# ClientJS: Migrating Java UI Clients to HTML 5 and JavaScript - An Experience Report

Udo Borkowski
abego Software GmbH
Aachen, Germany
Email: ub@abego-software.de

Patrick Muchmore
University of Southern California
Los Angeles, CA, USA
Email: muchmore@usc.edu

**Abstract:** *We were tasked with migrating an existing web based Java applet User Interface (UI) to a UI solution based on HTML 5 and JavaScript. This new UI addresses problems with multiple browsers, Java Runtime Environments (JREs), and operating system incompatibilities and allows for display on mobile devices that do not support JRE. This report briefly describes the approach we used for the ClientJS project, and summarizes some of the complications that were encountered along the way.*

## 1 Background and Motivation

CodoniX established in 1995 has a web based Electronic Health Record (EHR) which has used Java applet technology in its user interface for the last 12 years [1]. At the server side all of the medical knowledge is stored as XML files in a master Medical Knowledge base (KB) and represents >200,000 hours of clinical development. Each self-contained xml file or "Medical Problem" (MP) contains all of the clinical logic as well as graphical information to be rendered by the Java applet in a compatible browser.

When new modules were added to the system they employed the UI technology current at the time. As extensions to the application were made over a period of many years the resulting code base contains a mix of AWT and Swing based modules.

The original system was hospital based, but has grown to support physician clinics throughout the United States. In 2001, when the system was first designed there was a single major browser (IE) used by virtually all hospitals who uniformly used MS Windows. Over time that landscape has changed dramatically. There are now at least 4 major browsers constantly adding patches which often require different versions of JRE, add to that the overlay of stricter security considerations and certain business decisions by operating system and device vendors and the net effect has made using Java applets in web browsers progressively more difficult, or even impossible.

Because of the huge investment in the medical KB, any replacement for the applet based UI had to be 100% compatible with the existing clinical KB design. An analysis of current end user technologies revealed HTML 5 and JavaScript (JS) to be the best candidates for use as the base of the alternative UI, leading to the ClientJS project.

## 2 Prototyping

To better judge the feasibility of a pure HTML 5 / JavaScript solution we re-implemented a small vertical slice through one UI module using only HTML 5 and JS. One goal was to ensure the technologies worked well with the server backend. In addition, the experience acquired during this phase led to a better understanding of the technologies employed and the development environment they would require.

The prototypical re-implementation was successful. However, developing the prototype revealed some risks in using these technologies, e.g. the lack of a static type system in JS. It was also clear that manually re-implementing the legacy Java code in JS would be both time consuming (expensive) and error prone.

Moreover, it became clear that, at least initially, some non-clinical features outside of the KB would not be available in ClientJS. For example, the Java UI supports connectivity to hardware components such as signature pads and scanners for which no comparable browser based solution is available. This meant the ClientJS project could not fully replace the Java UI. Rather, both UIs would co-exist, at least for several years, and both UIs need to be maintained in parallel.

## 3 Java to JavaScript

The expected high cost of a manual re-write of the UI in JS, coupled with the need to maintain both the Java and ClientJS UI, led to the decision to reuse as much code as possible from the legacy Java UI in ClientJS. Ideally, there would be a single (Java) source base for the majority of both the Java and ClientJS UIs.

To make this work we looked for a "Java to JavaScript" compiler. Most of the products we found did not fit our needs, variously because they would have required us to also re-implement our Java UI code, did not support the full Java language, or did not seem to be mature enough.

We finally ended up with Java2Script (J2S) [2], an open source project providing an Eclipse plug-in that compiles Java source into JS. In doing so it emulates essential Java language functionality, such as classes, inheritance, and method overloading. Java2Script also includes a re-implementation of parts of the Java Runtime Environment. By design J2S makes it easy to mix Java and JavaScript code, and it even supports embedding JavaScript in Java source files.

## 4 Preprocessing

As previously mentioned certain Java modules, such as those providing specialized hardware support, do not presently have a JS/HTML equivalent. To accommodate these cases the legacy Java source code was annotated with preprocessor statements, similar to those used in C, to indicate which code segments could not be compiled into JS. In addition to marking code not applicable for ClientJS, we used the same approach to exclude Java code for 'non-essential' features. This reduced time-to-market for a JS/HTML based UI sufficient for most uses, and the missing features can be added in subsequent releases.

The preprocessor [3] reads the annotated legacy Java files and creates read-only Java files for the ClientJS project. The generated files are never edited manually; rather, any necessary editing occurs in the legacy source files. In addition to the preprocessor generated files ClientJS also contains manually edited source files of various languages, mainly consisting of Java, JavaScript, HTML, or CSS.

## 5 Runtime Library

Java2Script's runtime library does not cover all features of the standard Java Runtime Environment (JRE). In particular, support for the UI technologies used by CodoniX (AWT and Swing) is missing. Providing an implementation for these frameworks in the ClientJS context was one of the biggest challenges. Parts of the missing libraries were filled-in by reusing code from Apache Harmony [4], a project intended to develop an open source implementation of the JRE. Given the maturity of Java development tools, such as the Eclipse IDE, we typically favored Java over JS when writing our own code. When JS was used we tried to leverage mature third party JS libraries like jQuery [5]. We have only implemented those JRE features used by our system as developing a generally usable JRE would have increased the costs and time-to-market significantly.

## 6 Extra Benefits

Translating the AWT and Swing based UIs into HTML 5 and JavaScript has yielded benefits beyond the initial project goals. For example, since both AWT and Swing components are ultimately rendered via HTML the appearance of modules developed with the two Java UIs has been significantly unified.

Certain features can also be implemented far more easily in the new UI. For example, ‘Meaningful Use’ [6] certification depended on providing accessibility features that were easily implemented using HTML 5.

## 7 Problems and Challenges

While working on ClientJS we ran into problems in various areas. Here are some examples:

**Java2Script:** Although Java2Script is a mature project we discovered several bugs, both in the compiler and the runtime. The developers of J2S often fixed these issues very quickly; however, some issues remain open. Working around these bugs has often involved changing the legacy Java code, something we hoped to avoid. Also, Java2Script only functions as an Eclipse plug-in, and the lack of a standalone compiler made seamless integration into our normal build process impossible.

**Development Environment:** Eclipse was used as the main IDE; however, as the code was ultimately compiled to JavaScript debugging was often done using browser based tools. At times we experienced frequent crashes in the browsers, often with little information as to the cause. This was particularly true at the beginning of the project, and fortunately this situation has improved over time.

**Visual Appearance:** one particular module (the “Graphics“) of the existing Java applets uses pixel-based absolute positioning to layout the screen. Differences in how Java and HTML 5 treat borders, along with the use of proprietary fonts in the applet, made it a significant challenge to create an exact replica of the applet’s appearance using HTML.

**Multi Threading:** web browsers run JavaScript in a single thread which made it difficult to replicate applet features, such as blocking/modal dialogs, that are based on a multi-threaded model. In the JS/HTML world only non-blocking dialogs (with callbacks) are supported. To resolve this issue we had to change some legacy Java code.

## 8 Perspective

Performance of the ClientJS on desktop and laptop devices has been comparable to, and at times better than, the Java applet version. However, on mobile devices rendering a complicated screen may cause a noticeable decrease in performance which we suspect is related to less processing power. While some UI forms adapt well to varying screen sizes, others were designed with a desktop in mind and may be difficult to use on a smaller screen. For certain applications of the CodoniX system, such as communicating directly with patients, using a mobile device may be a natural choice. As such, improved support for mobile devices is one of the main goals for future versions of ClientJS.

### References


[1] CodoniX, Potomac, MD, USA www.codonix.com
[2] Java2Script. http://j2s.sourceforge.net
[3] jpp. https://github.com/abego/jpp
[4] Apache Harmony. http://harmony.apache.org
[5] jQuery. http://jquery.com
[6] Meaningful Use. http://www.healthit.gov/providers-professionals/meaningful-use-definition-objectives