Case Study: Making the Move to AJAX

What a software-as-a-service provider learned from using an AJAX framework for RIA development

Jeff Norwalk, Consultant

Small start-up companies often face a bewildering array of technical choices: how to deliver their application, what language to use, whether to employ existing components (either commercial or open source) or roll their own... and the list goes on. What's more, the decisions surrounding these choices typically need to be made quickly. This case study offers a realistic representation of the sorts of challenges a young start-up company faces when working to deploy a product on the Web. As with many startups, this is also a story that does not have a happy ending.

As fate would have it, SalesCentrix, the company at the center of this case study, was a venture that did not succeed. Work on the product discussed in this case study, AccountDynamics, was completed in 2006 while SalesCentrix was still a going concern. The company failed not long after that. Subsequently, the intellectual property rights to AccountDynamics were acquired by another fledgling organization in Vancouver, B.C. called Realm Dynamics, which ran into funding difficulties of its own in 2008.

Even though these organizations did not succeed, the RIA (rich Internet application) technology discussed in this case study had been deemed "successful" by SalesCentrix and, subsequently, by Realm Dynamics. Owing to business challenges that had nothing to do with AccountDynamics, however, the feature discussed here was never actually released.

If you, too, have had the pleasure of working for a young start-up firm, this story may strike you as being all too familiar. Our apologies for the unpleasant memories.—ACM Queue

SalesCentrix, a small start-up in Vancouver, realized in 2006 that something would have to be done with its SaaS (software-as-a-service) solution for synchronizing data between QuickBooks and Salesforce to let users take better advantage of the Web. For the most part, the SalesCentrix software served as a means to connect the two popular applications, doing its work seamlessly in the background without the need to deliver much in the way of a user interface. During setup, however, SalesCentrix users had to be able to capture and synchronize a baseline state accurately from their QuickBooks and Salesforce databases.

To facilitate this, a highly interactive desktop-like user interface, complete with a spreadsheet-like workspace—and designed to run primarily on the client side—seemed like the right way to go. The SalesCentrix team believed this capability should be something their customers would be able to employ immediately, without first having to download new versions of enabling software or install additional applications. The decision was soon made that this setup portion of the SalesCentrix application would need to be implemented as an RIA.

The next decision facing SalesCentrix had to do with which RIA technology to employ. In the course of building an earlier product the company's developers had gained experience with Java applets, but they rejected that alternative this time around because of some deployment and upgrade issues they had encountered. They had discovered just how complicated and messy deployments

could get when customers had incompatible versions of Java installed on their browsers. They had also learned how server-side applets tend toward bloat, leading to significant download and start-up times for any clients still using dial-up connections.

Flash, meanwhile, was something the SalesCentrix team considered to be primarily for multimedia presentations. They also recognized that developing for the Flash platform would take time away from development to learn ActionScript. Finally, since SalesCentrix was not a Microsoft shop, Microsoft technology was not even considered.

This essentially narrowed the field of candidates to only JavaScript-based solutions. With that realization, the SalesCentrix team quickly settled on AJAX (Asynchronous JavaScript and XML). Their comfort level with AJAX was quite high, both because of its general popularity with Web developers at the time (early 2006) and because the AJAX-based applications already launched by Google seemed to deliver just the kind of interactivity and performance they were hoping to achieve.

THE DECISIONS

Having settled on AJAX, the SalesCentrix team began to evaluate five development frameworks: Backbase, ClearNova ThinkCAP, Dojo, NextApp Echo2, and TIBCO General Interface. Their primary goal was to find something that would minimize coding in JavaScript and interacting with the DOM (document object model) directly, thus allowing them to steer clear of the client-side complexities that often characterize AJAX. They also believed that, with respect to maintainability, JavaScript was not suitable for a team of programmers working to build a complex system. In particular, JavaScript's lack of type checking and the absence of any means of defining strict interfaces between components would make it hard to build a system that would be sufficiently componentized and decoupled. Consequently, they felt the use of JavaScript would hamper their ability to build and maintain an MVC (model-view-controller) style architecture.

Another important consideration was that the SalesCentrix engineers wanted to avoid the learning curve they would have to climb to gain expertise in JavaScript. A far better option, they felt, would be to find something that included a visual design tool they could use to avoid JavaScript altogether. In evaluating the AJAX frameworks available to them, they also put a priority on easy cross-browser support and vendor stability (that is, a company they could be confident would not end up orphaning them). In terms of the attributes they were looking for, a good widget set, the ability to add their own components, and capabilities for graceful degradation were all especially important to them.

With all this in mind the SalesCentrix team downloaded all five frameworks, went through the hands-on tutorial available for each, and attempted to produce a simple application with each to get a feel for how their environments and processes actually worked. This was the SalesCentrix team's first introduction to AJAX; however, none of the frameworks required them to deal with it at low levels. In fact, they found working with AJAX frameworks to be very much like developing UI (user interface) applets with other frameworks they had used before.

The SalesCentrix team knew they would require tables for the display and manipulation of data. This proved to be an important point of differentiation in evaluating the frameworks, since not all of them included tables as part of their Getting Started programs. Some products, such as ClearNova ThinkCAP, did include tables, however, making it easy for the SalesCentrix engineers to test the capabilities they were looking to incorporate. This almost surely influenced their decision,

and ClearNova ThinkCAP, which offered many of the same features as the other products, ended up being their choice. They selected ThinkCAP not only because of how it dealt with tables, but also because of the company's perceived stability and some additional research indicating that the ClearNova product was the only one of the five frameworks that provided for graceful degradation when a user's browser did not support AJAX.

With that choice made, prototyping with ClearNova ThinkCAP began almost immediately. While part of the objective was to test the technology, the main goal was to rough out a demonstration of how they intended to implement the UI for their application. Accomplishing this didn't require implementing the whole application or hooking it up to a real back end, so the prototype came together quite readily. Better still, the stakeholders liked what they saw of the UI and approved it.

Shortly thereafter, however, GWT (Google Web Toolkit) was released. Although the SalesCentrix team considered ClearNova ThinkCAP to be a good product, they were not entirely comfortable with the prospect of being locked into its proprietary set of UI components. By contrast, GWT was open source, which meant SalesCentrix would not run the risk of getting locked into a particular vendor's product. Furthermore, even though GWT supplied an underlying framework, it offered developers greater design latitude than ClearNova ThinkCAP, including the potential to drill down to JavaScript.

The team then repeated many of the same steps just taken with the ClearNova ThinkCAP framework, working their way through GWT's hands-on tutorial and building a simple application that made use of tables so they could verify that GWT would allow them to accomplish the project ahead of them. Once satisfied, and despite the time and effort already sunk into prototyping with ClearNova ThinkCAP, the SalesCentrix team readily concluded that they would make the switch to GWT.

Beyond the latitude offered by GWT and its inherently nonproprietary nature, the SalesCentrix engineers were impressed that programming with GWT was done entirely in Java—which was not true of any of the other frameworks they considered—with the toolkit then generating the JavaScript equivalents required for each of the major browsers. SalesCentrix was an avid Java shop and thus felt that using Java—a higher-level, object-oriented language—helped (unlike JavaScript) to reduce code complexity and simplify code maintenance. GWT also offered a Swing-like library, which fit in nicely with the engineers' familiarity with Java's AWT (Abstract Window Toolkit) and Swing graphics libraries.

The architectural infrastructure of GWT would also automatically provide for the varying JavaScript requirements presented by different browsers, thus obviating the need for the SalesCentrix engineers to design their own mechanism. What's more, GWT, coded in Java rather than in JavaScript, offered the type of high-level design framework they were looking for—one that would allow them to create and maintain the kind of MVC architecture they were hoping to build.

Another consideration was that, being a Google product, GWT was perceived as coming from a stable and reliable source. The SalesCentrix team also felt that since Google clearly knew the AJAX space, GWT would prove to be both robust and efficient. Still, these were judgments that had to be made largely on faith, since GWT was only in beta release at the time. There really was no way of looking at existing applications built on top of GWT, either inside of or outside of Google.

Given that GWT was still in beta, the SalesCentrix team was fully aware that more issues, including a higher incidence of bugs and performance problems, might arise than might be expected

with a more finished product. Among the other drawbacks, there was no debugging capability at the JavaScript level and GWT's widget set did not include a scrolling-table widget. All such concerns were trumped by Google's prominence and the SalesCentrix engineers' familiarity and comfort with developing in Java. If the price for that familiarity was that they would have to build their own widgets, they were only too willing to put up with that. The lack of a GWT visual design tool (one of SalesCentrix's original criteria) was deemed to be manageable, given the development team's familiarity with Java and comfort level with coding the UI directly. They also liked most aspects of GWT's UI library and were impressed with how GWT handled communications between client and server.

LOOKING BACK

The first affirmation of the decision to go with GWT came with the discovery that it fit into the SalesCentrix build environment fairly painlessly. Ant was already being used as the infrastructure there, making it relatively straightforward to incorporate the GWT build into the existing build process. Also, since the SalesCentrix engineers were building the rest of their application in Java, GWT's Java orientation fit right in. Even though the GWT and Java compilers are different, the engineers found the spirit of building something with GWT to be quite similar to how a Java application is constructed, and thus very familiar.

All in all, their experience with the GWT framework proved to be quite positive. They found it to be stable, and even though they encountered bugs (as anticipated), fixes were made in a timely manner. The UI library allowed them to do what they wanted, and GWT offered a good debugging environment, albeit only at the Java level, which worked fine for dealing with coding errors and other straightforward problems.

In terms of networking, the client-server communications problem was already largely handled, although GWT's serialization/deserialization infrastructure required using a specific type of GWT servlet to implement the data communication facilities. Since SalesCentrix already had some server-side architecture in place to handle authentication, they had to make some adjustments in order to slot in the servlet.

Because GWT did not provide encryption, SalesCentrix used HTTPS for authentication. Beyond that, they had no concerns about AJAX-related security—at least none beyond the usual concerns that come into play with any page-based application.

They also encountered no real problems integrating GWT with the rest of their architecture. They were able to institute the MVC architecture they wanted, describing the experience as being akin to implementing it using a Swing-like application. The only slight wrinkle was that, because of the asynchronous nature of AJAX, they had to write their application such that all calls to the server were asynchronous rather than synchronous.

In the course of development, however, the engineers did find two performance problems with GWT. The first had to do with GWT's lack of a scrolling-table widget. Instead, they composed something that behaved like a scrolling table out of the nonscrolling table and scrolling window supplied with GWT. Nonetheless, GWT's table proved unable to handle even moderate amounts of data, which was quite alarming given the large SalesCentrix data sets.

The solution the SalesCentrix engineers came up with was to create a simple buffering component. The controller in their MVC architecture made calls to this buffer for data, which

was fetched asynchronously from the server and cached. The controller then populated the table incrementally with this data, essentially shuffling it in and out of the table on the fly. In this way they were able to bring over a sizable chunk of data in relatively short order. This was enough so that whenever a user moved up or down a page, the data could be made immediately available from the cache. Once the user moved up or down more than one screen at a time, however, a call had to be made back to the server to retrieve more data.

As work-arounds go, this proved to be reasonably successful. Although the scrolling experienced by users was not as smooth as it would have been with a fully populated table, the delays were quite minimal.

A far more serious performance problem related to serialization/deserialization, which at first was so slow as to render the framework virtually unusable, with 500 objects (400 KB) taking more than a minute to transfer. When the SalesCentrix engineers first experienced this latency, they weren't quite sure what was responsible. Through straightforward "printf debugging" they were quickly able to determine where in the program the delays were occurring. Then, with a test program designed to monitor the data fetch, they observed that the deserialization process was unusually compute-intensive, consuming 100 percent of CPU cycles for extended stretches.

Happily, a subsequent release of GWT improved on deserialization latency by an order of magnitude, so that the same operation required only a few seconds. Although still far from blazingly fast, this at least allowed for performance that fell within an acceptable range.

Another difficulty, which surfaced later in connection with the different instances of the application generated for various browsers, was that the interface didn't render consistently across all the different environments. The engineers managed to fix this at the Java level by using other options to lay out, combine, and compose widgets so they would appear the same regardless of the underlying browser.

LESSONS LEARNED

One unsurprising consequence of relying on a beta release was that when a fresh release of GWT came out in mid-2007, it didn't work right out of the box. Several changes in GWT layout and styling caused the SalesCentrix screens to display differently. It was only with a certain amount of integration rework that SalesCentrix was able to stay current and reap the benefits of the new release.

Also unsurprisingly, the inability to debug at the JavaScript level greatly hampered the SalesCentrix engineers' efforts to speed the population of tables and improve scrolling functionality for large data sets. Although a major factor was that GWT's native debugging capabilities are restricted to Java, SalesCentrix's own lack of JavaScript expertise was another obvious impediment. Still, any effort to debug generated JavaScript code would almost surely have proved to be problematic anyway.

In retrospect, the SalesCentrix team concluded it might have been better to implement their own widget at a more basic level, rather than compose it as they had out of existing widgets. The work-around they implemented worked well enough. Had it failed, the SalesCentrix engineers would have found themselves down at the level of JavaScript and the DOM working to build their own widget implementation from scratch. At the same time, they would have confronted the additional complication of finding a way to integrate their homegrown widget seamlessly into the GWT framework.

Probably the gravest concerns for SalesCentrix surfaced around the serialization/deserialization issue. So deeply embedded and fundamental did that problem turn out to be that, in all likelihood, had GWT's subsequent release not improved latency tenfold, it probably would have been a show-stopper—clearly illustrating the hazards of using early-stage toolkits. The fact that they didn't manage to uncover this problem until they were well along into development led them to perform a retrospective assessment of how they had managed to miss that in their initial risk analysis. Their early work and initial performance metrics with GWT had indicated there was more than enough bandwidth available between client and server to support the kind of local interactivity they were looking for, particularly with regard to the ability to scroll through thousands of records. The tests they used to pass data back and forth were quite simple, however, and didn't exercise GWT's data-transport mechanism in quite the same way as the final application would. In retrospect, they should have. And, yes, consider that a valuable lesson learned. ☐

LOVE IT, HATE IT? LET US KNOW

feedback@queue.acm.org

JEFF NORWALK (jeff.norwalk@norwalkconsulting.com) is a senior technical communicator and project manager. He is also a recovering attorney and ex-best boy, and currently resides in Manhattan.

© 2009 ACM 1542-7730/09/0400 \$10.00