Live Migration of JavaScript Web Apps

James Lo
Department of Computer Science
University of British Columbia
Vancouver, Canada
tklo@cs.ubc.ca

Eric Wohlstadter
Department of Computer Science
University of British Columbia
Vancouver, Canada
wohlstad@cs.ubc.ca

Ali Mesbah
Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, Canada
amesbah@ece.ubc.ca

ABSTRACT

Due to the increasing complexity of web applications and emerging HTML5 standards, a large amount of runtime state is created and managed in the user's browser. While such complexity is desirable for user experience, it makes it hard for developers to implement mechanisms that provide users ubiquitous access to the data they create during application use. This work showcases IMAGEN, our implemented platform for browser session migration of JavaScript-based web applications. Session migration is the act of transferring a session between browsers at runtime. Without burden to developers, IMAGEN allows users to create a snapshot image that captures the runtime state needed to resume the session elsewhere. Our approach works completely in the JavaScript layer and we demonstrate that snapshots can be transferred between different browser vendors and hardware devices. The demo will illustrate our system's performance and interoperability using two HTML5 apps, four different browsers and three different devices.

Categories and Subject Descriptors

D.3.2 [**Software**]: Programming Languages—*JavaScript*; E.2 [**Data**]: Data Storage Representations—*Object Representation*

Keywords

JavaScript, session migration, HTML5, JSON, DOM

1. INTRODUCTION

With the evolution of web technologies, browsers, and HTML5 [3] a great deal of application state is being offloaded to the client-side. In order to achieve more responsive apps, JavaScript is increasingly used to incrementally mutate the Document Object Model (DOM) in the browser to represent a state change, without requiring a URL change. Additionally, with new HTML5 APIs apps can feature advanced graphics, animation, audio, and video. Therefore, capturing and migrating a particular state of an app is not as simple as saving and loading a URL any longer. It requires developers to manually implement code for persisting the transient browser state (i.e. state that normally would be lost once a user closes a browser tab).

WWW 2013, May 13-17, 2013, Rio de Janeiro, Brazil
ACM 978-1-4503-2038-2/13/05.

While some libraries and APIs [2, 8] provide basic support for object persistence, developers are still obliged to register and track individual objects programmatically, which can be tedious and error-prone. Since persistence is well-known to be a crosscutting concern [9, 10], adding it to existing code is difficult because it requires changes scattered across various modules.

In this paper, we investigate the use of *session migration* to address this problem. Session migration is the act of transferring a session between browsers, possibly on different platforms, at runtime. We demonstrate a novel technique and tool, called IMAGEN,[1] for migrating client-side session state of web apps across different browsers and devices, which is publicly available. Our technique enables end-users to seamlessly capture the runtime client-side browser state at a desired instance, and later restore that state in a different browser and continue using the app from there. While there is some previous work on Web app migration [1], that system only supports persistence of simple objects, and not full applications which include Javascript function closures, event-handlers, and HTML5 media objects. Thus that work is not applicable to the applications that we showcase in this demo.

IMAGEN works through a combination of novel JavaScript transformations. Such transformations can be applied in two different ways: developer-initiated or user-initiated. The developer-initiated transformation is applied by a software developer to their code prior to application deployment. Alternatively, end-users can enable the transformation themselves by using a provided transformation HTTP proxy. Either way, no extra coding is required to achieve migration of sessions.

We will demonstrate both options in our demo. To the best of our knowledge, IMAGEN is the first tool to support client-side session migration of Javascript/HTML5 apps without requiring modifications to a web browser or operating system. An online video provides a demonstration of IMAGEN and the type of application migration that will be shown in the live demo.[2]

2. IMAGEN DESIGN

We present an overview of some challenges in session migration (Section 2.1), followed by a high-level architectural overview of the components involved in our approach (Section 2.2).

[1] IMAGEN means *image* in Spanish.
[2] `http://www.cs.ubc.ca/~wohlstad/imagenVideo.html`

```
1 //Attempt (and fail) to serialize the
2 //user's session by JSONizing 'window'
3 var snapshot = JSON.stringify(window);
4
5 //Attempt (and fail) to unserialize a
6 //user's session by assigning
7 //parsed string to 'window'
8 window = JSON.parse(snapshot);
```

Listing 1: Essence of Snapshot Imaging. This code fails horribly on regular apps but becomes possible using Imagen.

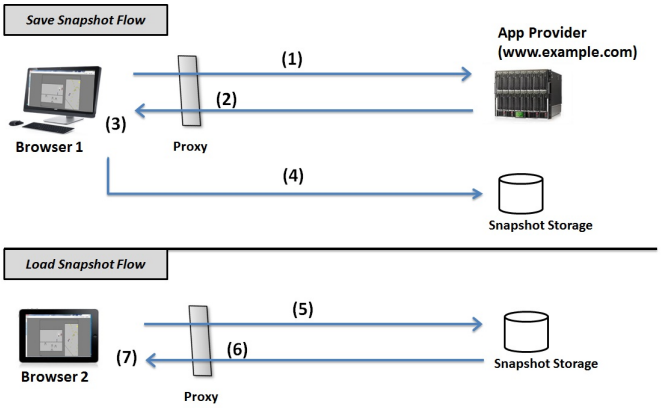

Figure 1: Imagen Architecture: (top) Starting up an app and saving a snapshot; (bottom) Loading a previously saved snapshot to a different device.

2.1 Challenges

In order to explain the technical challenges for session migration, we illustrate using an incorrect strawman implementation of saving/loading a snapshot image of some browser session (shown in Listing 1). In JavaScript, the global variable `window` provides a context from which both native browser APIs (such as the DOM) and application-specific state (in the form of JavaScript objects) can be accessed by programmers. JSON [5] is the popular serialization format of JavaScript and can be used to serialize (`JSON.stringify`) and unserialize objects (`JSON.parse`). Thus it would seem reasonable that to migrate a user's session, one might be able to simply `stringify` the whole `window` object (line 3). Ideally, this would return a string capturing all runtime state needed for migration. Then later, on another browser, the stringified snapshot could be parsed back into `window` (line 8) and the user's session would resume. Unfortunately, this will not work in practice.

Capturing a snapshot for migration is much more challenging, for a number of reasons:

1. *Function Closures.* In addition to objects, JavaScript state includes function instances called *function closures*. This kind of function/object hybrid is not easy to serialize.
2. *Event-handler state.* Event-handlers are the driving force of execution in JavaScript. They create a schedule of activity that is not supported by existing serialization mechanisms.
3. *HTML5 rich-media objects.* Modern web applications make use of rich-media objects from the HTML5 standard which have unique serialization requirements.

All of these problems need to be solved without introducing burden on the developer or end-user, in particular IMAGEN is designed to be:

1. *Generic and Interoperable*: End users should be able to migrate a variety of apps and should have the freedom to use a snapshot in any device of their choice.
2. *Automatic*: Enabling session migration should not require additional coding for developers and only minor setup configuration.
3. *Efficient and Scalable*: End users should experience the same level of interactivity as the original app. This means IMAGEN's overhead to the app's execution should be minimal.

Due to space considerations, we do not describe here the specific source-code transformations that we have implemented to meet these challenges. The interested reader is referred to our recently published paper [7], a preprint copy is available online[3].

2.2 Architectural Overview

We describe the components involved in the migration of a running app, as depicted in Figure 1. The flow of this architecture will organize the steps for live migration that will be shown in the demo.

The figure is divided into two: the top half (*Save Snapshot Flow*) and the bottom half (*Load Snapshot Flow*). First, a user starts to load an app in their browser and execute it as usual (1); e.g. entering the URL or navigating from a search page. In order to make migration possible, the JavaScript source code of an application is transformed by our system and instrumented with additional code. This can be done by the developers using a source code processor, prior to deployment. Alternatively, instrumentation can be transparently injected by an end-user by making use of a provided HTTP proxy which runs on the user's own machine (shown as (2) in the figure). Our technique supports both, and either way, no manual changes to the application code is necessary.

Ideally, there should be no noticeable change in the app's behavior after instrumentation. Using a simple GUI button (added to the bottom of each web page by the instrumentation) a user can take a snapshot at any point during execution (3). This snapshot is then saved to a secondary storage (4), either on a remote web service (*Snapshot Storage* in the figure), or on the user's local drive. Either way, the user is provided with a URL, which can be used to retrieve and load the snapshot.

Sometime later, the user opens a new browser, which can be on a different device. In the example figure, the user migrates the application from their desktop to a tablet (e.g., iPad). The user then enters the previously given URL in the new browser (5). If instrumentation was provided by the application developer, step (6) is not necessary. Otherwise at (6), the proxy redirects the user's browser to the original URL where the snapshot was taken. However, rather than returning the content at that URL, it returns the saved snapshot instead. This step allows the restored app to run in the

[3] `http://www.cs.ubc.ca/~wohlstad/imagen.html`

same browser security domain as the original application. After the app is loaded into the new browser, it seamlessly continues exactly where it had left off (7).

3. TOOL IMPLEMENTATION

Our implementation consists of two main parts: a source code transformation for JavaScript (written in Java) and a library of JavaScript functions.

The Java-based transformer is built on top of Mozilla's Rhino open-source project. It provides us JavaScript in the form of abstract syntax trees (ASTs) which we then analyze and transform with our own code. Our own Java code is 6,923 lines. The transformer can be run by a developer through the command-line to transform their JavaScript code. Alternatively, the transformer can be hosted in an HTTP proxy by any user.

Our JavaScript library is injected into a web app by the transformer (by inserting a `<script>` tag into the DOM). The library performs most of the work of saving and loading snapshots. It retrieves information that was stashed away by the instrumentation. This information is combined with other data available through `window` to build a JSON formatted snapshot file consisting of: function closures, plain JavaScript objects, event-handlers, media objects, the DOM, and optionally any cookies for the web app. To JSONize the DOM we make use an existing library called JsonML [6]. Our JavaScript library is 5,028 lines of code.

IMAGEN is open source and publicly available.[4] An online video provides a demonstration of IMAGEN and the type of application migration that will be shown in the live demo.[5] For more technical details of the approach and implementation, we refer the interested reader to our recent paper appearing in WWW 2013 [7].

4. LIVE DEMONSTRATION

In our demo, we will showcase the front-end, instrumentation proxy, and JSON storage aspects of IMAGEN.

We have tested IMAGEN thoroughly [7] on at least five feature-rich and interactive HTML5 applications, four of the most popular browsers (Chrome, Internet Explorer, Firefox, and Safari) and three types of devices (Windows PC, Mac, and iPad). During our demo, we will demonstrate a cross-browser session migration for two real-world web applications, namely Robots Are People Too (RAPT) and Color Piano. Specifically, we will migrate

1. A live session of playing the RAPT game from Internet Explorer on PC to Safari on iPad, and
2. A live session of using Color Piano from Chrome on Mac to Firefox on PC.

Both RAPT and ColorPiano make intensive use of client-side JavaScript for animation, media and maintaining application state. Through illustrating the session migration of these two applications, our goal is to highlight the transparency, efficiency, and interoperability of IMAGEN.

RAPT [11] is a performance intensive application. It is a side-scrolling two player platform game and features different challenges, drones, and rewards. It invites gamers to invest sufficient time and effort to finish levels and make progress. A gamer may want to persist or migrate her session of gameplay for any of the following reasons:

- *Strategy*: She perceives a risky move ahead and wants to seamlessly try again when it fails without repeating previous effort.
- *Time*: She has to work on something else and wants to close this game completely from the browser. A reason could be to free up some system resources since the game can be performance intensive.
- *Location*: She is going somewhere and wants to continue the game at another location or on another device.

The current version of RAPT does not provide a feature for saving progress during a game. By using IMAGEN, end users can persist and migrate such game state without requiring developers to provide any additional coding.

When migrating a session of RAPT, we will first show briefly our proxy that instruments the JavaScript source code. End users can host the proxy on their own machine, or on another server such as Amazon EC2. Then, all they have to do is just to specify the proxy's IP in any device's Internet Settings. Similarly, deploying IMAGEN is flexible because developers can host the proxy on their own application servers, instead of relying on end users.

To demonstrate the difference between the original and the instrumented application, we will first play the game and show that there is no feature to save your game in the original version. Next, we will run the version instrumented by our tool. We will highlight how IMAGEN adds an HTML menu bar on the application's DOM which allows users to *Save* and *Restore* their progress at any time (as in Figure 2). Next we will show how the responsiveness of the application remains nearly identical to the original.

To illustrate interoperability, we will keep the saved version running in Internet Explorer, and then we will open a saved snapshot in Safari. The focus here will be to show how the application state resumes in another browser within a few seconds.

To demonstrate some internals of our system, we will examine how the snapshot is saved as JSON data. We will pick a random level of the game and start playing it. At some point we will save the session, and then open the saved JSON file so the audience will see how the data structure of a snapshot is organized.

Color Piano [4] is a piano teaching animation. As a song is played, the app animates a slider of sheet-music notes and animates keys playing each note. Session migration would be useful here as it allows piano students to pause and resume at any particular positions in any songs they are practicing. ColorPiano does not include this feature in its current implementation.

Through demonstrating a session migration on ColorPiano, our goal is to illustrate how well IMAGEN works on HTML5 media objects such as HTML5 `<canvas>` and `<audio>`. The demo will also show that the music and animation are synchronized after migration, validating our support of JavaScript event-handling.

5. CONCLUDING REMARKS

We have presented a generic solution and tool, called IMAGEN, for session migration, which works in the JavaScript layer and also targets some HTML5 APIs. However there

[4] `http://www.cs.ubc.ca/~wohlstad/imagen.html`
[5] `http://www.cs.ubc.ca/~wohlstad/imagenVideo.html`

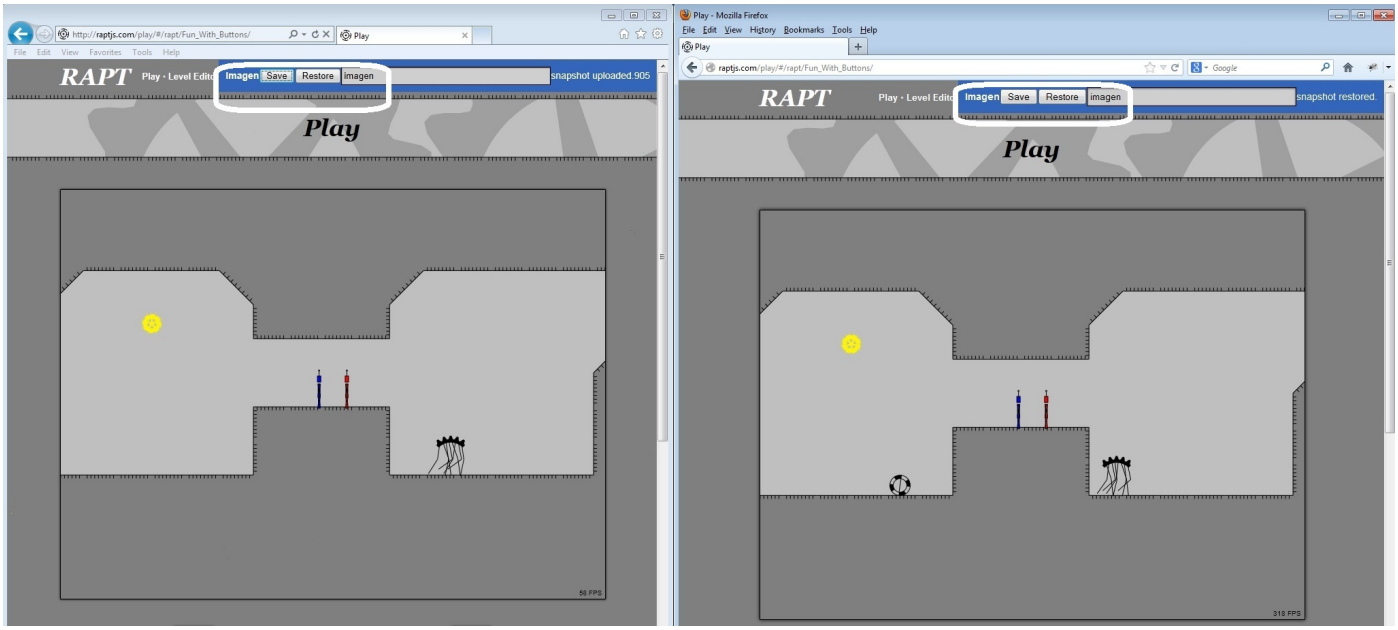

Figure 2: Imagen on Robots Are People Too (RAPT): A session of playing RAPT in Internet Explorer (left); the game state migrated to Firefox (right). The game state was saved in Internet Explorer (left) at some time t_0 and then this browser session was left running for n seconds. Then the previously saved state was loaded in Firefox (right). Note that the images are not identical because the game state on the left has moved forward in time and is now at the state corresponding to $t_0 + n$. On the other hand, the state loaded on the right correponds back to time t_0. The reader is encouraged to view the online video to perceive the effect in real-time. Imagen instruments RAPT's JavaScript and inserts a toolbar for end users to easily save and restore browsing sessions using a button (annotated by an ellipse on the top of the browser windows).

are still some APIs that are not covered by our current implementation, such as WebWorkers and GeoLocation. WebWorkers provides support for background computational tasks but since each worker has an isolated memory and cannot respond to UI events, our assumptions made in our implementation that depend on a single-threaded model still apply. While additional effort will be required to enable support, the fact that such APIs are being standardized should help making migration support feasible.

While we have not focused on debugging in this demo, it may be possible to use Imagen so that when an end user requires urgent assistance, she can instantly duplicate and share a session snapshot with developers who could inspect the state in a web developer tool such as FireBug. We plan to investigate such debugging support also in our future work.

6. REFERENCES

[1] F. Bellucci, G. Ghiani, F. Paternò, and C. Santoro. Engineering JavaScript state persistence of web applications migrating across multiple devices. In *Proc. of the Symposium on Engineering Interactive Computing Systems*, 2011.

[2] E. Benson, A. Marcus, D. R. Karger, and S. Madden. Sync kit: a persistent client-side database caching toolkit for data intensive websites. In *Proc. of WWW*, 2010.

[3] R. Berjon, T. Leithead, E. D. Navara, E. O'Connor, and S. Pfeiffer. W3C HTML5, 2012. http://dev.w3.org/html5/spec/.

[4] M. Deal. Colorpiano. `http://mudcu.be/piano/`.

[5] Introducing JSON. `http://www.json.org/`.

[6] JsonML. JSON Markup Language. http://www.jsonml.org.

[7] J. Lo, E. Wohlstadter, and A. Mesbah. Imagen: Runtime migration of browser sessions for JavaScript web applications. In *Proc. of International World Wide Web Conference (WWW)*, 2013.

[8] PersistenceJS. `http://persistencejs.org`.

[9] A. Rashid and R. Chitchyan. Persistence as an aspect. In *Proc. of Aspect-oriented software development*, pages 120–129, 2003.

[10] S. Soares, E. Laureano, and P. Borba. Implementing distribution and persistence aspects with AspectJ. In *Proc. of Object-oriented programming, systems, languages, and applications*, pages 174–190, 2002.

[11] E. Wallace, J. Ardini, K. Gishen, and P. Kernfeld. Robots are people too. `http://raptjs.com/`.