Next Up Previous Contents
Next: Freeing the Source **Up:** Open Sources Voices from **Previous:** The Future

Hardware, Software, and Infoware

Tim O'Reilly

I was talking with some friends recently, friends who don't own a computer. They were thinking of getting one so they could use Amazon.com to buy books and CDs. Not to use ``the Internet,'' not to use ``the Web,'' but to use Amazon.com.

Now, that's the classic definition of a ``killer application'': one that makes someone go out and buy a computer.

What's interesting is that the killer application is no longer a desktop productivity application or even a back-office enterprise software system, but an individual web site. And once you start thinking of web sites as applications, you soon come to realize that they represent an entirely new breed, something you might call an ``information application,'' or perhaps even ``infoware.''

Information applications are used to computerize tasks that just couldn't be handled in the old computing model. A few years ago, if you wanted to search a database of a million books, you talked to a librarian, who knew the arcane search syntax of the available computerized search tools and might be able to find what you wanted. If you wanted to buy a book, you went to a bookstore, and looked through its relatively small selection. Now, tens of thousands of people with no specialized training find and buy books online from that million-record database every day.

The secret is that computers have come one step closer to the way that people communicate with each other. Web-based applications use plain English to build their interface -- words and pictures, not specialized little controls that acquire meaning only as you learn the software.

Traditional software embeds small amounts of information in a lot of software; infoware embeds small amounts of software in a lot of information. The ``actions'' in an infoware product are generally fairly simple: make a choice, buy or sell, enter a small amount of data, and get back a customized result.

These actions are often accomplished by scripts attached to a hypertext link using an interface specification called CGI (the Common Gateway Interface). CGI defines a way for a web server to call any external program and return the output of that program as a web page. CGI programs may simply be small scripts that perform a simple calculation, or they may connect to a full-fledged back-end database server. But even when there's a heavy-duty software engine behind a site, the user interface itself is not composed of traditional software. The interface consists of web pages (which may well have been created by a writer, editor, or designer rather than by a programmer).

Information interfaces are typically dynamic. For example, Amazon.com's presentation of books is driven by sales rankings that are updated every hour. Customers can add comments and ratings on the fly, which then become a key part of the information-rich decision-support interface for purchasers. A site designed to help someone buy or sell stocks online needs to not only present updated share prices, but also the latest relevant news stories, insider trading information, analyst recommendations, and perhaps even user discussion groups. The information interface thus typically consists of a rich mix of hand-crafted documents, program-generated data, and links to specialized application servers (such as email, chat, or conferencing).

Information interfaces are not as efficient for tasks that you do over and over as pure software interfaces, but they are far better for tasks you do only rarely, or differently each time. In particular, they are good for interfaces in which you make choices based on information presented to you. Whether you're buying a book or CD at Amazon.com, or a stock at E*Trade, the actual purchase is a fairly trivial part of the interaction. It's the quality of the information provided to help you make a decision that forms the heart of the application you interact with.

The way the Web is transforming the whole computing paradigm was never clearer to me than back in 1994, before Microsoft had gotten the Web religion, and I shared the stage (via satellite) with Microsoft VP Craig Mundie at an NTT event in Japan. Mundie was demonstrating the planned interface for Microsoft's ``Tiger'' server, which was supposed to enable video on demand. The interface emulated Windows, with cascading menus responding to a kind of virtual remote control channel clicker.

It was pretty obvious to those of us who were involved in the Web that the right interface for video on demand, when and if it comes, will be a Web-like information interface. It's ironic that even then, Microsoft had the perfect interface for video-on-demand: its own CD-ROM-based movie encyclopedia, Cinemania. What better way to choose what movie to watch than to search by category, read a few reviews, watch a few film clips, and then, homework done, click on a hypertext link to start the movie? Cinemania has it all but the last step. It's not until hypertext-based information products are connected to network servers that their real power becomes apparent. Suddenly, information is not an end in itself, but an interface that allows a user to control an application space far too complex for a traditional software application. (Amazon.com clearly knows this: their purchase of the Internet Movie Database, a collection of user reviews and other information about movies, will put them in pole position not only for selling videotapes online, but as a future gateway for video-on-demand services.)

Information interfaces are particularly appropriate for decision-support applications, but they also make sense for one-time tasks. In a sense, the use of ``wizards'' for software installation is an example of the same trend.

There are also information applications that use a simpler, more software-like interface for user interaction, but provide dynamic information output. My favorite example is something that would have been virtually unthinkable as an application only a few years ago: getting maps and directions. A mapping site like `http://maps.yahoo.com` lets you type in two addresses, and get back a map and a set of directions showing how to get to one from the other.

So what does all this have to do with Open Source software?

There's an obvious answer: most of the technologies that make the Web possible are Open Source.

The Internet itself -- features like the TCP/IP network protocol and key infrastructure elements such as the Domain Name System (DNS) were developed through the open-source process. It's easy to argue that the open-source BIND (Berkeley Internet Name Daemon) program that runs the DNS is the single most mission-critical Internet application. Even though most web browsing is done with proprietary products (Netscape's Navigator and Microsoft's Internet Explorer), both are outgrowths of Tim Berners-Lee's original open-source web implementation and open protocol specification. According to the automated Netcraft web server survey (`http;//www.netcraft.co.uk/survey`), more than 50% of all visible web sites are served by the open-source Apache web server. The majority of web-based dynamic content is generated by open-source scripting languages such as Perl, Python, and Tcl.

But this obvious answer is only part of the story. After all, why is it the Web and not some proprietary technology that is the basis for the networked information applications of the future?

Microsoft actually was ahead of the curve in realizing the power of online multimedia. In 1994, when the Web started to take off, Microsoft's CD-ROM products like Encarta, their online encyclopedia, and Cinemania, their online movie reference, were ahead of the Web in providing online hyperlinked documents with rich multimedia capabilities. Microsoft even realized that it was important to provide information resources via online networks.

There was only one problem with Microsoft's vision of the Microsoft Network: barriers to entry were high. Publishers were expected to use proprietary Microsoft tools, to apply and be approved by Microsoft, and to pay to play. By contrast, anyone could start a web site. The software you needed was free. The specifications for creating documents and dynamic content were simple, open, and clearly documented.

Perhaps even more important, both the technology and the Internet ethic made it legitimate to copy features from other people's web sites. The HTML (HyperText Markup Language) pages that were used to implement various features on a web site could be easily saved and imitated. Even the CGI scripts used to create dynamic content were available for copying. Although traditional computer languages like C run faster, Perl became the dominant language for CGI because it was more accessible. While Perl is powerful enough to write major applications, it is possible for amateurs to write small scripts to accomplish specialized tasks. Even more important, because Perl is not a compiled language, the scripts that are used on web pages can be viewed, copied, and modified by users. In addition, archives of useful Perl scripts were set up and freely shared among web developers. The easy cloning of web sites built with the HTML+CGI+Perl combination meant that for the first time, powerful applications could be created by non-programmers.

In this regard, it's interesting to point out that the software industry's first attempts to improve on the web interface for active content -- technologies like browser-side Java applets and Microsoft ActiveX controls -- failed because they were aimed at professional programmers and could not easily be copied and implemented by the amateurs who were building the Web. Vendors viewed the Web in software terms, and didn't understand that the Web was changing not only what applications were being built but what tools their builders needed.

Industry analysts have been predicting for years that Perl and CGI will be eclipsed by newer software technologies. But even now, when major web sites employ large staffs of professional programmers, and newer technologies like Microsoft's Active Server Pages (ASP) and Sun's Java servlets are supplanting CGI for performance reasons, Perl continues to grow in popularity. Perl and other open-source scripting languages such as Python and Tcl remain central to web sites large and small because infoware applications are fundamentally different than software applications and require different tools.

If you look at a large web site like Yahoo!, you'll see that behind the scenes, an army of administrators and programmers are continually rebuilding the product. Dynamic content isn't just automatically generated, it is also often hand-tailored, typically using an array of quick and dirty scripting tools.

``We don't create content at Yahoo! We aggregate it,'' says Jeffrey Friedl, author of the book *Mastering Regular Expressions* and a full-time Perl programmer at Yahoo. ``We have feeds from thousands of sources, each with its own format. We do massive amounts of `feed processing' to clean this stuff up or to find out where to put it on Yahoo!.'' For example, to link appropriate news stories to tickers at `http://quotes.yahoo.com`, Friedl needed to write a ``name recognition'' program able to search for more than 15,000 company names. Perl's ability to analyze free-form text with powerful regular expressions was what made that possible.

Perl is also a central component in the system administration infrastructure used to keep the site live and current. Vast numbers of Perl scripts are continually crawling the Yahoo! servers and their links to external sites, and paging the staff whenever a URL doesn't return the expected result. The best-known of these crawlers is referred to as ``the Grim Reaper.'' If an automated connection to a URL fails more than the specified number of times, the page is removed from the Yahoo! directory.

Amazon.com is also a heavy user of Perl. The Amazon.com authoring environment demonstrates Perl's power to tie together disparate computing tools; it's a ``glue language'' par excellence. A user creates a new document with a form that calls up a Perl program, which generates a partially-completed SGML document, then launches either Microsoft Word or GNU Emacs (at the user's choice), but also integrates CVS (Concurrent Versioning System) and Amazon.com's homegrown SGML tools. The Amazon.com SGML classes are used to render different sections of the web site -- for example, HTML with or without graphics -- from the same source base. A Perl-based parser renders the SGML into HTML for approval before the author commits the changes.

Perl has been called ``the duct tape of the Internet,'' and like duct tape, it is used in all kinds of unexpected ways. Like a movie set held together with duct tape, a web site is often put up and torn down in a day, and needs lightweight tools and quick but effective solutions.

Microsoft's failed attempt to turn infoware back into software with ActiveX is rooted in the way paradigms typically shift in the computer industry. As a particular market segment matures, the existing players have an enormous vested interest in things continuing the way they are. This makes it difficult for them to embrace anything really new, and allows -- almost requires -- that new players (``the barbarians,'' to use Philippe Kahn's phrase) come in to create the new markets.

Microsoft's ascendancy over IBM as the ruling power of the computer industry is a classic example of how this happened the last time around. IBM gave away the market to Microsoft because it didn't see that the shift of power was not only from the glass house to the desktop, but also from proprietary to commodity hardware and from hardware to software.

In the same way, despite its attempts to get into various information businesses, Microsoft still doesn't realize -- perhaps can't realize and still be Microsoft -- that software, as Microsoft has known it, is no longer the central driver of value creation in the computer business.

In the days of IBM's dominance, hardware was king, and the barriers to entry into the computer business were high. Most software was created by the hardware vendors, or by software vendors who were satellite to them.

The availability of the PC as a commodity platform (as well as the development of open systems platforms such as Unix) changed the rules in a fundamental way. Suddenly, the barriers to entry were low, and entrepreneurs such as Mitch Kapor of Lotus and Bill Gates took off.

If you look at the early history of the Web, you see a similar pattern. Microsoft's monopoly on desktop software had made the barriers to entry in the software business punishingly high. What's more, software applications had become increasingly complex, with Microsoft putting up deliberate barriers to entry against competitors. It was no longer possible for a single programmer in a garage (or a garret) to make an impact.

This is perhaps the most important point to make about open-source software: it lowers the barriers to entry into the software market. You can try a new product for free -- and even more than that, you can build your own custom version of it, also for free. Source code is available for massive independent peer review. If someone doesn't like a feature, they can add to it, subtract from it, or reimplement it. If they give their fix back to the community, it can be adopted widely very quickly.

What's more, because developers (at least initially) aren't trying to compete on the business end, but instead focus simply on solving real problems, there is room for experimentation in a less punishing environment. As has often been said, open-source software ``lets you scratch your own itch.'' Because of the distributed development paradigm, with new features being added by users, open-source programs ``evolve'' as much as they are designed.

Indeed, the evolutionary forces of the market are freer to operate as nature ``intended'' when unencumbered by marketing barriers or bundling deals, the equivalent of prosthetic devices that help the less-than-fit survive.

> Evolution breeds not a single winner, but diversity.

It is precisely the idiosyncratic nature of many of the open-source programs that is their greatest strength. Again, it's instructive to look at the reasons for Perl's success.

Larry Wall originally created Perl to automate some repetitive system administration tasks he was faced with. After releasing the software to the Net, he found more and more applications, and the language grew, often in unexpected directions.

Perl has been described as a ``kitchen sink language'' because its features seem chaotic to the designers of more ``orthogonal'' computer languages. But chaos can often reveal rich structure. Chaos may also be required to model what is inherently complex. Human languages are complex because they model reality. As Wall says in his essay in this volume, ``English is useful because it's a mess. Since English is a mess, it maps well onto the problem space, which is also a mess... . Similarly, Perl was designed to be a mess (though in the nicest of possible ways).''

The Open Source development paradigm is an incredibly efficient way of getting developers to work on features that matter. New software is developed in a tight feedback loop with customer demand, without distortions caused by marketing clout or top-down purchasing decisions. Bottom-up software development is ideal for solving bottom-up problems.

Using the open-source software at the heart of the Web, and its simpler development paradigm, entrepreneurs like Jerry Yang and David Filo were able to do just that. It's no accident that Yahoo!, the world's largest and most successful web site, is built around freely available open-source software: the FreeBSD operating system, Apache, and Perl.

Just as it was last time around, the key to the next stage of the computer industry is in fact the commoditization of the previous stage. As Bob Young of Red Hat, the leading Linux distributor, has noted, his goal is not to dethrone Microsoft at the top of the operating systems heap, but rather, to shrink the dollar value of the operating systems market.

The point is that open-source software doesn't need to beat Microsoft at its own game. Instead it is changing the nature of the game.

To be sure, for all their astronomical market capitalization, information-application providers such as Amazon.com and Yahoo! are still tiny compared to Microsoft. But the writing on the wall is clear. The edges of human-computer interaction, the opportunities for computerizing tasks that haven't been computerized before, are in infoware, not in software.

As the new ``killer applications'' emerge, the role of software will increasingly be as an enabler for infoware. There are enormous commercial opportunities to provide web servers, database backends and application servers, and network programming languages like Java, as long as these products fit themselves into the new model rather than trying to supplant it. Note that in the shift from a hardware-centric to a software-centric computer industry, hardware didn't go away. IBM still flourishes as a company (though most of its peers have down-sized or capsized). But other hardware players emerged who were suited to the new rules: Dell, Compaq, and especially Intel.

Intel realized that the real opportunity for them was not in winning the computer systems wars, but in being an arms supplier to the combatants.

The real challenge for open-source software is not whether it will replace Microsoft in dominating the desktop, but rather whether it can craft a business model that will help it to become the ``Intel Inside'' of the next generation of computer applications.

Otherwise, the Open Source pioneers will be shouldered aside just as Digital Research was in the PC operating system business by someone who understands precisely where the current opportunity lies.

But however that turns out, open-source software has already created a fork in the road. Just as the early microcomputer pioneers (in both hardware and software) set the stage for today's industry, open-source software has set the stage for the drama that is just now unfolding, and that will lead to a radical reshaping of the computer industry landscape over the next five to ten years.

Next Up Previous Contents
Next: Freeing the Source **Up:** Open Sources Voices from **Previous:** The Future

Download this document: *[src.tar.gz][ps.gz][html.tar.gz][dvi.gz]*

Open Resources (www.openresources.com) Last updated: 1999-08-06