Earlham College

Computer Science Department

Senior Capstone, Fall 2012

Senior Research Paper

Author: **Leif DeJong**

December 12, 2012

Contents

1 Abstract

This paper looks at the history of user-centered design and covers some current methods for dealing with multi-platform interfaces in the context of the web. It attempts to determine what it means for an interface to adapt to a user's environment, characteristics, and situations. The paper proposes a new method for implementing an adaptive system that pays special attention to each user's particular needs and circumstances. The paper also looks at a software solution to the new approach and discusses the results.

2 Introduction

With the vast array of devices that are available in today's world, there is a need to create interfaces that are rendered on those devices appropriately. In addition, the tasks that users perform on these devices largely vary. Users have different characteristics and require different interfaces that depend on their needs and circumstances. For nearly a decade, these issues have been thought about and concepts such as adaptive and adaptable user interfaces have been proposed. So far, there is no one solution that directly attempts to solve all of these issues at once. Developers find it difficult to create compatible software as platform languages and hardware characteristics vary from device to device. In addition, creating interfaces that map uniquely to a given device is time consuming, inefficient, and practically impossible with the vast array of platforms that exist today. Attempting to determine a user's usage on a device presents itself as a challenge given the vast capabilities of a particular device. Regarding accessibility, many interfaces are not tuned to be helpful and intuitive for the physically disabled and more energy is spent on the average human than on the special cases. If it were somehow possible to identify user's unique characteristics, devices, and circumstances, one could tailor an interface to suit those individual needs creating a personalized and optimized environment for that user. That said, the rise of web technology including HyperText Markup Language 5 (HTML5), Cascading Style Sheets 3 (CSS3), JavaScript (JS), and HyperText Preprocessing (PHP) has opened the door to a universal language that developers can turn to.

3 Background Information

Before attempting to explain and focus on the idea of a universally accessible and uniquely tailored interface, it is important to look at the research that has already been done in this area. Charlie Peck from Earlham Colleges quotes, "don't reinvent the wheel." There are two philosophies at hand when dealing with interface adaptively. One involves creating an auto rendering fixed environment for the user called an Adaptive User Interface (AUI). The other involves creating a customizable and flexible environment called an Adaptable User Interface (ABUI). Both concepts fall under the category of Human-Computer Interaction (HCI) which is a subset of computer science that deals with developing software with the user primarily in mind. In addition to this, we will look what it means to be a multi-platform interface paying specific attention to the utilization of the web as a tool for accomplishing this.

3.1 Adaptive User Interface

An Adaptive User Interface (AUI) is an interface that adapts to the characteristics, behaviors, and preferences of the user. Generally speaking, the system monitors the user's interaction with the system and "adapts" its interface to match those characteristics. According to Stuerzlinger, most of modern software makes all user options available up front cluttering the environment, making it difficult to find specific functions, and leaving very little room for actual work to be accomplished [Stuerzlinger et al., 2006]. An example of this is Adobe Photoshop where most of the options are splashed on the desktop in the form of widgets cluttering the interface and leaving very little room for image manipulation. An adaptive approach to the Photoshop problem would monitor what the user does with the software and in turn make widgets available or unavailable depending on the widget's usage. According to Tufte, 10% of a user interface should be devoted to the admin controls while 90% should be given to the actual content at hand [Tufte, 1997]. AUIs can reduce a cluttered interface by displaying elements only appropriate to the user. This approach allows end users to have a personalized experience that is intuitive and understandable. The users are not bothered with irrelevant information and are allowed to focus in on the tasks at hand in an effective manner. In a world that demands results, tasks can be completed faster and with less effort than before, improving overall efficiency. Researchers have tried to come up with ways to accomplish this for nearly a decade and according to Bunt, for such an interface to become possible, one must be able to model cognitive processes and produce quantitative predictions of the user's behaviors based upon their previous interactions. These behaviors are then compared against an existing user model in order specify the nature of adaptation that fits that particular user's needs [Bunt et al., 2004].

Although there are many positive aspects to AUIs, Bunt states that most AUIs are known to have implementation flaws and cause unpredictable results that often do not map appropriately to the user. Users feel a lack of control for their environment and become confused with unexpected results [Bunt et al., 2004]. AUIs make the user interface less flexible and customizable as the rendering is determined by the program not the user. These issues may not be as important for some contexts but Stuerzlinger states that users seem to be surprised when their environment suddenly changes in the case of Adobe Photoshop's widgets appearing and disappearing[Stuerzlinger et al., 2006]. There has to be a certain amount of freedom to let the user choose their interface and decide how to interact with it.

3.2 Adaptable User Interface

An Adaptable User Interface (ABUI) according to Stuerzlinger, is one where the users are able to customize their environment by manipulating preferences provided by the developer. These customizations allow users to create fully personalized environments that match their needs and gives them the freedom to change those configurations on the fly [Stuerzlinger et al., 2006]. Studies by Bunt, show that users prefer the freedom of ABUIs. This particular interface reduces the time it takes for users to complete a task than if they were using the default configurations of that system. Bunt also points out that it is more effective for users to customize their environment up front in some sort of setup than on the go as they are using the system. There are two levels of customization that can be provided to the user. The first is called surface customization which

involves simplistic modifications to the UI. The second is called deep customization which involves changing the functions of the underlying system to reflect the user's preferences. Developers lean towards surface customization as it reduces the amount risk in manipulating deep system calls from the user's end [Bunt et al., 2004]. Apple does this by giving users the ability to change the appearance of the desktop background, system settings, and buttons in the top menus in applications such as Finder, but does not allow changes to core system files.

Although ABUIs give users the freedom to create their own environment, there are several challenges that face this approach. How do you motivate users to keep their systems up to date and how much customization do you allow them? If you provide user's with too much control, software often malfunctions and breaks. There has to be a balance between restricting customization and allowing absolute freedom. Users should not be confused by the flexibility of the system but should not feel that they have a lack of control.

3.3 The Hybrid Model

With AUIs and ABUIs, there are many factors that are called into question regarding their functionality. How much customizable freedom should be given to a user without endangering the system? How can the system prevent users from feeling that they don't have control over their environment? How can you ensure that the matching of the user model correctly maps to a specific user? AUIs and ABUIs seem to clash conceptually as they are quite opposite from each other but if they were to be combined, they can provide a very effective user experience. This is what I refer to as the hybrid model where AUIs help the user maintain a ABUI. In other words, the software helps the user customize his or her environment. Bunt states that the hybrid model can help users be selective in what features they want in their environment. An example of this involves an interface that monitors the user's activities and issues suggestions for how to make their system more personable and tailored to their needs. The users can approve or disapprove these prompts and the system will adapt based on those decisions. Bunt also suggests that it is important to help users maintain their personalized environment as their tasks evolve over time [Bunt et al., 2004]. This model can also help users customize their interfaces early on in an attempt to increase productivity. This hybrid model should assist users in exploring their environment, exploiting its full capabilities, and helping them understand which aspects of that environment are automatic. In addition, Kuhme states that users should also have the ability to take back control anytime upon their request and be able to revert to defaults or saved settings [Kuhme, 1993].

3.4 Multi-Platform User Interfaces

When looking at AUIs and ABUIs, it is also important to identify the context in which the software is being used. In modern times, the increase in multiple devices such as cell phones, smart phones, tablets, laptops, and large screen TVs has been so dramatic that developers are often forced to program for each of these devices separately and individually. As Foss puts it, "we have hit a point where screens sizes are moving in every direction, both bigger and smaller" [Joly, 2012]. Not only do they have to focus on the user's characteristics with reference to AUIs

but also how the interface renders on the user's device. According to Gajos, software should be able to render on any platform at the users command along with reflecting the characteristics of that individual user [Gajos and Weld, 2004].

The problems associated with this approach is the there is no way developers can possibly program for all kinds of devices effectively. For example, it is inefficient for developers to program for an iPhone™in Objective C and then replicate that same program for an Android™phone in Java. It is vital to tailor software to the needs and characteristics of the end user without replicating code. Factors such as cultural backgrounds, geographical locations, disabilities, and device capabilities have to be taken into account when thinking about the end user. Examples of these include language, symbols, units, currency, network types, network quality, and special needs devices. In addition, it is very important to know a device's capabilities and limitations when creating an interface tailored to it. With the array of language and hardware constraints, how does one begin to determine the best method for a multi-platform interface that unique addresses a given user?

4 The Web Context

If a device is capable of accessing the Internet in the 21st century, it is most likely capable of parsing web languages such as HTML5, CSS3, and JS. We can use these tools to program an interface once while maintaining compatibility from device to device. This becomes incredibly convenient and powerful as the Internet is becoming more and more popular. More people now are accessing the Internet on mobile devices than on conventional desktops [Wroblewski, 2011]. As a result of this, multi-platform design for the web becomes a priority to developers in this day and age. It is imperative to tailor websites to the devices they are rendered on and improve loading time for those with poor networks. The goal is to get information to more people in more countries faster.

When looking at the current state of most websites, one finds that they are designed for desktops that rely on fixed width displays. This resulted from designers wanting to control how their interface looked on a given browser. It was easier to fix the display and guarantee that it render the same way on every device than to make it render differently according to the screen size [Marcotte, 2011]. This has created huge problems as the fixed width displays are often distorted and too big to view on mobile devices, forcing users to pan and zoom. In addition, the same is true on large TV screens, except that the display of fixed widths is often too small to read. It is much more difficult for designers to rely on fluid width displays as those kinds of designs are prone to unpredictable appearances.

When mentioning network types and speeds, it is important to note that not everyone has high speed Internet access, especially in third world countries. In addition, devices such as smart phones are not capable of viewing high quality images and video so why make them download all that unusable information? When users have to wait a long time for a given page to load, they become frustrated and leave without the opportunity to take advantage of what that page has to offer.

These issues have to be addressed as fixed width displays are no longer the best approach

to web design, multiple renditions of the same program is inefficient, mobile devices often can't take advantage of high quality images and video, and network speeds vary from region to region. Developers need one way of working with code so that their software renders on any device with any given network effectively and efficiently. Instead of looking at every device as an individual entity, one must generalize the interface to match some categories such as tablets and smart phones versus specific devices such as iPhones®and iPads®.

When attempting to develop a method for multi-platform interfaces that uniquely address user's characteristics, one must know some of the tools that currently deployed today. In this next section, I will go over two approaches for multi-platform interface design. One method is called Responsive Web Design (RWD) coined by Ethan Marcotte and the other is Mobify JS created by Igor Faletski, John Boxall and Peter McLachlan.

4.1 Responsive Web Design

Responsive Web Design (RWD) is a method of using a browsers screen size to determine how a website's interface adapts to a given device. According to Marcotte, it consists of "a flexible grid-based layout, fluid images and media, and media queries...from the CSS3 specification." [Marcotte, 2011] RWD uses a technique called graceful degradation. According to Florins, this process involves identifying a source interface, designing for the most optimum or default case of that interface, and then creating transformation rules in order to produce code that targets a more specified environment. In order to accomplish this, you first split the user interface into chunks then create image and widget transformation rules allowing them to shrink and expand according to the device's screen size. Depending on the target, you then reshuffle the widgets and chucks so that they conserve more space and then remove any unnecessary widgets while maintaining the overall purpose of the interface [Florins et al., 2006]. The tools that make RWD possible is the theory of graceful degradation in combination with CSS3 media queries. Marcotte defines media queries as "an incredibly robust mechanism for identifying not only types of media, but for also inspecting the physical characteristics of the devices and browsers that render our content." [Marcotte, 2011] The specification allows you to identify a maximum and minimum browser width and override core CSS for that particular width range. This is the magic behind RWD that allows device specific design adaptation of the user interface. We can program for specific screen ranges that allow us to determine whether the device is in a particular category such as desktop, tablet, smart phone, etc. This allows us to generalize the code and avoid programming for every single device separately. RWD uses flexible images and media in addition to a grid system that helps elements fall into place depending on the device they are rendered on.

This problem with RWD is that it does not address network throughput nor address optimization of content. The approach is largely client side and redundant code in the form of media queries are loaded on the client's device and may never be executed. RWD is a key component in designing multi-platform interfaces that uniquely identify the user but it is not the answer to the approach.

4.2 Mobify

Igor Faletski, John Boxall and Peter McLachlan started a company in Canada called Mobify. The company takes a very different approach to multi-platform interface design with regards to the web than RWD does. The company leads an open source project called modify.js which is a library for taking existing fixed width websites and renders mobile versions of them. The script makes pointers to specified HTML tags and data that are copied over to a mobile template that can be rendered in a separate manner than the main site. The mobile template can use different styles and scripts and resides on a different URL than the main site. When the user updates the desktop version of the site, the mobile version will automatically sync as its data is referred from the desktop version. This is a client side implementation of mobile adaptation and does not deal with wide screen TV displays.

The main limitations of this approach is that it does not address network throughput and content optimization. In addition, if the HTML structure is changed on the desktop version of the site, it can potentially break the mobile version that refers it. This makes the mobile version less flexible and the scope of adaptation is very limited. The rendered is also executed on the client decreasing performance and efficiency [Mobify, 2012].

5 The New Approach

When talking about multi-platform adaptable interfaces, it is not enough to just talk about their design but also how they function and perform. There is a need to optimize a website's content depending on the recipient of that content. This involves a server-side processing strategy where one has to account for image and video optimization depending on network throughput and screen width. According to Wroblewski, developers should also use image sprites to group images into a single file, bundle and minify styles and scripts cutting down on HTTP requests, limiting dependencies on large JS libraries that are not effectively used, use appropriate HTTP headers to optimize server-client communication, and take advantage of CSS3 to replace heavy JS functionality. [Wroblewski, 2011] Optimizing web content is very important as every detail of the software adds up and learning how to manage that efficiently is crucial. The current techniques covered do not take the optimization of content into account limiting ways to best serve content to a specific user.

5.1 Clients, Servers, HTTP Requests, and Cookies

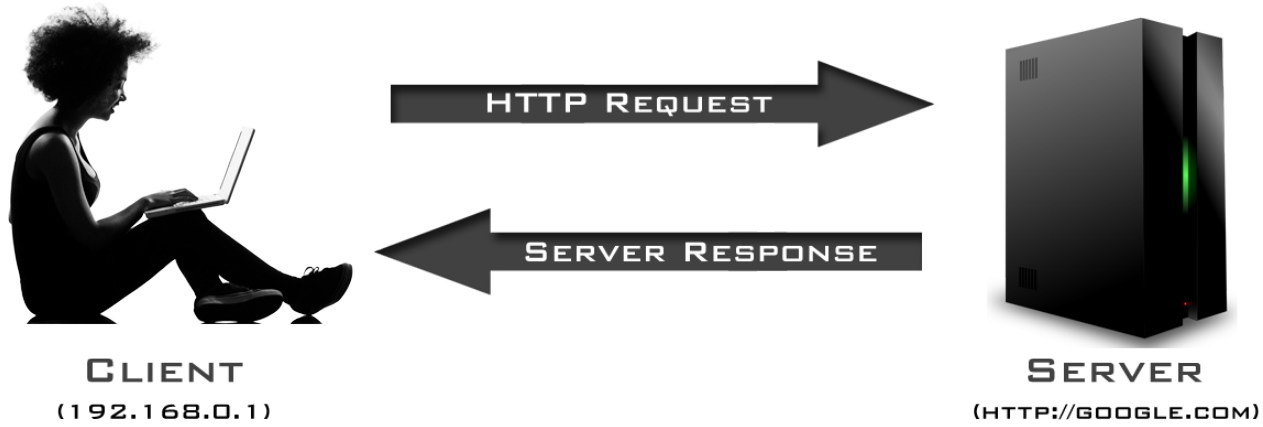

Figure 1: Clients-Server Communication

A client is defined as the user's device or a device that is making a request for some information coming from a remote source. A server is defined as the device holding the information that the client is requesting. When a client makes a request to the server, it makes a request called an HTTP request. The request is sent from the client and processed by the server. After parsing the request, the server sends the client back the information it requested. HTTP requests are incredible useful as they contain a lot of information about the user. Some of this information includes the client's IP address, the requested data, the client's user agent, and the client's language. The user agent is information about the client that includes the client's actual device, the browser, and the operating system running on that device. [W3, 2012] A cookie is a textile typically generated by the server upon receiving a HTTP request that is sent and stored in the client's web browser. Upon succeeding requests from the client, the cookie is sent back to the server, processed, and sent back to the client. The cookie can contain information about the user such whether they are logged in and what they were viewing before making another request. The information can be encoded in such a way that only the server can make sense of it as other resources will not be able to interpret what the contents of the cookie mean. The idea behind a cookie is to allow information to be shared and kept in sync between the client and the server. Using JS, a cookie can be made and kept up to date by the client. [Central, 2012] Based on the HTTP request information and the cookie, the server can know more about the client and render a suitable website that is appropriate and efficient. The idea is to take the majority of the adaptation away from the client's end and have the server render a site knowing the client's specific constraints.

5.2 Gathering Information

In order to produce a multi-platform interface, developers have to first collect as much information about the user as possible. If a developer can identify a user, the user's device, the device's constraints, and the way the user uses that device, it becomes possible to produce a personalized environment which matches those factors. From an HTTP request, information such as the user's device capabilities, the user's geographical location, and the user's language can be known. When the client makes the initial request to the server, information can be extracted from that request. Using server-side languages such as PHP or CGI, we can extract the information from the HTTP request and parse the data storing values into variables. One of the key pieces of this data when dealing with multi-platform adaptability is the user-agent. This information contains the user's device and software running on it. We can run this information against a pre-existing database until we hit a match. Once we have identified the device and the server can make sense of it, we can proceed to render a site that is appropriate to that platform.

5.3 Limitations of a HTTP Request

1	IP Address	159.28.60.103
2	Visit Type	Human Visit
3	User Agent	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_2) AppleWebKit/536.26.14 (KHTML, like Gecko) Version/6.0.1 Safari/536.26.14
4	Local Time	12 Dec 2012 12:24:42
5	Country	United States
6	Latitude	33.9777
7	Longitude	-118.4351
8	Client Referer	http://leifdejong.com/research/
9	Connection Type	keep-alive
10	Request Method	GET

Figure 2: HTTP Request Information (Server Side)

There are many limitations when solely relying on the information extracted from the HTTP request. The most important one deals with the user agent's information for determining the device capabilities and constraints. Devices such as the iPad mini ®and the Android Tablet ™have the same user agent information as their predecessors the iPad ®and Android ™. This makes it impossible to distinguish between these devices that have very different capabilities in terms of their screen size and hardware. Already it is time consuming and ineffective to keep track of all the millions of devices that exist today and store them in a database. The lookup costs of matching a user agent to that database is high and often unpredictable. It is also likely that the database may not contain information about the device that the server is looking up. Another important factor is that you can not determine network throughput from the information present in an HTTP request. This makes it impossible to render content without some client-side collection of information when optimizing the user's experience. In addition, referring back to the idea of a hybrid model with AUI and ABUI, the information in the HTTP request can not be modified leaving the user very little control over the nature of adaptation.

5.4 Generalizing the Device

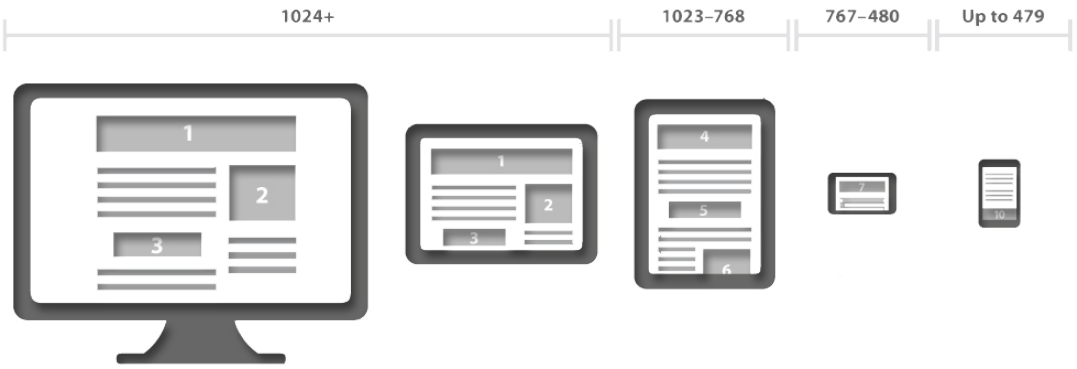

Figure 3: Device Categorization - http://www.zauraliyev.com

To address the problem that developers face when trying to program for every specific platform imaginable, one can generalize a device and have code that is guaranteed to work for that categorization. This can be done by identifying the screen width of a particular device and then coming up with ranges that define which category that device falls under. Some examples of categories include: desktops and above; laptops and tablets; portrait phones and landscape phones. Once a device is appropriately categorized, a set of rules can be applied to render content that matches that device and screen width. This approach is powerful because every single device imaginable will fall under one of these categories. In addition, this approach makes it trivial to change or add a new device as a category, such as a tablet mini, by making slight modifications to the screen ranges.

5.5 The Software Design

A better approach in developing multi-platform interfaces that are uniquely tailored to the characteristics of a specific user is to preform server pre-processing to optimize the website's framework. The client then does processing to determine the user's screen width and network speed. The server then does post-processing to produce optimized dynamic user content. The interface must be able to: identify the characteristics of the user; be flexible and scalable matching the dimensions of the user's device; have one code core instead of different renditions of the same software; and optimize content such as images and video depending on the network and the device.

5.6 Server Pre-Processing

Figure 4: Server Pre-Processing

When a client initially makes an HTTP request to the server, there is an opportunity for pre-processing to take place. The server can read the request, record the initial information known about the user, consolidate and compress styles and scripts, and compress and optimize the site's HTML framework. The user's content has not yet been optimized at this point as a result of the lack of information about the user's screen width and network throughput. In addition, the content can not be optimized until the user has confirmed the nature of adaptation as stated in the hybrid model for AUIs and ABUIs. This pre-processing can be written in any server-side language such as PHP and should also incorporate RWD concepts for better display. Once the pre-processing is complete, the rendered information is sent to the user.

5.7 Client Processing

1	Throughput	48.9 ms
2	Connection Speed	Fast
3	Device Type	Desktop and Above
4	Screen Width	1280 px
4	Language	en-us

Figure 5: Processing Information (Client Side)

Once the server sends the initial optimized HTML framework back the client now can start processing the user's onboard characteristics. The two pieces the framework cares about is the screen size and calculating the network throughput. The screen size can be extracted from the the Document Object Model (DOM) of the browser and measurements can take place to calculate network throughput. Once the network throughput is known, the framework can categorize the connection speed as fast, medium, and slow by matching it to a pre-defined range. The screen width is also categorized to determine the generalized device type as explained above. This initial gathering and categorizing of information can be done in JS as it is one of the only languages that can perform client-side operations.

5.7.1 Calculating Throughput

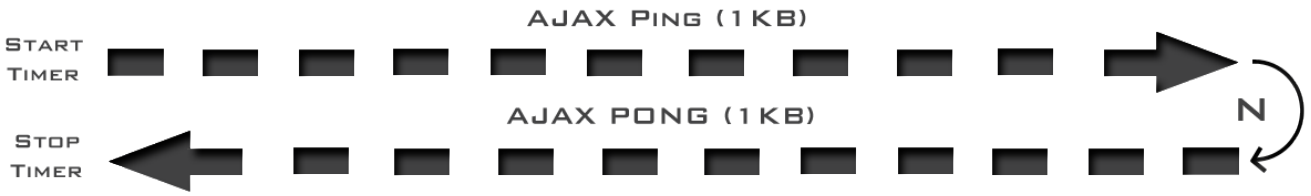

Figure 6: Calculating Throughput

To avoid reloading the framework, JS has a tool called Asynchronous JS and XML (AJAX) that allows the client to send and receive data from the server without refreshing the page. This can be used to calculate the network throughput by initially starting a timer and sending a load of 1KB to the server. When the server returns the 1KB load to the client, the timer is stopped and the lapsed time is calculated. This can be done n times and the results can be averaged to get the most reasonable measurement for network throughput.

5.7.2 User Verification

Once all the client's data has been calculated and collected, a dialog confirmation message can be issued to the user for verification of the findings. A button is available in the framework to change these confirmations on the fly. This allows the user to have some control over how the website adapts for their specific device. Once the user approves the information, the throughput and screen width along with other characteristics that have been calculated are stored in an array and written to a cookie so that the server can access this information on the subsequent HTTP request.

5.7.3 Dynamic Content

To conserve the information that was just processed by the client, AJAX can be used to bring in content and load it in the framework dynamically. The AJAX request is like any other HTTP request in that is holds some client information along with the contents of the cookie. This decreases redundant client processing and brings optimized pieces of content to the client faster. The server keeps processing and optimizing content and makes it available to the client on demand.

5.8 Server Post-Processing

Figure 7: Server Post-Processing

After sending the initial framework to the client, the server waits until it receives an AJAX request for content as stated above. The cookie is read and the information is stored into variables. Using some conditions, the server can determine the size and quality the images and videos should be by matching it to the device and network category as determined from client processing. The images and video are optimized along with any other content and sent back to the client where it is rendered in the framework.

6 Measuring the Software

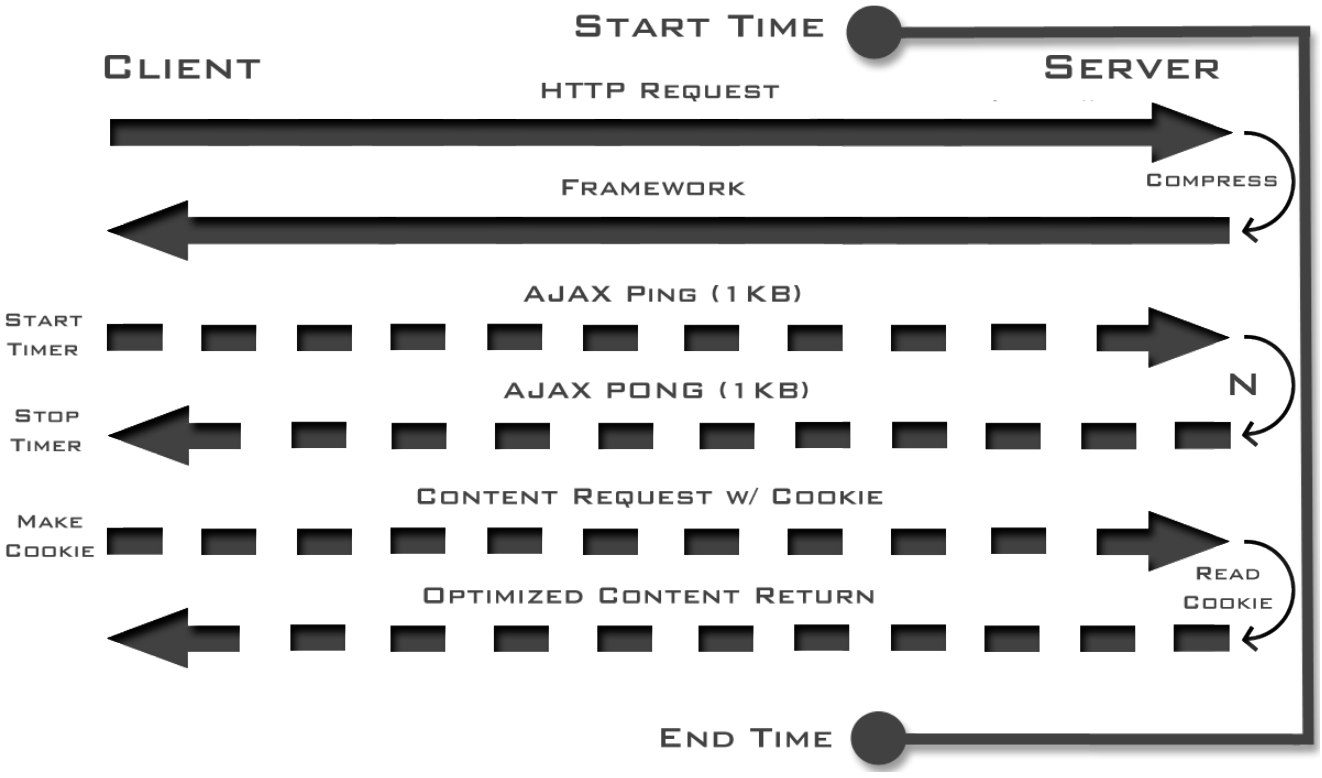

Figure 8: Measuring the Software

In order to test that this method is effective, one must be able to measure the loading speed of the framework with content and compare that to a static counterpart. A timer page is initially loaded on the client that has stop watch capabilities. The timer starts and the framework or static counterpart is then loaded into an iframe. Once the loading finishes, the timer stops and

the lapsed time is calculated. The developer can manipulate the screen width and network speed to run more tests on different conditions. This gives developers a reasonable way to measure the effectiveness of the interface and understand the optimal points.

7 Addressing Screen Types

When talking about the vast array of screen types, there is a solution called "Mobile First" developed by Luke Wroblewski. The approach states that an interface should be designed thinking about a mobile environment first. This forces developers to embrace the constraints of a mobile device upfront and understand the capabilities to fully maximize the user's experience. With regards to screen types (touch, pointer, etc..), this approach means that all elements of a site will be touch compatible and degrade appropriately depending on the screen width. JS allows touch events to be issued creating a compatible way to program touch directly into your interface. [Wroblewski, 2011] This method scales well because if an interface is optimized for touch, it will also easily support clicking and pointing on other screen types.

8 Dealing with Network Changes

Networks, whether wireless or physical, are irregular as speeds tend to fluctuate at an unpredictable rate. As a result, it is important to not find a balance for n where n is the number of AJAX requests in throughput caculation but also recalculate throughput to maintain accuracy. This can either be done on every content request or after a certain amount of time has lapsed. The cookie must always be updated and kept in sync with the server to minimize incorrect rendering of an interface.

9 Conclusion and Results

The amount of AJAX requests needed to determine network throughput creates a slower loading speed as communication is often slower than computation. A balance in the ratio of communication and computation has to be established. The number (n) of AJAX requests to calculate network throughput has to be proportional to the speed. For instance, if the network speed is fast, less tests have to be carried out. If the network speed is slow, more tests should be carried out to. In addition, there should be more tests carried out in a network that fluctuates more frequently than others. When looking through the results, I found that the adaptive interface produces better results than its static counterpart when the network is slow and the screen width is small. When the screen width is maxed and the network is fast, there is not much optimization taking place and the communication during the throughput calculation makes the site load slower than its static counterpart. Therefore this framework will help those with smaller devices and worse network speeds than those with larger devices and faster network speeds. Overall, this project was a success in creating a multi-platform interface that deals with user's unique characteristics and situations.

Bibliography

[Bunt et al., 2004] Bunt, A., Conati, C., and McGrenere, J. (2004). What role can adaptive support play in an adaptable system? In *Proceedings of the 9th international conference on Intelligent user interfaces*, IUI '04, pages 117–124, New York, NY, USA. ACM.

[Central, 2012] Central, C. (2012). Cookie central.

[Florins et al., 2006] Florins, M., Simarro, F. M., Vanderdonckt, J., and Michotte, B. (2006). Splitting rules for graceful degradation of user interfaces. In *Proceedings of the 11th international conference on Intelligent user interfaces*, IUI '06, pages 264–266, New York, NY, USA. ACM.

[Gajos and Weld, 2004] Gajos, K. and Weld, D. S. (2004). Supple: automatically generating user interfaces. In *Proceedings of the 9th international conference on Intelligent user interfaces*, IUI '04, pages 93–100, New York, NY, USA. ACM.

[Joly, 2012] Joly, K. (2012). One design to rule them all?. *University Business*, 15(2):49 – 50.

[Kuhme, 1993] Kuhme, T. (1993). A user-centered approach to adaptive interfaces. In *Proceedings of the 1st international conference on Intelligent user interfaces*, IUI '93, pages 243–245, New York, NY, USA. ACM.

[Marcotte, 2011] Marcotte, E. (2011). *Responsive Web Design*. Jeffrey Zeldman, New York, NY, USA.

[Mobify, 2012] Mobify (2012). http://www.mobify.com/mobifyjs/].

[Stuerzlinger et al., 2006] Stuerzlinger, W., Chapuis, O., Phillips, D., and Roussel, N. (2006). User interface facades: towards fully adaptable user interfaces. In *Proceedings of the 19th annual ACM symposium on User interface software and technology*, UIST '06, pages 309–318, New York, NY, USA. ACM.

[Tufte, 1997] Tufte, E. (1997). *Visual explanations: images and quantities, evidence and narrative*. Graphics Press.

[W3, 2012] W3 (2012). http://www.w3.org/protocols/rfc2616/rfc2616-sec14.html.

[Wroblewski, 2011] Wroblewski, L. (2011). *Mobile First*. Jeffrey Zeldman, New York, NY, USA.