HTML5

By Bob Frankston

I've been having fun building a home control application in HTML5. Figure 1 shows an image of part of the floor plan of my house with lightbulb icons showing the lights and other devices that I can control.

Understanding the past and future of HTML5 gives us an understanding of the world of bits, where one implements first and then discovers standards as experience coalesces into common practices. I was able to produce this application in a few days. Most of the time was spent learning about new facilities such as WebSockets and implementing the connection to my existing home control application.

This application can run in just about any modern browser from anywhere in the world. I can share it just by providing a URL. It is not a Web page as much as an application that uses the browser as a powerful virtual machine that can run on a wide range of devices, from those that fit in my pocket to large desktop computers.

History does not repeat itself as much as it has harmonics and echoes. The early personal computers gave us the freedom to explore new opportunities without the need to justify our experimentations to computer systems managers. Nor did we have to worry about a meter running.

HTML5 is very much a work in progress, and that is part of its appeal as the environment continues to evolve. The ideas in HTML5 evolved over the years as browsers competed. Various libraries, such as AJAX and JQuery, were written to give programmers the ability to take advantage of the new features despite the differences in the capabilities of each browser.

BROWSERS: THE FIRST HALF CENTURY

My first column ("Refactoring CE," *IEEE Consumer Electronics Magazine*, January 2013) described the accidental history of the Internet. In implementing the early packet radio networks, we discovered that we could solve communication problems without relying on carriers to preserve the meaning of our messages.

HTML5 arises from what I call presentation-centric computing. It is very different from the days when computers would read punched cards and then print out the results of their calculations.

We can go back to the early 1960s to Digital Equipment Corporation's first computer, the PDP-1 and MIT's pioneering timesharing system CTSS. Timesharing was the cloud computing of the day—a shared resource. JUSTIFY (http://www.dpbsmith.com/tj2.html), DITTO (http://manpages.bsd.lv/history/CC-205.pdf), and RUNOFF were some of the programs that would allow you to write a memo (or a book), insert markers for pages and paragraphs, and then have the document formatted to look good when printed. The ability to then edit the original file and print out a new copy was revolutionary in the days when any change would require retyping entire pages if not an entire document.

There is a direct path from those early programs to HTML. HTML built upon the ideas in the SGML format used for professional publishing to create a format for sharable documents. HTML files are standard files that can be accessed using any protocol, but the preferred protocol is HTTP, which looks very much like an e-mail header. The use of standard text

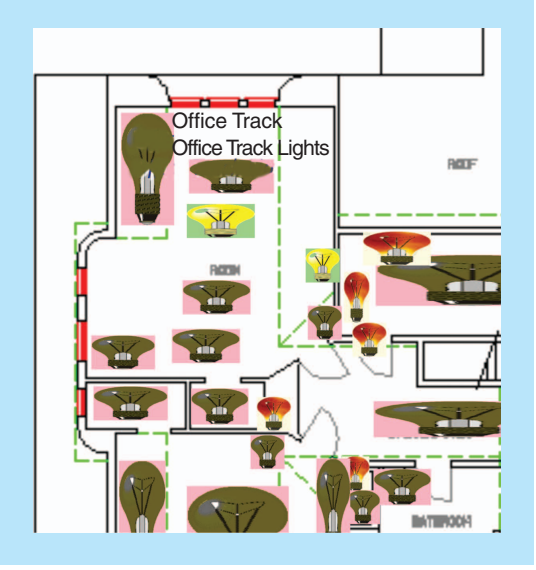

FIGURE 1. Part of the floor plan of my house. The lightbulb icons indicate the lights and other devices under my control.

Digital Object Identifier 10.1109/MCE.2013.2296836
Date of publication: 20 March 2014

files allowed people to implement and experiment without having to build special tools. Anyone could read an HTML file or an HTTP message, and that readability engendered a degree of trust.

This open format, along with browsers that ignored unrecognized tags, allowed for experimentation with new tags such as <table>. The addition of JavaScript and the ability to embed new objects enabled further evolution. The initial JavaScript took about ten days and has been evolving ever since.

The asynchronous loading of elements created an opportunity for the browser to expand its role from being a program that merely presented remote data to one that could have local smarts and fetch remote data as needed. I remember experimenting with those new capabilities in the 1990s. Steve Ward took this effort a step further with his CURL language in 1998, but the market has taken a slower route to local programming in HTML5.

The constraints of HTML and the browsers contributed to the success of the Web. First is the concern about making it safe to use browser-based applications. This means that the environment is not suitable for all applications, but limiting oneself to working within the browser removes a major barrier to the adoption of the application.

The asynchronous programming environment encourages a very responsive programming style while avoiding many of the complexities of using threads in native applications. Here too, there is a benefit in choosing to limit oneself to the browser environment.

FIGURE 2. An example of a QR code.

Programs can take advantage of the rendering heritage by manipulating objects and classes rather than having to do their own drawing. For example, one makes an object (or a whole class of objects) appear or disappear by simply setting the visibility property with a single line of code. The browser quickly updates the presentation to reflect that change.

The loose programming environment encourages experimentation, and users can tolerate most failures. Of course, a heavy-duty commercial site has high standards, but, for the most part, users can handle issues, especially if they are fixed by something as simple as a redisplay.

The JavaScript language is also continuing to evolve. TypeScript (which I am using) and CoffeeScript represent the future direction of JavaScript, and features from these preprocessors will become part of the standards over time.

Of course, an interpretative environment like JavaScript has to be slow. Or does it? It is amazing what one can accomplish when there is the incentive to improve. JavaScript performance has improved rapidly using such techniques as recognizing patterns at runtime, which is all the more reason to understand effective programming styles.

HTML5 TODAY

The process of implement first and standardize later has worked very well. Today's HTML5 is a plateau in a continuing evolution. Features that were formerly available in libraries are now natively available in current browsers. The development environments have also improved both within the browsers and externally.

Today's browser is a rich programming environment with strong text, video and audio presentation capabilities, and vector- and pixel-based graphics. The application can also use local storage on the platforms so it can function as a stand-alone application. WebSockets offers powerful communication capabilities. WebRTC brings voice communication capabilities to HTML5, basically making a telephone just another software component.

PhoneGap is one of the wrappers that can be used to make an HTML5 application run as a native application across many platforms. Adobe has a service that will take your application and adapt it to each platform for free (at least for modest use).

RESTful servers complement the browser by building on the HTTP stateless relationships. This makes it easy for the browser application to access both local and remote services. You can think of a RESTful interface as a Web form used by a program.

I can't begin to explain the full richness of today's HTML5 environment; all I can do is whet your appetite and encourage experimentation.

THE FUTURE?

It's not easy to separate the future from the present, as much of the future is here now in the form of ongoing experiments and capabilities. In my home control example, I had to create icons to represent the devices in my house and do the programming necessary so that clicking the icon would control the actual device and, conversely, reflect any changes in the state of the actual device in the icon.

This capability is already available in the form of widgets that can be embedded on Web pages, but the standards are not quite there. In fact, the idea of microformats in the form of XML objects is an old one, but such standards have had difficulty gaining traction on their own. But as standards and the common framework develops, they will come into their own.

As a strong reminder that this is not just about the Internet as such, we can transfer this information via NFC or QR codes (an example of which can be seen in Figure 2) or any other means. A URL is simply one way to provide a reference. It uses the DNS as its dictionary.

Just as personal computers pulled back control from the centralized computing cloud of the 1970s, the local computing capabilities of the HTML5 environment offer the possibility of increased local control of computing. The so-called cloud or shared capabilities will remain important, but the relationship has changed. Rather than just browsing the cloud, a browser is now a

(continued on page 67)

SOFTWARE TO RULE THEM ALL

Almost every enterprise hardware company is involved in open compute and many also in open storage approaches. There are a great number of hardware and, especially, software start-up companies that are enabling this rapidly growing storage market. Software to manage the movement and storage of content plays an increasing role, and it is changing the face of enterprise storage, particularly for consumer applications. The concepts of abstraction or virtualization are moving to a whole new level with what is coming to be called software-defined storage (SDS).

SDS is similar to the networking concept for simplifying data communication called software-defined networking. In SDS, the software plays an even greater role in the management of the content in the storage hardware. For instance, software RAID between distributed servers containing their own redundant storage and using persistent flash memory may lower the costs for higher performance as well as inexpensive mass storage.

Other concepts such as erasure codes are being used to create almost infinite levels of redundancy, providing protection from data loss even if multiple hard disk drives or other storage devices fail, and providing graceful performance degradation as problems occur. Also, flash memory is moving into use as persistent memory, which could replace some DRAM, except that the flash memory can retain the state of the machine even if it is turned off.

New solid-state storage technologies are in the wings, with performance closer to that of DRAM, which could take these ideas of persistent memory to new levels and increase overall system performance and reliability. Today, we are already seeing performance improvements as some applications, such as databases, use flash memory as primary storage rather than just a temporary cache.

THE SHAPE OF THINGS TO COME

Digital storage technology, whether in your smart mobile device, a local external storage device, or the cloud that runs your applications, will play a crucial role in future consumer electronics. The development of fast solid-state storage has encouraged the development of faster communications that are matched to the data rate capabilities of storage devices and systems that use flash memory for performance enhancement. As a result, interfaces like USB 3.2 and Thunderbolt are moving to 10 Gb/si and 20 Gb/si data rates, respectively.

Thunderbolt is based on the PCIe interface, and this high-speed communications bus is finding increasing use for enterprise and consumer applications. The next generation of flash devices in tablet computers will likely use a new MIPI M-Phy specification that provides compact PCIe interface storage products. Likewise, flash memory and, eventually, other solid-state storage will increase overall system performance and constitute an important tier in data center design for consumer applications.

As the total amount of consumer content increases, we will need more bulk storage, likely on object-based open storage platforms and perhaps managed by SDS software. These may use expected 6–8-TB shingled or even the new He-filled bulk storage magnetic recording hard disk drives, with perhaps some flash cache built in to provide some performance enhancement. Also, some tablet users may want the inexpensive storage capacity provided by hard disk drives, either in the tablet or available via Wi-Fi or personal cloud storage device or by conventional interfaces, such as USB.

The world of the future will be an interesting place and technology will help drive it. Digital storage is where our content is kept, whether in our consumer devices or the data centers that support them. Whether near or far, we need more and better digital storage.

CE

Bits Versus Electrons *(continued from page 63)*

peer that uses the cloud as a resource. Don't let the legacy use of browser confuse you.

In thinking about the future, it is useful to look at the accidental path that took us to HTML5. I am surprised at how complex the HTML platform is. I had advocated far simpler platforms. Perhaps it's similar to the tension between simpler processor architectures (RISC) and more complex ones (CISC). It's very expensive to create complex systems, but when investment is concentrated on a common goal, we can build very rich platforms. HTML5 has lots of special case quirks because it is more of an agglomeration of experiments than a single engineered whole. But, as a byproduct, we also have a very powerful general purpose platform that can be used for any purpose.

The reason I use the term "refactoring" is that complexity is not intrinsic. We can ignore the complexity of implementing the platform itself and instead take advantage of the (relative) simplicity of using the platform to create applications that take advantage of the new opportunities.

What happens to the telecommunications industry when WebRTC makes telephony just another application? What happens to the television business when a "smart TV" is just another generic device with a screen and an HTML5 engine? Stay tuned to find out.

CE