Streamlining the Web Development Process Through Unification

Jeremy Florence
Department of Computing Sciences
Villanova University
Villanova, PA 19085
jeremy.florence@villanova.edu

Abstract
As the technologies behind the web and its applications grow and become more advanced, developers find themselves in a bind. Users can choose from five major web, each with varying architectures and interfaces. To accommodate these diverse browsers, developers are now spending as much if not more time debugging their websites and applications as they are designing them. Even with the success of services such as Google Apps, Facebook, and Twitter, the of debugging times will inevitably slow down progress. This research project explores approaches of unification that can broaden the compatibility of the five browser, while still maintaining their unique attributes. The hoped for result of these approaches is to reduce both the development times and cost for projects.

1. Introduction
Although this topic has not yet garnered mainstream support as topics such as computer graphics or accessibility, this paper includes a definition of the topic itself. In the process, the paper highlights the areas in which further research can be done.

Sections 2 and 3 will look into ways in which browser unification can be done, along with the pros and cons of each method. Section 4 features a proposed method developed from sections 2 and 3 that can be developed and later implemented. Finally, section 5 will look into benefits of the advancement and implementations of this research.

2. Internal Browser Unification
Each of the five major browsers has a unique way of implementing code such as HTML and PHP and features that have become standard over the years. Though the implementations are different, many of the actions that end-users see are the same, such as bookmarks, tabbed browsing, and so forth. Because of those similarities, a possible way to unify the major browsers would be from the inside out. Inspired by the various distributions of Linux, a single unified core could be developed, on top of which each browser maker can build the various components, such as rendering engines, interfaces, and so on. The upside to this method is that the browsers would be able to maintain the attributes that set them apart, while giving developers a central development environment with which to work in. The downside is that there is no way to force the companies that make the browsers conform to a standard coding method. Given the closed source nature of Internet Explorer, Safari, and Opera, this option may not be completely viable.

3. External Browser Unification
A potentially more viable option for browser unification with examples already available, external browser unification refers to development tools that take the browsers in their current form and adapts ways (either on the server side or the client side) to pass information to each browser in such a way that the information works and is displayed correctly regardless of browser used. There are a number of ways to go about this, highlighted below.
3.1 Code Unification

While the inner workings of applications written in dedicated languages such as Java, Python, and Ruby tend to not have problems on their own, presentation can be compromised because of the ways that browsers read and show markup languages. XHTML 1.0 hoped to fix that problem by creating a much more strict standard for writing page, and the new HTML 5 hopes to continue that process.

The upside to relying on code unification is it spawns new growth and development at the code of web design, the code. HTML 5, for example, is based off of the old HTML 4.1 and XHTML 1.0 while adding new tags (the components that make up the language) and features. The downside here is similar to that of internal unification, in that there is no way to force developers to switch to HTML 5.[7]

However, a new Google App, Wave, which was created entirely in HTML 5, has the potential to persuade other to make the switch and begin propelling development forward.

3.2 Third Party Unification

Unification through third-party applications (such as browser add-ons, external programs, etc.) is possibly the most viable option at the moment. There are a number of examples of programs that are on the right track at the moment. One is X-marks, formerly a Firefox only add-on; it now runs on IE, Firefox, and Safari and allows a user to share his/her bookmarks and passwords across computers and browsers. As such, a Safari user at home can transfer his bookmarks to a Linux based laptop running Firefox and his Windows based work PC with Internet Explorer.

Another example, from the gaming world, is Steam and games that run on the platform. The way it works is that games that are attached to the service give users access to friends lists, text chat, and voice, regardless of whether or not the game originally supported those functions. This brings together players of single and multi player games through external support.

Finally, a third example, which is the inspiration for the proposal in section four, is a Firefox add-on called IE tab. This add-on allows for the user to run web pages in a separate tab within Firefox using the Internet Explorer rendering engine. While IE still has to be installed on the system, it allows the user to run two separate engines under one window, such that now rare websites and applications that would only run under Internet Explorer can be accessed without having to close or minimize Firefox and wait for IE to load.

The upside to using third party unification is that it allows for much more flexible and intuitive applications. However, as the browsers themselves evolve, so to would the applications attached. This can be counted if the project is open source and catches on quickly, as many open source apps with large followings tend to update frequently.

4. Proposal

After determining various ways in which the browsers can be united, a system for actually implementing some of the above methods was conceived. The system itself is made of two separate parts that can work together to ease developer woes, make deployment easier, and offer the end user an experience that does not change simply because they use one browser over the others.

4.1 The Development Browser

The first part of the two part experiment proposal can be used individually or in tandem with the second part to ease the woes of developers. The first part is a browser, codenamed Genesis, that was inspired by the Firefox add-on IE Tab. IE Tab itself works by accessing the local Internet Explorer files from within Firefox, effectively allowing a user to run both Firefox and Internet Explorer in the same window.

Genesis takes this one step further by adding the rendering engines of Chrome, Safari, and Opera as well, while implementing the additional feature of being able to edit code from within
the browser and updating automatically. This system would eliminate the need for developers to have to run each of the main browsers separately and have the code open in another program and have to alt-tab and refresh continuously. [3]

4.2 The Server-Side Script/Application
The second part of a two part experiment proposal is a server-side script, codenamed Revelation, that is meant for deployment. After noting what changes need to be made to Javascript files, CSS files, etc, using Genesis, instead of having to deploy a large number of files with minuter changes, Revelation would allow a smaller number of files to be used, and, based on the browser making the request, dynamically change the corresponding CSS/Javascript files to fit the browser based on parameters entered into Revelation. While no language has been chosen for this application, likely candidates include Java and PHP.

5. Benefits
As mentioned above, the main reason for this research is to reduce the additional development time and cost for porting/adapting websites and applications to multiple browsers. Regardless of the method used, there are other benefits to this research as well. For example, it would be easier for new developers, even those with little to no design background, to come in and create new applications and sites.[4] Creating and implementing new design patterns would be easier as well.[2] Finally, projects such as a browser-based version of Folding@home would be easier to create.

References