



# Reports of the Web's Death Are Greatly Exaggerated

**Tommi Mikkonen**, *Tampere University of Technology, Finland*

**Antero Taivalsaari**, *Nokia Research Center, Finland*

**The development of entirely new types of Web-based software systems built to leverage the vast capabilities of the World Wide Web allows the use of dynamically downloaded applications and services from any type of terminal, including both desktop computers and mobile devices, implying radical changes in the ways people develop, deploy, and use software.**

In a 2010 *Wired* magazine article, Chris Anderson and Michael Wolff provocatively claimed that “the (World Wide) Web is dead.”<sup>1</sup> They based this claim on two main arguments. The first was that the amount of Internet traffic generated by webpage downloads has decreased dramatically over the years compared to the traffic generated by video and music downloads. The second argument was that users soon will no longer surf webpages with a traditional Web browser because, for the vast majority of Web services such as e-mail, Facebook, Twitter, and Skype, they will prefer custom-built native applications over open, unfettered Web browser access. Anderson and Wolff argued that the trend toward such apps will be even more evident in the mobile device space, in which—according to the authors—the use of Web browsers has already lost the battle against custom-built native apps.

Anderson and Wolff’s first argument has been widely refuted in the press and on the Web ([www.smallfish-bigpond.com/2010/08/wired](http://www.smallfish-bigpond.com/2010/08/wired)). The observation that the amount of

text-based Web traffic is insignificant compared to other types of traffic, while literally true, is highly misleading because, at the same time, webpage and Web browser usage has increased dramatically—nearly exponentially, in fact. Consequently, the notion that video and music downloads generate most network traffic is irrelevant and in no way confirms that the Web itself is dead. Furthermore, Web browsers initiate the vast majority of bandwidth-consuming data traffic on the Internet—that is, it can be claimed that to a substantial degree, the music and video downloads are just “subtypes” of the more general Web traffic. The fact that such traffic consumes more bandwidth than text-based HTML, Cascading Style Sheets, and JavaScript code downloads is self-evident and irrelevant from the viewpoint of attempting to prove the Web’s demise.

Interestingly, Anderson and Wolff’s argument about the transition from open, browser-based Web access to custom-built native apps has generated surprisingly little discussion. While it is tempting to think that the popularity of custom-built native applications for mobile devices would somehow confirm that use of the open Web will fail in the mobile device space,<sup>2</sup> it’s likely that the trend toward such custom applications is only temporary. In fact, we believe that the use of open Web applications will eventually surpass the use of custom native applications on mobile devices.

## TOWARD THE WEB AS A SOFTWARE PLATFORM

The World Wide Web is arguably the most powerful medium for information-sharing and distribution in the history of humankind. Therefore, it is not surprising that use of the Web has spread to many areas outside its origi-

nal intended use. These days, everyday artifacts such as photos, music, videos, newspapers, and technical documents are widely available on the Web. Online banking and stock trading have become commonplace. Official documents that used to be difficult to access, such as municipal zoning regulations, government budget documents, and tax records, are now readily accessible on the Web. Various industries such as banking, financial services, book and electronics retailing, and music and video distribution have undergone dramatic transformations. Web-based services such as Facebook and Twitter have altered the meaning of social life.

More recently, the Web has begun to transform the political landscape as well, making it easy even for ordinary people to disseminate and discuss political information, and to arrange grassroots campaigns for various causes—much to the chagrin of people in power, who were previously able to centralize and control the flow of information in mass media.

It is quite possible that so far we have seen only the tip of the iceberg with regard to the impact the Web will eventually have on the broader society.


### Software system deployment

Given that the majority of useful information is available on the Web, the Web browser has become the most common—or often the only—computer program that most people use. Therefore, although the Web browser was not originally designed to be a software platform, the Web has become more important as a deployment environment for various types of software systems. Systems that previously required substantial investments or distribution costs are increasingly available as services on the Web, as evidenced by the popularity of cloud computing systems and the recent success of companies such as Salesforce ([www.salesforce.com](http://www.salesforce.com)).

From the software development viewpoint, the Web offers numerous benefits compared to conventional binary end-user software:

- *Instant worldwide deployment.* Web applications distributed over the public Web require no middlemen, distributors, or app stores. Once a webpage or application is deployed on the Web—whether in Tampere (Finland), Dar-es-Salaam (Tanzania), Kyoto (Japan), or Menlo Park (California)—it is instantly available to everyone connected to the Web.
- *No manual installations or upgrades.* Web applications are generally available simply by entering the correct URL in the Web browser. Unlike binary applications, they do not require manual installation. Furthermore, the user typically runs the latest version; HTML5-based applications also can support offline use without an active network connection.

- *Platform independence.* Web applications running in the Web browser are intended to be platform-independent; from the end user's viewpoint, the Web browser will effectively assume the role of an operating system. Although there are still various compatibility issues between different browsers,<sup>5</sup> these issues are relatively minor in comparison to developing software for currently available operating systems. Libraries such as Dojo (<http://dojotoolkit.org>) and jQuery (<http://jquery.com>) are available to hide some of the most annoying differences.
- *Ubiquitous, seamless access to data.* The Web enables data sharing on a scale that exceeds the capabilities of any previous systems. Basically, most of the important public data in the world is already available in digital form on the Web. The availability of such data has given rise to entirely new types of applications and mashups that dynamically combine data in unforeseen ways. Technical details related to data access are irrelevant; service providers are responsible for managing the software infrastructure, access interfaces, and databases, and for ensuring their compatibility.



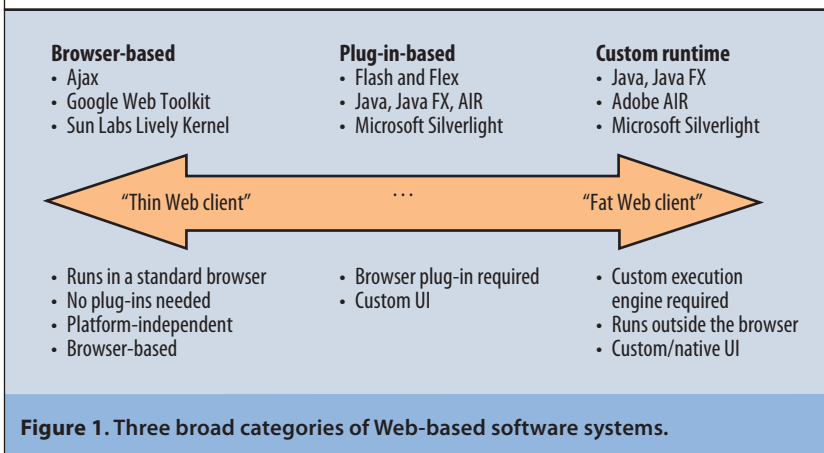
**From the software development viewpoint, the Web offers numerous benefits compared to conventional binary end-user software.**

While the ability to share data is the Web's ultimate strength, these other aspects are critical from the perspective of application development and deployment. The capacity for worldwide deployment is incredibly powerful, especially when combined with the notion of “zero-installation software”—software that does not require any manual installation or upgrades. Such software can evolve at a pace that is almost beyond the imagination of conventional binary software developers, who are accustomed to offering “nano releases”—software upgrades occurring on a daily basis or perhaps even hourly.

### Web-based software systems

Because the Web is not yet a uniform platform, discussions about Web-based software frequently refer to very different types of systems. Figure 1 illustrates three broad categories of Web-based software systems:

- *browser-based*—systems that use only the software that the standard Web browser offers;
- *plug-in-based*—hybrid systems in which the Web browser uses a custom runtime such as Microsoft Silverlight; and



- *custom runtimes*—systems in which application execution requires a special runtime environment, such as Microsoft’s Common Language Runtime (for Microsoft Silverlight) or the Java Runtime Environment; applications run outside the Web browser.

Browser-based systems offer the best application portability across different types of computers and operating systems, but these systems have various limitations related to the lack of low-level platform or device APIs and interactive graphics capabilities. The developers are essentially at the mercy of the limited APIs available in the standard Web browser. Custom runtimes can offer comprehensive graphics capabilities, extensive libraries, and customized development and debugging tools, but at the cost of application portability—basically, such applications do not run in a standard, open Web browser. In many companies, the installation of custom runtimes or additional browser plug-ins is explicitly prohibited for security reasons; this further limits the use of such systems for the development of truly portable Web applications.

A fourth category of Web-connected applications—*custom-built native apps*—is extremely popular today. For example, on Apple’s iPhone and iPad devices, as well as on Google’s Android devices, users typically apply custom-built native apps rather than a Web browser to access Facebook, Twitter, and many other popular Web services. Although they are not actually Web applications, these apps use the same network protocols to access the back-end services as the Web browser does. Developers can use the native graphics libraries to customize the look and feel of such apps to fit the specific needs of the application and the device; these apps can also leverage device-specific features much more comprehensively than a pure Web application could.

The downside of such apps is that they are strictly platform-specific. Apps developed for the iPhone run only on the iPhone, so several different implementations—composed with different platform-specific tools—are

needed to run the app on the Android, BlackBerry, Symbian, or other commonly used target platforms.<sup>4</sup> In many cases, a separate app is needed for each version of the target device. Such fragmentation is what effectively killed Sun’s (now Oracle’s) once highly successful Java ME platform.<sup>5</sup>

Unlike pure Web applications, a native app requires conventional installation. The user usually must download the application binary from a specific location, such as Apple’s App Store (<http://store.apple.com>). To introduce new features, the user typically must

download and install a new version or upgrade the application explicitly by device-specific means. This is clumsy and inconvenient for the user because the system might be unavailable while the download and upgrade are in progress.

### Open Web application manifesto

Mozilla’s recently published Mozilla Manifesto<sup>2</sup> declares that the Web is a global public resource that must remain open, accessible, interoperable, and secure. In line with these principles, the Web must be built around public, open standards rather than using proprietary technologies that serve specific business interests.

Among the four categories of application development technologies, the only one that fulfills the promise of truly open Web applications are browser-based systems—applications that require nothing more than a standard Web browser to run. Because they rely on proprietary technologies, additional plug-in components, or custom runtimes, the other approaches cannot deliver the Web’s true flexibility as an application platform.

While native apps can offer considerable gains in terms of performance, usability, and API coverage in the short term, in the long run, convergence will lead to the development of open Web applications that eschew the use of native software—apart from the Web browser or a compatible runtime environment—in favor of pure Web technologies. The rationale is based on the power of qualities such as instant worldwide deployment; no manual installation or upgrades; platform independence; and ubiquitous, seamless access to data, with the ability to flexibly combine data from different sources.

In the near term, there are still numerous challenges in moving toward truly open Web applications. For example, the evolution of the Web browser itself is progressing rather slowly, and many of the APIs needed for developing comprehensive applications simply are not yet available. Further, using the Web in mobile devices poses special challenges.

## The Web versus the mobile Web

The software industry is currently experiencing two parallel paradigm shifts—one toward Web-based software and the other toward Web-enabled mobile devices. The resulting convergence of desktop, mobile, and Web application development has already unleashed the development of entirely new types of Web-based software systems.

While the underlying needs to communicate and access information are the same in both the desktop and mobile environments, the ways people consume content and use applications with various terminals and devices in these environments are fundamentally different.

In the mobile space, the time span of users' actions is usually significantly shorter than in the desktop space. Users wish to perform rapid, targeted actions instead of participating in long-lasting sessions; actions must be simple yet focused, and they must be accomplished with ease, using minimal keystrokes or finger presses. Often, these actions are performed while the user is walking, driving a car, or otherwise distracted.

The different usage modalities and smaller screen sizes have a significant impact on application design; generic webpages geared toward laptop or desktop computer users are not usually ideal for mobile use. In addition, performance or network connectivity issues can make Web applications nearly unusable on mobile devices.


Given these challenges, it is not surprising that the mobile Web's evolution diverged from the general Web's evolution early on. The prime evidence of this early divergence is the now infamous Wireless Application Protocol (WAP), which was basically a parallel Web stack—including its own markup language, transport protocols, and browser solution—specifically designed for resource-constrained mobile devices. While the technical and commercial arguments behind WAP seemed sound, the assumption that people would want to use a custom browsing solution on mobile devices—separate from the general Web—turned out to be a huge mistake.

Somewhat surprisingly, companies that currently promote the use of proxy browsers are repeating the same mistake. Proxy browsers preprocess the information from the public Web and transliterate it into formats that fit better on the smaller screens of mobile devices. Although the technical arguments behind such proxy browsers are sound—decreased bandwidth costs, faster downloads, more optimal screen size usage, and so on—in reality, people usually prefer having access to the open, unfettered Web if such a choice is presented to them. However, many currently available low-to-mid-range mobile devices do not offer such a choice.

The WAP experience taught us that no matter how powerful the technical arguments in favor of custom Web runtime solutions specifically for mobile devices might

be, the allure of the open Web is so powerful that people generally do not want to be limited by such solutions.

In the early 2000s, the mobile Web developer community expressed significant interest in Web widgets—prepackaged, purpose-built Web applications that can be installed and executed in mobile devices. The difference between Web widgets and general-purpose Web applications is that the installation formats and practices for Web widgets have been predefined specifically to resemble those of traditional (native) applications. Again, although there are sound technical arguments in favor of such solutions, Web widgets actually represent the worst of two worlds: they combine the use of conventional, rigid, explicit installation practices (familiar from binary applications) with languages and tools that usually result in applications that run slower and are often harder to maintain than their native counterparts.



**While the technical and commercial arguments behind WAP seemed sound, the assumption that people would want to use a custom browsing solution on mobile devices turned out to be a huge mistake.**

## LIVELY WEB SYSTEMS FAMILY

During the past six years, we have constructed several systems for studying the use of the Web as a platform for real applications. Two of these systems—Lively Kernel ([www.lively-kernel.org](http://www.lively-kernel.org)) and Lively for Qt (<http://lively.cs.tut.fi/qt>)—have been made available publicly as open source projects, and they have been used for application development on a wide range of systems from desktops and laptops to mobile devices.<sup>5,6,7</sup>

Lively Kernel, originally developed at Sun Microsystems Laboratories, is a Web application environment that supports desktop-style applications with rich graphics and direct manipulation capabilities, but without the installation or upgrade hassles associated with conventional desktop applications. The system can also function as an integrated development environment, making it self-supporting so that it can improve and extend itself dynamically in the same fashion as Smalltalk programming environments—yet the system requires only a Web browser for its execution. In fact, the entire Lively Kernel system is just a webpage.

Architecturally, Lively Kernel is built around three principal components:

- *JavaScript engine.* We used the JavaScript engine available in the Web browser as a fundamental building block for the system. The system itself, as well as all





Figure 2. Lively Kernel (left) and Lively for Qt (right) running on a Nokia N900 device.

the applications and tools in it, were written entirely in JavaScript.

- *Expressive graphics API.* A fundamental requirement for a modern Web programming environment is the availability of a powerful graphics API that provides support for direct drawing, direct manipulation, and a rich set of predefined graphical widgets. For improved efficiency and development flexibility, the APIs should support a procedural—not just declarative—development style. At the implementation level, the Lively Kernel graphics API is mapped either on the scalable vector graphics (SVG) API or the canvas API available in the Web browser.
- *Asynchronous HTTP networking.* All the networking operations in both systems are performed asynchronously using the Ajax XMLHttpRequest feature.<sup>8</sup> Using asynchronous networking is critical so that the system can perform all networking requests in the background without impairing its interactive response.

Although one of our original goals was to run Lively Kernel on mobile devices, our early attempts to accomplish this failed badly.<sup>6</sup> The primary reason for this failure was that the JavaScript engines and graphics capabilities of the mobile Web browsers available in the mid-2000s simply were not powerful enough for our needs. For these reasons, we started working on another system that would exploit the underlying resources in a more closely coupled fashion.

The resulting system—Lively for Qt—was the logical successor of Lively Kernel in the sense that it implemented a similar, fully interactive, malleable, and lively Web programming environment.<sup>7</sup> However, a key difference was that we built Lively for Qt explicitly to leverage Qt (<http://qt.nokia.com>), an existing, rich, mature, well-documented application framework. Qt supports a rich set of APIs, widgets, and tools that run on most commercial software platforms, including Mac OS X, Linux, and Windows. More

important to us, Nokia had recently acquired Qt, with the intent that Qt libraries would eventually become available on all mobile devices produced by Nokia. This meant that we could rely on Qt libraries being available on Nokia devices without needing to separately download any additional plug-in components or custom libraries.

A similar concept, although independently developed, was later used in Palm's (now HP's) WebOS,<sup>9</sup> which introduced its own set of widgets that have been “baked into” a Web runtime, so that those widgets can be instantiated and used directly from Web applications.

Over the years, our research team members (<http://lively.cs.tut.fi>) have built numerous applications—games, desktop-style applications, applications that combine local data with Web content, as well as mashups that combine data from numerous websites into a single integrated experience—to demonstrate these systems' capabilities. When using Lively Kernel, the applications appear as individual windows inside the browser, while Lively for Qt applications are usually presented as desktop items or as full-fledged stand-alone applications, comparable to native applications. Figure 2 shows screenshots of some Lively Kernel and Lively for Qt applications running on the Nokia N900 mobile device.

In the mobile space, our research efforts culminated in a prototype GSM phone that we built in collaboration with a start-up company using JavaScript as the primary programming language for all the applications. This meant that any of the phone's applications could be downloaded dynamically over the air, run without explicit installation, and be updated transparently and seamlessly. However, in practice most of the system applications were still stored locally (cached in advance) for faster access.

## REMAINING CHALLENGES

The 2010s will be an interesting decade in the software industry. We anticipate a major battle between proprietary native application technologies and the open Web, which will determine the software industry's future for years to

come. Although there will be considerable disagreement about the outcome of this “battle of the decade” in the near term, we would hesitate to bet against the Web in the long run because the distribution model for pure Web apps is so much more powerful. No installation is required, the application suites of computers and mobile devices can be entirely dynamic, and applications and upgrades can be made available worldwide instantly. Further, given the significant impact that the Web has already had on so many industries, it seems obvious that it will eventually revolutionize the distribution of software as well. In the larger scheme of things, the current hoopla around app stores and native apps is likely only a temporary solution.

Some important technologies and initiatives will improve the use of the Web as an application platform.

## HTML5

The forthcoming HTML5 standard ([www.w3.org/TR/html5](http://www.w3.org/TR/html5)) adds numerous features to the existing HTML standards. Although HTML5 is a general-purpose Web standard, many of the new features are aimed squarely at making the Web a better place for desktop-style Web applications. The additions to the earlier versions of the HTML specification include support for automatic application updates (via HTML5 cache manifests), offline applications, local storage, a 2D canvas graphics API, built-in audio and video support, drag-and-drop and context menus, and cross-document messaging. The next generation of HTML5-compliant Web browsers will also support several other important W3C standards, such as Web sockets, Web workers, and file system and geolocation APIs.


## WebGL

Developed by Mozilla, the Khronos Group, and a consortium of additional companies including Apple, Google, and Opera, WebGL ([www.khronos.org/webgl](http://www.khronos.org/webgl)) is a cross-platform Web standard for a hardware-accelerated 3D graphics API. The main feature that WebGL brings to the Web is the ability to display 3D graphics natively in the Web browser without any plug-in components. Based on OpenGL ES 2.0, WebGL uses GLSL, the OpenGL shading language. WebGL runs in the HTML5 canvas element, and WebGL data is generally accessible through the Web browser’s document object model interfaces.

In our opinion, WebGL is an exciting, but still mostly overlooked, standard that will dramatically change the perception of the capabilities of the Web browser specifically, and Web applications more generally. WebGL’s most dramatic impact is that it will eliminate the “last safe bastion” of conventional binary applications. Although the majority of end-user software has already started migrating to the Web, so far it has been difficult to convince game developers to take Web-based software seriously. This is partly because suitable development APIs were not

available, and partly because until recently the execution speed of Web-based software was inadequate for CPU-hungry gaming applications. However, with the recent introduction of high-performance JavaScript engines such as Google’s V8 and Microsoft’s Chakra, the situation has changed dramatically.

In combination with HTML5 and other new Web standards, the Web browser will have support for high-performance 2D and 3D graphics, Web sockets, video streaming, audio, Cascading Style Sheets, scalable vector graphics, Web workers, file handling, geolocation, and many other features. When combined with high-performance JavaScript engines, the availability of such capabilities will reset the expectations about what pure Web applications can and cannot do, allowing new types of applications that run in a standard Web browser. Google’s Body Browser (<http://bodybrowser.googlelabs.com>) serves as an example of the new possibilities.



**The main feature that WebGL brings to the Web is the ability to display 3D graphics natively in the Web browser without any plug-in components.**

## Standardization efforts

While HTML5 and related World Wide Web Consortium (W3C) standard activities will play a critical role in turning the Web into a compelling application platform, the feature set that an HTML5-compliant Web browser offers is still somewhat incomplete for real-world applications. For instance, the platform/device APIs that are under definition will still offer only limited access to features that are available in personal computers and mobile devices today.

Further, given that it defines the necessary standards in multiple, separate activities, the current standardization work likely will not yet result in a consistent platform. For example, Microsoft currently has no plans to support WebGL in its IE9 browser. In general, since many vendors have a vested interest in ensuring that their native software development platforms remain competitive, Web platform standards probably will remain one or two steps behind native application platforms.

We predict that another major round of standardization efforts will be necessary within the next five years or so to establish a more complete Web application platform beyond HTML5. A critical goal in that standardization activity will be to more comprehensively “virtualize” the underlying operating system and device capabilities, as well as ensure that the necessary security mechanisms are in place to access all the platform capabilities securely.

## Open Web versus one Web?

The open Web concept does not necessarily imply one Web. As envisioned in the W3C Mobile Web Best Practices document,<sup>10</sup> one Web means that the same information and services should be available to users irrespective of the device they are using. While the overall goal is highly desirable, it is important to take into account the different input mechanisms, usage modalities, and screen size limitations on different types of terminals and mobile devices. Consequently, Web standards should not aim at establishing only “lowest common denominator” or “most common denominator” platforms. Instead, they should offer rich capabilities and support a broad variety of input mechanisms so that application developers and service providers can create compelling applications and services for all types of systems and devices. Plenty of interesting challenges still remain in this area.

**T**he World Wide Web's massive popularity is turning the Web browser from a document viewing tool into a general-purpose host platform for various types of services, including desktop-style Web applications. HTML5-style Web applications require no installation or manual upgrades, and they can be deployed instantly worldwide without middlemen or distributors. Conventional binary applications are at a significant disadvantage compared to Web-based software that can be deployed instantly across the planet.

So far, several obstacles have hindered the development and deployment of full-fledged, truly interactive Web applications. These obstacles have been especially apparent in the mobile device space. Emerging standards such as HTML5 and WebGL will eliminate many of the limitations

in this area. Although numerous challenges remain, we believe that the transition toward Web-based applications will eventually lead to the end of the binary end-user software era. In the future, the use of conventional binary programs will be limited to system software, while researchers will use Web technologies to develop the vast majority of end-user software—even for mobile devices. **□**

## References

1. C. Anderson and M. Wolff, “The Web Is Dead. Long Live the Internet,” *Wired*, Sept. 2010, pp. 118-127; 164-166.
2. Mozilla, “The Mozilla Manifesto”; [www.mozilla.org/about/manifesto](http://www.mozilla.org/about/manifesto).
3. A. Taivalsaari et al., *Web Browser as an Application Platform: The Lively Kernel Experience*, tech. report TR-2008-175, Sun Microsystems Laboratories, 2008.
4. T. Wasserman, “Software Engineering Issues for Mobile Application Development”; [http://works.bepress.com/tony\\_wasserman/4](http://works.bepress.com/tony_wasserman/4).
5. R. Riggs et al., *Wireless Devices with the Java 2 Platform*, Micro Edition, 2nd ed., Addison-Wesley, 2003.
6. T. Mikkonen and A. Taivalsaari, “Creating a Mobile Web Application Platform: The Lively Kernel Experiences,” *Proc. 24th ACM Symp. Applied Computing (SAC 09)*, ACM Press, 2009, pp. 177-184.
7. T. Mikkonen, A. Taivalsaari, and M. Terho, “Lively for Qt: A Platform for Mobile Web Applications,” *Proc. 6th Int'l Conf. Mobility Technology, Applications, and Systems (Mobility 09)*, ACM Press, 2009, article no. 24.
8. D. Crane, E. Pascarello, and D. James, *Ajax in Action*, Manning Publications, 2005.
9. M. Allen, *Palm WebOS*, O'Reilly Media, 2009.
10. World Wide Web Consortium, *Mobile Web Best Practices 1.0, Basic Guidelines*, W3C recommendation, 29 July 2008; [www.w3.org/TR/mobile-bp](http://www.w3.org/TR/mobile-bp).

**Tommi Mikkonen**, a professor of computer science at Tampere University of Technology, Finland, pioneered mobile software development in Finland and has arranged numerous courses on software engineering and mobile computing. His research interests include cloud computing, Web programming, embedded systems, and mashup development. Contact him at [tommi.mikkonen@tut.fi](mailto:tommi.mikkonen@tut.fi).

**Antero Taivalsaari**, a Distinguished Engineer at Nokia Research Center in Helsinki, Finland, played a seminal role in the design of the Java Platform, Micro Edition (Java ME). His research focuses on Web application technologies and Web-based software development, especially for mobile devices. Contact him at [antero.taivalsaari@nokia.com](mailto:antero.taivalsaari@nokia.com).

Together, Mikkonen and Taivalsaari lead the Lively Web Programming Research Team at Tampere University of Technology. For further information and a full list of research team publications, see <http://lively.cs.tut.fi>.

## COMPUTING THEN

Learn about computing history  
and the people who shaped it.

[http://computingnow.  
computer.org/ct](http://computingnow.computer.org/ct)



Selected CS articles and columns are available  
for free at <http://ComputingNow.computer.org>.