The Future of Squeak

Dan Shafer

The WeTalk Network, Inc.

Introduction

Friedrich Nietzsche once said, “Our destiny exercises its influence over us even when, as yet, we have not learned its nature; it is our future that lays down the law of our today.”

If ever there was a topic to which Nietzsche’s thought could be applied, it is Squeak. We have not yet learned the nature of the destiny of Squeak because it continues to unfold before our very eyes and because we are about the business of creating that destiny. Yet, to an extent not attained by other programming languages and environments, Squeak has always been *about* the future. Its future has in fact determined many of the ways it works and thinks today.

In this chapter, we’ll take a look at where Squeak might well be headed over the next two to five years. To choose a shorter time horizon would result in a chapter that would be obsolete almost before the book could see the light of day. To project further would be foolhardy in the extreme given the uncertainty not only of Squeak but of the world in which it will play its role.

Much of the content of this chapter derives from conversations I have been privileged to have with the group of Squeak developers and insiders known affectionately as “Squeak Central.” Some of it also comes from extensive conversations I have had with my colleague, Laurence Rozier, who is among the most forward-thinking object-aware software professionals I have met. Rozier has been carrying Squeak’s message to me and thousands of others for many years, as he carried the Smalltalk and JavaScript and other object-oriented language message before and since.

I have also sprinkled a few of my own predictions liberally through the text. What emerges is not a guaranteed future or even a “certified” future bearing the imprimatur of the Squeak Central team or of the Squeak community. Rather, this is an attempt to have some fun speculating about where our favorite programming language and environment might take us and where we might guide it.

The chapter is divided into two major sections. In the first, we’ll focus on the future of the Squeak kernel, the heart of the beauty, as it were. In the second, we’ll concentrate on applications and implementations that Squeak may facilitate in the relatively near term as the Squeak community broadens, deepens, and learns.

Some Basic Concepts

Because this chapter is about the broad-scoped future of Squeak, I use a couple of terms in ways that are probably not completely technically accurate. Certainly they aren't entirely precise.

When I talk about Squeak in this chapter, I mean to include not only what is clearly and inherently part of what the Squeak Central team is building (i.e., the kernel and virtual machine along with such core additions as Morphic, SqueakToys, and the Swiki), but also important and widely used classes and packages being developed by others. Comanche is one example of this kind of extension that I will bring under the umbrella

name of "Squeak" because it feels to me like the future of Squeak with Comanche is richer than its future without that Web server.

The other term that is hard to define and confine not just in Squeak but in Smalltalk dialects of all types is "application." Traditionally, this has meant a file on which a user could double-click with a mouse and which ran as a sort of stand-alone piece of functionality from that time forward. The problem is that this kind of analogy doesn't resonate with people. In your real life, you don't have things lying around on your desktop (particularly not a trash can!). You may not even have a desktop.

In the world of Smalltalk and Squeak, though, things are much more natural, though this naturalness means we have to think more carefully about what we are doing than we would have to if we weren't so conditioned to metaphors that don't map to our real lives. In some cases, an application might exist as a set of objects the user simply adds to his or her system and which extend what the user can do. In that scenario, there is presumably an underlying environment, whether a Squeak image or an operating system or a chip with Squeak embedded. In some cases, transparent background downloads of packages might make the very existence of something even vaguely resembling an application irrelevant. There would be no such thing as a "system" or an "environment"; rather the Squeak "thing" would in fact become the environment or the system. This kind of behavior will be particularly predominant, I suspect, when it comes to portable devices and TV set-top boxes. In still other cases, packaging might well be used to create what looks more like the traditional double-clickable application.

On some levels, this explanation is part of a much larger issue which centers on how humans interact with and perceive computers. In just the past few years, we've seen the perception of a computer as a desktop where stuff (documents and applications, principally) is stored to a sort of transport mechanism. "I just went over to Yahoo! and looked it up" has become such a part of the way we think and talk about our experience that Web sites like yahoo.com are clearly places in our minds, just like Grandmother's house. In no real sense, of course, to we "go" to Yahoo. But that's what it feels like and it's how we describe the activity. This has deep implications for how we build the user interfaces of the future, how we help people make the shift from the analog world to the digital world of their computers and other electronic devices. Squeak and Morphic are destined to play a major role in that evolution.

The Squeak Kernel in the Future

In speculating about the future of the Squeak kernel, we are in some sense on relatively firm ground, at least as far as prediction itself can ever be on a solid foundation. The kernel, after all, is in the figurative control of Squeak Central. Though its design clearly allows for extension by other members of the Squeak community, Squeak Central plays a sort of central arbiter role akin to that played in other Open Source community projects by those who are ultimately seen as the keepers of the vision.

This is not to suggest, however, that other members of the Squeak community cannot or will not influence the direction of the kernel. They clearly have and they just as clearly will continue to do so.

Small is Good, Modular is Even Better

The size of the Squeak kernel will ultimately be quite small, almost certainly smaller than 1MB. The Version 1.18 Squeak release image only occupied 968K on the Macintosh, with VM support adding another 290K.

Development work past that early release of the technology bloated the image somewhat but Squeak Central has repeatedly assured the community that it is intent at some point on getting back to these size parameters if not even smaller.

In a message posted to the Squeak mailing list in late 1999, Dan Ingalls said, "[W]e still need to extend our reach…before we will really know what the kernel should be and how it wants to be seen by users." In that same email message, Ingalls warned, "It's going to get worse before it gets better, but it *will* get better."

The size of the kernel is important and interesting, but even more significant is the concept of Squeak modularity. "The monolithic Squeak environment will go away," predicted Squeak Central's Ted Kaehler. He points to then-recent development of ImageSegments as an important bit of technology in terms of making Squeak modular. "ImageSegments are capsules of live objects. We are using them to make stand-alone importable projects with their own Internet URLs," Kaehler says.

ImageSegments is a facility that can begin with a defined list of root objects and write to a binary file all of those objects and other objects to which the original objects point. This allows Squeak developers to create remarkably stand-alone pieces of functionality which can be readily imported into a different Squeak image.

Combined with ImageSegments, the Squeak Central team has been perfecting a technique which will allow developers to create Morphic "SqueakPages" which can be saved to a server as external objects and dynamically loaded into a running Squeak image as needed.

In addition to the obvious advantages ImageSegments present for the delivery of Squeak functionality in small packages, they also facilitate the kind of robust project and change management that is essential to the success of environments like the Web, which must be subject to being updated at any given moment.

The Internet and Distributed Computing Models

As Squeak becomes more and more modular, it lends itself increasingly to distributed computing applications. Older, monolithic versions of Smalltalk were notoriously unfriendly to networking; even creating networked applications was a major undertaking. Squeak, by contrast, is coming of age in an era when the Internet in general and the World Wide Web more specifically are integral parts of the software landscape. It acknowledges this fact and thrives in it.

"We now foresee," Ingalls has said, "the possibility of an extended universe of Squeak worlds, distributed over the Internet as Web pages, downloaded and internalized…and swappable in a reasonable memory footprint…."

This ability to save, send, store, and dynamically reload segments and pages will allow Smalltalk, for the first time, to play a crucial role in the creative development process where collaboration is important. The implications of this capability, as we will see in the second major part of this chapter, are staggering.

It is almost certain that within the next two years, Squeak will also support a notion very close to that of the Java applet: a bit of relatively stand-alone functionality that can be downloaded, embedded into a Web page (or some similar future form factor) on the client, and executed.

Meanwhile, the Audience Evolves

People encountering Squeak for the first time are often bewildered by the question of whether, in a world of the Internet and Java programming, there is room, let alone a need,

for yet another programming language. For those who are already comfortable creating software in Java or C++, Squeak Central has a simple answer: for you, there isn't.

"We expect to grow Squeak in the direction of a significantly broadened user base rather than by attracting or even attempting to attract Java and C++ programmers," Scott Wallace of Squeak Central says.

Expect Squeak's evolution along these lines to focus on the Squeak Toy environment and extensions to it, built on top of the Morphic substrate that has emerged as everyone's preferred way of interacting with and managing Squeak development. The idea is to attract an increasing audience of users who are not programmers but rather multimedia authors and others who can bring to the table contributions that, in Ingalls' words, "are more generally enlightening and educational, enabled by Squeak."

Squeak Central has indicated it plans to focus significant energy on increasing support for first-time users in coming versions of Squeak. Specifically, the team plans to explore four aspects of Squeak to see what kinds of improvements can be made to make Squeak more accessible to new users and non-programmers:

- experiments with alternative syntaxes
- making it easier to create simple applications
- a new, streamlined programming framework focusing on an integrated "object operating table"
- integration of "SqueakToy" scripting tiles

Discussions of audience among the members of Squeak Central often devolve to trying to find a word to describe the user base, composed as it inevitably is of both programming users and what have traditionally been called "end" users as if there were in fact a theoretical end point to the use process. Alan Kay and others have begun adopting the word "omni-user" to encompass all types of empowered users accessing Squeak capabilities and services.

Fitting Squeak Into Your Hand

One of the most exciting and interesting areas of Squeak development has been the emergence of hand-held devices capable of running (and in some cases even supporting development in) Squeak in small memory footprints and on small form-factor screens. Sharp has two Personal Digital Assistants (PDAs) available only in Japan at this writing which support Squeak and on which Squeak programs operate.

Squeak has also been successfully ported to Microsoft's WinCE platform and the Cassiopeia from Casio, among other small form-factor devices.

"We are," says Wallace, "coming closer and closer to the famed DynaBook dream" of Alan Kay's early career.

The portability of Squeak even to places where no operating system yet exists has been a reality for Squeak Central from early in the product's life. In the first year of its development, the Squeak team brought in an undergraduate college student who had never seen Squeak. They asked him to port Squeak to a bare microprocessor then being developed by one of the major semiconductor firms. That chip had only a minimalist software design kit (SDK) available, and it was purely text-based.

"In three weeks," recalls John Maloney of Squeak Central, "this guy had Squeak ported to this chip. Most of that time was eaten up getting peripheral drivers to work so

it could display graphics on a screen in color and respond to keyboard input." The team says that when Kay saw the demo for the first time, he immediately began to demonstrate to others who were present the implications of this. Once you've done the port, a huge body of work and content is suddenly immediately available in the new enviornment.

He asked the student to let him "drive" the demo and proceeded to astound the audience of semiconductor executives by showing off some of his best demonstrations running on this bare-chip implementation of Squeak.

3-D in Squeak

3-dimensional graphics are important elements of realistic multimedia and virtual-reality worlds, but creating them on any modern computer platform within the reach of the average person is so difficult that almost nobody attempts the task. The Squeak community has long been interested in overcoming this obstacle to multimedia creation.

It has a strong ally in the person of Randy Pausch (see Chapter X, "Alice in a Squeak Wonderland," by Jeff Pierce for a detailed discussion of this technology), whose teams first at the University of Virginia and later at Carnegie-Mellon University have created an experimental 3D tool called Alice. This tool has been ported to Squeak from its origins in Python (an object-oriented Open Source scripting language) and is under active use by dozens of teachers, multimedia developers, and tool designers who are striving to make it as easy to create 3D objects as it is to create 2D documents in a word processor.

The Squeak team has felt strongly that 3D support belongs directly in the kernel, or at least at the heart of the language. In late 1999, Squeak Central integrated the 3D engine in the language with the Morphic interface technology which is rapidly becoming the core of the Squeak development environment. This allows the projection of 3D images onto Morphic canvases.

"Our real goal here," explains Maloney, "is to create a place where 2D and 3D objects can both play on the same screen or canvas without any performance penalty. When we accomplish that, we will have gone a very long way toward the Holy Grail Pausch and his colleagues envision."

This will allow for the first time the seamless integration of 3D with existing text and graphical content, a development whose impact on computing it might be difficult to overstate. The entire segment of the software industry which is interested in exploring the 3D representation and visualization of 2D data would be greatly accelerated in its efforts by the availability of this functionality.

Taking a page from the book of Macintosh (not coincidentally the OS of choice for Squeak Central, most of whom were part of Apple Computer at some point), the idea of a 3D Finder has some serious interest among Squeak developers. Rozier has had visions of such a top-level interface for many years, having created an environment called HyperOffice more than 15 years ago. HyperOffice, which Rozier first built in Framework and later re-deployed in Digitalk Smalltalk/V and JavaScript, uses the concept of a building with floors dedicated to certain functionality (like the accounting department) and offices or rooms where individuals keep their projects and documents.

In its Framework incarnation, the program used crude 3-D file drawers but by the time it reached its Smalltalk/V incarnation under the name CyberTalk, it boasted a strong representation of a 3-D office environment.

Users can "beam" between offices, use other users' computers with appropriate permission, and generally get the feeling of inhabiting a real-world building while they accomplish their work goals.

Rozier is, at this writing, finishing a Morphic project bringing this work up to date and implementing it in Squeak. By the time you read this, it will be available through Rozier's Web site, The Pattern (http://www.thepattern.com).

"Extending the flat 2D desktop to a set of rooms and places to navigate among is the next logical step in system user interface design," says Maloney. "Whereas in today's Squeak environment, we switch projects by clicking on icons representing windows and their contents, in the future, we will run down hallways into adjacent and distant rooms containing our various projects."

Major Graphic Changes Coming Soon

In some ways, 3D graphics implementation alongside 2D graphics is a sub-set of a larger future direction Squeak is taking. Squeak Central has indicated it is at this writing close to being able to take full advantage of the graphics acceleration hardware on each kind of machine on which Squeak runs. At the same time, Andreas Raab is finishing up a "Just in Time" version of the graphics engine in Squeak that will compile efficient new, machine-native code for each bitmap movement on the screen. Speed, then, lies just around the bend.

One of the major reasons Squeak Central chose to undertake this step is to permit Squeak to take full advantage of hardware graphics acceleration.

Once this new bitmapped imaging is in place, Squeak Central plans to reconstruct Morphic from scratch, preserving as much of the existing Morphic as possible in the process but making the new Morphs more lightweight objects that can be more easily and compactly manipulated and scripted.

There is also very little doubt that the Squeak community plans to create a blizzard of new widgets for user interface design and construction, probably on top of Morphic. In early 2000, the Squeak mailing lists were abuzz with conversations on the subject and several volunteers were already gathering resources and energy to tackle the problem. Once these widgets are available, it will become easier to create business applications that look and feel like native platform applications. (Not everyone believes that this is either good or necessary, but since Squeak allows for such widget kits, it is guaranteed that they will be built.)

As the new Morphic is completed and widgets are built and ported to the new environment, Ingalls has signaled his intent to create a Squeak version of a 1988 programming concept he and other members of the team presented at the annual Object-Oriented Programming Systems and Languages (OOPSLA) conference called Fabrik. This is a visual "wiring" development environment which would facilitate the assembly of Morphs and widgets into full-blown interfaces and applications.

It is safe to bet that over the next two years, Squeak will become a visually programmable environment in which the creation of easily packaged stand-alone applications with native look-and-feel will become an eminently achievable goal even for a non-programmer.

What Belongs in the Kernel?

In the very early days of Smalltalk, someone suggested that the world of computing had had some of its ideas out of kilter for some time. Asked why there was so much operating-system-like functionality in Smalltalk, Squeak Central's Ingalls replied, "An operating system is a collection of things that don't fit into a language. There shouldn't be one."

Ever since then, the Smalltalk and Squeak worlds have become involved in frequent, largely friendly debates about what should be in the language kernel, what should be in the operating system, and what should be in the language but not in the kernel.

With modularity emerging as a touchstone of the new versions of Squeak that will emerge in the near future, this discussion merits even closer examination. Before the existence of ImageSegments or any equivalent idea, there was little if any *practical* distinction between a Smalltalk image and the Smalltalk kernel. There were, to be sure, technical differences, but to the application builder or end user, the line was largely indistinct and irrelevant.

Now that we are able to externalize and load on demand major pieces of functionality, it is inevitable that the tendency is going to swing strongly toward eliminating from the kernel anything that doesn't need absolutely to be present all the time.

This will mean that some of the elements contained in today's "basic" or "core" image (which remains for practical purposes indistinguishable from what is technically the kernel) will now be externalized. Multimedia graphics, for example, will probably become an externally loadable set of objects rather than integrated into core Squeak. Similarly, Web servers, swiki capability, speech recognition and synthesis, MIDI interfaces, and many other pieces of Squeak that veteran Squeakers have come to think of as core elements of the environment will be loaded as needed.

The other side of this coin is that in a connected world – even where that connection might be file-based URLs on local hardware – the distinction between what is contained in the distributed image and what is stored externally to be loaded as needed becomes largely irrelevant. If you use a bit of code that calls some routine that isn't in the current image you or your user is running, Squeak simply uses the URL of the missing object(s) to load the functionality on the fly and completely transparently.

All of this leads to a suggestion that we will almost certainly see a Squeak-based operating system – or, to be somewhat more accurate, Squeak running in the absence of an operating system – in the near future. Indeed, there has already been one abortive attempt at such a project[1] which even empowered the creation of device drivers in Squeak. ("Imagine being able to write device drivers for new peripherals and form factors entirely in Squeak Morphic," Squeak Centralist John Mahoney enthuses.)

Why would one want to create or use a Squeak OS? When asked that question, Maloney said, "I can see a future in which everything you might want to fine-tune or control on your computer is right there for you to access through a Squeak Morphic interface. You can change anything. You are in control."

The advantages of such an approach, Maloney says, "are transitional. You can just sit there in native mode and real time and manage everything. Given a low-latency Squeak, this is entirely feasible. "All of the missing elements that used to constitute gaps between Squeak and OS capability are now perfectly capable of being filled."

---

[1] A project conducted at Interval Research, a product and company incubator, had a working version of a Squeak OS on a handheld device, but the project was shelved in favor of implementing Microsoft's WinCE instead. The outcome was foreseeable given that Interval is funded in large part by Microsoft co-founder Paul Allen, a lifelong friend of Microsoft founder Bill Gates.

Some Reasonably Clear Directions

Over the next few years, it is fairly certain that Squeak will be applied to a broad range of problems and needs. Many of these are obvious; they derive either from the use to which computers are already being put or they have been made clearly a conscious part of the way Squeak works and is being created.

But some of the ways we are almost certain to see Squeak evolve may not be readily apparent from our present vantage point.

Squeak Gets Corporate MIS Buy-In, a la Linux

IBM, if not uniquely at least unusually among large computer companies, understands the ultimate value of Smalltalk. Its VisualAge development environments for Smalltalk, Basic, C++, and Java, are all based on work done entirely in Smalltalk several years ago. To this day, the company maintains a very active and moderately large group of Smalltalk developers and related programs and technologies.

At the same time, IBM has more recently become an avid advocate of the Open Source movement of which Squeak is also a part. When in late 1999, Big Blue opened its arms to the Linux world and began supporting that free operating system, the rules of computing began a major upheaval, the entire effect of which has yet to be felt.

But IBM is not alone among large global corporations embracing Linux, which has with blinding speed become all but the *de facto* server operating system of choice. Companies like Red Hat Software, VA Research, and LinuxCare have sprung up around this phenomenon and have enjoyed immense market approval in public stock offerings. At this writing, the Linux train is just leaving the station and it's already traveling at the speed of sound.

Squeak may well enjoy the same kind of success in a few years. Rozier says, "Squeak at this point is very nearly in a parallel position with where Linux was two years ago," and he cites the following criteria as evidence to support his contention:

- It is completely Open Source.
- It is surrounded and enhanced by a small but fanatical group of developers and fans.
- It enjoys just enough central management control from Squeak Central to keep it from veering so far off course as to become unusable while allowing for individual creativity.
- It is ubiquitous: it already runs on more platforms today than Linux does and it seems destined to continue to spread to new chips and operating systems all the time.
- It is in widespread use in academic settings where it can grow and be nurtured without the pressure of productization or profitability.

At the same time, Squeak enjoys some advantages over where Linux was at relatively the same point in its growth:

- Because it is a language and a development environment and a deployment platform, Squeak is more useful to a much larger audience. Very few people make decisions about operating systems and even fewer do more than simply run them. Programming environments are a horse of a different color, because they allow mere mortals to create applications, solutions, and projects.

- Built as it is on a very long-term language base going back to Smalltalk-80, Squeak enjoys a legacy of books, experienced programmers, documentation, sample applications, and real programs that form a sound basis for newcomers to learn it.
- It is pre-equipped to deal with the changing world of the Internet and the Web. Whereas it is sometimes difficult to adapt Linux to new technologies as they emerge, Squeak is inherently adaptable and extensible. It includes classes that allow straight-forward abstraction to deal with networking and multimedia needs and issues.
- As we have indicated earlier, Squeak's nature supports dynamic management of its configuration. This in turn results in a huge win for users and for corporate IS/IT departments supporting those users. As the speed at which business is conducted continues to accelerate, reducing the time to adjust to change is a major economic win. Squeak will play a key role in facilitating this cost-reduction by bringing user-configurable and easily programmable configurators and change mechanisms to the desktop.

Ultimately, for these and a number of other reasons related to some of the specific applications we'll be looking at later in this chapter, Squeak could become the next big story in corporate computing.

If Squeak has a downside in all of this, it is simply that it relies on Squeak Smalltalk syntax, which programmers accustomed to C, C++, and Java often find too cumbersome. Even that problem is relatively easily addressed in the "assimilation" strategy of Squeak, though. Anyone can write a module that shows methods in a particular syntax and accepts changes in that syntax. "We have been experimenting with new looks for Squeak with kids in mind," Kaehler says, "but anyone could create, for example, a C-syntax module."

Squeak's Potential to Alter Education Approaches

From its inception, of course, Squeak has been viewed primarily as a language to enhance and support the education of the world's children. The very first Smalltalk efforts were aimed at developing a language that would be accessible to children; in fact, that goal gives the language its name.

Squeak is, if anything, more clearly deliberate in its aim at education as its primary market space. The Squeak Central team are all strongly if not exclusively education-focused. Wallace says, "One of our big dreams for Squeak is for it to disconnect learning from the necessity of sitting in rows in a classroom."

Kaehler envisions a time when a small number of hand-held devices in a remote Third World classroom will be able to connect to a collection of the world's great works of literature such as that embodied in Project Gutenberg. "Can you imagine what learning could take place in such a free-form environment where Squeak could be used to open up all those avenues of learning and wisdom?" he says wistfully.

Rozier is a strong believer in the global importance of this kind of distributed education. "Imagine how much the world will be able to learn from those brilliant, sophisticated wise men and women of indigenous populations once we have a language rich and expressive enough with which we can communicate with one another."

Much of the potential for Squeak to alter radically how students are taught rests in its historical roots in Simula and the degree to which an object-oriented architecture lends itself to the creation of simulations. The SqueakToys capability executed in Morphic in the current release of Squeak is an excellent example of how simulation can be brought to

bear in a learning environment where students add to their knowledge and wisdom by experimentation and experience rather than by passive reception of lectures and limited feedback loops.

Squeak as Hand-Held OS: At the Amusement Park

Today's cellular phones and PDAs allow their users very little control over their behavior or their appearance. Because of their small memory footprints and their primitive interfaces (did you ever try to enter a long, complicated email address into a cell phone with its simple keypad?), these devices tend to stay in fixed configurations. If you want a phone with more functionality, you toss your old one and buy a new one.

Squeak has the potential to change that situation dramatically. If Squeak can run on top of a raw semiconductor CPU, it can certainly replace the operating system running in a cell phone or a PDA. And if it can do that, it can open up worlds of possible extensions, enhancements, and unheard-of functionality for users of these small devices.

Imagine, for example, a family going to a theme park in the not-so-distant future. When they arrive at the gate and pay their admission fee, instead of a fistful of tickets or a paper or magnetic badge, each member of the party is given a palm-sized device with a color LCD screen running a Squeak application.

Throughout their stay at the park, the family members can communicate with one another via text messages or short voice data bursts. Built-in global positioning system (GPS) software and hardware will ensure that lost parents can be easily located by their children. Want to ride the most popular attraction at the park? Rather than traipsing a mile over to the ride to find out there's a 2-hour line, just tap on your Squeak application's screen and get an instant update on the wait. Want to buy something? Just beam your credit card information into the infrared receiver on the cash register of the vendor.

Oh, by the way, you might not want to leave this wonderful little device behind when you go home. That's OK. You can buy it and take it home with you where you can use it to keep up with the latest news about the amusement park owner's business, new cartoons and movies, cable broadcasts, video releases, and to keep up with your own personal email account at the home address of some famous rodent.

All of that and more is certainly well within reach with Squeak as an embeddable system.

In fact, Rozier believes that Squeak might enjoy extremely widespread adoption by telecommunications companies and hand-held device manufacturers. "It is, after all," he points out, "the only Open Source technology suitable to meet the needs of creating both servers and devices needed by these companies. The future of small is Squeak Smalltalk."

There is, in fact, another level on which Squeak's potential importance in the world of operating systems is becoming obvious. At its heart, Squeak has a virtual machine (VM). This virtual machine essentially **replaces** the notion of an operating system with the concept of a true computing environment. In the earliest days of Smalltalk, Dan Ingalls described the operating system as a place to put things that didn't fit into the programming language, going so far as to sayh that there shouldn't even **be** an OS.

Squeak is, in many respects, an operating environment that is at once richer and more malleable than the typical chip-dependent OS. It happens also to have a very powerful and accessible programming language (Smalltalk dialect) in it, but that feature is almost incidental for some purposes and many users.

Squeak Supplants Netscape and Internet Explorer

There is widespread agreement among those who are building content and sites for the Worldwide Web that they are being severely hampered by the inability (some would say refusal) of AOL/Netscape and Microsoft to create a Web browser that is even remotely capable of accessing all the kinds of content people would like to experience.

Browsers were first designed to be fairly passive display pieces. They could be pointed at various URLs and from those sites they could retrieve and display static text and graphics. Over the first two or three years of the Web, browsers evolved rapidly so that they supported interaction, form completion, animated graphics, and then even multimedia. But their mechanisms for supporting these additional types of content were primitive and lacked standards support.

To make your browser show you a movie, for example, you have to have the site owner help you figure out what kind of add-on functionality in the form of a browser plug-in you need. It must then guide you to the place where you can obtain that plug-in, download and install it. Then you often have to close your browser application and restart it, remembering where you were when this whole ordeal began.

When the browser manufacturers tried to implement a more standards-based solution to users' expressed desires to have more fluidity and interactivity in their browsers, they fumbled the ball again. They decided to promulgate something called dynamic HTML (DHTML). But DHTML isn't a single technology; it is a term that describes the rather unhappy marriage of JavaScript scripting, HTML markup, and extensions to both the language and the markup that facilitated the creation and manipulation of individual objects in the browser.

This idea, which is at its heart right and helpful, became ensnared in the usual standards battle with the ultimate result that both Microsoft and AOL/Netscape implemented it so differently that it is all but impossible to write truly cross-browser DHTML pages without a lot of experience and much trial-and-error.

Morphic, particularly with its extended Macromedia Flash player, could provide the solution to this browser dilemma. If an Open Source Squeak project were started with the intent of producing a freely distributed browser that would natively understand how to deal with Flash content and which could be easily adapted – programmatically or behaviorally – to deal with other kinds of content, it would fill a huge gap. It could, in fact, become the new browser of choice, first in corporate settings and behind firewalls where the browser decision is centralized, and later in homes and schools interested in the adaptability of this new design.

Rozier says this will happen. "First," he predicts, "we'll see helper applications and browser companions done in Squeak. When people see the unlimited potential for extension and enhancement Squeak tools represent, the move to a Squeak-based browser, perhaps surrounding Flash as a multimedia file format, will be too compelling to resist."

Squeak-Based Comanche Becomes Cross-Platform Personal Web Server of Choice

Today, there is no cross-platform personal Web server. Linux has Apache, Microsoft Windows has Internet Information Server (IIS), Macintosh has WebSTAR, large-scale UNIX machines have Netscape, and there are a number of other variants, but none of them is cross-platform or ubiquitous.

Why should we want such a beast?

The magic of the Web is the degree to which it allows every individual who owns a node on the Internet to become a publisher of content. Whether the audience for that

publishing effort is a group of co-workers on a LAN or the entire known world on the Web, all of us need or want to share information with others.

Historically, the Internet is built on the TCP/IP protocol suite, which in turn was created to prevent single points of failure on a communications network. That means that every node has some important role to play and that nodes tend to be relatively equal to one another. The Web builds further on those ideas.

Furthermore, people obviously *want* to share or publish information in numbers that were unheard-of before the advent of the Web. How else explain the raging success of such businesses as Geocities and Xoom, which find themselves with *millions* of ordinary people creating and publishing Web sites for the world to see?

In some ways, however, the idea of requiring people who want to share information with others to upload their information to someone else's computer is anti-Internet. If you want to publish content, you have a personal computer almost by definition. So why can't you just turn your computer into a Web server?

You can.

All major home computers today come with a personal Web server. The problem is that these Web servers are closed, proprietary, and limited.

So along comes Comanche, a Squeak-based, open source Web server built on the original Squeak Pluggable Web Server (PWS; see Chapter XX for full details). Comanche is a project being driven principally by Bolot Kerimbaev and Stephen Pair, with the usual supporting cast of many others helping with debugging and feature enhancement.

For all the reasons that Squeak should gain widespread corporate acceptance similar to that attained by Linux in the recent past, Comanche should become the personal Web server of choice in the future. It's cross-platform, Open Source, infinitely extensible, and eminently usable.

Given the boundless nature of the underlying Squeak environment, Comanche becomes a true Web platform in its own right. It will be trivial for people to publish Web sites or Swikis (see Chapter XX) on their own local machines and extend those sites with functionality only dreamed about by today's Web site designers and developers.

Rozier puts it this way: "Linux continues to try to find a suitable client environment at the same time as Windows struggles to become a decent server. Squeak is becoming the first environment to provide the best of both worlds, a truly distributed peer network."

Squeak and the Microsoft .NET Initiative

In mid-2000, Microsoft announced what it characterized as a revolutionary new product strategy which will see all of its products and technologies converge on an interoperable Internet. At the core of this new initiative – which in many ways is not nearly as astounding as Microsoft would have us believe, building as it does on dozens of existing Open Source projects and products – is a language-independent virtual machine. Much of the work on this VM was done by Smalltalk guru David Simmons, who created SmalltalkAgents and Smalltalk 2000. In fact, a Smalltalk-based scripting language called SmallScript will be incorporated into the .NET strategy when it is released some time in the second half of 2001.

This has interesting potential implications for Squeak as well. Simmons has invited the Squeak community to join the SmallScript and .NET initiatives and to port the VM and other technologies to the Squeak platform. At this writing, a number of Squeak advocates are getting ready to do just that.

Ultimately, this could prove to be a huge win for Smalltalk as it enters into a mainstream, widely adopted new application platform, the interoperable World Wide Web.

Squeak Web Server and Scriptable Intelligent Agents

Mobile agents have been an important area of computer research for a number of years. Like other kinds of objects and components, agents need a backplane, a place from which to be dispatched to undertake their masters' bidding and to which they return with their results. At the same time, their value increases with the network effect of an increasingly diverse collection of nodes on which they can operate.

By providing the platform for a truly distributed peer network (see previous section) Squeak will also become the launch pad for these agents. Rozier has spent the past 15 years perfecting a technology he calls Scriptable Intelligent Agents (SIAs) and he sees Squeak as the ultimate platform on which these agents can be nurtured and trained.

The first place such agents might well find acceptance in the world of the Web will be in the emergence of 24/7 custom pricing. A Web site called Priceline.com emerged in late 1999 with a custom pricing model that was sufficiently unique that it was granted a U.S. Patent. The concept was simple: consumers could go to the site, indicate an interest in purchasing, say, an airline ticket to Los Angeles for any time in the week of March 13, and a desire to be able to buy it for $200 or less. Within an hour, Priceline.com infrastructure would search out and identify a vendor willing to match the user's requirements.

Imagine how powerful this idea becomes if you can dispatch your own SIAs to the Web to locate the best prices –within your parameters – for goods and services you wish to purchase. You are no longer tied to a single site's limited resources but you are free to have your agent roam cyberspace looking for bargains and deals and partners. The fundamental shift in pricing this technology represents is ground-shaking in its implications.

And Squeak plays a key role in this development because of its ease of customization, accessible programmability, and incorporation of personal Web servers as "farms" for agents.

Squeak as a Multimedia Tool and Platform

By its nature and origins, Squeak is intensely graphical. As it has rapidly evolved in the past year or so, Morphic has made it even more graphical. It should come as no real surprise, then, that Squeak has also become a fertile playground for multimedia developers and producers, and that much of its near-term future evolution is likely to concentrate in this area.

We've already touched on the issue of including 3D capabilities in the kernel and on the implications of the Alice 3D project being ported to Squeak.

Andreas Raab, at this writing the latest addition to Squeak Central, has been working on multimedia aspects of the language for some time. He implemented the Macromedia Flash player, which was among the first full-scale Squeak tools to demonstrate the feasibility of what Rozier calls "Borg-like assimilation" of external standards and technologies. It is already possible to load any file stored in the publicly documented Macromedia Flash file format into Squeak and play the resulting movie. In the process, we can get our hands on the individual objects within the Flash movie and thus create whole new types and levels of interaction.

But beyond 2D and 3D graphics, beyond even the ability to play Flash and Virtual Reality Modeling Language (VRML) files directly in the Squeak environment and in Morphic worlds, the Squeak community has always been strongly interested in sound. The use of music, voice, and sound effects in computing has not been well-developed. On the Web and in programming environments like Squeak, even less has been done with determining precisely how audio influences ought to be brought to bear on the user experience.

Stephen T. Pope, one of the early Smalltalkers, has for many years been working on MIDI tools and interfaces. His work is included in the Squeak image as of this writing. It represents some of the most interesting and advanced work being done with incorporating musical capabilities into computers.

Already this work has been built into an interactive MIDI Jukebox and a piano-roll type of interface for composing, arranging, playing, and editing music.

Squeak developers have done much with audio engines. Work will now be focused on the creation of great voices for these engines and in the area of file compression. The SqueakTime asynchronous file reader included in the current releases of Squeak are not very efficient at compression, resulting in large file sizes and often poor playback.

In the near future, expect to see an MPEG player (assuming licensing and royalty issues can be worked out satisfactorily).

Ultimately, its developers expect to see Squeak spawn a successful effort to create a dynamite tool for multimedia development that will allow composers and producers the ease of drag-and-drop incorporation and mixture of 2D and 3D graphics, MIDI and compressed sound and music, and animations including Flash movies.

Combining the best of what we've learned from watching thousands of developers use commercial tools, Squeak could form the basis for the creation of a new generation of multimedia development aids by making an infinitely extensible platform core to the success of the tool. Much as Adobe has maintained its leadership in the photo-manipulation market by opening its widely used Photoshop to plug-ins, so Squeak, by allowing extensions to be coded in Morphic and Smalltalk, could allow the end users of a Squeak-based multimedia tool to extend it at will. The results will no doubt be astonishing.

The First Fully Extensible Collaboration Tool?

In building and extending Squeak, the Squeak Central team and the broader community have almost coincidentally managed to create an environment for collaboration that is probably already better than anything on the commercial market.

The sharing of files and live objects between Squeak images is easy and becoming trivially transparent as this is written. The Morphic environment empowers individuals to share graphical representations of their ideas very quickly; the narrow but enthusiastic and successful use of "whiteboard" technology will be dwarfed by what is possible in Squeak.

"We are all interested in generic collaborative tools," Wallace says. "We use them all the time and we know where they are strong and where they are weak. A really general-purpose collaborative tool, infinitely extensible in Squeak, of course, is definitely in the offing if for no other reason than that it grows naturally out of the work we are doing."

Clearly the notion of taking collaboration and its big sibling, online community, to the next level, would greatly benefit from Squeak and Morphic. Not only because it is object-oriented but also because the components already available to facilitate collaboration – and others to come in the near future – make Squeak a powerful and

infinitely extensible platform for the construction of discussion boards and for meaningful real-time interaction with whiteboard-supported chat, graphical instant messaging, and other such feature sets. All of this can be easily integrated into the underlying system because of Squeak's nature as a VM and OS-replacement.

Web-Based Games

The first Web-based game written in Squeak, "Oceanic Panic," made its debut in kiosks at Disney's Epcot Center as this was being written. More are under development.

Games delivered and experienced on the Web would have a number of advantages over those played on either personal computers or dedicated proprietary game-playing platforms:

- They would be easier to install than those on PCs because they would carry their infrastructure – sound support and the like – around with them.
- They would lend themselves more easily to multi-player scenarios since they would by nature be networked and sharable.
- They could be personalized, customized, and extended using Smalltalk, Morphic, and other, similar programming and development techniques that are at the core of Squeak.

Expect to see a huge increase in the number, variety, and kinds of Web-based games and fun educational applications as Squeak moves more and more into this arena.

Finally, the Dynabook?

During the early days of Smalltalk, Kay was going around talking about and describing his vision for a new type of device he dubbed the Dynabook (dynamic book). He saw this device as featuring a graphical interface, connected to external links in some then-unknown way, and embodying a way for children (its users) to program, or extend, it. His ideas grew out of his early experience with object-manipulation programs during his days at the University of Utah.

It seems that Kay has infected most, if not all, of the Squeak Central team with his notions so that it is difficult to have a conversation with any of them about the future of Squeak without the Dynabook being mentioned repeatedly.

Certainly Squeak has moved the marker a long way toward the realization of Kay's dream. Out of the Morphic interface has grown Ingalls' and Kaehler's work with ActiveEssays, which in many ways are the content Kay would have chosen as a primary type of information to be presented on the Dynabook.

The fact that Squeak can be deployed on highly portable, even hand-held, devices, and that such devices are easily connected to wireless networks over which Squeak runs quite efficiently and effectively certainly lead to the conclusion that the Dynabook is finally within reach.