to the users of a particular language will normally not be accepted. At most, one or two Smalltalk-related papers will now be selected for an OOPSLA conference. Generally, the same will be true of C++ or any other language. This does not mean that only one or two good papers exist, but to publish more would result in an unbalanced conference. If eighteen of the twenty-three papers at this year's OOPSLA were the papers from the USENIX C++ Conference, only C++ programmers would attend OOPSLA.

The obvious solution is that we need a Smalltalk conference. This is not a totally new idea; others have suggested it in the past, but nothing has happened so far. Why? It's easy for an individual such as myself to get excited about the idea. I know what has to happen. I lived through the organization of the first OOPSLA. I could get on the phone and start calling people to get them involved... but wait, reality starts to kick in. Organizing a conference takes a lot of work and entails considerable financial risk. I run a small business. Can I afford to take the time away from my clients and employees? Could I carry the financial burden? Well, it was a nice idea, but back to work.

What is really necessary for the organization of a successful conference is an organization to back it. For OOPSLA, this was the ACM. For the C++ Conference, it is USENIX. USENIX describes itself as follows:

> *"The USENIX Association is a not-for-profit organization of those interested in UNIX and UNIX-like systems. It is dedicated to fostering and communicating the development of research and technological information and ideas pertaining to advanced computing systems, to the monitoring and encouragement of continuing innovation in advanced computing environment, and to the provision of a forum where technical issues are aired and critical thought exercised so that its members can remain current and vital. To these ends, the Association conducts large semi-annual technical conferences and sponsors workshops concerned with varied special-interest topics..."*

A successful users group must be a response to a real need, a "pull" from the user community.

What about Smalltalk? Unfortunately, there is no comparable Smalltalk user's organization. Past efforts to create such an organization have been unsuccessful. Like conferences, user organizations take a considerable investment of time and money. Past efforts were "pushed" by vendors or individuals who had neither the time nor financial resources to be successful. A successful users group must be a response to a real need, a "pull" from the user community. In addition, it must have the backing

of large, corporate sponsors that can afford to commit people and financial resources to its success.

Perhaps, today, the environment is ripe for such an organization. There are now numerous large corporations that are making strategic commitments to Smalltalk. These are the organizations that really need and can afford to support a Smalltalk users group and conference. My final remarks are to my colleagues in these organizations.

You and your organization have made a commitment to Smalltalk. Its future success is critical to your future and success. This requires a dynamic, vibrant community of Smalltalk users. Take control of your future, get involved and organized. Put together an organization, sponsor conferences and workshops, encourage standards. You know who you are, you know you have the need. So do it. If you don't know who your counterparts in other corporations are, then call me at (503-242-0725) and I will get you connected. Let's make Smalltalk succeed! ❁

Allen Wirfs-Brock can be reached at Instantiations, Inc., 921 SW Washington, Ste. 312, Portland, OR 97205, or by phone at (503)-242-0725.

SOFTWARE REVIEW

reviewed by Jim Salmons

WindowBuilder: An interface builder for Smalltalk/V Windows

WindowBuilder, from Acumen Software, is a User Interface Management tool which greatly facilitates the rapid development of Smalltalk/V Windows applications. As its name implies, WindowBuilder enhances developer productivity by providing a "construction set" tool with which to interactively design application windows and dialogs in a "what you see is what you get" manner. Once you are satisfied with your design, WindowBuilder creates a new class to encapsulate your design, generating the Smalltalk methods which bring it to life.

At a list price of $149.95, this is a potent rapid application development tool which should be included in any Smalltalk/V developer's environment. Though there is room for improvement, this initial release of WindowBuilder is a much needed enhancement to Smalltalk/V Windows.

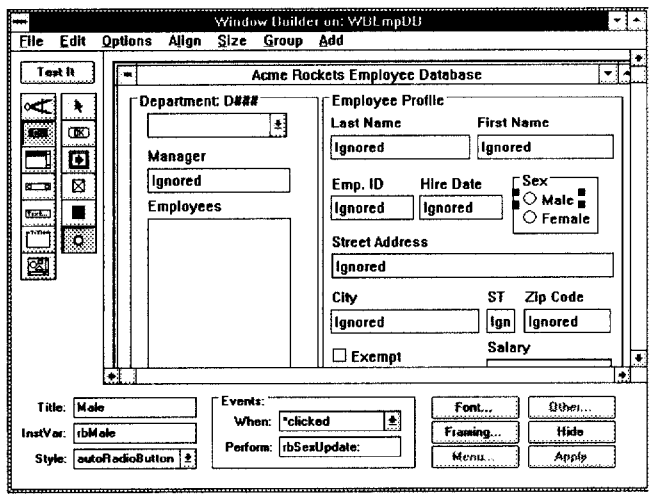


Figure 1. WindowBuilder tool building a database application window.

HOW DOES WINDOWBUILDER WORK?

WindowBuilder consists of software and a ninety-five page manual. The WindowBuilder tool and its associated classes are easily installed by filing in a single Smalltalk source file. Thirty-one classes are added to the base Smalltalk/V Windows environment. Some of these classes implement the WindowBuilder tool itself, but many are refinements and enhancements to the base system's window user interface Control classes. In addition to new classes, Acumen has made a significant number of modifications to methods in the base Smalltalk/V Windows classes.

Once filed in, a WindowBuilder menu is added to your Transcript window menubar giving you quick access to creating new and editing existing WindowBuilder windows and dialogs. WindowBuilder defines a new abstract class, WBTopPane, from which new windows and dialogs subclasses are created.

Figure 1 shows WindowBuilder in use to create a relatively complex application window. To place the **Male** RadioButton in the **Sex** GroupBox, as shown, the tool palette on the upper left side of the WindowBuilder window is used first to select a primary icon to place "Button" objects, after which a "RadioButton" secondary icon is selected. A crosshair cursor then appears to target the button's placement in the GroupBox.

The newly placed button displays "selection handles" to indicate that it is the active object. The Attributes Pane along the bottom of the WindowBuilder window is used to specify a default title, associated instance variable and Windows-specific style attributes. The Events group includes a **When** ComboBox which allows you to choose events to which the selected object will react. In this case, the RadioButton associated with the **rbMale** instance variable will react to a **clicked** event by sending its parent window the **rbSexUpdate** message. The Events group can be used to specify as many **when:** *event* **perform:** *method* associations as required by your design.

A well-implemented group of alignment options make it easy to create a clean window or dialog design. The **Distribute Horizontally** and **Distribute Vertically** options, which space objects evenly between two outermost selected objects are particularly useful and relatively rare in user interface design tools.

A Framing Parameters Editor is provided to specify the complex relationships among window control objects when the window is resized. In Figure 2, the Framing Parameters Editor is being used to specify that the upper-left corner and bottom of a ComboBox are fixed relative to the Parent window's top left dimension while its right dimension is scaled to the window's new size.

A Menubar Editor makes is easy to design dropdown menus to be added to your window designs. As with your basic window or dialog design, WindowBuilder generates the often complex and error-prone source to the methods which create and initialize your menus.

Working in concert, the tools provided by WindowBuilder make quick work of designing a window or dialog. As an interactive tool, you can save your design to its own WBTop-

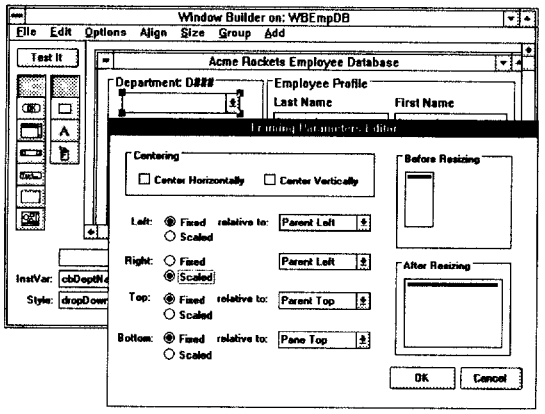


Figure 2. WindowBuilder's Framing Parameters Editor.

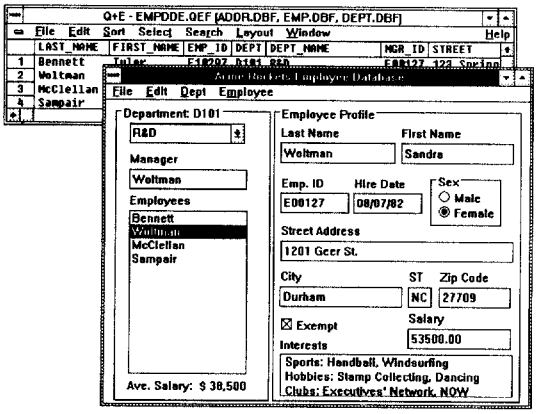


Figure 3. A WindowBuilder-built DDE client application window and its Q&E database server.

Pane subclass at any time and a **Test It** button is provided to generate an instance of your design.

Once you have the design worked out, you may then open a Class Browser on your window's WBTopPane subclass. To complete the implementation, you simply complete the "shell methods" which WindowBuilder generates based on your **when:perform:** and menu item action specifications.

To make all these WindowBuilder features immediately accessible to you, a cogent manual is provided. It includes an overview of the components and functionality of graphical user interfaces, a "Quick Peek" introductory tutorial, a user's guide, an extended example tutorial, a reference section and an index. WindowBuilder is so intuitive, however, that you hardly need the documentation.

USING WINDOWBUILDER TO CREATE A DDE DATABASE CLIENT APPLICATION

About three-quarters of my development session was spent implementing the DDE communication between Smalltalk/V and Pioneer Software's Q&E database engine (Fig. 3). The development of the window design was truly painless using WindowBuilder. Since WindowBuilder generates empty methods based on the control and menu event specifications of your design, it is essentially a "fill in the gaps" process to make the application fully functional.

Had I not been using WindowBuilder, I anticipate my experimental development effort would have easily doubled. WindowBuilder makes Smalltalk/V Windows a viable choice as a consultant's rapid application development environment.

WINDOWBUILDER'S BRIGHT SIDE

WindowBuilder is a vastly improved way to develop a Smalltalk/V Windows user interface when compared to writing raw source code. As a consultant, I would only recommend Smalltalk/V Windows for corporate client development projects if it were enhanced with WindowBuilder.

The Framing Parameters Editor and the Align menu features of WindowBuilder are particularly useful and are often not implemented as well in other user interface builders which I have used.

WindowBuilder is extensible. WindowBuilder is provided in source code and its interface includes a facility for adding your own new Subpane classes. If you create, or purchase, a set of interface components such as ToggleSwitch or ThermometerGuage objects, you could include them in your WindowBuilder designs.

Acumen supplies a WindowBuilder run-time file. Once you have an application built based on a WindowBuilder user interface, you can create a lean image with the classes and method changes required to implement the interface but not the WindowBuilder tool itself.

“WindowBuilder is a vastly improved way to develop a Smalltalk/V Windows interface when compared to writing raw source code.”

A WINDOWBUILDER WISH LIST

The most glaring problem I had with Version 1.0 was the lack of a Z-order editor. Windows uses a Z-order list to determine the order through which the window "focus" will progress under keyboard control. In a data entry application, you often want to make an entry and tab to the next logical field. It is surprising that Acumen did not provide any means to control and reorder this all-important aspect of a window or dialog design. The current workaround for the lack of a Z-order editor is to cut and paste the **addSubpane:** blocks in the **addSubpanesTo:** method. In a window as complex as the DDE Database example, this is incredibly tedious.

Experienced Smalltalk/V Windows and Smalltalk/V PM developers probably noticed that WindowBuilder uses WBTopPane as the parent of application windows rather than the more flexible and powerful ViewManager class. In many circumstances, the multi-window views supported by ViewManager designs are not required. When use of ViewManager is desirable, it is possible to add each WBTopManager subclass as a view of your application's ViewManager instance and set the owners of all Subpanes of your WindowBuilder windows within your window's TopPane to the ViewManager instance. While this is possible, I would like to see a clean and easy "Link to ViewManager Instance" option in a future release of WindowBuilder.

Also, there is no easy way to save WindowBuilder designs as subclasses of other WindowBuilder subclasses. I would like to be able to encapsulate reusable instance variables and methods for a DDE client WindowBuilder window in a new abstract subclass of WBTopPane. New DDE-based WindowBuilder designs could be created as a subclass of this abstract class. Currently, the only way to do this is to create your new design as a subclass of WBTopPane, file it out, remove it, edit the source and file it back in as the subclass of your abstract subclass of WBTopPane.

Where truly high performance is required or where multiple instances of a window or dialog may be active at one time, it is often desirable to compile a window or dialog using the Microsoft Resource Compiler from the Windows Software Development Kit or similar tool. Stored in a dynamic link library (DLL), such resources blast onto the screen when created and may take advantage of DLL shared run-time functionality. A "Write WindowBuilder Design to DLG Script" which could be fed to the resource compiler would be useful.

Finally, WindowBuilder does not fully implement the user interface standards of the Windows and Presentation Manager supported Common User Access (CUA) protocol. CUA defines the "proper" way a keyboard interface should work in terms of tabs between control groups and arrow keys moving within a group's items, etc. While a WindowBuilder window may have the "look" of a CUA-compliant window or dialog, the user access interaction misses the mark in terms of these subtle "feel" requirements.

HOW WINDOWBUILDER STACKS UP

WindowBuilder is a welcome addition to any Smalltalk/V Windows developer's toolkit. WindowBuilder will enhance the productivity of the new as well as experienced Smalltalk/V Windows developer.

By comparison, Digitalk's forthcoming Smart Parts product (demoed for nearly a year as the "Look and Feel Kit") has the potential to establish an entirely new programming paradigm for Smalltalk application development. Smart Parts will be a radical departure from traditional Smalltalk development procedures. While Smart Parts will be revolutionary, WindowBuilder is a solid evolutionary extension to Smalltalk/V development.

Try it. You will like it. Thanks, Acumen, and keep up the good work. ●

PRODUCT INFORMATION

WINDOWBUILDER
RETAIL PRICE: $149.95
SYSTEM REQUIREMENTS:
SMALLTALK/V WINDOWS,
MICROSOFT WINDOWS 3.0 OR LATER

ACUMEN SOFTWARE
2140 SHATTUCK AVENUE, SUITE 1008
BERKELEY, CA 94704
(415)-649-0601

Jim Salmons is President of JFS Consulting of Lexington, South Carolina. JFS Consulting specializes in the documentation of object technology products and object-based user interface revision control systems. With his partner, Timlynn Babitsky, Jim is coeditor of The International OOP Directory, *published by SIGS Publications. Jim and Timlynn are also Exhibits Cochairs of the annual ACM OOPSLA Conference.*

28 Oct-1 Nov 1991

Church House Conference Center, London

presented by

JOURNAL OF OBJECT-ORIENTED Programming — OBJECT magazine

Wang Inst of Boston Univ

sponsored by

computing

Program at a glance

- O-O design
- Software development using Smalltalk
- Putting objects to work
- Introduction to C++
- Objects by teamwork
- O-O analysis
- O-O databases
- C++ strategies and tactics
- O-O Windows programming using application frameworks
- Lessons learned from O-O projects
- Planning the software industrial revolution
- Synthesis — analysis and design
- Towards successful O-O development
- Which OOA&D should I choose?
- Reusability in practice
- Making O-O systems work
- Putting objects in DLLs
- O-O analysis — what could be more precise?

PLUS: A C++ workshop, BoF sessions, technical paper presentations, & exhibits.

EXHIBITORS

AI International
Borland International
Boston University Corp Ed Ctr
CACI Products
CGI/Yourdon
CNS
CRIL
Cocking and Drury
Computer Manuals
Computing
DataFlex Services
Euroline Systems
Glockenspiel
Harlequin
LBMS Education & Training
Logic Programming
Mark V
Object International
Program Now
Rational
Semaphore Training
SIGS Publications
VALBECC
Zortech

Learn OOP from the gurus at SCOOP EUROPE

SCOOP-Europe presents a diversified program of OOP-related topics. Featuring the thought leaders in the technology, this five-day event offers over forty intensive tutorials, lectures, and technical paper presentations — plus a large Exhibits area.

Learn the latest state of activity from such notables as:

Larry Constantine — original developer of structured design

Peter Coad — author of O-O ANALYSIS and O-O DESIGN

Grady Booch — O-O design pioneer and author of O-O DESIGN

Michael Jackson — Founder of Michael Jackson Systems, publisher of JSP & JSK methods

Brad Cox — inventor of Objective-C, founder of Stepstone

Tom Love — OOP pioneer and noted trainer and consultant

Tom Atwood — President of Object Design, O-O database pioneer

Marie Lenzi — Editor, OBJECT MAGAZINE and HOTLINE ON OBJECT-ORIENTED TECHNOLOGY

Meilir Page-Jones — noted industry writer and consultant

Chris Stone — President of the Object Management Group

plus Steve Cook, Rob Murray, Frank Ingari, and other industry pioneers.

If you are using object-oriented technology, or even considering its usage, you should attend SCOOP-Europe.

To receive a detailed brochure, call **071.259.2032**, *fax* **071.373.9430**, *or return card by mail.*

❑ Yes, I want to stay current on object-oriented technology. Send me a detailed brochure.

Name_____

Title_____ Company _____

Address _____

Postcode _____ Country _____

Phone_____ Fax _____

Return to SCOOP-Europe, c/o Boston Univ., 43 Harrington Gardens, London SW7 4JU, UK

WHAT THEY'RE SAYING ABOUT SMALLTALK

Excerpts from industry publications

... Smalltalk/V has been realized in DOS, Macintosh, and Windows versions. Much of the code can be used across all the environments. Objects can be stored in text form, and "filed out" and "filed in" from system to system. Because Smalltalk is an interpreter, these objects can be introduced into a running system. The potential exists to network together Smalltalk systems on a number of different platforms, and to let them exchange objects in real time. This is not something you can do with C++. Smalltalk/V has the most thorough tutorial of any of the packages reviewed here. The manual is a complete course in the language, and the example files give you working code for most applications. Smalltalk is not like other computer languages. Instead of being like a musical score, it is more like a jam session in which you create both new instruments and new musicians as you go along. When you've constructed your band, you're ready to play. Smalltalk is extraordinarily interactive, and the ideal environment for creative people. Accessibility is excellent, limited only by Smalltalk's unusual syntax. Every element of Smalltalk, from object creation to debugging and running the application, takes place with a single Windows application. Integration is total. The assistance to clear thinking, Smalltalk's clean handling of the Windows environment, its integration and rich data taxonomy, and its potential for interplatform development, make Smalltalk/v the winner among all the packages we have surveyed ...

Breaking into Windows, Birrell Walsh,
Microtimes, 6/19/91

... Only the interpreted object-oriented systems such as Smalltalk, object-oriented Lisp, and various proprietary object-oriented development systems, have a clear edge over Nextstep in programmer productivity. Because these systems are unsuitable for producing commercial applications due to their poor performance, huge size, or restrictive licensing policies, it is hard to refute the commonly heard claim that the Nextstep environment is the most productive mainstream development environment available today ...

The Next Next, Scott Raney,
UNIX World, 7/91

... Smalltalk is not about to replace COBOL, but it is finally maturing into a viable choice in application development, especially for users looking for a tool to speed development of advanced graphical user interfaces in client/server applications ... As a dynamically compiled language built on reusable objects and a virtual interface that uses machine-independent, intermediate code, Smalltalk is also easily portable between the platforms it supports ...

... there is still no widely accepted development methodology for Smalltalk or for any other object-oriented environment. In addition, many users are still making the transition to the relational model and structured programming techniques. "Most [IS developers] still don't know what to do with objects. They're still traumatized from making the migration to the RDBMS," says Natasha Krol, application program director at the Meta Group in Stamford, Conn. Smalltalk also faces increasingly stiff competition not only from other object-oriented languages such as C++ but also from new GUI-building tools, such as Easel from Easel Corp, and Actor from Whitewater Group ...

Smalltalk Grows Up, Jeff Moad,
Datamation, 7/15/91

IBM will postpone its scheduled announcement of support for object-oriented technology in AD/Cycle until later this year ... IBM had planned to announce by the end of this month support for the Digitalk Smalltalk language in its strategic software development environment. When it does make the announcement, IBM said, it will also provide a more substantial statement of direction for AD/Cycle and object-oriented languages as well as methodologies ... The AD/Cycle announcement will focus more on how object-oriented technology will affect the whole development life cycle ... rather than on an individual product ... IBM still plans to include Smalltalk in AD/Cycle and will also recognize C++ as an AD/Cycle language ...

IBM puts off object-oriented support, Rosemary Hamilton,
Computerworld, 6/17/91

... GUIs, however, are not a prerequisite of OOP programs, But, the two have become closely identified because GUIs increase the size and complexity of programs to the point where traditional programming methods cannot manage them effectively. OOP, on the other hand, can easily accommodate the programming of GUIs. In fact, Smalltalk, one of the first completely object-oriented languages, incorporates a graphical environment of menus, windows and scroll bars ...

Programming with Modules,
Chemical Engineering, 6/91

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.

Logic Arts announces VOSS: virtual object storage system for Smalltalk/V

Logic Arts' virtual object storage system, VOSS, is available now for Smalltalk/V 286. Voss object management facilities include: persistent storage, transparent access, virtual collection and virtual dictionary, multikey access, a class restructure editor, and import/export, in which administration facilities provide for backup, restore, renaming, import/export between machines, or access over a network. VOSS also features performance tuning: the control panel allows cache size and other parameters to be tuned for optimum performance, according to the degree of object volatility and random v. sequential access to virtual collections. Many of the new classes are independently reusable.Smalltalk/V286 source code is supplied. VOSS requires Smalltalk/V286 and MS-DOS.

For further information, contact Logic Arts, Ltd., 75 Hemingford Rd., Cambridge CB1 3BY, UK, (0223) 212392, or fax (0223) 245171.

Tigre ships multiplatform rapid GUI application builder

Tigre Object Systems, Inc. of Santa Cruz, CA, is now shipping the Tigre Programming Environment. Tigre implements the capability to build graphical user interface applications quickly for instant use on multiple computer platforms and heterogeneous networks. Color applications created by Tigre run without modification on Windows 3.0. Macintosh II, Sun/3, Sun SPARCstation, IBM RS/6000, Digital DECstation, Hewlett-Packard's HP 9000 Series 300 & 400, Apollo Series 2500, 3500, 4500, Sequent superminis, and on mixed networks of these. Additional platforms will follow. Tigre, a fully object-oriented system, uses Objectworks\Smalltalk Release 4 by ParcPlace Systems as its scripting language.

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

Digitalk announces Smalltalk/V developer conference

Digitalk, Inc. announced their first developers' conference, Smalltalk/V Dev Con '91. The conference will take place September 11–13 at the Universal City Hilton and Towers in Universal City (Los Angeles), CA. Sponsored by Digitalk and BYTE magazine, the conference will include a wide range of technical topics, panel discussions, speakers, and product demonstrations. Events include: sessions on design, management issues, application delivery, Smalltalk/V internals, integrating with other languages, integrating with other products, etc., as well as panel discussions, and industry guest speakers.

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

Instantiations announces new engineering tools and version management for Smalltalk

Instantiations, Inc. announced that it has developed a powerful new set of software engineering tools to support developers using Objectworks\Smalltalk called Application Organizer Plus. The product is an integrated set of tools that give Smalltalk users new ways to structure applications, manage code, and optimize reuse and was specifically designed to provide these capabilities without sacrificing the freedom and high level of interactivity that are the essence of Smalltalk programming.

Application Organizer Plus provides the Objectworks\Smalltalk developer with version management, improved code modularity, enhanced reusability, smaller delivered applications, new browsers, and workspace enhancements.

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

Digitalk ships new release of Smalltalk/V Windows

Digitalk, Inc. announced a new release of Smalltalk/V Windows, which combines Digitalk's widely used object-oriented programming environment with Microsoft Windows 3.0. Smalltalk/V Windows Release 1.1 contains an icon editor, performance improvements, better memory utilization, and many new programming examples demonstrating usage of Windows features.

Smalltalk/V Windows includes standard Smalltalk/V features such as source code browsers, inspectors, and push-button debuggers. In addition, Smalltalk/V Windows provides interfaces to dynamic data exchange (DDE), allowing information to be shared between Smalltalk/V programs and other programs and dynamic link libraries (DLLs), providing a mechanism for calling code written in other languages from within Smalltalk/V. Smalltalk/V Windows source code is compatible with Digitalk's Smalltalk/V PM programming environment for OS/2.

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

Digitalk announces royalty-free runtime

Digitalk, Inc. announced new versions of Smalltalk/V DOS and Smalltalk/V Mac that include royalty-free runtime. Smalltalk/V Windows and Smalltalk/V PM are already royalty-free.

The Smalltalk/V DOS Version 3.0 runtime system allows developers to create standalone executable applications and includes integrated EGA/VGA color. Registered users of earlier versions of Smalltalk/V may purchase an upgrade that includes a new manual.

The new version of Smalltalk/V Mac allows developers to create standalone, double-clickable applications with no additional royalty payments. Prior to this new policy, there was a per-copy charge for runtime applications. Registered users of earlier versions of Smalltalk/V may purchase an upgrade.

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

ParcPlace announces 4GL application development tool for Objectworks\Smalltalk

ParcPlace systems announced the availability of FACETS, a development tool for use when building applications in Objectworks\Smalltalk Release 4. FACETS, supplied by Reusable Solutions, was designed to help create screen-based, data-intensive applications such as order entry, financial processing, and other database-oriented 4GL applications. In conjunction with Objectworks\Smalltalk Release 4, FACETS provides an extendable object-oriented 4GL development environment and series of on-screen forms that guide the user through the rapid generation of interface components.

FACETS is fully compatible with Objectworks\Smalltalk Release 4 and fully portable across all supported platforms, and allows full connection to Servio's Gemstone interface for powerful database connectivity.

For further information, contact ParcPlace Systems, 1550 Plymouth St., Mountain View, CA 94043, (800) 759-PARC.

Object Technology International announces an object-oriented team development environment for OS/2 and Windows

Object Technology International, Inc. (OTI) announced the immediate availability on OS/2 and Windows of Release 1.0 of ENVY/Developer, an object-oriented team programming environment. With ENVY/Developer, development teams using Smalltalk may work concurrently on both OS/2 and Windows, sharing code and data using the tools provided by the environment.

The environment supports the full manufacturing lifecycle including prototyping, development, interactive debugging, performance analysis, packaging/delivery, and maintenance of large systems written in Smalltalk, and provides all tools required to realize the benefits of object-oriented software development. ENVY/Developer is currently the only toolset for delivering large systems incorporating advanced object-oriented technology.

For further information, contact Object Technology International, Inc., 1785 Woodward Dr., Ottawa, Ontario K2C 0P9, Canada, (613) 228-3535, or fax (613) 228-3532.

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.

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:

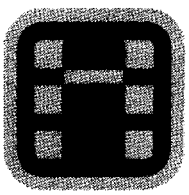
Project-focused team and individual learning experiences.

Smalltalk Development Tools

Application Organizer Plus™ Code Modularity & Version Management Tools

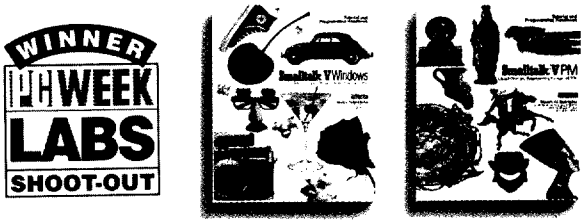
See our new Multi-User/Shared Repository Team Tools At OOPSLA 91!

Smalltalk! Nobody Does It Better.



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

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 trouble-shooting 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

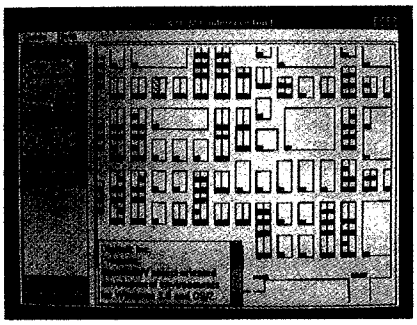
NCR has an integrated test program development environment for digital, analog and mixed mode printed circuit board testing.

MIDLAND BANK

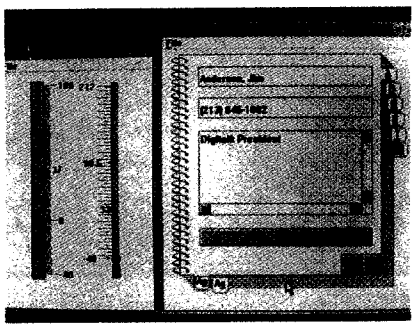
Midland Bank built a Windowed Technical Trading Environment for currency, futures and stock traders using Smalltalk V.

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.