The Past, Present, and Future of Smalltalk

L. PETER DEUTSCH

Chief Scientist, ParcPlace Systems

1 ABSTRACT

Smalltalk has firmly established itself as a contributor to software technology, a practically useful application development environment, and a participant in the commercial marketplace. This paper reviews its evolution, surveys its current roles, and offers some opinions about its future.

2 INTRODUCTION

There are few programming languages in existence today that have made major contributions to software technology, have evolved through multiple generations of implementation and engineering, and are living and flourishing in the marketplace. This author believes that Smalltalk has firmly established itself as a member of this select group, along with the venerable FORTRAN, Lisp, and Algol/Pascal. The intent of the present paper is to:

- Review and evaluate in some detail the technical evolution of Smalltalk, specifically with respect to the emergence of those concepts that are now recognized as fundamental to object-oriented software design, and the user interface innovations that have inspired so many later systems.

- Survey the present role of Smalltalk in the marketplace and in research, as a language, a development environment, and a basis for applications.

- Offer a number of opinions about the technical challenges for Smalltalk in the future, and also a few comments about its competitive future.

For the purposes of this paper, the watershed of Smalltalk history — the transition from a widely discussed but practically inaccessible set of experiments, to a technology available to researchers and later vendors — is the publication of the three Addison-Wesley Smalltalk books by Xerox authors in 1983. I have chosen to call the pre-1983 evolution of Smalltalk within Xerox the "past," and the developments since then the

"present."

3 THE PAST

3.1 Ancestry and Initial Concepts

The original goals for Smalltalk were described by Alan Kay in the early 1970s. The initial sketches that formed the basis for Smalltalk were heavily influenced by the idea of classes as an organizing principle (taken from Simula-67), of turtle graphics (taken from the LOGO project at MIT), and of what is now called "direct manipulation" interfaces (inspired by the Sketchpad drawing system, developed by Ivan Sutherland at MIT Lincoln Laboratories in the early 1960s, and by Kay's Ph.D. thesis on the FLEX Machine).

Kay sometimes referred to Smalltalk as a language and system for "children of all ages". The connotations of this vision included the desire to make "simple things VERY simple, and complex things VERY possible." In the early years of Smalltalk development at Xerox PARC, the vision of the individual "playing" with a relatively simple system dominated the evolution of the language and environment, and the research projects tended to be small, self-contained, and heavily oriented toward the interactive.

3.2 Smalltalk-72

Between 1971 and 1975, Kay's group at PARC designed and implemented the first real Smalltalk language, environment, and applications. This system included a remarkable number of technical innovations:

- The language was based entirely on the Simula concepts of class and message. There were no built-in operations, only predefined (machine coded) procedures that were invoked in the same way as user-written procedures. Even the initialization of objects at creation time, which is often implemented specially in object-oriented languages, was handled with a special init message. (Hewitt's work on Actors began around the same time, but Smalltalk-72 was the first usable, running system based on this approach.) This characteristic has remained one of the hallmarks of Smalltalk even today, and distinguishes Smalltalk from all other object-oriented languages in widespread or commercial use.

- The language had no fixed syntax. Each class was responsible not only for its own behavior and state definition, but even for parsing the token stream that followed a mention of an instance. This made it easy for a programmer to create a syntactic style of his/her choosing, but very difficult for anyone other than the author to read and understand a program. We can see this as an example of the project's bias toward the single-user, user-as-author paradigm.

- *Originally, Smalltalk was even closer to Actors in that Kay intended that each object*

have its own control state, and that function-like as well as physical-object-like objects would be supported. While some control structures were actually implemented that way, objects did not, in fact, have a useful control state, and the idea was abandoned in later Smalltalk systems.

The innovations in the environment were equally radical. Kay originally envisioned a portable machine with a flat-panel display, but this display technology did not exist at the time; indeed, bitmapped displays were considered expensive and arcane. Nevertheless, in 1971-72, PARC took the daring position (to a considerable extent at Kay's urging) that their research hardware would use bitmapped (CRT) displays. This daring step enabled Kay, Dan Ingalls, and the other members of what was then called the Learning Research Group (LRG) to implement:

- Bitmap-based text in multiple sizes and styles, with user-editable fonts: until recently, this was the state of the art in interactive systems, only now being superseded by outline- or stroke-based font descriptions and smart renderers.

- "Turtle" graphics as the basic paradigm for creating pictures: this was the first good example of linking algorithmic activity directly to a visual display, and has proven excellent for introducing young students to graphics, but quickly falls short as one wants to produce more interesting visual displays.

- Later, the BitBlt (also known as RasterOp) function as a fundamental primitive for bitmap graphics: this is still the basis of all commercial Smalltalk graphics, and an important component of all bitmap-based graphics models such as that of X Windows.

- A multi-window environment, including a class library that provided window capability to any application: the concepts have taken root in the industry, although the original implementation had serious problems (for example, it provided structurally for neither tiled nor overlapping windows).

The Smalltalk-72 system further took the point of view that there was no reason for a separate operating system, since the object paradigm could manage all hardware resources at least as easily as any other approach. (Subsequent evolution of both Smalltalk and operating systems has, in this author's opinion, strongly validated this belief, modulo issues of protection and of controllable resource allocation.) Consequently, not only the display, mouse, and keyboard handlers (down to a very low level), but even the file system and (later) network protocols were implemented in Smalltalk.

In addition to the innovations in language and environment, the Smalltalk-72 system was the arena for many interesting application experiments:

- Multiple-object, real-time animation;
- Simulation, both real-time and traditional, including dynamic (animated) display;
- Several different forms of music input and (both visual and audible) output;
- Several different painting and drawing programs.

In retrospect, Smalltalk-72 did a remarkable job of capturing many aspects of Kay's visions, in both its strengths (near-instantaneous interaction, personal control, a media-oriented system, a simple and powerful programming paradigm) and its weaknesses (primarily a heavy orientation toward a single, isolated user who creates the applications him/herself, and toward toy-size, non-combinable applications).

3.3 Smalltalk-76

By 1975-76, it had become clear that the lack of attention to issues of performance and scaling were hampering further investigation of the vision of personal computing that was driving the Smalltalk effort. Kay's group proceeded with a major redesign of all aspects of the Smalltalk system. In the language area:

- The idea of inheritance and a subclass hierarchy, which had been added to Simula, was also incorporated into Smalltalk. This hierarchy was used both for inheriting specification (e.g., Integer as a subclass of Number) and for inheriting implementation (e.g., Dictionary as a subclass of Set). Subsequent experience has confirmed that the former is both theoretically and practically valuable, while the latter tends to create difficulties in subsequent evolution and often reflects insufficient understanding.
- The syntax of the language was fixed. This enabled compilation into an efficiently interpretable, compact (bytecoded) instruction set. Given that the only machines of interest to the group were high-speed, microcoded engines with relatively small memories, this was an excellent choice at the time. While bytecode interpretation remains a viable implementation choice for Smalltalk on today's fast microprocessors, it is being complemented and perhaps superseded by various forms of compilation (more on this subject below).

There were also three important innovations in the underlying language implementation:

- In order to make most efficient use of the limited memory of the Alto hardware (128K bytes, of which 60K were taken up by the display frame buffer), Smalltalk-76 incorporated a novel object-oriented virtual memory system called OOZE (Object-Oriented Zoned Environment). It relied heavily on the speed of microcode (it *required a hash table lookup for every reference to an object!), but it did an excellent job of packing a useful working set of objects into a small memory. Indeed,*

subsequent experience has confirmed that paging is practically useless for Smalltalk systems, whereas OOZE was highly practical. OOZE also included a clever mechanism for efficiently making an automatic checkpoint of the virtual memory roughly every 30 seconds, which provided a crude but sometimes very useful way to recover from fatal (user or system) errors. The checkpoint mechanism gave the group a strong appreciation for the value of a stable (disk-based) record for disaster recovery: this was provided through a different and more robust mechanism (stable logging of source code changes) in later Smalltalk systems.

- Not only classes and compiled code, but Contexts (stack frames) were made into first-class objects. While this was a step away from Kay's original idea that every object carried its own control state, it suited the new compilation approach better, and it provided exactly the appropriate hook for the new debugger (see below).

- Explicit multiprocessing was added to the system, in the form of Process and Semaphore classes. Again, this was a step away from distributed control. In retrospect, concurrency has always been a problematic area in Smalltalk, and none of the subsequent experiments in this area have been entirely satisfactory. It would be interesting to build a Smalltalk system with the extreme concurrency opportunities that characterize Actors or Concurrent Prolog, but to the author's knowledge this has not yet occurred.

Smalltalk-76 enabled LRG researchers to break the size barrier, and a large part of this was due to the creation of the Browser by Larry Tesler. The Browser is a five-paned window that provides a view of a four-level tree structure. The first and third levels, called "category" levels, provide a user-specified organization of, respectively, classes and messages within a class; the second and fourth levels reflect the language entities of classes and messages respectively. Each level is represented by a scrollable list of names: simply selecting a name with the mouse brings up the selected subtree in the next pane. The fifth, large pane displays and allows editing of the selected leaf (method within a class). The Browser was a startling innovation at the time, and is still superior to the macro-scale navigation facilities in most environments.

The other major environmental innovation was the development of a user interface class library that included multi-paned windows, scroll bars, pop-up menus, scrollable and selectable lists, and a multi-font text editor. These components were all reusable through subclassing.

Smalltalk-76 also introduced a manager for overlapping windows. An interesting aspect of this window manager, which it shared with Smalltalk-72, is that control responsibility was completely distributed: each window decided when to give up the (single thread of) control, and was responsible for making sure it only displayed within its assigned boundaries. *In retrospect, this feature, while consistent with a pure object-oriented philosophy, turned out to create major debugging problems. We see this as a particular case of the problem, alluded to earlier, of replacing an operating system with*

an object-oriented application structure: there is no responsible entity that can make any guarantees about resource allocation.

As with Smalltalk-72, Smalltalk-76 spawned several interesting research applications:

- A constraint-based language and constraint satisfaction system, called ThingLab, that was the most sophisticated effort of its kind for many years following.
- A completely visual programming environment, designed for constructing educational software, called Programming By Rehearsal.
- A simulation kit, including visual animation and the ability to restrict users of the kit to a "filtered" subset of the system classes.
- An experimental "dynamic book," based on an information retrieval system called FindIt, that foreshadowed many later hypertext systems.
- Additional experiments in animation, including 2-dimensional with collision detection and 3-dimensional wire-frame.

3.4 Smalltalk-78

By 1977-78, LRG saw that microprocessor technology was reaching the point where it might be possible to realize another part of the original vision, namely, the creation of (a more realistic prototype of) a portable, relatively affordable personal computing system. This led to a combined hardware/software project called the NoteTaker. Based on dual Intel 8086 processors with a custom display, bus, and package, it executed a variant of Smalltalk-76 called Smalltalk-78, implemented primarily by Bruce Horn and Ted Kaehler. Performance was sluggish, and various non-technical difficulties led to the project's cancellation after only 10 systems had been built, but it was a persuasive demonstration that it would very soon be possible to implement Smalltalk on a conventional processor.

Aside from the hardware, the interesting technical innovation in Smalltalk-78 was a compromise on the first-class objecthood of Contexts: Smalltalk-78 used a linear stack, one per process, with a consequent danger of dangling references. This tension between the desire for clean object semantics for Contexts, and the desire for the efficiency of a linear stack, was only resolved two implementation generations later (in PS, described below).

3.5 Smalltalk-80

In 1979-80, partly in the wake of the NoteTaker project, the attention of LRG was increasingly drawn to the possibility of propagating Smalltalk beyond PARC. The group *designed and implemented yet another generation of Smalltalk systems, this time with some changes specifically aimed at leveraging the progress occurring in the hardware*

and software worlds outside Xerox:

- The Smalltalk-76 character set included quite a number of special characters, some of them left over from Smalltalk-72. The Smalltalk-80 system used the ASCII character set (actually, the Teletype character set; the small but significant difference has been a source of annoyance ever since). This decision met with heated opposition within the group at the time, but has turned out to be essential for the acceptance of the system in the world.

- The Smalltalk-76 primitive methods included the ability to directly access any memory location in the machine, and used this ability for some miscellaneous functions like reading the mouse position, setting the cursor coordinates, and reading the calendar clock. The Smalltalk-80 system removed this ability, adding half a dozen new primitive methods for the necessary system functions. Again, this was a key decision in the move towards system portability.

- The Smalltalk-80 language introduced the concept of metaclass, to provide a way of talking about behavior (messages) that were specific to an individual class. Motivated primarily by the desire to have class-specific creation messages that did not require making up a "blank" instance and then initializing it, metaclasses have proven confusing to many users, and perhaps in the balance more confusing than valuable.

- The Smalltalk-80 language also made blocks (isolated pieces of functionality supplied as the arguments for enumerators) first-class objects. In retrospect, this proliferation of different kinds of instantiation and scoping was probably a bad idea: a simpler lexical scoping story, as in Scheme, or a dynamic lookup arrangement, as in Self, would have been easier to explain, and might have been no harder to implement.

Even though metaclasses and blocks complicated the language model, they helped complete the object-oriented "story," in that there was now no data structure visible in any way to the programmer that was not a first-class object.

Besides the changes in the language and the primitive methods, the Smalltalk-80 system introduced a new architecture for interactive applications, called Model-View-Controller (MVC). This architecture calls for completely separating the Model (the data, often persistent, that represent the state of a simulation or other application), the View (the algorithms and formatting for mapping the Model to the screen), and the Controller (the algorithms and tables that map user actions into editing operations on the Model): the purpose of this separation is to enable the development of independently reusable Model, View, and Controller components. While MVC has proven somewhat difficult for application writers to master, and while it has certain limitations (e.g., it does require a certain amount of cooperation from the Model to make viewing possible at all), it has been recognized as a profound improvement over most of its predecessors and many of its successors.

New experimental applications developed in Smalltalk-80 at PARC included:

- The Galley Editor: a document editor that allowed mixing text paragraphs, paint-style graphics, and animations. Besides numerous features characteristic of today's desktop publishing systems, the ability to include dynamic images set the Galley Editor ahead of most systems developed since then.

- The Alternate Reality Kit: a simulation of physical reality, complete with friction and gravity forces, that evolved into a complete visual programming environment.

3.6 Publication

By 1981, a significant number of Smalltalk researchers at PARC felt it was important to take direct action to propagate Smalltalk beyond PARC. Adele Goldberg, who had replaced Alan Kay as head of the group, and Dave Robson, a long-time group member and major technical contributor, decided to write a series of books about Smalltalk, of which the first would describe the Smalltalk-80 system architecture in sufficient detail that others could implement the Virtual Machine (instruction set executer, memory manager, and primitive methods) on their own hardware. Goldberg contracted with a number of hardware vendors for them to receive pre-publication drafts of the book, in exchange for which they promised to actually implement the Virtual Machine and report all problems they encountered. This unusual experiment produced four completed Virtual Machine implementations (by DEC, Apple, Hewlett-Packard, and Tektronix), and a book which numerous others have used to implement the Smalltalk-80 Virtual Machine since that time. (The book, *Smalltalk-80: The Language and its Implementation*, was published by Addison-Wesley in 1983 and is often referred to in the Smalltalk community simply as the Blue Book.)

One of the immediate benefits of publication was the inception of a research project at Berkeley, under the direction of Prof. David Patterson, which produced two interesting new pieces of Smalltalk-relevant technology:

- David Ungar, one of Patterson's students, created a new memory management technique called generation scavenging. Based on some published but unimplemented work by Hewitt and Lieberman at MIT, generation scavenging proved to have better pragmatic properties than any of its competitors (reference counting, garbage collection, or semispace copying), and is now used in all commercial Smalltalk systems.

- Based partly on results from the PS research at PARC (described in the next section), Patterson's group designed, fabricated, and largely debugged a VLSI RISC processor, called SOAR, that was designed for efficient Smalltalk execution. Even though the SOAR system never reached full operation, simulations and measurements had a significant effect on subsequent hardware, specifically the Sun SPARC processor.

4 THE PRESENT

Since Smalltalk emerged from its Xerox nursery, it has spawned a wide variety of technical and business activities. I would characterize the former as composed of:

- Engineering work to discover how to make Smalltalk systems more competitive with other languages on standard platforms;

- Research to explore design extensions and alternatives;

- Development of libraries and applications, both for commercial purposes and to further understanding of the benefits and limitations of the object-oriented approach.

4.1 Engineering

As mentioned earlier, the first major step towards a usable Smalltalk on standard platforms was taken at Xerox PARC starting around 1978, with the development of an experimental implementation using the Intel 8086 processor. The first Smalltalk system running entirely on stock hardware was also developed at PARC, starting in 1981 and essentially completed by 1984. This implementation, known as PS (for Portable Smalltalk), was an implementation of the Smalltalk-80 Virtual Machine in MC68000-family macroassembler language. Key technical advances included:

- The development of an entire macroassembler language programming environment, in Smalltalk. This environment included an incremental assembler and linker and a symbolic debugger, but it used the Smalltalk Browser and change management facilities. This development was a dramatic example of the easy reusability that can result from thorough use of object-oriented design: in effect, a very large part of an entire environment was reused in conjunction with a very different language. (The assembly language environment was never released or demonstrated outside Xerox, but it is mentioned in Biggerstaff's recent book on software reusability.)

- The development of a new implementation technique, called dynamic translation, that combined the conceptual simplicity of bytecode interpretation with some of the speed of compilation. The key idea is to represent both code and stack frames (Context objects) in two different forms, and to convert between them automatically as needed. One form is execution-oriented: code is machine code, and stack frames are in the form most natural for the machine (similar to what C might use). The other form is access-oriented: both code and frames are represented like other objects. These ideas were presented in a paper in the January 1984 ACM Principles of Programming Languages conference.

Dynamic translation continues to be the leading implementation technology for high-performance Smalltalk systems. *In 1987, ParcPlace Systems created a retargetable dynamic translator, written in C, which today generates code for products running on the*

MC68000 family, the Intel 80386, the Sun SPARC, and the MIPS R2000, and, on an experimental (not product) basis, the DEC VAX.

The other branch of Smalltalk engineering is represented by Smalltalk/V, a highly tuned bytecode interpreter running on the IBM PC and compatibles, and recently reimplemented for the Apple Macintosh. Smalltalk/V carefully selects a large subset of the original published Xerox Smalltalk functionality, adds user interface classes, and makes it available to low-budget users.

4.2 Research

Quite a number of languages, of varying degrees of public visibility, owe their conceptual origins to Smalltalk. A group at Tektronix, concerned about the apparent impossibility of disentangling a Smalltalk application from the development environment, proposed and implemented a good deal of a language called Modular Smalltalk, which adds module facilities and some other more stringent visibility and access mechanisms. Two groups, one at MIT and one in Japan, have developed languages they call Concurrent Smalltalk, which add explicit concurrency constructs to the Smalltalk-80 language. Objective-C, a commercial product, adds Smalltalk-style classes (with inheritance) and messages on top of C, as does another product called Complete C. In this author's opinion, the most interesting language descendent is Self, developed at Stanford. Self has multiple inheritance, but uses prototypes rather than classes, and is considered by many in the academic community to be a simpler and more powerful (albeit less structured) language than Smalltalk.

Two research groups are focusing on issues of implementation performance. The Self research group at Stanford has developed a sophisticated dynamic translator. Their current system runs about twice as fast as the best Smalltalk implementation on the same hardware, and promises to come within a factor of 2 or 3 of the performance of C within the next year. Another group, at the University of Illinois, is working on an optimizing compiler based on type declarations, for a dialect called Typed Smalltalk.

Numerous groups are engaged in research using Smalltalk as a base technology. Some of the longer-established projects include work on industrial (including real-time) applications at Carleton University in Canada, work on constraint programming at the University of Washington, and work on hypertext at Tektronix.

4.3 Commercial Activity

As of this writing, the author knows of two flourishing vendors of Smalltalk implementations in the U.S. (ParcPlace Systems, offering the Smalltalk-80 system, and Digitalk, offering Smalltalk/V); two in Europe (Smalltalk Express in the UK, offering a Smalltalk-80 system with their own VM, and Georg Heeg in Germany, with Smalltalk-80 systems on several platforms in addition to those supported by ParcPlace); and several in Japan (Fuji Xerox, Sony, and NEC). A number of companies (generally small ones) are offering library packages or development tools that extend the base systems,

and there is a growing list of application products that use Smalltalk as the underlying technology, but where the Smalltalk language and development environment are not visible to the end user. These applications span a wide variety of domains:

- CASE (process management, group coordination, code generation, version management);
- Data base interface;
- Simulation;
- Music;

and many more.

5 THE FUTURE

5.1 Application Platform Issues

Now that Smalltalk implementations are available on relatively affordable platforms with subjectively acceptable performance, third party application developers are starting to use it as the base for their products. Issues that Smalltalk vendors will have to address to satisfy these developers include:

- Space: even Smalltalk/V, the most compact of the commercial Smalltalk systems, takes over 300K bytes of RAM; Smalltalk-80, with its more comprehensive facilities, requires around 2M bytes. Both general engineering and modularization of the class library are likely to be required to make the space consumption competitive with C-based solutions.
- Speed: even the ParcPlace Smalltalk-80 implementations, the fastest available, are typically a factor of 5 or more slower than C on computation-intensive problems (although the graphics and interaction performance are perceived by users as competitive). If Smalltalk is to become a serious application delivery environment, better compilation techniques will be required.
- Stability: Smalltalk systems are so malleable, and commercial software environments are evolving so rapidly, that vendors both can and feel a need to improve their offerings frequently. However, application developers want stability in their technology base. Reconciling these two desires will be an important issue.
- Closed delivery: Smalltalk systems currently provide no boundary between the development tools and any kind of runtime class library, and assume that the source code for the entire application is available for inspection and modification. Developers want only a library, and need to be able to protect their code. Some

work has been done in these directions, but more is required, especially to support applications that may still provide the user with some controlled way to write Smalltalk code. Smalltalk also offers the potential, now just starting to be realized in other environments, of dynamically loading multiple applications that can share their data in a natural way; understanding how to resolve clashes of symbolic names, and to represent code in a protected format that can still be integrated into a running system, are open questions.

5.2 CASE and Groupware Issues

As both developers and researchers create more ambitious applications using Smalltalk, issues of scale and process management become relevant, as they have been for some time in other language environments. Smalltalk offers some potential leverage in this area, not because of any special properties of the language, but because the Smalltalk environments already make it natural to think of source code, object code, users, files, etc. as objects that programs can manipulate.

5.3 Language Issues

The Smalltalk-80 language, as defined by the Blue Book, is a de facto standard. However, both Digitalk and ParcPlace have made minor modifications, and there is some feeling that the language is too complex in some areas (different kinds of name scopes and lifetimes) and too weak in some others (modularization, concurrency), and that developers' and customers' interests might be served by some form of standardization.

5.4 Competition

As Smalltalk has moved further into the commercial arena, it has encountered competition in all three of its traditional areas (language, environment, and application).

- In the language arena, the chief competitor is C++, which offers more familiar syntax, efficiency closer to that attainable by assembly code, and static type checking (generally considered, even by many Smalltalk devotees, to be a feature, rather than a drawback, for applications). Smalltalk offers much simpler syntax and semantics, and automatic storage management; in this author's opinion, the latter provides tremendous productivity leverage. The next few years are likely to see each language adopt some of the best features of the other.

- Several class libraries for C++ and other object-oriented hybrids have appeared on the market. The most successful of these is probably MacApp, which is nevertheless tied to a single vendor (Apple) and hardware platform (Macintosh). The promise of reusability still seems to be more fully realized in the Smalltalk world than in others, perhaps because its environment makes it easier to restructure classes in the face of new understanding, and because (unlike C++) programs do not *have to commit themselves by declaring the names of the superclasses for their object references.*

- The Smalltalk development environment is still unmatched, especially given that it is fully portable between different machine architectures and operating systems. However, other environments have adopted variants of the Browser, and the Smalltalk environment is still very strongly oriented towards single-person use. Moving beyond this orientation is the most important challenge that vendors of Smalltalk development environments will have to face. There are some research results and a couple of third-party products in this area, but much more remains to be explored.

Besides these obvious competitors, a new kind of software environment has emerged in the market, exemplified by Apple's HyperCard. These products allow rapid construction of visually appealing applications with limited semantic breadth (HyperCard, for example, only provides a single very simple data structure) by people without training in traditional concepts of programming. Another important challenge facing Smalltalk vendors is understanding how to combine this kind of ease of development for simple applications with the more sophisticated power of full Smalltalk for more experienced developers.

5.5 Cohabitation with Platform Window Systems

The Smalltalk-80 user interface classes were based on a single graphics primitive, BitBlt, that directly modified the display hardware frame buffer. Today, industry standard operating systems require applications to use a graphics library and a window system provided by the platform, often with very large space and time overheads. (The X Window System, for example, typically requires 1M bytes of address space and imposes a minimum time overhead of several thousand machine instructions for the simplest request.) In this author's opinion, these systems are a giant step in the wrong direction for personal computers and workstations, but they represent the state of the market, and Smalltalk must adapt to them to be successful. Technical issues to be resolved include:

- The definition of an appropriate object-oriented API (application program interface) to window systems and graphics libraries: there are a large number of alternatives in each area, and the choices in the latter (e.g., bitmap-oriented, device-independent stencil/paint, display lists, or hierarchical structures) have a profound effect on application structure.

- The issue of control flow: Smalltalk assumes it has full control of the CPU, and implements input by polling the keyboard and mouse, but window systems use a variety of models (asynchronous interrupts, asynchronous call-back, or polling), and it seems very likely that Smalltalk will have to change its input model to some form of event-driven or call-back structure.

- *The nature of graphical objects, specifically explicit bitmaps: X Windows, for example, may be able to do operations on bitmaps stored in the window server an*

order of magnitude more efficiently than on bitmaps that must be passed to it from the client, but a server may have space limits, and it may not offer specialized image operations (such as seed fill, rotation, or dithering) that a client could do efficiently. Understanding how to share or trade these responsibilities, while not unduly compromising a pure object-oriented information model, is likely to be a challenge.

5.6 Collaboration with Other Languages

Just as in the area of window systems, Smalltalk must evolve to allow its application developers to take advantage of the enormous body of valuable software that is available in other languages, primarily C. Data base, communications, and high-quality graphics functionality are all available, and they must be made accessible. Again, there are a number of challenging technical issues:

- Different languages have different representations for the same data types. For example, Smalltalk systems often use tag bits to distinguish integers from other data types. Conversion between representations must be nearly automatic, if inter-language communication is to be practical.

- Smalltalk represents control (Contexts and Processes) very differently from C. Simply keeping track of calls from Smalltalk to C and back raises significant implementation questions. Similar issues arise with respect to interrupts, exceptions, and blocking.

- C and Smalltalk have totally different approaches to storage management: C is manual, with no assistance from the language or library, whereas Smalltalk is automatic, but requires a more structured environment in which all references can be found. (C++ takes a middle position.) Bridging the gap in a safe and efficient manner will be another key challenge.

6 CONCLUSION

Smalltalk has grown and thrived in many environments, and has shown little sign of exhausting its potential. We look forward to the coming years with confidence that Smalltalk will continue to contribute to the software world in creative and valuable ways.

7 ACKNOWLEDGEMENTS

The Smalltalk community has grown tremendously, especially in the past five years, and the names and contributions of the original principals are well known by now. I would especially like to acknowledge Adele Goldberg for her assistance and support, as research lab manager, as company president, and as indefatigable proofreader.

7.1 Trademark Notices

Digitalk, Inc. HyperCard is a trademark of Apple Computer, Inc. Macintosh is a trademark of McIntosh Laboratory Inc., licensed to Apple Computer, Inc. The X Window System is a trademark of the Massachusetts Institute of Technology.

8 REFERENCES

In addition to the specific references listed below, several of the Xerox PARC research projects are documented in technical notes (available to the public): FindIt (Steve Weyer), Programming By Rehearsal (Laura Gould, William Finzer), and ThingLab (Alan Borning).

Goldberg, A. *Smalltalk-80: The Interactive Programming Environment.* Addison-Wesley, 1984.

Goldberg, A., and Robson, D. *Smalltalk-80: The Language and its Implementation.* Addison-Wesley, 1983.

Krasner, G., ed. *Smalltalk-80: Bits of History, Words of Advice.* Addison-Wesley, 1983.

BYTE magazine, August 1981 and August 1986 issues.

Proceedings of OOPSLA '86 (Portland, Oregon), *'87* (Orlando, Florida), and *'88* (San Diego, California). Association for Computing Machinery, New York.

Ungar, D. "The Design and Evaluation of a High Performance Smalltalk System." Ph.D. thesis, University of California, Berkeley, 1986. Also published by MIT Press in the ACM Distinguished Dissertation series.

Scheifler, R. W., Gettys, J., and Newman, R. *X Window System C Library and Protocol Reference.* Digital Press, 1988.