Dartmouth College
Dartmouth Digital Commons

Dartmouth College Undergraduate Theses | Theses and Dissertations

6-1-2013

Nimbus A Novel Multi-Device Operating System Shell

Sang Jin Lee
Dartmouth College

Follow this and additional works at: https://digitalcommons.dartmouth.edu/senior_theses

 Part of the Computer Sciences Commons

Recommended Citation

Lee, Sang Jin, "Nimbus A Novel Multi-Device Operating System Shell" (2013). *Dartmouth College Undergraduate Theses*. 84.
https://digitalcommons.dartmouth.edu/senior_theses/84

This Thesis (Undergraduate) is brought to you for free and open access by the Theses and Dissertations at Dartmouth Digital Commons. It has been accepted for inclusion in Dartmouth College Undergraduate Theses by an authorized administrator of Dartmouth Digital Commons. For more information, please contact dartmouthdigitalcommons@groups.dartmouth.edu.

Nimbus

A Novel Multi-Device Operating System Shell

Sang Jin Lee
Advisor: Lorie Loeb

Dartmouth Computer Science Technical Report TR2013-736

Figure 0: Nimbus on all officially supported platforms: Windows 7, Android 4.2, OSX, and iOS.

Table of Contents

Abstract

Web technologies have advanced considerably in the last decade. With the introduction of HTML5 and CSS3 web standards, websites are no longer static documents, but rich applications that are capable of rivaling those written in native code. Similarly, internet infrastructure and adoption has reached a point where "always connected" and cross-platform applications such as Facebook, Twitter, Google Drive, Spotify, Dropbox, and Youtube are becoming dominant. While these web applications have acclimated users to cross-platform and cross-device experiences, operating systems have lagged behind, offering subpar experiences across devices. We present the design, prototype implementation, and evaluation of Nimbus, an operating system shell built on web technologies that aims to provide a compelling cross-platform and cross-device OS experience. We demonstrate how Nimbus can be used to create a seamless user experiences across platforms through realtime synchronization of application data and UI interactions. We also examine the advantages and disadvantages of implementing Nimbus as an OS shell, rather than a standalone cloud operating system, such as Jolicloud or Google Chrome. Nimbus is a disruptive approach to designing an operating system shell that allows developers to easily create multi-device applications that provide seamless experiences for users across all of their devices.

1) Introduction

Over the past decade, web technologies have evolved tremendously. With the introduction of HTML5 and CSS3 web standards, websites are no longer just static documents, but rich applications that are capable of rivaling those written in native code. With the number of global internet users reaching nearly 35%, and the number of internet users in OECD nations reaching well over 70% [10], internet infrastructure has also reached a point where web-reliant products and companies such as Google, Facebook and Twitter are increasingly prevalent. A large part of this is due to a rise in the variety of devices available through which people can connect to cyberspace. In America, as of 2012 over 60% of adults own a laptop computer, 58% own a desktop computer, 31% own a tablet computer, and nearly 90% own a mobile device [20]. While rich web-based applications like Facebook, Twitter, Youtube, and Spotify have acclimated users to cross-platform experiences that span across their laptops, tablets, and mobile smartphones, operating systems have lagged behind, offering subpar experiences across devices.

We present the design, prototype implementation, and evaluation of Nimbus, an operating system shell and several example applications built on web technologies that aim to demonstrate compelling cross-platform and cross-device OS experiences. Through websockets and asynchronous data requests, Nimbus synchronizes not only file data, but also application data across platforms in realtime.

Unlike previous work that focuses on cross-device communication and interaction at the application level, or work that explores cloud operating systems within the frame of distributed computing [4, 6, 7], our approach focuses on user-centric design and explores cross-device operating systems through the lens of the end user experience.

Nimbus enables developers to easily create web applications with a native look and feel that offer seamless interactions across OS environments (Windows, OSX, Linux, iOS, and Android) and devices (laptops, tablets, and smartphones). We demonstrate some of these cross-platform interactions through rich media applications including an audio player, video player, photo viewer, text editor, and presentation tool. Through these example applications, we also provide a UI and UX framework for designers looking to design cross-platform applications.

We also examine the advantages and disadvantages of implementing Nimbus as an OS shell, rather than a standalone cloud operating system such as Jolicloud or Google Chrome. By developing Nimbus as an OS shell designed to work in tandem with existing operating systems, we allow for full backwards compatibility with the parent OS, while maintaining a

native look and feel. Nimbus takes a novel approach to designing an operating system shell that enables developers to easily create multi-device applications that provide seamless experiences for users across all of their devices. Figure 1 illustrates the structural layout of Nimbus, and shows how Nimbus can integrate seamlessly into an existing OS.

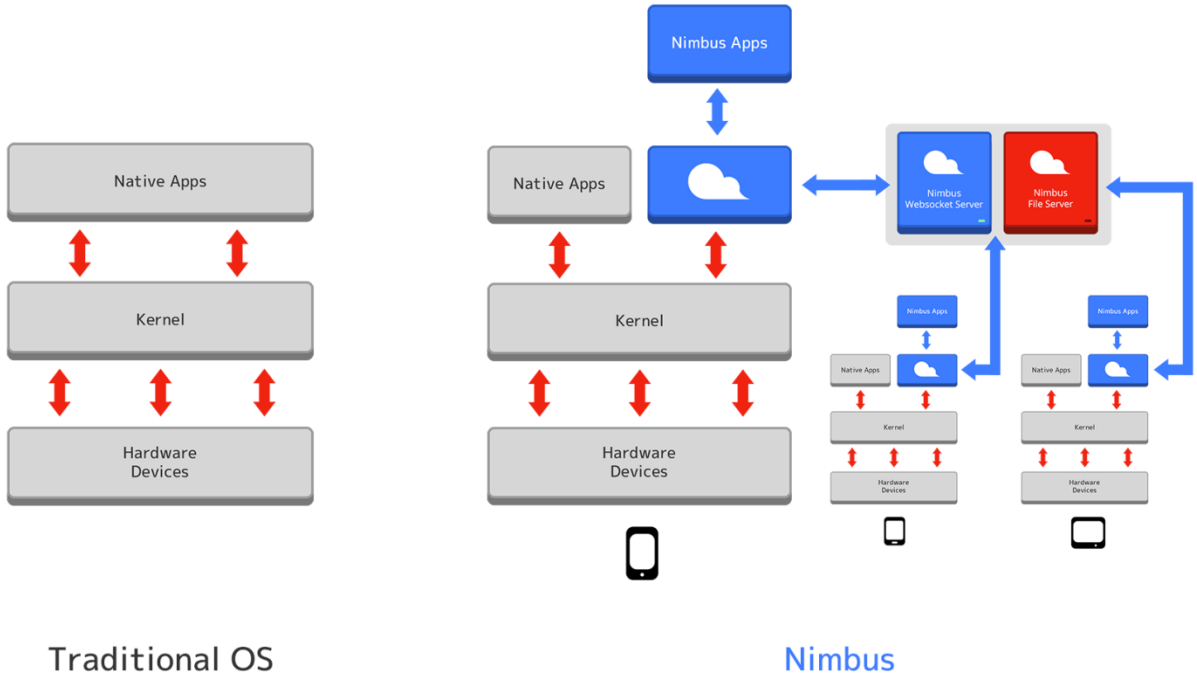

Figure 1: Nimbus integrates into your existing OS, providing an extra layer of connectivity across devices

2) Related Work

The concept of a cloud-based operating system extends far before the rise of the internet in the 1990s and 2000s. The first class of cloud operating systems emerged in the 1980s, when enterprises and educational institutions explored the concept of distributed computing with operating systems such as *Clouds* [4]. More recently, however, cloud operating systems and applications supporting cross-device interactions have been a hot topic in the consumer space, starting with file synchronization services [5, 21, 22], remote control interfaces [2, 12, 16], and more recently collaborative office and productivity software [1]. These products demonstrate the value of the cloud not only as a computational resource, but also as a means to offer compelling cross-platform and connected experiences. Many of these products, however are primarily single applications bound to proprietary software ecosystems. Our concern is to design an operating environment that provides these connected experiences on a global level for consumers and simplifies the design and development of cross-platform applications.

Most of the current consumer cloud-enabled operating systems are full stack packages that require users to install and boot to a completely new operating system with different apps and affordances. In the case of JoliOS and ChromeOS, users must completely abandon their current operating system, meaning that any native applications currently installed on their devices will be inaccessible [14, 24]. Both JoliOS and ChromeOS attempt to compromise by also offering limited in-browser interfaces for users who do not wish to completely abandon their current OS [11]. The result is a subpar experience that feels unwieldy. In our approach, we design Nimbus as a software layer that replaces only the OS shell, leaving the rest of the operating system intact. This allowing users to experience the product as a native application (without any unwanted browser chrome) and enjoy applications that take advantage of the cloud while also retaining their library of applications installed on the parent OS.

Recent work has also established that realtime cross-device interaction is often cumbersome in practice, and that one solution to creating fluid and seamless interactions across devices is to take advantage of the input methods afforded by one device and translate them to an appropriate action on another device [23]. We build on this cross-device approach by creating metaphors that translate across keyboard, touchscreen, and mouse inputs. We also create a UI framework that encourages consistent and logical applications that span across multiple devices and screens.

3) Experience & Aesthetic Design

> *How might we design an operating environment that enables users to have a seamless experience across devices?*

3.1) Cross-Platform Experience

Nimbus has been designed from the ground up as a multi-device and multi-platform operating environment.

The user experience for Nimbus was designed around principles of user centered system design [19], and we targeted users with multiple devices and optimized for users with smartphones and laptop/desktop computers.

Based on findings that many users with both desktops and smartphones prefer to share information across their devices [15], the initial concept for Nimbus was to create a data management layer that would synchronize files across desktops and mobile devices (a service that would synchronize all of a user's files files, as opposed to selectively synchronizing folders as is the case with products such as Dropbox [5]). After early user tests and feedback, however, we quickly realized that file synchronization alone did not offer the seamless experience we were targeting. As a result of this early testing, we quickly iterated and evolved our concepts to focus on realtime data synchronization between devices.

As a result of our initial user tests, we realized that the bulk of user interactions that contribute to seamless cross-platform experiences could be divided into two categories: application data, and file data. While file data synchronization occurs in the background to sync files that have been dumped to the disk, application data synchronization occurs in realtime, synchronizing user interactions and temporary application data in running applications (figure 2). This dual-tier approach to data synchronization combines the features of products such as Apple's Airplay and Box's Onecloud [1, 22], and serves to create a holistic representation of user data across platforms. This combination is key to providing a seamless experience across devices.

File Data Sync	App Data Sync
Saved files	User interactions
	Clipboard Items
	Current Application state information

Figure 2: Differences between File data synchronization and App data synchronization in Nimbus

In our prototype implementation of Nimbus, file information, such as user preferences and other data stored to the local file system, is updated asynchronously when an application modifies an existing file. Application information, including user interactions and relevant application data stored in memory, is synchronized across devices in realtime (figure 3). Ultimately, by providing this framework for syncing file and application data, Nimbus makes it simple for developers to create applications that span across multiple devices and provide seamless cross-platform experiences.

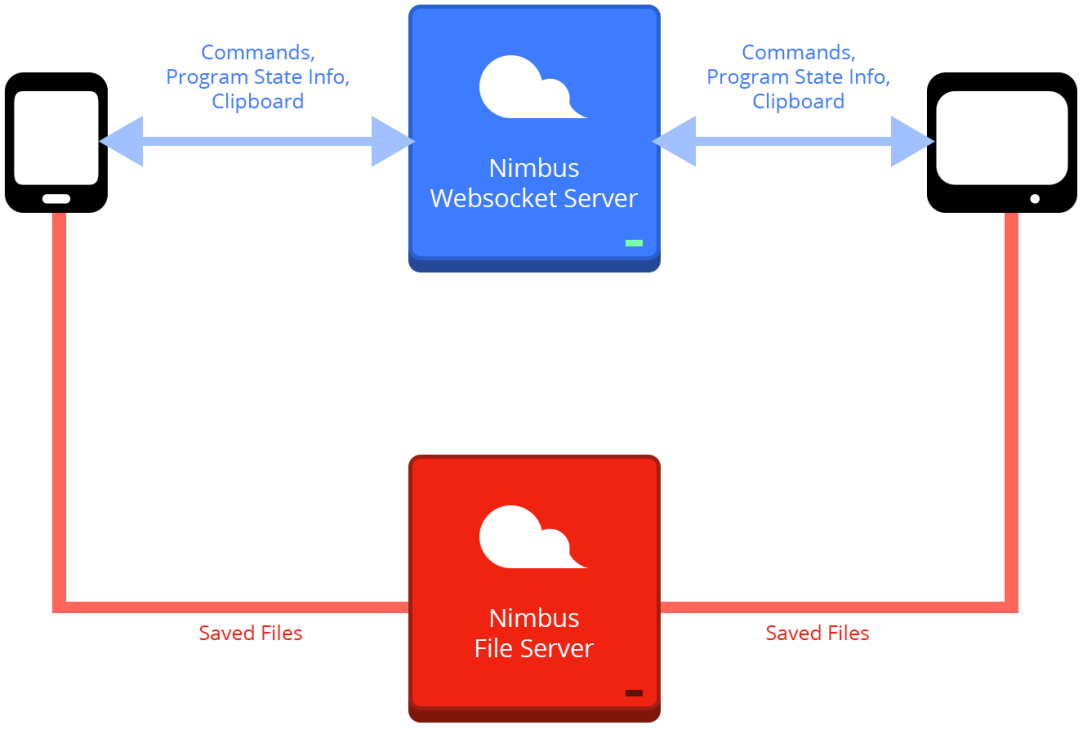

Figure 3: File synchronization (red) and App data synchronization (blue). Blue path is synchronized in realtime. Red path is synchronized more slowly.

3.2) UI Framework and Guidelines

Designing applications for multiple screens and devices can be a daunting task, especially given the novel realtime cross-device capabilities afforded by Nimbus. In order to ameliorate the design and development of multi-device applications, Nimbus provides a specialized UI framework for designing applications for a wide array of devices and screen sizes, as well as a set of guidelines for designing experiences that take advantage of multiple devices. These guidelines are built on the principles of direct manipulation interfaces [9] and are ultimately designed to give users a sense of control and ease of use across platforms.

The Appbar and Appscreen

Nimbus applications are launched from the Appbar on desktops and from the Appscreen on mobile and tablet devices (figure 4). All Nimbus applications are placed in the Appbar and Appscreen, and can be arranged into stacks, which are expandable folders of applications (figure 5). The Appbar floats at the top of the screen, above the user wallpaper, while the Appscreen displays icons floating above the user wallpaper.

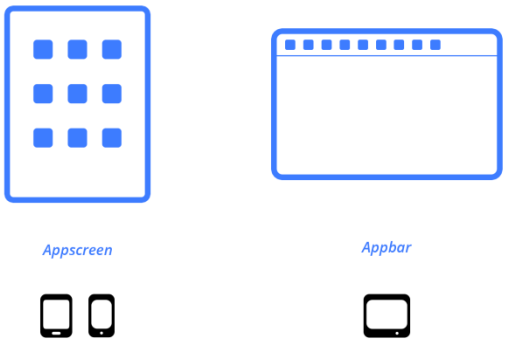

Figure 4: Appscreen and Appbar

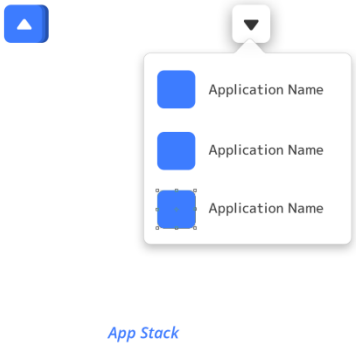

Figure 5: Application Stacks

Gestures

Although Nimbus allows for a variety of input methods including multitouch, mouse, keyboard, and accelerometer, in order to maintain usability across platforms, critical user interactions (such as those involving app navigation and app management) must use single-point taps on touch-enabled devices, and mouse clicks or keyboard shortcuts on keyboard and mouse enabled devices. Moreover, we encourage designers to consider the

strengths and weaknesses of various input methods on each of their target devices. For instance, an image viewer application on desktops may use keyboard shortcuts to navigate through images, as well as swipes on touch-enabled devices (figure 6).

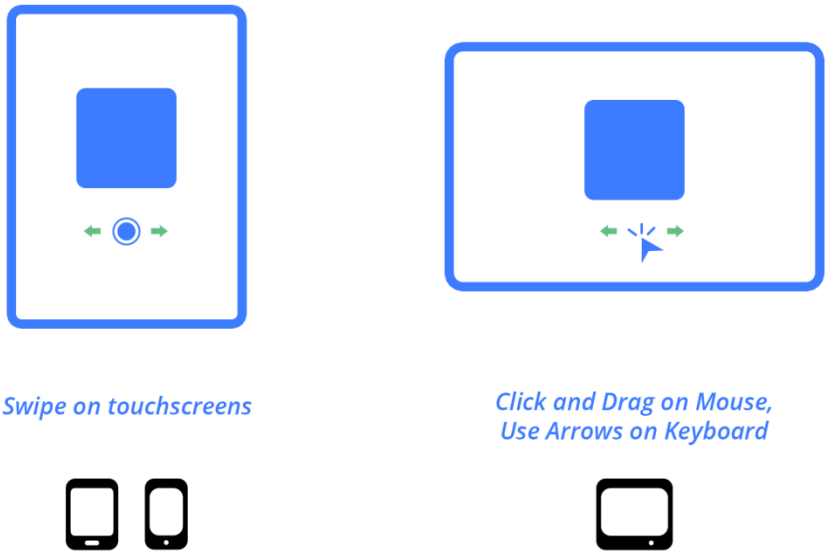

Figure 6: Multiple user inputs map to same interface action

Nimbus Default Cross-Platform Interaction Metaphors	
Multitouch/Mobile Device Interaction	**Mouse & Keyboard Interaction**
Tap / Double Tap	Click / Double Click
Two finger tap	Right Click
Tap & Hold / Two finger tap and hold	Left Click & Hold
Swipe in DIRECTION	Click and drag in DIRECTION
none	Hover
Multi-Finger (3+ Finger) Swipe	*none*
Accelerometer Tilt	*none*
Vertical drag	Mousewheel scroll
Pinch In/Out	CTRL + Mousewheel Scroll In/Out

Figure 7: Default Cross-Platform Interaction Metaphors

For this reason, we encourage designers to first determine what actions users should be able to perform within their application, and afterwards, determine for each target

platform, map those actions to the appropriate user input. We discourage designers from using hover actions on desktop devices, as this has no matching metaphor on touchscreen devices (figure 7).

Default App Layout

Taking inspiration from the principles of responsive web design [17], Nimbus provides a default three-row, two-column layout for applications. The left column (gray outlined box in figure 8) houses top-level app navigation, while the right column has a stream of application content. On smaller screens and windows, the left column is hidden by default, but slides into view when toggled via the menu button in the top-left. Navigation between app screens for multi-screen apps is handled with a navigation toolbar located at the bottom of the screen on mobile/tablet devices and at the top of the window on desktops (figure 8). Applications can be closed by tapping or clicking on the close button in the top-right corner. On devices that support multiple windows (desktop), apps can also be maximized, restored, and minimized in a similar fashion.

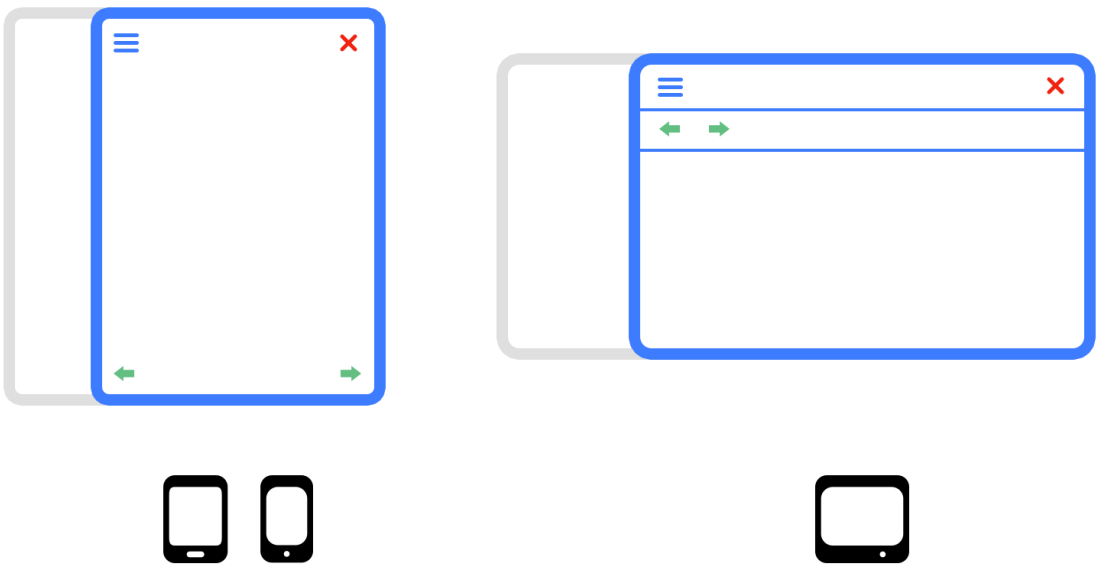

Figure 8: Two Column Responsive Layout

Multi-Device Applications

For multi-device applications, designers should consider the strengths and weaknesses of each platform and deliver content and create interactions based on their evaluations.

While we certainly encourage designers to create consistent app interfaces across devices, we also encourage designers to experiment and create asymmetric interfaces, where the interface for an application on one device may not closely mirror that of the same application on another device. Figure 9 showcases possible advantages of an asymmetric interface in a cross-platform game that utilizes the mobile device touchscreen and sensors as a wireless game controller that controls the game environment on a desktop or desktop-connected television.

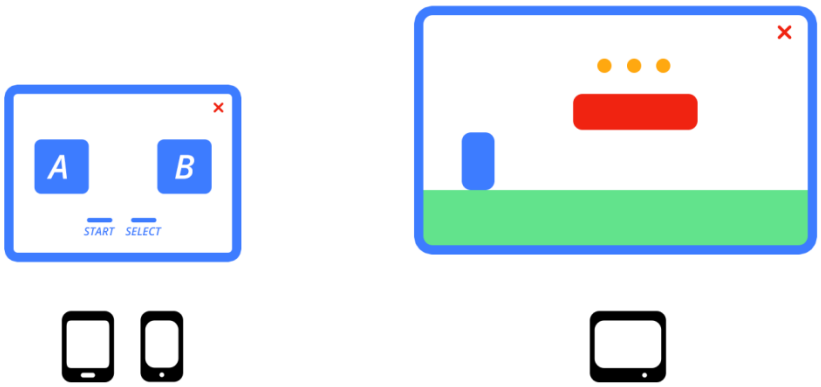

Figure 8: Example Asymmetric Interface.
Game controller interface on mobile device, Game world interface on desktop

3.3) Visual Design

The Visual Design for Nimbus focuses on clean, neutral colors and lines. The goal behind this semi-minimalistic approach to the visual design in Nimbus is to create an unobtrusive user interface that puts content first, and encourages users to interact with the OS.

White and off-white are used to create whitespace (negative space) around icons and other UI components. In addition, color saturation and contrast are used to make interact-able UI elements (such as icons and buttons) distinct from their surrounding whitespace. This contrast also serves to draw the user to interact-able areas of the screen. Darker gray solid lines are also used to create visual boundaries between list items, and containers. Blue is used as a semi-neutral color for containers that users can interact with (window borders, and default icons). Figure 10 shows a screenshot of the Nimbus desktop interface.

The primary shapes in nimbus are rounded rectangles, with a border radius of 4px. The slightly rounded edges create depth and soften overall image of the OS, making it appear

more organic and inviting without losing the structure afforded by sharp, unrounded rectangular containers.

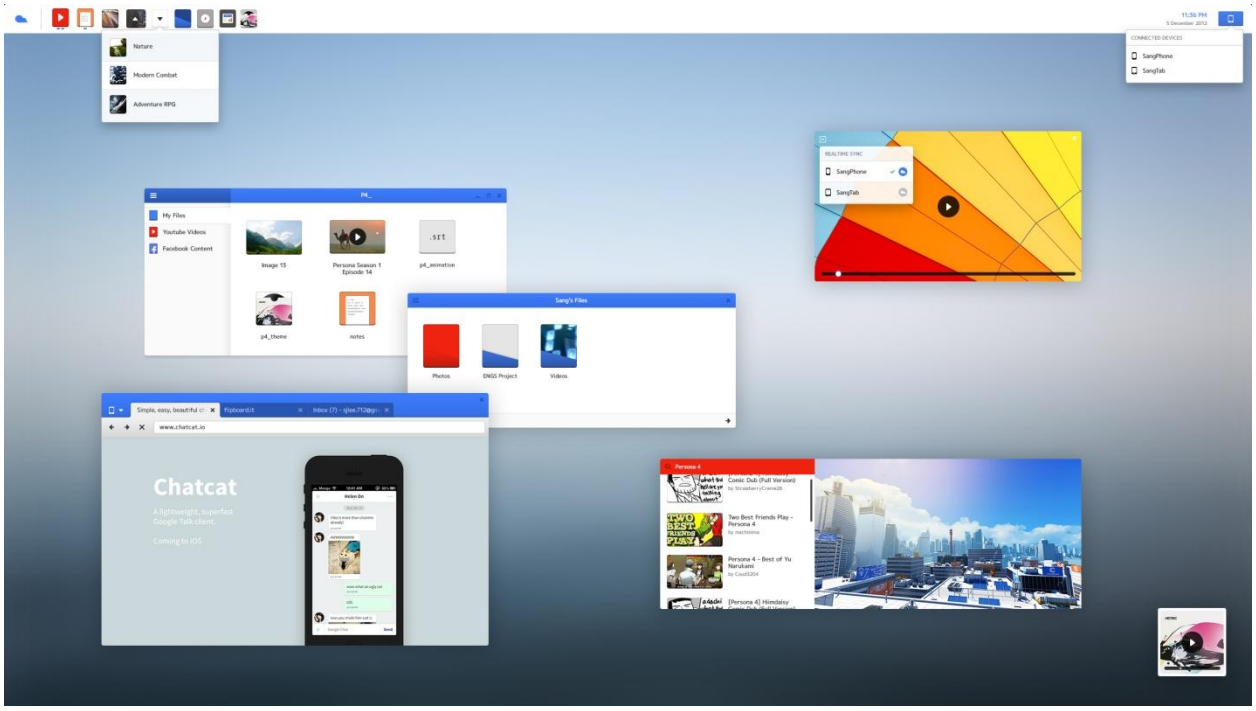

Figure 10: Screenshot of the Nimbus Desktop Interface with the Photo, File Manager, Web Browser, Video, and Music Apps.

Faint drop shadows, gradients, and value adjustments are used to create depth and volume. On desktops, drop shadows are also used to indicate the z-order of windows and to further establish the boundaries of each window.

Transparent UI components are used sparingly, but are encouraged for use in media applications where designers may wish to maximize the screen real estate by overlaying UI components on top of content.

4) Implementation

Due to its cross-platform and cross-device nature, Nimbus is built on web technologies that are standardized across devices. The prototype Nimbus GUI is rendered entirely in CSS3 and HTML5 using the webkit rendering engine and is built on Google's Chrome Packaged App [25] interface. The shell is also scripted entirely in Javascript, using HTML5 APIs and the Chrome Packaged App interface to gain access to device hardware, including serial ports, bluetooth radios, and accelerometer data. On Windows desktops, Nimbus is capable of replacing the default shell, while on all other supported platforms, Nimbus runs on top of the OS shell, providing an extra layer of app synchronization. Figure 11 illustrates how Nimbus can be integrated into an existing operating system and allow for applications to communicate with other devices.

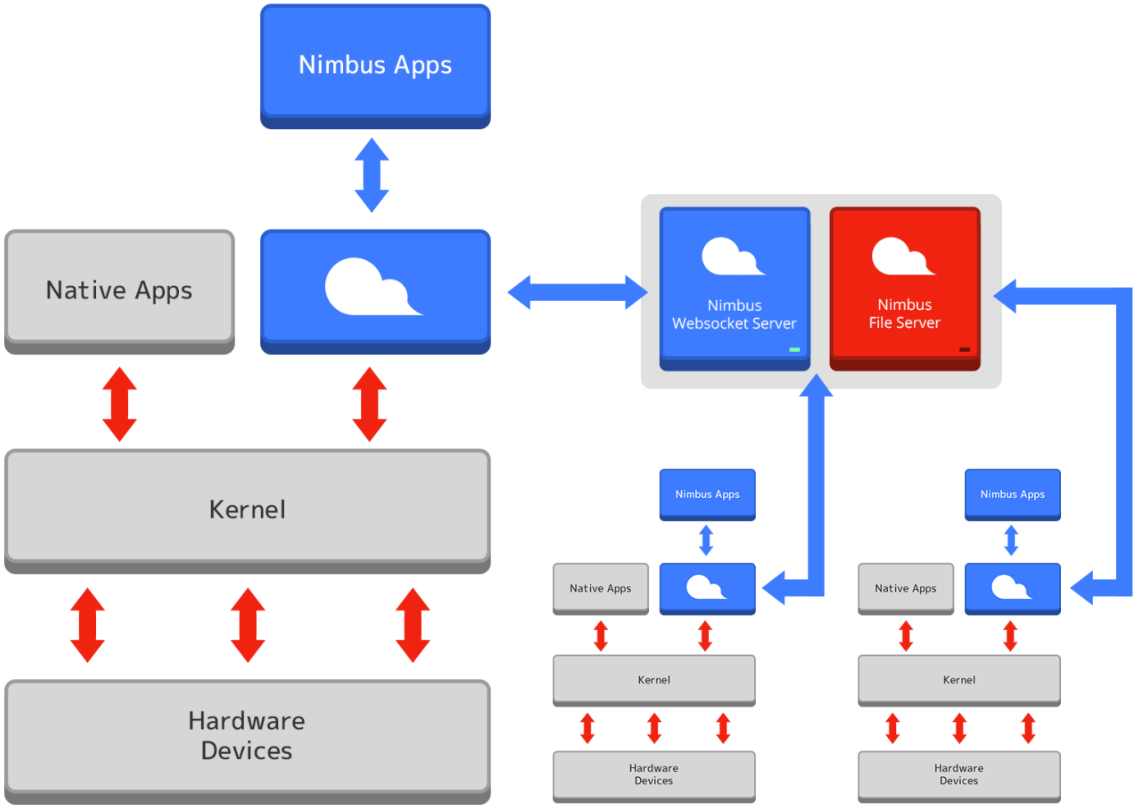

Figure 11: The Nimbus stack layout. Nimbus is designed to add an extra layer of functionality on top of the preexisting operating system

4.1) Synchronizing User Interactions

Nimbus uses secure HTML5 websockets to transmit user interactions to a websocket server running on NodeJS.

User interactions are first packaged into a Javascript object as JSON. The application then posts the JSON information to the parent Nimbus process through the cross-frame message posting API [8]. The parent Nimbus process then stringifies the JSON and transmits it through a secure websocket connection. The server then reconstructs the stringified JSON, checks it to ensure that the commands encoded in the JSON are valid, and then restringifies the JSON and broadcasts it to all devices that are currently connected for the user. After receiving a message, each client then reconstructs the stringified JSON, checks to see if it should execute the command, and then routes the command to the appropriate application through a cross-frame message. Figure 12 illustrates how user interactions flow from one device to another.

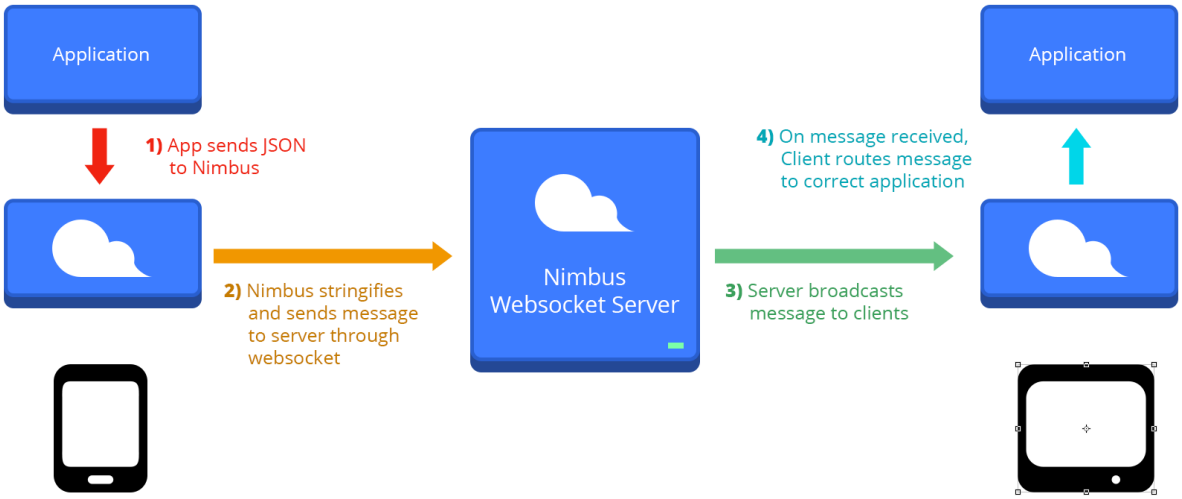

Figure 12: Example pipeline for sending a user interaction from a mobile device to desktop.

These JSON commands are very lightweight (normally fewer than 20 bytes) and allow for one device to share data in realtime with other devices without requiring a large amount of bandwidth on each device. Furthermore, the server is able to process and route these commands quickly, given the non-blocking I/O model of NodeJs [18].

4.2) Synchronizing File Data

Nimbus synchronizes user files on a different server and protocol than the realtime NodeJS server. This is to ensure that user commands are relayed as quickly as possible across platforms and are not blocked by CPU-intensive file synchronization. Nimbus applications can use the XMLHttpRequest object to send and receive binary data. Rich media objects such as audio and video files are streamed directly from the server to client devices, while images can either be served as blob: URLs or cached locally (figure 13).

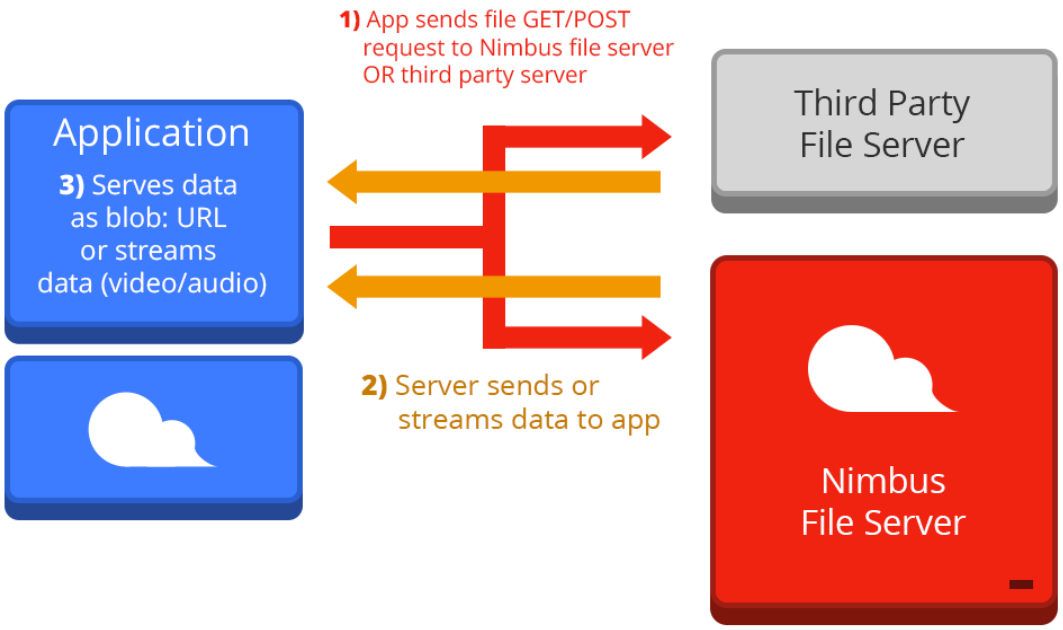

Figure 13: Example pipeline for requesting files from Nimbus server or Third Party server (for example, youtube)

4.3) High-Performance UI and Graphics

Although Nimbus is rendered using web technology, our goal was to ensure that the UI would render components with the same speed and fidelity as a native shell. In order to maintain a high level of performance throughout the Nimbus GUI, we take full advantage of hardware accelerated CCS3 3D transformations on mobile and tablet devices, as well as hardware accelerated CSS3 2D and 3D transformations on desktop devices [13], allowing for incredibly smooth animations on all devices. In addition, the Nimbus UI utilizes

advanced effects such as alpha transparency and drop shadows, rendering an interface that offers near-native levels of performance and graphical fidelity.

4.4) Security Considerations

Because Nimbus is capable of transmitting sensitive user interaction data and file data across the internet, several security measures have been implemented in order to minimize unwanted access to user data, and to prevent malicious code from executing on the user's device. The prototype implementation of Nimbus therefore fully complies with the Chrome Packaged App Content Security Policy (CSP) [3], which prevents inline scripts and external resources (except for video and audio resources) from being directly embedded within any part of the application.

Nimbus also only synchronizes files and command data with known, whitelisted servers that are part of the Nimbus server ecosystem, preventing man-in-the-middle attacks where hackers might impersonate a Nimbus websocket or file server in an attempt to gain privileged user information. The websocket server also checks request origins and user ids and passwords to ensure that commands are only coming from a whitelisted set of domains and not forged requests coming from a 3rd party. This serves to protect users from hijacking attempts where hackers might attempt to gain control of a user's device through a forged connected device.

5) Design Process & Evaluation

Nimbus has been tested in limited capacities for its usability. The bulk of user tests were focused on the cross-device experiences afforded by Nimbus, as this is the core value proposition of Nimbus over competing products.

Initial Tests

Initial tests were conducted on mock prototype software that emulated a small operating system environment on a Windows PC and an iPhone that featured a simple text editor program, which allowed users to create and save files on one device, and open the same file on another device. In these initial tests, users commented that the cross-platform experience was very similar to apps such as iAnnotate or Evernote, which automatically synchronize file data across platforms using existing file synchronization services such as Box. Ultimately, users felt that the value added by file synchronization was insufficient for them to use Nimbus as a solution for cross-platform applications.

Addition of Realtime "Beaming" and Remote Control features

Based on these initial user tests, we identified that a large portion of the user experience was interrupted when users moved from the smartphone text editor to the desktop editor. Users spent a large amount of time launching the application and searching for the corresponding text file to open. As suggested by the strengths of direct manipulation interfaces [9], we simplified this complex user process of launching and synchronizing applications across platforms into two simple user actions. We realized that the key to creating a seamless user experience across devices was to allow for users to launch and "beam" applications from one device to another (figure 15). This "beaming" action allows users to send an application in its current state from one device to another device. In addition, we chose to make the cross-platform experience even more compelling by allowing users to use their device as a remote control.

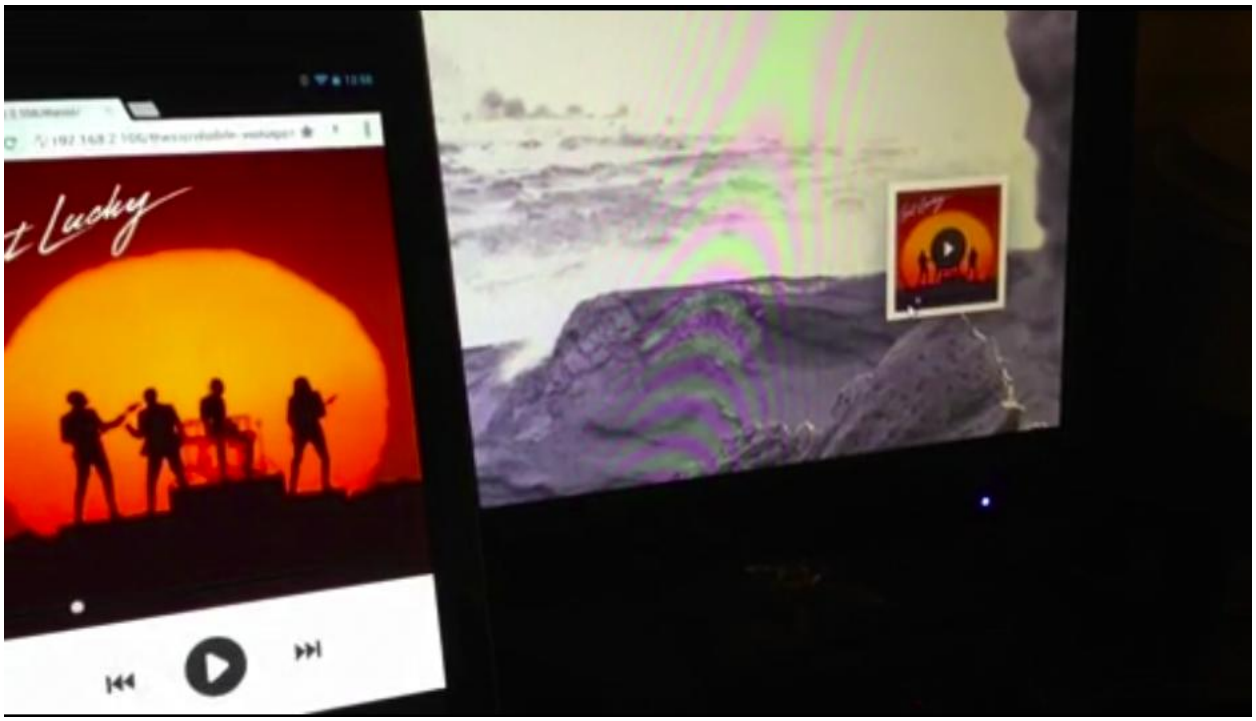

Figure 14: Image of Nimbus remote control feature test

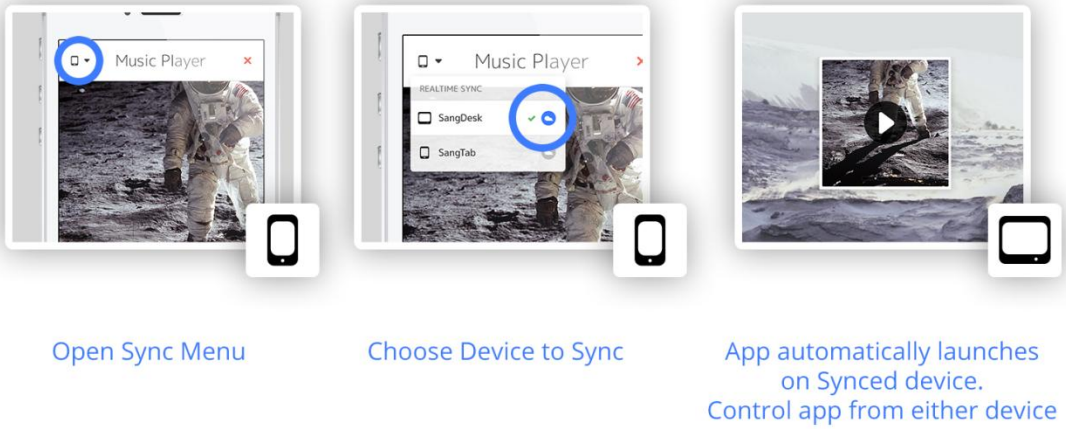

Figure 15: Example app beaming scenario.
Apps that you beam to another device automatically sync their data,
so you can control the app from either device

Evaluation

We tested these beaming and remote control features on our second major prototype, which featured a photo viewing application, video application, presentation application, and music application. Test users felt that the addition of these realtime features created much more compelling and seamless interactions across devices. Users also commented that the app beaming mechanic created a great workflow across devices, where users could start listening to music on their smartphone, and move seamlessly their desktop, and back to their mobile device. Some users felt that it was at times confusing to tell what applications were running on which devices. Users were also worried that devices could become desynchronized and felt that with three or more devices, control over an application could be confusing. As a result we limited control over applications to just two devices and introduced a global popup UI element that shows all devices that are synchronized to the current application. We also added a "global sync" feature, which allowed users to synchronize all actions (app launch, app close, and all in-app actions) across devices in the "global sync" array.

6) Limitations and Future Considerations

Although web technologies have indeed advanced considerably over the past few years, the HTML5 API is still relatively feature-sparse. For instance, low-level access to devices such as the GPU, network controllers, sound card, and battery are severely limited or nonexistant. Future studies may investigate whether a native wrapper for Nimbus could allow for low-level access to hardware devices, and explore the kinds of experiences that could be possible if devices could share hardware resources as well as data.

Another limitation is performance on older devices. HTML5 performance can vary greatly across devices and while most current generation devices do not suffer any serious performance issues while running Nimbus, even devices such as the iPhone 4 or Google Nexus One that are a few generations old do not support all of the advanced websocket features in Nimbus and often have serious performance issues. In addition, Nimbus relies heavily on consistent network quality and low latency, especially for its realtime application data synchronization, and poor network connectivity severely limits the synchronization capabilities of the shell.

Future work should focus on gracefully degrading app performance as network connectivity is reduced. It is also important to consider how the application will scale with a large userbase. Future studies could investigate the optimal strategy for implementing Nimbus on a large scale - whether to adopt a centralized system where all customers using Nimbus have their data piped through a central cluster of data centers, or whether to allow customers to install Nimbus on their own personal clouds. The current UI framework could also be expanded to include more default items, such as sliders, progressbars, radio forms, dropdown menus, and other extended UI elements. Finally, it would be very interesting to explore Nimbus as a "public" operating system, where many users in an institution could be connected to the same Nimbus session and share data and application control with each other across applications and devices in realtime.

7) Conclusion

We presented the design, prototype implementation, and evaluation of Nimbus, an operating system shell and several example applications built on web technologies that aim to demonstrate compelling cross-platform and cross-device OS experiences. Although one could argue that Nimbus just provides a simple wrapper for developers to create pre-existing live sync applications such as Google Documents or Apple Airplay, we believe that Nimbus is an important development because it enables and encourages designers and developers to start creating cross-device applications that provide users with features and experiences that could not have been possible on just one device. In addition, we believe that Nimbus provides an incredible value proposition for increasing number of users who own more than one device and are dissatisfied with the current disjointed and cumbersome state of connectivity across their myriad of devices. Nimbus has great implications in the future of not only OS design, but also networked applications. It takes a unique approach to designing an operating system shell that allows developers to easily create multi-device applications that provide seamless experiences for users across all of their devices.

8) Acknowledgement

I would like to thank everyone at the PF user interface community for their support, critique, and insights.

I would also like to extend a heartfelt thanks to my advisor Lorie Loeb, who supported me throughout this project and through my entire Dartmouth career. I truly believe that I would not have pursued my passion of UI design if not for her help and guidance throughout my college years.

9) References

[1] "Adobe Creative Cloud / Features : Tools and Services." *Adobe Creative Cloud*. Adobe, n.d. Web. 15 May 2013.

[2] "Apple - AirPlay - Play Content from IOS Devices on Apple TV." *Apple.com*. Apple, n.d. Web. 15 May 2013.

[3] "Comply with CSP." *Chrome Apps*. Google, n.d. Web. 15 May 2013. <http://developer.chrome.com/apps/app_csp.html>.

[4] Dasgupta, P., R. J. LeBlanc, Jr., and W. F. Appelbe. "The Clouds Distributed Operating System: Functional Description, Implementation Details and Related Work." *Distributed Computing Systems, 1988., 8th International Conference on* (1988): 2-9. Print.

[5] "Dropbox - Features." *Dropbox*. Dropbox, n.d. Web. 15 May 2013.

[6] Faybishenko, Yaroslav. Secure Platform Independent Cross-platform Remote Execution Computer System and Method. Yaroslav Faybishenko, assignee. Patent US5757925 A. 25 July 1996. Print.

[7] Gill, Phillip. "The First Cloud OS." *At Oracle: News*. Oracle, Jan. 2012. Web. 15 May 2013.

[8] Hickson, Ian. "HTML5 Web Messaging." *W3C Candidate Recommendation 01 May 2012*. W3C, n.d. Web. 15 May 2013. <http://www.w3.org/TR/webmessaging/>.

[9] Hutchins, Edwin, James Hollan, and Donald Norman. "Direct Manipulation Interfaces." *Human-Computer Interaction* 1.4 (1985): 311-38. Print.

[10] International Telecommunications Union. "Infrastructure Data." *The World Bank*. The World Bank, n.d. Web. 15 May 2013.

[11] "Introducing Jolidrive: Get Your Life Together." *Jolicloud Blog*. N.p., n.d. Web. 15 May 2013. <http://www.jolicloud.com/blog/2013/03/12/introducing-jolidrive-get-your-life-together/>.

[12] "Introducing Xbox SmartGlass." *Xbox*. Microsoft, n.d. Web. 15 May 2013.

[13] Jackson, Dean, Edward O'Connor, Dirk Schulze, and Aryeh Gregor. "CSS Transforms." *W3C Working Draft 11 September 2012*. Ed. Simon Fraser. W3C, n.d. Web. 15 May 2013. <http://www.w3.org/TR/css3-transforms/>.

[14] "JoliOS." *Jolicloud*. Jolicloud, n.d. Web. 15 May 2013.

[15] Kane, Shaun K., Amy K. Karlson, Brian R. Meyers, Paul Johns, Andy Jacobs, and Greg Smith. "Exploring Cross-Device Web Use on PCs and Mobile Devices." *Human-Computer Interaction – INTERACT 2009 Lecture Notes in Computer Science* 5726 (2009): 722-35. Print.

[16] "Kijjaa." *Kijjaa*. KIJJAA, Ltd, n.d. Web. 15 May 2013.

[17] MARCOTTE, ETHAN. "Responsive Web Design An A List Apart Article." *A List Apart*. N.p., 25 May 2010. Web. 15 May 2013. <http://alistapart.com/article/responsive-web-design>.
[18] "Node's Goal Is to Provide an Easy Way to Build Scalable Network Programs." *Node.js*. Joyent, Inc, n.d. Web. 15 May 2013. <http://nodejs.org/about/>.
[19] Norman, Donald A., and Stephen W. Draper. *User Centered System Design: New Perspectives on Human-computer Interaction*. Hillsdale, NJ: L. Erlbaum Associates, 1986. Print.
[20] "Pew Research Center's Internet & American Life Project." *Device Ownership*. Pew Research Center, n.d. Web. 15 May 2013.
[21] "Product Features." *Google Drive*. Google, n.d. Web. 15 May 2013.
[22] "Product Features Overview." *Box.com*. Box, Inc., n.d. Web. 15 May 2013.
[23] Schmidt, Dominik, Julian Seifert, Enrico Rukzio, and Hans Gellersen. "A Cross-device Interaction Style for Mobiles and Surfaces." *DIS '12 Proceedings of the Designing Interactive Systems Conference* (2012): 318-27. Print.
[24] Sengupta, Caesar, and Matt Papakipos. "Google Official Blog." *: Releasing the Chromium OS Open Source Project*. Google, n.d. Web. 15 May 2013.
[25] "What Are Packaged Apps?" *Chrome Apps*. Google, n.d. Web. 15 May 2013. <http://developer.chrome.com/apps/about_apps.html>.