DOI:10.1145/1461928.1461945

2^W is a result of the exponentially growing Web building on itself to move from a Web of content to a Web of applications.

BY T.V. RAMAN

Toward 2^W, Beyond Web 2.0

FROM ITS INCEPTION as a global hypertext system, the Web has evolved into a universal platform for deploying loosely coupled distributed applications. As we move toward the next-generation Web platform, the bulk of user data and applications will reside in the network cloud. Ubiquitous access results from interaction delivered as Web pages augmented by JavaScript to create highly reactive user interfaces. This point in the evolution of the Web is often called Web 2.0. In predicting what comes after Web 2.0—what I call 2^W, a Web that encompasses all Web-addressable information—I go back to the architectural foundations of the Web, analyze the move to Web 2.0, and look forward to what might follow.

For most users of the Internet, the Web is epitomized by the browser, the program they use to log on to the Web. However, in its essence, the Web, which is both a lot more and a lot less than the browser, is built on three components:

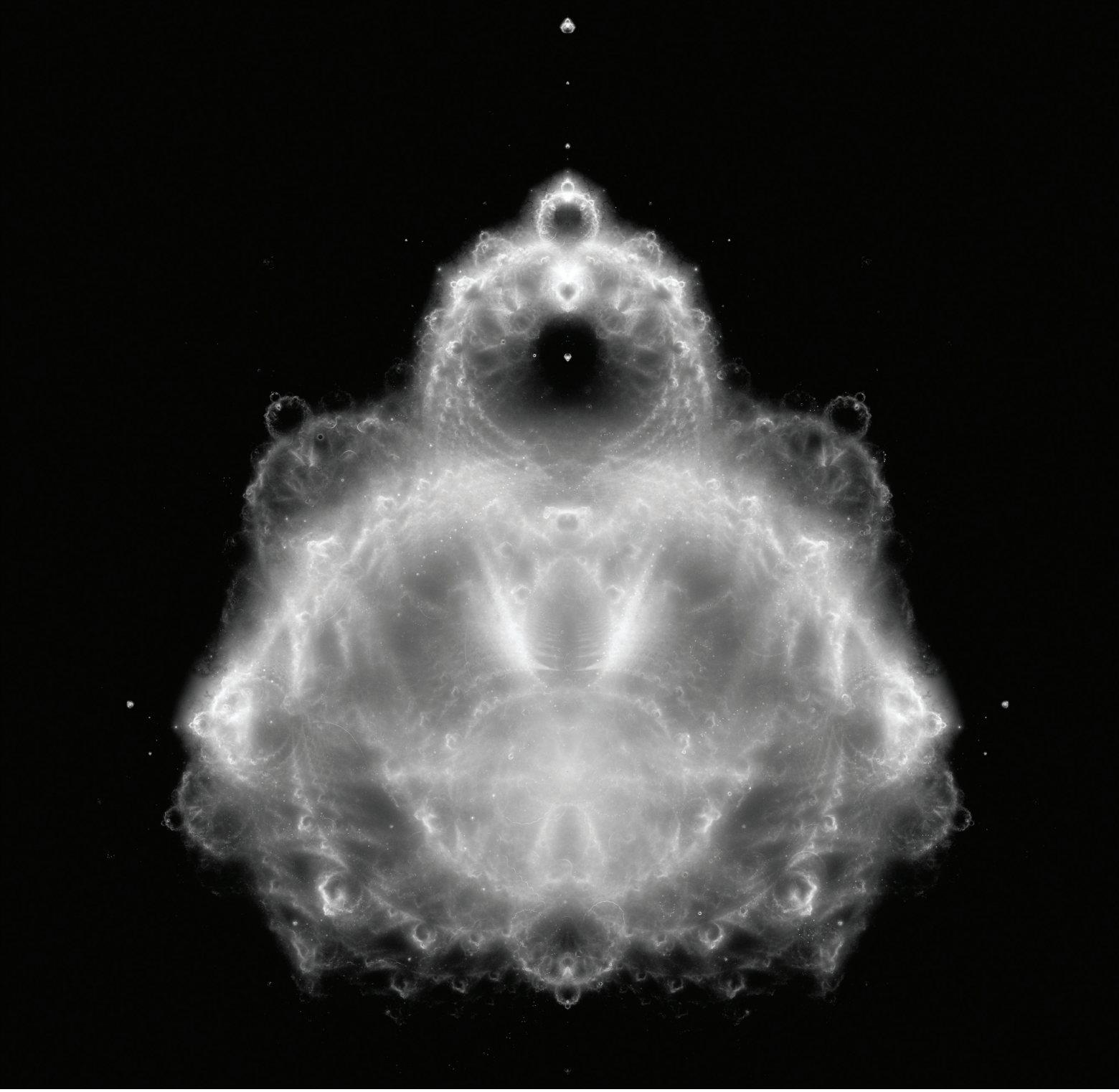
URL. A universal means for identifying and addressing content[6,7];

HTTP. A protocol for client-server communication[5]; and

HTML. A simple markup language for communicating hypertext content.[8]

Together, they constitute the global hypertext system. This decentralized architecture[35] was designed from the outset to create an environment where content producers and consumers come together without everyone having to use the same server and client. To participate in the Web revolution, one needed only to subscribe to the basic architecture of Web content delivered via HTTP and addressable via URLs. This yielded the now well-understood network effect that continues to produce exponential growth in the amount of available Web content. In the 1990s, the browser, a universal lens for viewing the Web, came to occupy center stage as the Web's primary interface. Deploying content to users on multiple platforms was suddenly a lot simpler; all one needed to enable universal access was to publish content to the Web. Note that this access was a direct consequence (by design) of the underlying Web contract, whereby Web publishers are isolated from the details of the client software used by their consumers. As Web browsers began to compete on features, this began to change in what became known as the browser wars, 1995–1999[36]; browser vendors competed by introducing custom tags into their particular flavors of HTML. This was perhaps the first of the many battles that would follow and is remembered today by most Web developers as the `blink` and `marquee` tag era marked by visual excess.

In 1997, HTML 3.2 attempted to ease the life of Web developers by documenting the existing authoring practice of the time. HTML 3.2 was in turn followed by HTML4[28] as a baseline markup language for the Web. At the same time, Cascading Style Sheets (CSS)[9] were introduced as a means of separating presentational information (style rules) from Web-page con-

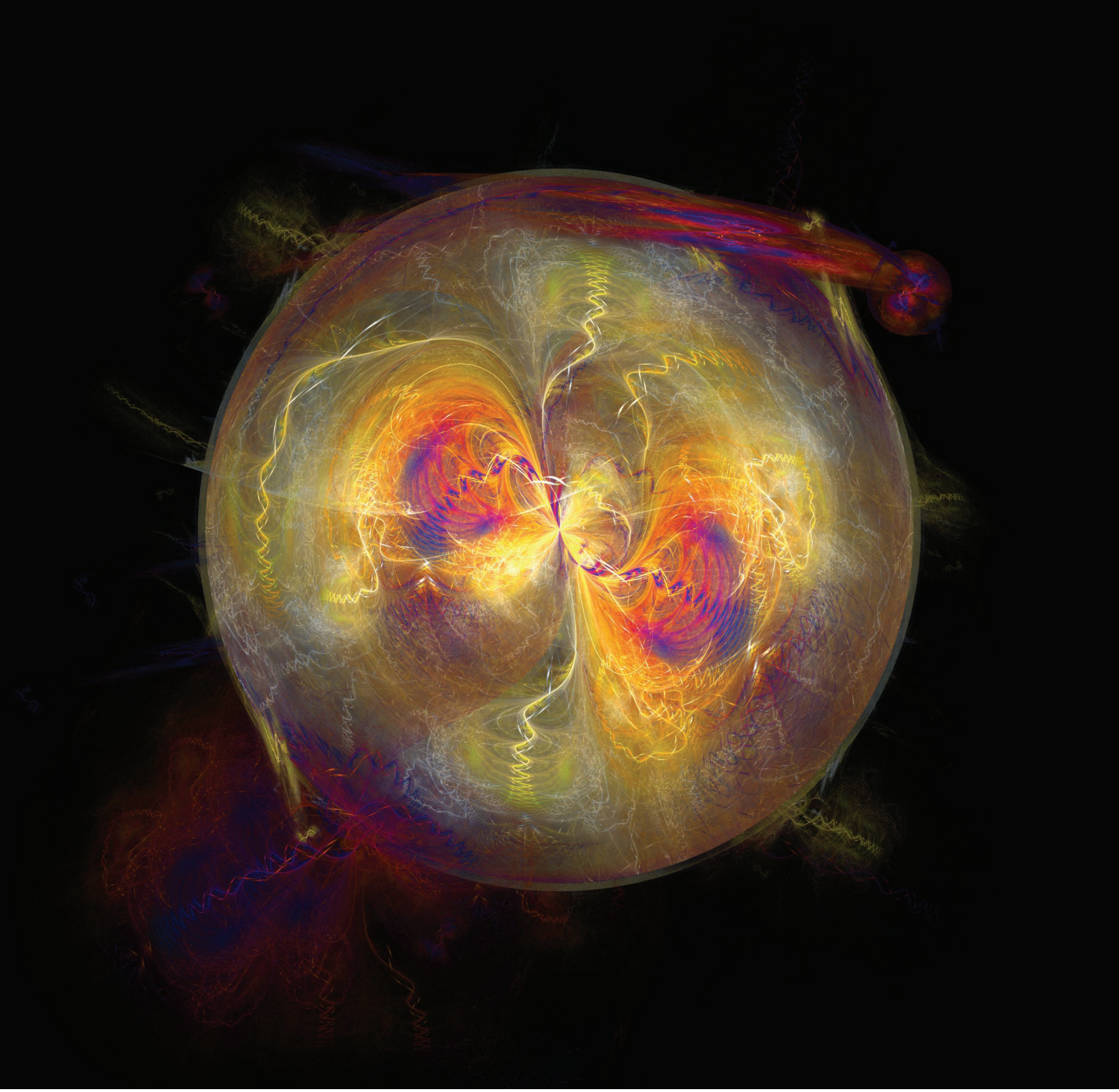


The self-similar repeating nature of fractals is a metaphor for the growth of the entire Web. This image by Jared Tarbell is a revisualization of the familiar Mandelbrot set; www.complexification.net/gallery/machines/buddabrot/.

IMAGE BY JARED TARBELL

tent. CSS enabled Web developers to flexibly style their content and was in part responsible for reducing their urge to invent new HTML tags purely to achieve a particular visual effect. But by 1998–1999, the browser wars were all but done, with Web developers coding mostly to the then-dominant browser, Microsoft's Internet Explorer 5. New features were no longer necessarily exposed via new tags in the HTML vocabulary; with CSS, a developer could easily create new presentational structures using the generic `div` and `span` tags. The behavior of constructs appearing in Web pages could be customized via JavaScript[18] and the HTML Document Object Model (DOM).[2,23] Thus, as the browser wars came to a close with the Web appearing to settle on HTML4, the Web community was already inventing a new highly interactive Web.

On the negative side, the dominance of a single browser during this period meant that all new behavior outside the scope of the HTML4 specification was implemented based on Internet Explorer; worse, that implementation in turn was a result of reverse engineering various features from the previously popular Netscape browser. This was particularly true with respect to interactive elements created through JavaScript and the HTML DOM, while incompatibilities between the CSS specifications and the predominant



Dreams 243.06260 and 243.06540 (page 58) were created by software artist Scott Draves through an evolutionary algorithm running on a worldwide cyborg mind consisting of 60,000 computers and people; ScottDraves.com.

implementation within Internet Explorer made it virtually impossible for Web developers to create content that would play consistently across multiple browsers.

Note that this period also saw significant movement away from Tim Berners-Lee's original vision of the Web. Web authors had started down the slippery slope of authoring for the dominant browser, thereby losing sight of the Web contract that had carefully arranged for Web content to be independent of the software that consumed it.

This breach might have seemed insignificant at the time, at least with respect to deploying Web content. The network effect that led to exponential growth in Web content during the 1990s meant that the Web had already taken off and that the slowdown in the network effect resulting from content coming to depend on a particular class of Web browsers did not immediately hamper growth. But the increasing interdependency between creator and consumer was not without cost; despite high hopes, the first round of the mobile Web fizzled in early 2000 partly because it was impossible to support mainstream Web content authored for a desktop browser on small devices like cellphones and PDAs. The problems that resulted from Web authors coding to a particular browser involved

IMAGE BY SCOTT DRAVES

additional hidden costs that became obvious by 2002 with the move from Web content to Web applications. By then, HTML, which began as a simple markup language, had evolved into three distinct layers:

HTML4. The markup tags and attributes used to serialize HTML documents;

CSS. The style rules used to define the presentation of a document; and

DOM. The programmatic interface to the parsed HTML document, used to manipulate HTML from within JavaScript.

The HTML4 specification went only so far as to define the tags and attributes used to serialize HTML documents. The programmatic API—the DOM—was defined within a separate specification (DOM 2) and never fully implemented by Internet Explorer. Making matters worse, CSS 2 was still under construction, and only parts of CSS 1 had been implemented in Internet Explorer.

Authoring Web content was now fraught with risks that would become apparent only over time. Authors could, with some trouble, create Web pages that appeared the same on the browsers of the time, at least with respect to visual presentation. However, when it came to styling the layout of a document via CSS or attaching interactive behavior via DOM calls, the shape of the underlying parsed representation proved far more significant than just visual appearance on a screen. As Web developers increasingly sought to add visual style and interactivity to their Web pages, they discovered incompatibilities:

Visual layout. To achieve a consistent visual layout, Web authors often had to resort to complex sets of HTML tables; and

Inconsistent DOM. Programmatic access of the HTML DOM immediately exposed the inconsistencies in the underlying representation in browsers, meaning that such programmatic calls had to be written for each browser and moved the Web further down the slippery slope toward browser-specific content.

But even as the hard-won Web looked like it would be lost to browser-specific Web content, the Web community was building on its earlier success of having a widely deployed universal browser that could be controlled (if poorly) via HTML and JavaScript. The Web had moved from mostly static content to documents with embedded pieces of interactivity. Web developers soon came to exploit the well-known fact of client-server computing: that even in a world of powerful servers, there are more compute cycles per user on a client than there are compute cycles on a server. Islands of interactivity implemented via JavaScript within HTML evolved into highly interactive user interfaces. The introduction of XML HTTP Request (XHR)[34] across the various browsers freed Web developers from having to do a complete page refresh when updating the user interface with new content. This set the stage for Asynchronous JavaScript and XML (AJax) applications.[19]

Discovering Web Applications

From late 1999 to early 2004, the line between content and applications on the Web was increasingly blurred. As examples, consider the following static (document-oriented) content and interactive (dynamic) applications:

Online news. News stories delivered in the form of articles enhanced with embedded video clips and interactive opinion polls; and

Shopping. Shopping catalogs with browsable items with interfaces, as well as real-time auction sites, enabling users to buy and sell.

This evolution from Web content to Web applications was accompanied by the progressive discovery of the Web programming model consisting of four Web components:

HTML. Markup elements and attributes for serializing Web pages;

CSS. Style rules for determining the final visual presentation;

DOM. Programmatic access to the parsed representation of the Web page; and

JavaScript. Open-ended scripting of the Web page through the DOM.

Here, the HTML, DOM, and JavaScript formed the underlying assembly language of Web applications. Though there is a clean architectural separation among content, visual-presentation, and interaction layers, note that this programming model was discovered through Darwinian evolution, not by design. A key consequence of this phenomenon is that content on the Web does not necessarily adhere to the stated separation. As a case in point, one still sees the HTML content layer sprinkled with presentational font tags, even though one would expect CSS to exclusively control the final presentation. Similarly, the content layer (HTML) is often sprinkled with script fragments embedded within the content, either in the form of inline script elements or as the value of HTML attributes (such as `href` and `onClick`).

Another key aspect of this phase of Web evolution was the creation of Web artifacts from Web components. Think of them as online information components built from Web-centric technologies—HTML, CSS, JavaScript—accessed over the network via URLs (see the figure here). A user-configurable component includes a customizable greeting, along with a photograph. Note that all of its aspects are constructed from five basic Web technologies:

Metadata. Component metadata encapsulated via XML;

Presentation. Content to be presented to the user encoded as plain HTML;

Style. The visual presentation of the HTML, controlled via CSS;

Interaction. Runtime behavior specified by attaching a JavaScript event handler (script) that computes the appropriate greeting based on the user's preferences and updates the HTML with the appropriate content; and

URLs. All resources used by the component—the photograph, the CSS style rules, the set of script functions—are fetched via URLs.

Toward Web 2.0

The first phase of the Web—Web 1.0—concluding in 2000 was characterized by bringing useful content online through the application of Web technologies to information (such as weather forecasts) in order to make them available on the Web to millions of potential users worldwide. A consequence was that a vast amount of useful content was now available—addressable via URLs and accessible over HTTP—with the requisite content being delivered via HTML, CSS, and JavaScript.

The next phase of this evolution—Web applications—saw the creation of useful end-user artifacts out of content already available on the Web. As an ex-

Web gadget built entirely from Web components—HTML, CSS, and JavaScript—displays a greeting and photograph both customizable by the user; the photograph is accessed via a URL.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Module>
 <ModulePrefs title="..."/>
 <UserPref name="myname"/>
 <UserPref name="myphoto"/>
 <Content type="html"><![CDATA[
 <div id="content_div"></div>
 <style type="text/css">...</style>
 <script type="text/javascript">
 // Get userprefs
 var prefs = new gadgets.Prefs();
 function greet () {
    // Get current time
    var today = new Date();
    var time = today.getTime();
    var html = "";
    // Display appropriate greeting
    html += ...
    // Display photo if asked to
    if (prefs.getBool("photo") == true) {
    html += ...
    }
    element.innerHTML = html;
    }
    ...
    gadgets.util.registerOnLoadHandler(greet);
 </script>
 ]]>
</Content>
</Module>
```

RESTful Web APIs from major Web applications laid the software foundations for Web 2.0.

Service	Resource
Amazon	XForms Book
Google	Search Hubbell+Labrador
eBay	ASTER
Yahoo!	Browse Autos

ample, weather forecasts were available on Web 1.0, but XML HTTP Request and the ability to asynchronously refresh the content displayed in a Web page through JavaScript callbacks enabled Web sites to integrate weather forecasts into the context of user tasks (such as travel reservations). In addition to being built from Web technologies, a travel site that integrates a variety of information sources in order to deliver a complete task-oriented interface uses the same Web technologies when constructing its constituent components. Note that Web 2.0 is a result of applying Web technologies to the Web. Described differently, Web 2.0 is a consequence of Web(Web()); or writing $\mathcal{W}$ for the function Web, a more apt notation for Web 2.0 would be W^2.

Web As Platform

The notion of the Web as a new platform emerged in the late 1990s with the advent of sites providing a range of end-user services exclusively on the Web. Note that none of them had a parallel in the world of shrink-wrap software that had preceded the Web:

Portal. Yahoo! Web directory;
Shopping. Amazon online store;
Auction. eBay auction site; and
Search. Google search engine.

In addition to lacking a pre-Web equivalent, each of these services lived on the Web and, more important, exposed the services as simple URLs, an idea later known as REpresentational State Transfer, or (REST) ful, Web APIs.[16,17] All such services not only built themselves on the Web, they became an integral part of the Web in the sense that every Google search, auction item on eBay, and item for sale on Amazon were URL addressable (see the table here).

URL addressability is an essential feature of being on the Web. The URL addressability of the new services laid the foundation for Web 2.0, that is, the ability to build the next generation of solutions entirely from Web components. The mechanism of passing-in parameters via the URL defined lightweight Web APIs. Note that in contrast to all earlier software APIs, Web APIs defined in this manner led to loosely coupled systems. Web APIs like those in the table evolved informally and came to be recognized later as programming interfaces that could be used to build highly flexible distributed Web components.

That all of these services heralded publication of a new platform was reflected in the O'Reilly Hacks Series, including: *Google Hacks*[10]; *Amazon Hacks*[4]; *Yahoo! Hacks*[3]; and *eBay Hacks*.[22]

The Web had thus evolved from a Web of content to a Web of content embedded with the needed user-interaction elements. Content embedded with user interaction evolved into Web applications that could over time be composed exclusively from Web components. Being built this way and living exclusively on the Web, the new software artifacts came to form the building blocks for the next generation of the Web. Together, they define the Web as a platform with certain key characteristics:

Distributed. Web applications were distributed across the network; application logic and data resides on the network, with presentation augmented by the needed user interaction delivered to the browser;

Separable. The distributed nature of Web applications forced a cleaner separation between application logic and the user interface than in the previous generation of monolithic desktop software;

Universal. By delivering presentation and user interface to a Web browser, Web applications were more universally available than were their earlier counterparts; coding to the Web platform—or using HTML, JavaScript, and CSS[21]—enabled developers to create user interfaces that could be consistently accessed from a variety of platforms;

Zero install. With user-interface enhancements delivered via the network, users did not need to install Web applications; and

Web APIs. Web applications exposed simple URL-based APIs that evolved bottom-up that were easy to develop, document, and learn and quickly became a key enabler for Web 2.0.

Aggregations, projections, and mashups are all a direct consequence of the user's need to consume information in a form that is most suited to a given task.

User-Centric Access

As increasing amounts of information was moved onto the Web in the late 1990s, end users had a problem: To access all the information required for a given task, they needed to connect to myriad Web sites. This was true on the public Internet, as well as on corporate intranets. Moreover, the information being accessed had to be customized for the user's context (such as desktop or mobile access). The desire to deliver user-centric information access led to the binding of mobile user interfaces to Web content, another example of specialized browsing.[29,33]

A user interface designed for a large display is inappropriate for viewing on small-screen devices like cellphones and PDAs. The distributed nature of Web applications—and consequent separation of the user interface—enabled Web developers to bind specialized mobile user interfaces.

At the same time, the need to provide a single point of access to oft-used information led to portal sites that aggregated all the information onto a single Web page. In this context, the various items of information can be viewed as lightweight Web components. The environment in which these components are hosted (such as the software that generates and manages the Web page) can be viewed as a Web container. Thus, common actions (such as signing in) were refactored to be shared among the various Web applications hosted by the Web container, a piece of software managing the user's browsing context.

Web components hosted in this manner relied on the underlying Web APIs discussed earlier to retrieve and display content on behalf of the user. But as long as such aggregations were served from portal sites, users still needed to explicitly launch a Web browser in order to access their information. This turned out to be an inconvenience for frequently viewed information, motivating the move by Web developers toward Web gadgets, small pieces of Web-driven software that materialize on the user's desktop outside the Web browser. Such Web aggregation has moved over time from the server to the client where it materializes as widgets or gadgets.

Viewed this way, Web gadgets are specialized browsers. Rather than requiring the user to explicitly navigate to a Web site and drill through its various user-interface layers before arriving at the target screen, these gadgets automate away a large part of the user actions by directly embedding the final result into the user's Web environment. Finally, Web gadgets have escaped the confines of the Web browser to materialize directly on the user's desktop. Users no longer had to explicitly launch a Web browser to access the gadgets. However, the gadgets themselves continue to be built out of Web components. As an example, a typical iGoogle gadget consists of several components:

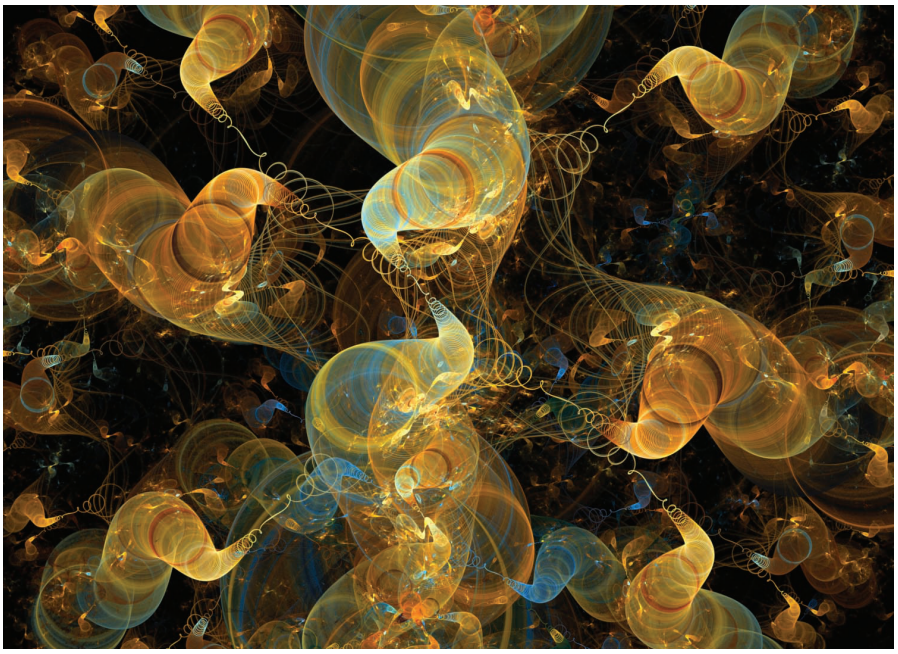
XML. A small XML file encapsulating metadata about the gadget;

HTML. The markup used to render the gadget;

CSS. Style rules to specify the final visual presentation; and

JavaScript. JavaScript functions used to retrieve and inject the relevant information into the HTML DOM before it is presented to the user.

Web gadgets relying on lightweight Web APIs, Rich Site Summaries (RSS) (letters also sometimes used to mean Really Simple Syndication), and Atom feeds[26] helped the move toward specialized browsing; retrieving information from a Web site did not always require a live human to directly interact with the user interface. Today, RSS and Atom feeds form the underpinnings of Web APIs for content retrieval. In the simplest cases, they enable



content sites to export a set of article titles and summaries. In more complex cases, such feeds are used in conjunction with newer protocols (such as the Atom Publishing Protocol[13]) layered on top of HTTP to expose rich programmatic access to Web applications. Together, these various feed-oriented APIs enable a variety of task-oriented Web tools ranging from bulk upload of data to custom information access. Note that this class of software services consists entirely of Web components.

Web gadgets thus provide specialized browsing functionality and are hosted in a variety of environments ranging from server-side containers to client-side user environments. In all cases, the hosting environment provides a number of services:

Back end. Access the Web to retrieve, filter, and format the requisite information;

Front end. Render the formatted information as HTML for realizing the final presentation and user interface;

Configuration. Provide the user interface affordances to allow users to customize the final experience by configuring the look and feel of the interface; such configuration includes adding, removing, expanding, or collapsing the gadget;

Preferences. Manage user preferences for gadgets within a container;

Single sign-on. Delegate common tasks (such as authentication) to the container, so users do not need to login to each Web application; and

Caching. Cache content to provide an optimized browsing experience.

The Web container thus provides the environment or evaluation context for Web widgets. I return to this pivotal role played by such container environments later when I address the evolving social Web.

A key aspect of all Web technologies is that the user has final control over visual presentation and user interaction. CSS emphasizes the C in Cascading by enabling users to cascade and consequently override the visual presentation chosen by the content creator. Similarly, scripting makes it possible for end users to drastically alter the interaction behavior of Web content. This flexibility was first leveraged in 1999 by Emacspeak[31]; the modules *websearch* and *url-templates* provided task-oriented Web wizards using REST APIs and XSLT Web transforms.[12] Later, similar functionality was brought to mainstream users by Greasemonkey,[27] a Firefox extension enabling them to attach arbitrary scripts to Web content. The success of Greasemonkey has been built upon by projects like Chickenfoot from MIT[25] and CoScriptor from IBM Research,[24] both providing higher-level user automation when working with Web interfaces. The ability to inject behavior into Web pages by scripting the HTML DOM was also successfully leveraged to create Google-AxsJAX,[11,30] a JavaScript library that helps developers across the Web enhance the usability of Web interfaces, with special focus on users with special needs (such as visual and hearing impairment).

Beyond Web 2.0

So here is where we stand:

- The Web, which began as a global hypertext system, has evolved into a distributed application platform delivering final-form visual presentation and user interaction;
- The separation between application logic and user interface enables late binding of the user interface,[14,15,32] promising the ability to avoid a one-size-fits-all user interface;
- More than URL-addressable content, the Web is a distributed collection of URL-addressable content and applications;
- It is now possible to create Web artifacts built entirely from Web components; and
- The underlying Web architecture ensures that when created to be URL-addressable, Web artifacts in turn become the building blocks for the next set of end-user Web solutions.

I described Web 2.0 earlier as the result of applying the Web function to itself, that is, Web 2.0 = Web^2 (). Let $\mathcal{W}$ denote the set of all URL-addressable information. Examining the properties of today's Web, we see the following additional properties with respect to $\mathcal{W}$:

Aggregation. New Web artifacts can be created by aggregating existing elements of the Web; when assigned a URL, such aggregations become elements of $\mathcal{W}$;

Projections. Information available on the Web can be filtered to suit the user's browsing context; such projections when made URL-addressable themselves become elements of $\mathcal{W}$; and

Cross-products. Discrete elements of $\mathcal{W}$ can be integrated into a single view to create Web mashups.

Note, too, that the notion of Web mashups can be generalized to cover cases where one brings together data from more than a pair of sites. Such cross-products are not limited to integrating data from multiple sources into a single view; instead, one can also integrate multiple views of the same piece of data (such as a visual representation that displays historical data both as a

IMAGE BY SCOTT DRAVES

table of numbers and as a histogram). Similarly, a multimodal view of a page, supporting both visual and spoken interaction, is also just one more type of view-to-view mashup. Bringing all this together, we can pose the question: What is the size of this Web to come? In theory, we can combine arbitrary subsets of W using the techniques I've outlined here. Each combination can in turn be deployed on the Web by making it URL-addressable and expressed mathematically as:

$$\binom{|W|}{0}+\binom{|W|}{1}+\binom{|W|}{2}+\ldots+\binom{|W|}{|W|}=2^{|W|}$$

User-Oriented Web: A Total Perspective

This number $2^{|W|}$ is extremely large and growing quickly as we build on the success of the Web; here, I denote this set 2^W. Fans of Douglas Adams's *Hitchhikers Guide To The Galaxy*[1] probably feel like they are now well entrapped within the total perspective vortex. But just as in the case of Zaphod Beeblebrox, the solution is not to focus on the totality of the Web but instead on the individual; 2^W exists for the user. As we move to a highly personalized social Web, each element of 2^W exists as it is perceived and used by a given user.

A significant portion of our social interaction increasingly happens via the Web. Note that a large portion of the impetus for the move from Web 1.0 to Web 2.0 and later to the predicted 2^W is due to user needs; aggregations, projections, and mashups are all a direct consequence of the user's need to consume information in a form that is most suited to a given task. Though the resulting set 2^W might be immense, most of these elements are relevant only when used by at least one user. Users do not use Web artifacts in a vacuum, but in a given environment or evaluation context provided by a given Web container.

Web content when combined is far more useful than its individual components. Likewise, Web applications used by collaborating users create a far richer experience than would be possible if they were used by users in isolation. Users typically converge on the use of such artifacts via popular Web containers, making the various APIs available by a given container a key distinguishing factor with respect to the types of interactions enabled within the environment. For example, OpenSocial from Google (code.google.com/apis/opensocial/), which describes itself as "many sites, one API," defines a set of APIs that can be implemented within a Web container. These APIs then expose a common set of services to gadgets being hosted within the container. Likewise, the Facebook platform provides an API for developing gadgets to be hosted in the Facebook container,[20] which can provide access to a user's contact list, enabling the various gadgets within it to provide an integrated end-user experience.

Conclusion

The Web has evolved from global hypertext system to distributed platform for end-user interaction. Users access it from a variety of devices and rely on late binding of the user interface to produce a user experience that is best suited to a given usage context. With data moving from individual devices to the Web cloud, users today have ubiquitous access to their data. The separation of the user interface from the data being presented enables them to determine how they interact with the data. With data and interaction both becoming URL-addressable, the Web is now evolving toward enabling users to come together to collaborate in ad-hoc groups that can be created and dismantled with minimal overhead. Thus, a movement that started with the creation of three simple building blocks—URL, HTTP, HTML—has evolved into the one platform that binds them all. C

References

1. Adams, D. *The Restaurant at the End of the Universe.* Ballantine Books, 1980.
2. Apparao, V., Byrne, S., Champion, M., Isaacs, S., Jacobs, I., Hors, A.L., Nicol, G., Robie, J., Sutor, R., Wilson, C. et al. *Document Object Model (DOM) Level 1 Specification.* W3C Recommendation, World Wide Web Consortium, 1998; www.w3.org/TR/REC-DOM-Level-1.
3. Bausch, P. *Yahoo! Hacks.* O'Reilly Media, Inc., Sebastopol, CA, 2005.
4. Bausch, P. *Amazon Hacks: 100 Industrial-Strength Tips & Tools.* O'Reilly Media, Inc., Sebastopol, CA, 2003.
5. Berners-Lee, T., Fielding, R., and Frystyk, H. *Hypertext Transfer Protocol 1.0*, 1996; www.w3.org/Protocols/Specs.html#HTTP10.
6. Berners-Lee, T. *Universal Resource Identifiers in WWW.* Internet Engineering Task Force, 1994; www.w3.org/Addressing/rfc1630.txt.
7. Berners-Lee, T., Masinter, L., and McCahill, M. *Uniform Resource Locators.* Internet Engineering Task Force Working Draft 21, 1994; www.ietf.org/rfc/rfc1738.txt.
8. Berners-Lee, T. and Connolly, D. *Hypertext Markup Language, Internet Working Draft 13*, 1993.
9. Bos, B., Lie, H.W., Lilley, C., and Jacobs, I. *Cascading Style Sheets, Level 2 CSS2 Specification.* W3C Recommendation, May 1998; www.w3.org/TR/REC-CSS2.
10. Calishain, T. and Dornfest, R. *Google Hacks: 100 Industrial-Strength Tips & Tools.* O'Reilly Media, Inc., Sebastopol, CA, 2003.
11. Chen, C.L. and Raman, T.V. AxsJAX: A talking translation bot using Google IM: Bringing Web-2.0 applications to life. In *Proceedings of the 2008 International Cross-Disciplinary Workshop on Web Accessibility*, 2008, 54–56.
12. Clark, J. et al. *XSL Transformations Version 1.0.* W3C Recommendation. World Wide Web Consortium, 1999.
13. de hOra, B. *The Atom Publishing Protocol.* 2006; bitworking.org/projects/atom/.
14. Dubinko, M. *XForms Essentials.* O'Reilly Media, Inc., Sebastopol, CA, 2003.
15. Dubinko, M., Klotz, L.L., Merrick, R., and Raman, T.V. *XForms 1.0, W3C Recommendation.* World Wide Web Consortium, Oct. 14, 2003; www.w3.org/TR/xforms.
16. Fielding, R.T. and Taylor, R.N. Principled design of the modern Web architecture. *ACM Transactions on Internet Technology 2*, 2 (2002), 115–150.
17. Fielding, R.T. *Architectural Styles and the Design of Network-based Software Architectures, Chapter 5 Representational State Transfer.* Ph.D. Dissertation, University of California, Irvine, 2000.
18. Flanagan, D. and Novak, G.M. Java-Script: The definitive guide. *Computers in Physics 12*, 41 (1998).
19. Garrett, J.J. *Ajax: A New Approach to Web Applications.* White paper, Adaptive Path Inc., 2005.
20. Graham, W. *Facebook API Developers Guide.* firstPress, 2008.
21. Hickson, I. HTML 5 Working Draft. W3C Working Draft. World Wide Web Consortium, 2008; www.w3.org/TR/html5/.
22. Karp, D.A. *eBay Hacks.* O'Reilly Media, Inc., Sebastopol, CA, 2005.
23. Le Hors, A., Le Hegaret, P., Wood, L., Nicol, G., Robie, J., Champion, M., and Byrne, S. *Document Object Model Level 2 Core Specification.* W3C Recommendation. World Wide Web Consortium, 2000; www.w3.org/TR/DOM-Level-2-Core.
24. Leshed, G., Haber, E.M., Matthews, T., and Lau, T. CoScripter: Automating & sharing how-to knowledge in the enterprise. In *Proceedings of the 26th Annual SIGCHI Conference on Human Factors in Computing Systems* (2008).
25. MIT CSAIL. Automation and customization of rendered Web pages. In *Proceedings of the ACM Symposium on User Interface Software and Technology* (2005).
26. Nottingham, M. *Atom Syndication Format.* Internet RFC 4287 Internet Engineering Task Force, 2005.
27. Pilgrim, M. *Greasemonkey Hacks: Tips & Tools for Remixing the Web with Firefox.* O'Reilly Media, Inc. Sebastopol, CA, 2005.
28. Raggett, D., Le Hors, A., and Jacobs, I. *HTML 4.01 Specification.* W3C Recommendation REC-html401-19991224. World Wide Web Consortium, Dec. 1999; www.w3.org/TR/ html401.
29. Raman, T.V. Specialized browsers. Chapter 12 in *Web Accessibility*, S. Harper and Y. Yesilada, Eds. Springer, 2008; emacspeak.sf.net/raman/publications/.
30. Raman, T.V. Cloud computing and equal access for all. In *Proceedings of the 2008 International Cross-Disciplinary Workshop on Web Accessibility* (2008), 1–4.
31. Raman, T.V. Emacspeak: The complete audio desktop. In *Beautiful Code*. O'Reilly Media, Inc., Sebastopol, CA, 2007.
32. Raman, T.V. *Xforms: XML-Powered Web Forms.* Addison-Wesley Professional, 2003.
33. Raman, T.V. *Emacspeak: Toward the Speech-Enabled Semantic WWW.* 2000; emacspeak.sf.net/raman/.
34. van Kesteren, A. and Jackson, D. *The XML HTTP Request Object.* W3C Working Draft. World Wide Web Consortium, 2008; www.w3.org/TR/XMLHttpRequest/.
35. World Wide Web Consortium Technical Architecture Group. *Architecture of the World Wide Web.* W3C TAG Finding REC-webarch-20041215, 2004.
36. Yoffie, D.B. and Cusumano, M.A. Judo strategy: The competitive dynamics of Internet time. *Harvard Business Review 77*, 1 (1999), 70–81.

T.V. Raman (raman@google.com) is a research scientist at Google Research, Mountain View, CA.