THE WEB AS ENABLING TECHNOLOGY FOR SOFTWARE DEVELOPMENT AND DISTRIBUTION

Peyman Oreizy • University of California at Irvine • peymano@ics.uci.edu
Gail Kaiser • Columbia University • kaiser@cs.columbia.edu

Omnia mutantur, nos et mutamur in illis. (All things change, and we change with them.)
—Matthias Borbonius
Delicie Poetarum Germanorum, i. 685

When confronted with a new technology, we instinctively consider it within the context of existing work and practices. Such is the case with the World Wide Web. But the Web is more than a new technology for leveraging existing work. It is, in fact, an enabling technology with the potential to change software development as dramatically as the transistor and microprocessor changed computer architecture.

An enabling technology changes the fundamental assumptions ingrained in a discipline. The microprocessor, for example, changed the reliability, cost, circuit density, and performance assumptions underlying hardware design. As a result, new applications and design approaches for hardware systems became feasible.

Just as the microprocessor changed the fundamental assumptions of hardware design, the Web changes some of the assumptions underlying software development. Thus it has the potential to change our notion of the software artifact and the collaborative processes used to construct it.

WHAT ASSUMPTIONS ARE CHANGING?

It is impossible to accurately predict the impact of the Web on software development. Rapid change and complex technological issues combine to make any prediction highly suspect. We instead focus on the changing assumptions themselves without trying to predict their consequential effects.

- *Accessible, cheap, direct customer channel*. The Web has dramatically reduced the costs and delays associated with distributing information. The result is a cheap, direct, and easily accessible communication channel between customers and software vendors that makes software distribution a potentially collaborative process.

  Customers have already begun using the Web to find and compare products, request product features, submit bug reports, and receive product support. Indeed, the Web provides an astonishing amount of information about commercial software products. Surprisingly, software vendors provide only a small part of this product information. Users themselves produce a large percentage by collaborating to compile, maintain, and distribute FAQs; by managing product-related Web sites; and by participating in Usenet newsgroups, mailing lists, and Internet Relay Chat (IRC) discussions.

  Although electronic software distribution (ESD) was popular among shareware vendors before the advent of the Web, few large commercial software vendors used the technology. The ubiquity of the Web has mobilized the software industry toward ESD for secure sales and distribution of software. Compared with traditional distribution channels, which require product packaging, shipping, warehousing, and retail shelf space, ESD is significantly faster and less expensive. Thus, new distribution models, such as weekly updates, become cost-effective, as does the distribution of small or inexpensive applications. Alternative software pricing and licensing models, such as per-use, per-function, subscription, rental, and lease, also become more practical.
- *Remote, frequently updated resources*. High software distribution costs and long lead times forced commercial software vendors to adopt product development cycles of 12 to 18 months or more. The Web significantly reduces these costs and lets vendors provide software updates, documentation, tutorials, help files, answers to FAQs, and bug fixes as they become available, in a timely, incremental, and cost-effective manner. As a result,

products released via the Web typically have significantly shorter development cycles and are commonly referred to as "developed on Internet time."

- *High software distribution costs* have traditionally discouraged large-scale usability testing of early prototypes. Long development cycles combined with limited end-user feedback increase product development risk. Using the Web as a cheap, direct, ubiquitous customer channel for deploying early prototypes enables direct collaboration with more customers. Aqueduct's Profiler,* for example, instruments software systems to collect operational usage data and transmit it to the software vendor for detailed analysis. Such technologies, built atop the Web, provide better customer feedback and facilitate vendor and user collaboration.
- *New medium of software distribution.* As the available communication and distribution medium has evolved, so has the software artifact. We can expect the software artifact to continue evolving to effectively utilize the Web's unique properties. This change has far-reaching consequences, which we discuss later.
- *Large, globally accessible information space.* The Web provides a globally shared, hyperlinked information space in which the physical location of information is largely transparent to users. By significantly reducing the difficulty and costs associated with browsing and extending shared information spaces, the Web has facilitated geographically distributed, collaborative development of complex products. The Apache Web Server project* and Madefast* are examples of such development. Recent efforts to extend the Web infrastructure with distributed authoring facilities, such as WebDAV,* hold promise in furthering collaborative development.
- *Internet-based collaboration tools.* Although the recent flurry of Internet-based collaboration tools, such as Mirabilis' ICQ* and Microsoft's Netmeeting,* are not based on Web technologies per se, the Web provided a critical mass of users for such tools. Netmeeting's application-sharing and videoconferencing capabilities, for example, have been used for customer support. Technical support personnel can remotely connect to customers' machines and interact with their applications to resolve problems. Geographically distributed developers can use similar real-time collaboration tools for remote code inspection and debugging.
- *Large information space searches.* A large information space requires a practical and efficient mechanism for locating information. Keyword-based Web indexing technologies, such as AltaVista* and Excite,* make it feasible to effectively search unstructured information. Categorization-based indexing, exemplified by Yahoo!,* is an effective way to search smaller information spaces.

  Both keyword- and categorization-based techniques have been used for Web-based software component repositories. For example, Gamelan,* a popular Java component and resource repository, has indexed more than 10,000 artifacts. Such mechanisms help developers locate relevant reusable components, designs, and expertise.
- *Simplicity, extensibility, and standardization.* The Web is based on several simple, extensible standards (such as URL, HTTP, and HTML). Simultaneously achieving these properties has resulted in significant benefits: simplicity facilitated adoption, extensibility facilitated evolution and customization, and standardization facilitated interoperability and heterogeneity. Software developers strive to build systems with similar benefits but, arguably, are rarely as successful. We should not underestimate the importance of these properties.

Given the Web's evolving nature, this list of assumptions will undoubtedly grow. The rest of this column explores just the idea that the Web is a new software medium. By addressing this assumption in detail, we hope to clearly demonstrate the Web's dramatic potential for changing the software artifact and the collaborative processes by which it is developed.

THE WEB AS A NEW SOFTWARE MEDIUM

The software artifact has already evolved to leverage the unique properties of past communication and distribution media such as magnetic tapes, floppy disks, CD-ROMs, and networked PCs. The CD-ROM's tremendous capacity, for example, provided a cost-effective medium for distributing large software systems and content. As a result, multimedia reference and entertainment software evolved as a new genre of software systems that took advantage of the medium's unique properties.

The Web represents a new medium of information exchange based on a globally networked, distributed, and linked information space. Even though the medium is still young, some of its effects on software are already apparent. Miniature applications, or applets, have become commonplace on the Web. Several interesting characteristics set applets apart from traditional software applications:

- Applets are automatically downloaded, installed, and executed without user involvement.
- Applets reside on Web pages, executing automatically when the user arrives at the page and stopping when the user leaves the page.
- Applets are embedded within the information provided on a Web page, as opposed to traditional applications, which create and embed the information.
- Applets are dynamically, incrementally, and transparently loaded on an as-needed basis.
- Applets are cached by the browser for efficiency.
- For security reasons, applets have restricted access to the client environment.

In effect, the software artifact has inherited several properties of the Web medium. In the context of these six applet characteristics, software behaves like just another browser content type. Consider the actions a Web browser performs to display a Web page: the browser retrieves and parses the HTML text, retrieves any embedded elements, determines the content type of each embedded element, and invokes the appropriate content handler to display the content. In the case of an embedded applet, the browser's content handler is a virtual machine interpreter that regards the content as executable instructions.

Although these characteristics seem odd at first, they make sense when we realize that on the Web the goal is to communicate information. Whether we use HTML, animated GIFs, applets, or Dynamic HTML is inconsequential. *On the Web, information matters; software is incidental.*

Until recently, the Web was based solely on a "pull" content model, with users explicitly requesting information from a provider. But new technologies such as Marimba's Castanet* have extended the medium to support a "push" content model as well. In the push model, users subscribe to an information source that automatically transmits the information to them whenever it changes, much like television and radio.

Software distributed using the push model has several unique characteristics:

- Customers "subscribe" to software, enabling software updates to be automatically downloaded and installed without user involvement. This blurs the traditional distinction between software versions.
- Fine-grained control over software subscription enables customers to tailor the software to their particular needs.

Push technology also enhances Web-based collaboration, since either party may initiate interaction. Several users who have registered interest in the same piece of information, for example, are automatically notified whenever any one of them updates that information. For instance, when code components are modified, software developers working on dependent code can be notified automatically of updates that affect their work. This eliminates the need to periodically poll for changes to the information.

EVALUATING EXISTING TECHNOLOGIES

Existing technologies provide only some of the capabilities needed to effectively leverage and experiment with the Web's unique aspects. Let's examine the effectiveness of four current technologies in enabling software systems to leverage the Web for communication and distribution.

Browser Plug-Ins

Web browser plug-ins* enable independent extension of content types, thereby enabling third-party developers to extend the Web browser without changing browser source code. Each plug-in encapsulates the functionality necessary to display a particular content type as a software component (typically implemented as a dynamic link library). Web browsers augment their internally supported content types with those supported by plug-ins. If a plug-in provides the viewer for a particular content type, the browser dynamically loads and executes the plug-in to display the content within a particular window region.

The plug-in mechanism is adequate for supporting new content types but fails to leverage several of the Web's unique properties. For example, users must explicitly download and install plug-ins, and that typically requires quitting and restarting the browser before new plug-ins can be used.

ActiveX and Java Applets

ActiveX* components and Java* applets are the two most popular mechanisms for implementing applets. From our perspective, the most significant differences between the two are that they adopt different security models and that ActiveX components are applets written for specific platforms, whereas Java applets are bytecode interpreted and chiefly platform independent. The ActiveX security model resembles that of a retail software store. The user is asked to approve the installation and subsequent execution of an applet on the basis of the name of the company that developed the applet. Once approved, an applet has unrestricted access to the client machine. With Java applets, most Web browsers implement a "sandbox" security model whereby applets execute within a severely restricted environment on the client machine. This restricted execution environment prevents applets from acting maliciously, so that users needn't make security decisions.

Unlike plug-ins, ActiveX and Java applets can be automatically downloaded, installed, and executed with little user involvement. But both suffer several shortcomings that restrict how much they can leverage the medium. Both technologies restrict the applet's execution context to a single Web page. Thus, an applet's execution context cannot be preserved beyond a Web page unless the context is transmitted to the Web server as the user leaves the source page and is subsequently downloaded when the user arrives at the destination page.

The sandbox security model further restricts Java applets because they cannot tailor their behavior to the user's environment, since they are prevented from querying and otherwise accessing the client machine. Furthermore, only applets downloaded from the same Web page are allowed to communicate directly with each other. Applets are otherwise restricted to communication with their host machine, and applets downloaded from different Web pages must communicate through their respective hosts.

Castanet

Marimba's Castanet* represents early efforts at extending the Web medium to support the push model for software deployment. Two components make up Castanet's functionality: the Castanet Tuner and the Castanet Transmitter. The Castanet Tuner executes on client machines, letting users subscribe to and receive content from multiple content providers. The Castanet Transmitter executes on the server, maintaining a list of subscribers and efficiently distributing the content to them as it changes.

Content distributed through Castanet is browser independent and persists until the user explicitly cancels the subscription. This alleviates the execution context limitations inherent in the Java applet and ActiveX mechanisms. The Castanet Transmitter can also personalize content on the basis of information collected from individual users.

Castanet can distribute any content, including software applications, but during subscription users must approve content containing software. Once approved, applications have unrestricted access to the client machine. Java applications may optionally execute within a restricted environment similar to that of a Web browser without user approval.

PROTOTYPING A FLEXIBLE ENVIRONMENT FOR APPLETS

Although existing technologies have enabled software systems to leverage some aspects of the Web medium, their current limitations and assumptions prevent us from freely exploring the medium. For example, existing technologies discourage—and in some cases prevent—interapplet communication.

At UC Irvine, we are prototyping an environment that overcomes some of the limitations found in existing technologies. Although our prototype is in many ways incomplete, initial experiments have been encouraging.

Our environment consists of a Web browser, a component repository, a command shell, and a component integration tool. The command shell works much like a Unix command shell in that it lets users compose behaviors by combining components from the repository. Components in the environment adhere to a canonical structure that requires them to communicate exclusively using a message broadcast mechanism. Through the command shell, users can directly control the message routing mechanism and modify the component bindings during runtime.

New components are added to the repository using the browser. A user who locates a desired component on the Web installs it by selecting its hyperlink. The browser responds by downloading the binary file representing the component and invoking the component integration tool. This tool places the component in the repository and executes its installation script. As part of installation, the component can examine and modify the user's environment or install other components.

Although simplistic, our environment is unique in that the explicit communication model encourages intercomponent communication, even if the components are from different vendors. Components may also query and adapt to the user's changing environment.

CONCLUSIONS

The Web's unique properties raise many interesting issues and questions for software developers. Emerging Web-related technologies should compel us to change fundamental assumptions and reevaluate our approaches to software development and distribution. ■

ACKNOWLEDGMENTS

Mark Bergman, Erin Bradner, Debbie Dubrow, Peter Kammer, and Roy Fielding provided valuable insights on this work.

URLs FOR THIS COLUMN

***ActiveX** • www.microsoft.com/activex/
***AltaVista** • www.altavista.digital.com/
***Apache Web Server project** • www.apache.org/
***Aqueduct Software** • www.aqueduct.com/
***Excite** • www.excite.com/
***Gamelan** • www.gamelan.com/
***Java** • www.javasoft.com/
***Madefast** • madefast.stanford.edu/
***Marimba's Castanet** • www.marimba.com/
***Microsoft's Netmeeting** • www.microsoft.com/netmeeting/
***Mirabilis' ICQ** • www.icq.com/
***Netscape's Plug-in Developer's Guide** home.netscape.com/eng/mozilla/3.0/handbook/plugins/pguide.htm
***WebDAV** • www.ietf.org/html.charters/webdav-charter.html
***Yahoo!** • www.yahoo.com/