Rich Internet Application UI Development Best Practices

WHO SHOULD READ THIS?
Technical Architects, Developers

WHAT WILL YOU LEARN?
For the first time, applications created for use on the World Wide Web are rivaling those of the desktop. Dubbed Rich Internet Applications (RIAs), they attempt to offer the user a familiar user interface with the functionality and responsiveness expected from modern applications.

RIA technologies enable selective manipulation of Web page content without completely re-displaying an entire page. While this offers great capability in terms of creating a more engaging user experience as well as making more efficient use of physical resources, there are subtle implications as well.

This paper discusses how browser limitations impact development of RIAs and provides tips and techniques to help developers implement successful rich applications. Specifically, two key areas are addressed: optimizing performance and achieving maintainability.

What is an RIA?
Rich Internet applications (RIAs) are Web applications that have the features and functionality of traditional desktop applications. RIAs typically transfer the processing necessary for the user interface to the Web client but keep the bulk of the data (i.e., maintaining the state of the program, the data, etc.) back on the application server.
UNDERSTANDING BROWSER LIMITATIONS

Without a clear understanding of browser environment limitations as they relate to implementation technologies (i.e., DOM Scripting/AJAX), RIA development can lead to a sluggish UI that is frustrating to use. Regardless of sophisticated functionality, users will not embrace it.

RIA technologies impact legitimization of JavaScript as a production-ready language and the acceptance of more sophisticated client side processing. As a result, AJAX, DOM Scripting and event capturing have become standard “tricks of the trade” when developing engaging Web pages.

Although existing Web technologies provide the ability to develop much more sophisticated user interfaces, the basic platform—the browser—has not changed (other than some standardization and bug fixes). Developers who add AJAX capabilities to a Web page to unobtrusively fill in a list box or validate data often do not find this to be an issue. However, a true RIA interface built from the ground up using extensive DOM Scripting has to contend with much more than an occasional AJAX call. To achieve the sophistication of a desktop application, the RIA UI must offer intensive code/processing capabilities such as “drag and drop,” event processing and heavy DOM manipulation. Even though browsers offer JavaScript support to allow these types of interactions to take place, the browser’s original capability was designed to display static content in large “page size” gulps. Other than some expanded JavaScript support, little has changed on the browser side of the equation. Add to this the differences in each browser’s implementation of “standards” and the browser is hardly the environment to offer a rich user experience.

The following sections of this whitepaper address this complex situation. Techniques and guidelines are offered to help developers avoid creating unwieldy UI’s.

RIA DESIGN

Understanding how to effectively develop RIAs is only one part of creating an engaging and effective rich user experience. Equally important is RIA design. Learning how to design for RIAs can be a considerable challenge, especially for designers who are used to static HTML Web sites.

RIA design embraces a completely different paradigm than an HTML Web site. Some examples of these differences are outlined in the chart on the next page.
### RIA CHARACTERISTICS COMPARED TO HTML WEB SITES

<table>
<thead>
<tr>
<th>HTML Web Sites</th>
<th>Rich Internet Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAGE NAVIGATION</td>
<td>PAGELSS NAVIGATION</td>
</tr>
<tr>
<td>Users click from page to page to obtain additional content.</td>
<td>Users may navigate by drilling deeper on a single page.</td>
</tr>
<tr>
<td>SCREENS</td>
<td>MODULES</td>
</tr>
<tr>
<td>The primary building block of an HTML Web site is a screen.</td>
<td>With most RIAs, the primary building block is a module or component of the screen. The content on this module can change or the module itself can morph into something different without affecting the rest of the page.</td>
</tr>
<tr>
<td>SIMPLE INTERACTIVITY</td>
<td>COMPLEX INTERACTIVITY</td>
</tr>
<tr>
<td>In most cases a click brings up a new screen.</td>
<td>Panels slide in and out, content is dragged from one location to another, floating panels appear, etc.</td>
</tr>
<tr>
<td>LIMITED INTERACTION</td>
<td>EXTENSIVE INTERACTION</td>
</tr>
<tr>
<td>Although a Web site can display limited rich content, such as video, audio and animations, the content is displayed in separate views with limited interaction.</td>
<td>Rich content is an integrated component of the application with extensive interaction and communication possibilities.</td>
</tr>
</tbody>
</table>

To fully exploit RIA capabilities, designers must consider the following concepts and options when designing for RIAs:

- **Think RIA**: Use a design process that harnesses the power of RIAs to achieve business goals.
- **RIA design tools**: Choose the right design tools to accommodate the greater complexity of RIAs.
- **RIA documentation**: Learn the most effective way to document RIAs with flows, wireframes and other approaches.
- **RIA accessibility**: Master special techniques to make your RIAs accessible to users with disabilities.
- **RIA testing**: Develop sophisticated testing plans to verify the usability of complex RIAs.

For more information, visit www.tandemseven.com/ria.
OPTIMIZING PERFORMANCE

The primary issue for performance in an RIA is the treatment of external resources such as style sheets, external JavaScript files and graphics. Several techniques have been proven to minimize the performance impact on the use of these critical resources.

REFERENCING EXTERNAL RESOURCES

The first consideration for improved UI performance is the positioning of references to external artifacts within the HTML code, including Cascading Style Sheets (CSSs), graphics and JavaScript files. Browsers operate in a “top down processing” model. As each resource is referenced it is processed. This model makes the order and placement of external resource references critical to performance.

• CSS

Styles are applied by the browser as each HTML element is displayed or rendered. However, if a relevant new style is introduced after an element has been displayed the element(s) will be re-displayed with the new styling applied. In addition to the incremental processing overhead of applying styles to page elements that have already been displayed, the visible changes of re-styled elements can be visually disconcerting. To avoid this situation, references to external CSS files should be placed in the HTML prior to the body tag to ensure the style definitions are available when elements are initially displayed.

• GRAPHICS

Individual graphics are typically loaded in a “just in time” fashion. Practically speaking this means that these resources are either loaded or taken from the cache on demand as <img> tags are encountered. Some techniques load all of the graphics into the cache prior to any page rendering. However, doing so on a regular basis causes the page to appear sluggish in its initial load. Interspersing JavaScript in the body of the HTML to pre-load the image into the cache immediately prior to the rendering stops the appearance of the big red “X” graphic placeholder, but all subsequent processing also pauses while the JavaScript is being processed. This gives a start-stop appearance to any other JavaScript-based rendering. It also places obtrusive code throughout the HTML. Attempting to control graphics loading is a situational decision, but if you are going to let the browser control the loading of graphics files, a good rule of thumb is to simply let the browser load graphics as needed when <img> tags are encountered in the page.
• JAVASCRIPT
Along with the growth of RIA UI development comes a large volume of JavaScript. Rather than a few lines of JavaScript that focus on button rollover and form editing, it is now common to see thousands of lines of JavaScript on one page. Since the browser will parse and process JavaScript in a top down fashion, how you place large pieces of JavaScript code is more important than ever. Even with efficient loading techniques as described below, the processing step takes a noticeable amount of time.

An immediate action a developer can take is to place JavaScript code at the bottom of the HTML document. This defers the extensive parsing/processing of JavaScript until the basic loading and rendering of the HTML document is complete. As always this cannot be done in a dogmatic fashion. The tenets of progressive enhancement¹ and graceful degradation² require that some of the visual treatment of the rendering is modified using JavaScript. As with CSS, placing the JavaScript too late in the load process can cause the content to be rendered and then noticeably re-rendered. Care must be given in using this approach by correctly placing the reference to JavaScript files to balance a delay with a smooth rendering of the page.

OPTIMIZE LOADING OF EXTERNAL RESOURCES
One of the most often ignored aspects of RIA UI development is the haphazard way external resources are loaded. While many treat the actual loading of these resources as "black box," there are several things an astute developer can do to improve the performance and efficiency of the resource loading process.

• REDUCE THE SIZE OF EXTERNAL RESOURCES
HTML, Javascript and CSS files are often full of comments, white space and line breaks. While this is great for readability and maintenance, there is a price to pay in increased file size and, therefore, load time. Stripping files of comments, white space and line breaks when publishing them to the production environment—a process commonly known as “minification”—can reduce the size of text-based resources by 30 percent. Programs such as JSMin will “minify” JavaScript without attempting to hide or alter the text in any other way. (Note the programmatic modification of code exposes the code to name collisions and other assorted unexpected traps. Do this cautiously.)

Once files have been "minified," more dramatic reductions in file size can be gained through a few simple adjustments to the Web server. Most modern Web servers can compress specified files before they are sent across the wire to the browser through a technique known as "GZipping." Browsers are able to detect "GZipped" files and will immediately decompress them before use. This technique commonly reduces file size by more than 50 percent.
LIMIT THE NUMBER OF HTTP REQUESTS
Another beneficial practice is to minimize the number of HTTP requests from the
browser caused by loading separate resource files. Most browsers have a default limit
of two resources that can load simultaneously from a single domain. In this case, the
ramification is that breaking up the JavaScript, CSS and images results in additional
HTTP requests. This situation can be addressed in three ways:

1. Combine all similar resources into a single file. Place all JavaScript into one file
   and all CSS into another. The YSlow extension [http://developer.yahoo.com/yslow/] of
   Firefox’s Firebug will place all of your JavaScript files into one for you. Consolidating
   JavaScript files prior to “minification” can offer a noticeable improvement in load
   performance. Attention should be paid to the size of the consolidated JavaScript
   files. Anecdotal evidence suggests that once the file size exceeds 500K there is a
   performance penalty.

2. For the consolidation of graphics or image files, an approach that is gaining
   wide acceptance is the use of CSS Sprites. With roots in the performance tuning
   of early video games, Sprites are simply the consolidation of separate but related
   images into a single file visually separated by enough “white space” to provide a
   margin around each image. Rather than using <img> tags to display an image on
   a page, use a container such as a <div> with the Sprite as the background image
   and the appropriate CSS positioning to have the needed graphic appear in the
   container. Just as the consolidation of JavaScript and CSS files into fewer larger files
   improves performance by reducing the number of HTTP requests, the use of Sprites
   also significantly reduces the number of HTTP requests. While this technique is
   particularly effective when multiple graphics are used to represent discrete states of
   an element, such as a button or rollover, the approach offers significant performance
   enhancements to any image-oriented page.

3. Load external resources from separate domains. Obtain external resources from
   different domains with no more than two resources coming from each domain (or
   alias). Be aware that more than four simultaneous domains cause a great deal of
   thrashing as the browser struggles to multi-thread.

While there are several methods to improve the performance of a browser’s ability to
load external resources, developers must continually observe the impact on performance
of applying different techniques as more and more resources are loaded to support the
RIA UI.
OPTIMIZED CODING
While size and appropriate reference to external resources are important to performance, the actual coding of JavaScript and HTML can have significant impact on overall performance. There are several easy ways to implement practices that can improve the performance of the application user interface.

• JAVASCRIPT
To keep pages responsive, minimize the amount of direct DOM manipulation. Repeated DOM processing to progressively construct a user interface component makes unnecessary demands on the browser through the DOM API. Constructing a new element as an HTML string and then adding the innerHTML value of the parent element is a better practice. This becomes a very efficient exercise in string manipulation rather than a constant call to the DOM parsing/traversing capabilities of the container. Similarly, when an existing element or structure must be manipulated in any significant way it is more efficient to copy it to a string (again, innerHTML), manipulate it with efficient JavaScript string handling functions and replace it when the processing is complete. The developer must be practical. To simply change a single attribute on an element by extracting, manipulating and replacing the element might be overkill.

Another common pitfall in the name of progressive enhancement is searching the DOM for elements based on very general attributes. For example, searching a document for all table cells <TD> contained within elements that are assigned a particular class is very inefficient. Since browsers do not create an index of elements by class, this approach requires every element in the document to be examined. However, browsers do create an index for elements by tag type. Performing an indexed search for elements with the tag types expected to have the desired class and then examining the reduced result set is a better method. Be as specific as possible when searching for DOM nodes.

• HTML/LAYOUT
Using tables to specifically control positioning is a very inefficient technique. Tables were introduced to allow the tabular presentation of data. For this purpose they are a great tool and, when combined with some JavaScript magic, they can be sortable, pageable and dressed very nicely to highlight important information. As a layout manager, however, they are, by their nature within the HTML framework, very inefficient. The basic rendering of a table requires the HTML parser/rendering engine to read the entire table so that it has a complete internal description of the complete table prior to beginning the rendering. This means the table structure must be processed twice. Considering that it is not unusual to see tables nested within table cells, the developer of a responsive RIA UI must consider the consequences of this approach. A much more efficient choice is to use CSS positioning for page layout.
ASYNCHRONOUS CALLS (AJAX)

One of the most popular innovations in the development of RIAs is the ability to asynchronously retrieve data and other resources on demand. This technique generally employs a group of technologies collectively known as AJAX. While not overly complex to implement, AJAX can have an unexpected impact on performance. Typically AJAX relies on a server to return a resource or execute a program. The performance of AJAX, and hence the application overall, is dependent upon the responsiveness of the server as well as the network and other infrastructure that ties the components of the application together.

When developing applications using AJAX or any other technique that dynamically retrieves resources, consideration must be given to the responsiveness of the application and how the application will address potential delays and slowdowns. In many cases work can continue while the application retrieves needed resources. However, if the application must wait for the completion of the asynchronous call, the developer and designers must agree on some method for blocking use or otherwise informing the user that there is an “expected” or understandable delay and that processing is still continuing. If interaction with the application must be halted temporarily, one effective method of blocking is covering the visible window with an opaque HTML element such as a `<div>` with a clear, prominently displayed notification message. Regardless of the method used, good practice is to keep the user informed of the state of the application at all times, but particularly when events are occurring that could mislead the user into believing that the application is not working.

ACHIEVING MAINTAINABILITY

With the increased volume of JavaScript and CSS in the modern RIA UI, the difficulty in maintaining the codebase increases significantly. No matter how well crafted the original UI, inevitable updates and enhancements will, without proper precaution, make it increasingly difficult to maintain over time.

ESTABLISH A COMMON DOCUMENT STRUCTURE

A common document structure makes it easier to understand what is included (and what is not) in any given document. General practice is to include a description of the document structure in comments at the top of the document. Including a description of any constants or structures that might be found is also wise. While this documentation does not need to be overly descriptive it does need to guide the new developer to the desired section of the document. Remember, the “minifier” will remove these comments from the production code library to prevent them from impacting the size of the code base.
ESTABLISH STANDARD CODING CONVENTIONS

Standard coding conventions help ensure future developers will only need to understand one section of the code base to understand the entire code base, regardless of how many authors were involved. Douglas Crockford of Yahoo provides a good set of base constructs at http://javascript.crockford.com/code.html. Be warned that a blanket set of conventions will not work for all projects and should be tailored for each project. For example, Crockford rages against the use of the keyword eval. However, there are projects where a large amount of JavaScript might be returned via AJAX calls. Eval might be deemed to be absolutely appropriate for these projects.

PROVIDE LIBERAL INTERNAL DOCUMENTATION WHERE NEEDED

Clever naming standards and clean code will not make the code base self-documenting and eliminate the need for comments. A good test is to go to an RIA application, look at the JavaScript and determine if the code can be followed without comments—probably not. Provide ample comments to explain what is happening when using a convoluted test or esoteric algorithms. Remember: When “minifying” files for production, comments are removed resulting in no impact on performance.

Since every situation is unique and a balance must be struck between performance, maintainability and the standards put in place, developers should not blindly adhere to any set of practices. Be certain that the deviation being undertaken provides significant worth and requires documentation. Strive to conform but, when the price of conformity is outweighed by the gain from the occasional deviation, make the “smart” decision.

IN SUMMARY

Recent improvements in both technology and development techniques enable the developer to provide a vastly enhanced user experience. Because these improvements rely on the browser—an underlying technology that has not seen a similar revolution in capability—developers must be vigilant in how these expanded capabilities are applied.

By addressing the most common performance problems encountered in the development of RIAs, developers are free to focus on presentation and functionality of an application. Furthermore, by implementing standard practices that promote maintainable code while not impairing the performance of an application, programming for performance is not done at the expense of maintainability. Standard document structures, coding conventions and liberal documentation in the testing/maintenance environment will go a long way in keeping the code base maintainable for current and future developers.

No rules are absolute and no single set of practices will work in all cases. However, the practices presented in this paper provide a solid foundation for the development of high performance RIAs.
ABOUT THE AUTHORS

DEAN JENNINGS
Dean is a Principal User Interface Architect with TandemSeven. He specializes in using leading edge technologies to develop frameworks and architectures that focus on user interaction. Dean was an early adopter of rich user interactions, now known as RIA technologies, and is a recognized expert in underlying RIA technologies, including JavaScript/ECMAScript, CSS, DHTML/XHTML and cross-browser compatibility. Dean personally designed and developed several RIA/AJAX-based frameworks that utilize normal HTML and lightweight JavaScript function calls, allowing developers with minimal RIA technology experience to develop such applications with ease. Dean earned a BS from Emory University in Atlanta. He can be reached at djennings@tandemseven.com.

STEVE WYMAN
As Vice President of Technology at TandemSeven, Steve is responsible for leading the company's technology efforts and managing its development process. He provides clients with a range of capabilities, including project management, technical oversight and development and delivery of mission critical applications. Throughout his career, Steve has worked with numerous Fortune 500 companies and has led international teams in a 24/7 development cycle. Steve earned a BA and MBA from the University of Maine, and an MSCS from Rensselaer Polytechnic Institute in Hartford, Connecticut. He can be reached at swyman@tandemseven.com.

Footnote information

ABOUT TANDEMSEVEN

TandemSeven designs, architects and builds user interfaces for world-class business applications and portals. We specialize in creating usable, intuitive interfaces for complex global applications. Headquartered in the greater Boston area, TandemSeven has designed and developed usable applications for numerous global companies, including 1-800-FLOWERS, Campbell Soup Company, Citi, FXall, Hasbro, LeapFrog Enterprises, Morgan Stanley, Orbitz Worldwide and Siemens Corporation.

For more information, visit Tandemseven online at www.tandemseven.com, or call 508-746-6116.