Web Applications and the Evolution of the Hardware Abstraction Layer

Jacob Dilles

George Mason University

March 15 2009

A computer is defined as a machine that manipulates data according to a list of instructions, however it has come to mean something more. Operating a modern Core class possessor that can perform 2.4 billion 64-bit operations per second (Intel Corporation, 2009) by listing one instruction at a time would not be very useful. However the first 1960 era computers, like the ENIAC, were only programmable with machine-code instructions entered by hand using switches and patch cables. This was not a significant problem at the time, when the longest program was 304 batch-operated instructions, took days to write and 5 to 10 seconds to run, and the punched card reader limited data input rate to 250 bytes per second. (Ballistic Research Laboratories, 1955).

However computers grew more capable at an exponential rate while steadily decreasing in cost. By 1970 machines, like the relativity affordable DEC PDP-7 that were capable of fairly advanced timesharing, expedited the development of the operating system - an interface between the physical computer hardware and an abstract environment that allows more than one program to be run simultaneously. Due to the wide variety of hardware architecture at the time, early operating systems had to be written for a specific machine, and were not consistent or interchangeable between computer manufactures and models. In 1973 the Unix operating system developed at Bell Laboratories was ported from PDP-11/20 assembly to the C programming language, and could be used on any machine with a C compiler with minimal modification. Copies of Unix with the source code were distributed to universities across America – including, most notably, the University of California, Berkeley - where researchers were allowed to modify and extend the operating system.

Over the next 15 years, Unix evolved rapidly, forking several times with additional capabilities, libraries, and functions. In 1988 the Institute for Electrical and Electronics Engineers published standard IEEE-1003, now also known as POSIX, which formally specified the user and software interfaces to a "Standard Operating System" (IEEE, 2001). By this time, affordable home computers (in contrast to minicomputers and mainframe computers, which were several times the cost of a house, and car, respectively) had become available. The OS, a widespread availability of high-level to machine code compliers, and more reasonable storage capacities measured in megabytes finally separated the user and their software from the physical machine. By 1990, it would be reasonable to expect a piece of software to run on any machine with the same operating system, regardless of the physical hardware in the machine.

The exponential growth continued, and even the most cursory overview of the 90's to present would be beyond the scope of this paper, however modern operating systems are still defined by the ability to provided the same experience to users – and the same contract to developers – across any hardware configuration. Modern systems also have been widely ported and Microsoft Windows, Linux, Sun Microsystems SunOS, Red Hat, Mac OS X (just to name a few) will all run on the same x86 architecture.

While giving the user a choice of operating systems encourages competitive innovation, it presents a significant challenge to software developers. Binary executable format, file system structure, and graphical user interfaces (GUI) are all operating system specific; a text editor for Windows 7 will not on Linux. Similarly, they are not forward compatible, so the Windows 7 text editor cannot be run on Windows 95. Because APIs vary so widely between systems, developing a consumer application for a target audience that spans multiple platforms is an arduous task. Each additional platform requires extensive porting, testing, and development time, increasing overhead and complexity.

There are two solution tracks to solve this problem. One utilizes a software abstraction layer, a kind of virtual machine (VM), which is a piece of software native to the operating system that it is run on. The VM acts as a translator between OS specific functions and a common instruction set made available to developers. Just as the hardware abstraction layer in the OS creates a consistent user experience across a variety of hardware, the software abstraction layer creates a consistent user experience across a variety of operating systems.

One successful implementation of a VM is the Sun Microsystems Java Virtual Machine (JVM). It is available for almost every consumer OS, and a remarkably wide range of embedded MCUs. Every JVM understands Java Byte Code, and will execute a program with the same results (or as close to as possible) regardless of the underlying OS and hardware. (Lindholm & Yellin, 1999) This means that a Java application, such as the productivity suite OpenOffice, will run on Windows, Apple, and Linux operating systems with no changes whatsoever to the executable. The adoption of Java was initially slow due to performance penalties associated with the additional software layer (Jelovic), however innovations such as Just In Time compilation, HotSpot compilation, adaptive optimizations, advanced caching and garbage collection have brought newer Java VMs extremely close to their natively compiled equivalents (Lewis & Neumann, 2004). The JVM and development tools are funded and freely distributed by Sun Microsystems, a valid strategy considering the estimated four billion JVM enabled devices worldwide.

The second solution to the cross platform development problem is the Web Application, where the application is executed on a remote server and displayed in a web browser. It is important to remember that centralized computing is not a new concept. Around 1970, virtually all non-administrative computer usage was remote terminal access to a time-sharing mainframe, primarily due to the size, cost, and operating requirements of hardware at the time. Terminals like the VT100 started "dumb" with minimal processing capability – just enough to remember the last 24 or so 80 character lines of received text and blink a cursor – a keyboard, a monitor, and a serial line or telephone modem to connect directly to the mainframe.

At that time, a number of mainframes in universities and government agencies had connections to other mainframes, and were collectively known as ARPANET. The specification of TCP/IP in 1974 and complete conversion of ARPANET to TCP/IP in 1982 coincided with cheaper, smaller, and more reliable computers; it became practical to run a basic shell on the terminal itself, exchanging data with the mainframe with packets switched over the network – which was less expensive than a direct line. In 1989 the European CERN network and the Australian AARnet were linked via TCP/IP with ARPANET, marking the beginning of the worldwide network we know as the Internet.

Eventually in the nineties, the entire OS moved to the terminal, in the form of the Internet enabled personal computer. (Wikipedia contributors, 2009)

Before 2000, the majority of personal computer Internet connections were 56 kilobit/second or slower, which is insufficient for any real time transfer other than text. However by May 2007, an OECD report indicated over half of all residences in the United States had broadband access (FCC defined as 768kbps or faster), accounting for 81% of active Internet users. (OECD, 2007) Higher speeds allow real time transfer of multimedia – graphics, sound, animations, etc. – that may be used to build a web based GUI.

Running a web application has many advantages. A web browser is bundled with almost every desktop operating system, so running web applications does not require obtaining and installing a VM. Since the application is hosted remotely, there is no need to distribute bug fixes and version upgrades to each user. License control is simplified, and the specifications of the client machine become irrelevant. Some downsides to using web apps include the lack of consist ant user interface options, limited local machine access, increased network load, and offline access. Offline access is one of the biggest challenges web applications face: without an Internet connection the user has no access to their data.

The open source, freely distributed Gears (previously Google Gears) browser extension is a noteworthy attempt to address the offline issue. It is available for Apple, Linux, Windows, and Windows Mobile, and is downloaded and installed separately from the browser. The Gears framework provides many features, including a HTML server and

SQL database that allow web applications to run offline. Since Gears is a JavaScript extension, it is possible to write an app to run on both Gears/Non-Gears clients by disabling features (like offline access) on the clients that do not support it. (Google, 2009)

Creating a web application with the quality and style of a desktop application UI is significant challenge. Web browsers were originally designed to display the text based HTML, not complex graphics or interactive displays found in desktop applications. Improvements have been made, such as the adoption of cascading style sheets (CSS) that allow the dynamic positioning of elements on a webpage to allow a more 'solid' looking interface (W3C, 1998), Extensible Stylesheet Language Formatting Objects (XSL-FO) but there are subtle nuances between layout engines in browsers that may disrupt a properly rendered display. (Butler, Giannetti, Gimson, & Wiley, 2002) Adobe Flash (previously Macromedia Flash), Java applets, and Microsoft Silverlight have contributed to the user experience, but a developer can not guarantee any of these technologies present on a client's machine, and not all of the technologies are available for every platform.

However this reoccurring problem of platform dependence has limited the proliferation of web applications. As more devices access the Internet, the problem of creating UIs for each device type grows worse. By using software to create a presentation to match the delivery device's capabilities, developers would need to create only one version of their Web content. (Butler, Giannetti, Gimson, & Wiley, 2002) According to Gomez, et al.: "The mushrooming of ad hoc developed Webbased applications is generating great problems when the enterprise faces aspects such as its maintenance or evolution, which are basic requirements in such a changing environment as the web.

Furthermore, this new brand of software products differ from traditional information delivery systems in that they require non-trivial functionality to be provided to the user, and ad hoc development processes have already proven unsuccessful to address functionality issues. In order to keep the possibility of failure to a minimum, the development process for such applications, either new or migrated, should evolve in a sound scientific way in order to adapt to the new environment."(Gomez, 2001)

Looking past the patchwork of available supplemental technologies, systematic development of maintainable platform independent web applications is in the same problem space as the software abstraction layer – both have the goal of a device-independent front-end specification. Other differences are not as great as they first seem: The offline capable Google Docs and platform independent OpenOffice software package are remarkably similar, both requiring an Internet connection and file download at some point, allowing files to be saved, edited, and printed, and providing a consistent interface regardless of operating system or hardware, essentially an extension of hardware abstraction layer.

While we have made significant progress, technology does not stand still, and OpenOffice is on the track to obsolescence despite its portability. The ultimate goal in UI/hardware independence is to remove the notion hardware altogether. With wide spread high-speed Internet access, third generation wireless data coverage, and IEEE 802.11N WLAN becoming commonplace, there is no justification for keeping physical devices separate. In the best interest of the user (and developer), the future is a single virtual machine that operates on any device – cell phone, desktop, laptop, tablet, refrigerator, etc… - without additional effort. It is not far off, enterprise "cloud" servers -

which present a virtual environment to applications but hop between physical hardware automatically to balance load and counter hardware failure - are extensively used for their low cost and high efficiency. For the consumer, web applications are the next step towards the hardware abstraction goal.

The advantages to designing web applications are too great to dismiss: Ease of backup and recovery of centralized data, maintenance and upgrade simplification, consistent and repeatable interfaces that increase productivity and decrease learning curve area, and access to all application features regardless of system privileges, operating system, hardware, and geographic location. All software developers should keep this in mind during the design phase, even niche markets that traditionally do not require platform independence, as these benefits will increase the value and utility of their product.

JSD

Works Cited

Ballistic Research Laboratories. (1955, December). *A Survey of Domestic Electronic Digital Computing Systems.* Retrieved Febuary 2009, from Facts about Antique Computers : http://ed-thelen.org/comp-hist/BRL-e-h.html#ENIAC

Butler, M., Giannetti, F., Gimson, R., & Wiley, T. (2002, October). Device Independence and the Web. *IEEE Internet Computing* , 81-86.

Cerf, V., Dalal, Y., & Sunshine, C. (1974, December). *SPECIFICATION OF INTERNET TRANSMISSION CONTROL PROGRAM.* Retrieved Febuary 2009, from Network Working Group Request for Comments : http://tools.ietf.org/rfc/rfc675.txt

Gomez, J. C. (2001). On Conceptual Modeling of Device-Independent Web Applications: Towards a Web Engineering Approach. *IEEE , 8* (2), 26-39.

Google. (2009). *Architecture*. Retrieved Febuary 27, 2009, from Gears API Developers Guide: http://code.google.com/apis/gears/architecture.html

IEEE. (2001, January). *The Open Group Base Specifications Issue 6.* Retrieved March 2009, from Unix.org: http://www.opengroup.org/onlinepubs/000095399/download/susv3.zip

Intel Corporation. (2009, January). *Intel Core2 Duo Processor E8000 and E7000 Series Datasheet.* Retrieved Febuary 22, 2009, from Intel Corporation: http://download.intel.com/design/processor/datashts/313278.pdf

Jelovic, D. (n.d.). *Why Java Will Always Be Slower than C++*. Retrieved Febuary 25, 2009, from Jelovic Engineering: http://www.jelovic.com/articles/why_java_is_slow.htm

Lewis, J., & Neumann, U. (2004). *Performance of Java versus C++*. Retrieved Febuary 26, 2009, from J.P. Lewis research: http://www.idiom.com/~zilla/Computer/javaCbenchmark.html

Lindholm, T., & Yellin, F. (1999). *The Java Virtual Machine Specification* (Second Edition ed.). New York, New York: Prentice Hall.

OECD. (2007). *Broadband Growth and Policies in OECD Countries.* Directorate for Science, Technology and Industry. OECD.

W3C. (1998, May 12). *Cascading Style Sheets, level 2 CSS2 Specification*. (World Wide Web Consortium) Retrieved March 2, 2009, from World Wide Web Consortium Recommendations: http://www.w3.org/TR/2008/REC-CSS2-20080411

Wikipedia contributors. (2009, March 15). *History of the Internet*. (Wikipedia, The Free Encyclopedia.) Retrieved March 15, 2009, from Wikipedia: http://en.wikipedia.org/wiki/History_of_the_Internet

2055 words double-spaced 12 point Times New Roman