



Rich Internet Applications: Design, Measurement, and Management Challenges

Table of Contents

1. Executive Summary	3
2. Overview	3
3. The Behavioral Characteristics of Rich Internet Applications	3
3.1. The Evolution of the Web Page	3
3.2. From Web Pages to Rich Internet Applications	4
3.3. The RIA Behavior Model	4
3.4. Browser/Server Interaction	5
3.5. Server-Side Elements	6
3.6. The Client-Side Engine	6
3.7. The Importance of Context	7
4. Measuring RIA Responsiveness	8
4.1. Measurement Foundations	8
4.1.1. Reasons for Measuring Applications	8
4.1.2. Active and Passive Measurement	8
4.1.3. Location of Measurement Probes	9
4.2. Complications When Measuring RIA Responsiveness	9
4.2.1. Greater Variety of Possible Behaviors	9
4.2.2. Increase in Concurrent Activity	10
4.3. RIAs: What and Where to Measure	10
4.3.1. RIAs: What to Measure?	10
4.3.2. Comet: Pushing the RIA Model Even Further	11
4.3.3. Measuring Push Technology	12
4.3.4. RIAs: Where to Measure?	12
4.4. RIAs: How to Measure Responsiveness	13
4.4.1. Emulating the User's Actions	13
4.4.2. Emulating User Think Times	13
4.5. Implications for Measurement Tools	14
4.5.1. Table 1. Summary of the Implications of Rich Internet Applications for Measurement Activities	14
5. Designing and Managing Rich Internet Applications	14
5.1. Availability: A Measure of Overall Quality	14
5.2. Responsiveness: Achievable, but not Guaranteed	15
5.2.1. Improving Responsiveness Involves Tradeoffs	15
5.2.2. Will Prefetched Content Be Used?	15
5.2.3. Unintended Communication Overheads	16
5.2.4. Will Offloaded Processing Be Faster?	16
5.2.5. Making Meaningful Comparisons	16
5.3. Clarity Requires Distributed Application Design	17
5.4. Utility Depends on Everyone's Contribution	18
5.4.1. The Need for Agile Processes	18
6. Resources	19

1. Executive Summary

The Web is already the platform for doing business efficiently and quickly. As the penetration of high-speed and broadband Internet access increases, Web technologies continue to evolve to deliver new user experiences and increased application utility. The Rich Internet Application (RIA) is another step in that evolutionary process.

The Rich Internet Application reflects the gradual but inevitable transition of Web applications from the simple thin-client model of a traditional Web browser to a richer distributed-function model that behaves more like the desktop in a client/server model. Today these richer user experiences are being implemented with technologies such as Flash, Ajax, and Java, using standard Internet and Web protocols.

Because RIAs are significantly more complex than traditional browser-based Web applications, they pose new design, measurement, and management challenges. To implement RIAs successfully, enterprises must reevaluate their approaches to service level management (SLM) and consider several new challenges including:

- how to measure an RIA user's experience of response time;
- how to break apart into meaningful components the time it takes to complete a business transaction;
- how to monitor the performance of a production application, and alert when its behavior is abnormal;
- how to change development and systems management processes to ensure successful implementations.

In each of these areas, introducing RIAs will probably require most enterprises to change what they do today.

2. Overview

This paper is presented in four parts:

1. We describe the evolution of traditional Web pages and applications and review the behavioral characteristics of the emerging class of Rich Internet Applications.
2. Because effective SLM methods depend on being able to measure performance, we describe how these more complex applications affect today's measurement tools and methods.
3. We consider how introducing RIAs affect Web SLM processes that were originally intended for designing and managing the performance of traditional Web applications.
4. We provide a list of useful resources referenced in this paper.

3. The Behavioral Characteristics of Rich Internet Applications

3.1 The Evolution of the Web Page

The Web was originally intended to help researchers share documents as static pages of linked text formatted in HTML. From there, Web pages quickly evolved to include complex structures of text and graphics, with

In a traditional Web application, all processing is done on the server, and a new Web page is downloaded each time the user clicks.

In a Rich Internet Application, some processing is transferred to the user's computer ("the client").

plug-in programs to play audio and video files or to stream multimedia content. Web developers supplement the basic browser function of rendering HTML by invoking code (*scripts*) on the user's computer (*the client*). These scripts can create interface elements such as rollover effects, custom pull-down menus, and other navigation aids. They can also execute UI methods, for example, to validate a user's input in an HTML form.

These script capabilities, while they enhance a user's interaction with individual Web pages, do not change the fundamental model in which application logic runs on the server and executes between Web pages after the user clicks. This behavior is said to be *synchronous*, that is, after each click the user waits while the server handles the input and the browser downloads a response page. In e-commerce, a typical user interaction involves a series of Web pages, which represent steps in a larger process that comprise a Web application.

3.2. From Web Pages to Rich Internet Applications

Recently, Web developers have been evolving a new model—the Rich Internet Application (RIA), which is "a cross between Web applications and traditional desktop applications, transferring some of the processing to a Web client and keeping (some of) the processing on the application server".

As with most computing advances, several technologies are vying for acceptance as the de facto standard way to build RIAs. The main contenders are Adobe's Flash suite, Java applets, and the collection of Web technologies known as Ajax, a term coined in 2005 by Adaptive Path's Jesse James Garrett. Garrett explained that Ajax is "really several technologies, each flourishing in its own right, coming together in powerful new ways.

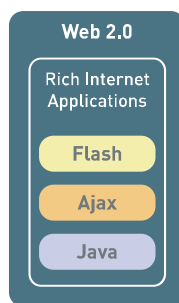


Figure 1: Relationship between Web 2.0, RIA, Ajax, Flash

Ajax incorporates:

- standards-based presentation using XHTML and CSS;
- dynamic display and interaction using the Document Object Model;
- data interchange and manipulation using XML and XSLT;
- asynchronous data retrieval using XMLHttpRequest;
- JavaScript binding everything together."

In practice, we can think of Ajax as a development concept or approach, as various technologies can substitute for those specified by Garrett. Some developers even claim that "Flash is Ajax", and others advocate using them together. See AFLAX. Complicating any analysis of RIA technology is the massive amount of hype surrounding both "Web 2.0" (a superset of RIA) and Ajax (a subset of RIA). Figure 1 illustrates these relationships.

The RIA Behavior Model summarizes three factors that together determine RIA performance.

The solid black arrows in Figure 2 trace the flow of a Web page download.

3.3. The RIA Behavior Model

A reference model is useful for establishing a shared frame of reference or conceptual framework to structure subsequent discussions of a subject. Figure 1 introduces the RIA Behavior Model, which represents the principle elements to be considered in any discussion of RIA behavior, and in particular RIA performance and management.

Note that the model does not address the complex human forces that determine perceptual, cognitive, and motor behaviors. It merely represents a few behavioral outcomes that are relevant in the context of an interaction between a human user and a Rich Internet Application. At the highest level, the model illustrates three major aspects (indicated by the color coding in the figure), each of which influences application performance:

- The application’s design and usage environment, or context (upper row, green)
- The user’s expectations and behavior (lower left, yellow)
- The application’s behavior when used (lower right, blue)

We now describe the model by stepping through what happens during a user interaction with a browser-based application. In the process, we note some key issues that affect the responsiveness of RIAs.

3.4. Browser/Server Interaction

If we consider a Web browser to be the simplest form of client engine, then the solid black arrows trace the flow of a traditional Web page download. The user *clicks* on a link in the browser, the browser sends *requests* to one or more servers. *Servers respond to client requests*, and when enough of the requested *content* arrives on the client (in the browser cache), the browser displays it and the user can *view it*. The user’s experience of response time is the elapsed time of the entire process from click to view.

Even a single Web page download typically involves many round trips between client (browser) and server, as most Web pages are an assemblage of content elements such as CSS files, script files, and embedded images, each of which is downloaded individually by the browser.

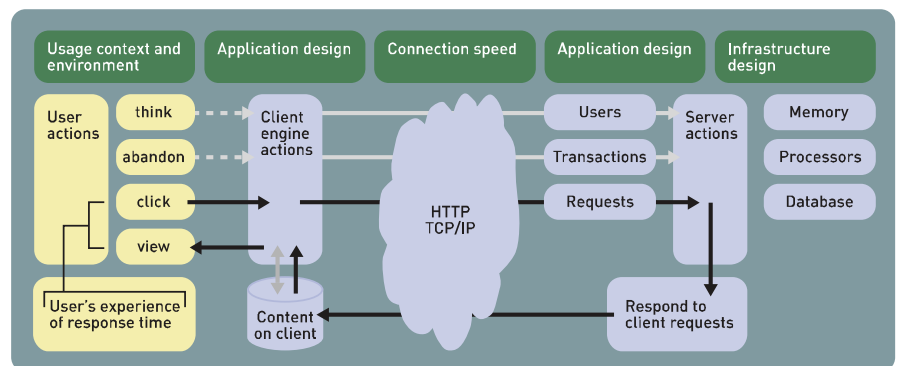


Figure 2: The RIA Behavior Model

Server performance is influenced by the number of concurrent users and concurrent transactions.

In a traditional synchronous Web application, this process repeats several times. Because applications usually require an exchange of information, at least one of the requests the browser sends to the server will typically be an HTTP POST (as opposed to the much more common HTTP GET request), to upload some data a user has entered into a form. Consider, for example, shopping at Amazon.com as a return visitor. At minimum, even if the application recognizes you from a cookie, you must reenter your password to confirm your identity. But after that, the site already has your personal information and you can complete your transaction by clicking on the specified buttons on each page as they are presented to you.

3.5. Server-Side Elements

Servers must field requests concurrently from many users. No matter how powerful the server, every concurrent user consumes a small share of the server's resources: memory, processor, and database.

Web servers can respond rapidly to stateless requests for information from many concurrent users, making catalog browsing a relatively fast and efficient activity. But a user's action that requires the server to update something (such as clicking a button to add an item to a shopping cart) consumes more server resources. So the number of concurrent transactions—server interactions that update a customer's stored information—plays a critical role in determining server performance.

In the model, the grey arrows and the boxes labeled *Users* and *Transactions* indicate that server performance is strongly influenced by these concurrency factors. Servers typically perform uniformly well up to a certain concurrency level, but beyond that level (the knee of the curve), transaction performance quickly degrades as one of the underlying resources becomes a bottleneck. As a result, seemingly small changes in application behavior or in the infrastructure serving the application may have a significant effect on the user's experience of response time if those changes extend transaction durations.

People who design and test back-end systems already know that behavioral variables such as user think-time distributions and abandonment rates per page have a significant influence on the capacity and responsiveness of servers under load. Now RIAs (as indicated by the dotted lines in Figure 2) give application designers the flexibility to design applications that attempt to take account of such behavioral variables.

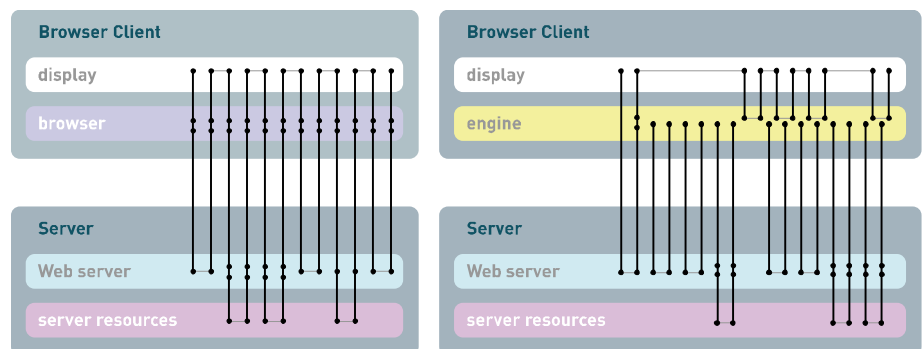


Figure 3a (left): The communication model of a traditional Web application
Figure 3b (right): The communication model of a Rich Internet Application

Adding a client-side engine separates user's interaction with the application from the browser's communication with the server.

Designers can use asynchronous communication to make applications more responsive, but the result will be more complex, and success is not guaranteed.

The success of any application design will depend on how well it matches the users' needs, their way of thinking, and their behavior.

3.6. The Client-Side Engine

Although implementations differ, all RIAs introduce an intermediate layer of logic— a client-side engine—between the user and the Web server. Downloaded at the start of the session, the engine handles display changes and communicates with the server.

Adding this layer allows developers to build Web applications with characteristics that the Gartner Group has described as “between the fat but rich client/server model and the thin but poor Web based UI model.”

Adding a client-side engine does not prevent an application from implementing the traditional synchronous communication style. But it also allows the user's interaction with the application to happen asynchronously— independent of communication with the server. Figures 3a and 3b illustrate how the asynchronous pattern of user-to-server communication implemented by RIAs differs from the synchronous behavior of a traditional Web application. In a Rich Internet Application:

- information can be fetched from a server in anticipation of the user's input;
- in response to an input, the screen can be updated incrementally instead of all at once;
- multiple user inputs can be validated and accumulated on the client before being sent to the server;
- responses to some user inputs can be generated without communicating with the server;
- processing previously handled by servers can be offloaded to the client desktop.

Designers can exploit these possibilities to make their applications more responsive, and most writers advocating Ajax and RIAs assume that the architecture guarantees a more responsive user experience. Indeed, in the best examples, users *will* spend less time waiting for the server to respond; however, employing these techniques will inevitably lead to a more complex design than the traditional synchronous Web application. The challenge of SLM is to ensure that the outcome is actually a more responsive user experience. Despite the optimistic claims being made for Ajax and Flash, there are no guarantees. In practice, RIA responsiveness will depend on several factors that we discuss later in this paper.

3.7. The Importance of Context

This brings us to the crucial importance of the design and usage environment or context, represented by the upper row of green boxes in the model. The users' satisfaction with any application depends on the usage context and environment, that is, how well the application design matches the users' needs at the time, their ways of thinking, and their behavior when using the application.

A user's experience of response time depends on the combined behaviors of the client and server components of the application, which in turn depend on the application design, the underlying server infrastructure design, and, of course, the user's Internet connection speed. The most effective RIA will be one whose creators took into account these factors at

each stage of its development life cycle, and who created the necessary management processes to ensure its success when in production.

4. Measuring RIA Responsiveness

4.1. Measurement Foundations

Before considering the particular challenges of measuring Rich Internet Applications, we first review some fundamental aspects of Web application measurement.

4.1.1. Reasons for Measuring Applications

As Keynote CEO Umang Gupta likes to say, “there are apps people and ops people.” This handy way of subdividing the world of information technology pinpoints an essential division that is reflected in many areas, such as users and systems, clients and servers, developers and administrators. Accordingly, the two main reasons for measuring the performance of a distributed application are to determine how quickly users can achieve their goals, and to discover how a system behaves under increasing load. The first focuses directly on the users’ experience, the second investigates underlying server behaviors that, in turn, will determine what users experience.

Within these two broad categories, measurement activities and tasks may focus on a variety of possible sub-goals. Ten of the most common motivations for measuring applications are listed in Table 1. Those addressing the first goal are conventionally called measurement, while those addressing the second are referred to as load testing, or simply testing. However, there is considerable overlap between them because they share many technical problems.

4.1.2. Active and Passive Measurement

A crucial factor in any measurement process is the target—in this case the particular mix of application software and hardware behaviors that are measured. A general term for such a mix is a measurement workload. A workload may be either real (produced by the actions of real users of the application), or synthetic (produced by computers that emulate user behaviors).

Measurements of real workloads are referred to as passive measurements, because the act of measurement involves simply observing an application’s behavior under normal usage conditions and recording what happens. Active measurements, in contrast, are obtained by generating synthetic application traffic. For example, one might measure a system’s maximum capacity by emulating a mix of user actions and increasing the number of simulated users until a component of the system saturates.

Note that the passive and active measurement approaches differ only in the way application traffic is generated—both still require mechanisms to measure how the system behaves in response to that traffic. Passive measurements must capture the behavior and experience of real application users, while active measurements must do the same for synthetic users. So both approaches must deal with the same set of technical complications created by the need to measure Rich Internet Applications.

Measurement usually focuses on the user’s experience or on the system’s behavior. Despite differences, these two classes of measurement activity share many technical problems.

Active monitoring tools measure synthetic workloads, while passive tools measure the system activity generated by real users.

For measurement tools, both real and synthetic RIA workloads will pose the same challenges.

Passive monitoring tools can infer client-side response times from TCP and HTTP traffic, using packet sniffing techniques.

The variety of possible RIA behaviors creates new opportunities for developers to make performance-related mistakes, requiring more systematic approaches to measurement.

Although active measurements do impose extra traffic, they rarely distort a system's behavior sufficiently to affect the validity of the results. For a typical e-business application, the number of additional active measurements required to sample the system and obtain useful data is usually insignificant compared with normal traffic volumes. Thus normal levels of application responsiveness and availability can be measured using either active or passive methods. Load testing on the other hand normally involves active measurement of a portion of the system that is isolated from real users for the purpose of the test.

4.1.3. Location of Measurement Probes

It may seem that to measure a user's experience of responsiveness, tools would need to collect measurements from users' workstations, or from measurement computers programmed to generate synthetic actions that imitate the behavior of a typical user. Surprisingly, this is not always the case for traditional Web applications.

The synthetic measurement approach does require computers to mimic both a user's actions and their geographical location. But passive measurement software can reside either on the client machine or on a machine that is close to the server, provided that it can observe the flow of network traffic at the TCP and HTTP levels. Because these protocols are synchronous and predictable, a tool that can read and interpret packet-level data can infer the user's experience of response time by tracking HTTP messages and the times of underlying TCP packets and acknowledgements.

Such a tool is called a packet sniffer, or protocol analyzer. Packet sniffing has a bad name in some quarters, being associated with malicious snooping by hackers. But in the right hands, it is a legitimate analysis technique used by some Web measurement tools to deduce client-side performance without installing any components, hardware or software, anywhere near the users.

4.2 Complications When Measuring RIA Responsiveness

Introducing a client-side engine makes RIAs significantly harder to measure than traditional Web applications. Additional complications arise in two areas:

4.2.1. Greater Variety of Possible Behaviors

To make meaningful statements about an application's performance, you must first decide what to measure. Enterprises will typically measure application usage patterns (*scenarios*) that are common, or important by reason of their business value.

When an application uses only a standard browser, its behavior is simple, known, and predictable, limiting the number of interesting scenarios to be measured. Adding a client-side engine separates user interface actions from server requests and responses, giving application designers many more options. It allows developers to build an application that uses creative, clever ways to transfer display elements and portions of the processing to the client.

Such freedom to improvise inevitably makes measurement harder. The custom application behaviors encoded in the client-side engine make the

Concurrent client/server interactions make it difficult for measurement and analysis tools to correlate seemingly separate pieces of data with a particular application activity.

We can no longer think of a Web application as a series of Web pages.

We can no longer assume that the time it takes to complete a page download corresponds to something a user perceives as important.

RIA milestones must be specified in advance to allow analysis tools to group measurement data into 'logical pages' or tasks.

result more complex and its usage less predictable than a traditional browser-based application. This increases the number of possible usage patterns, complicating the task of determining the important scenarios and measuring their performance.

4.2.2. Increase in Concurrent Activity

It is easy for a measurement tool to sit on a server and measure all requests for service—and this kind of measurement has its uses, especially when load testing or investigating bottlenecks. But because of the variety of implementation possibilities, a common problem when measuring RIAs is that related requests may appear to originate from separate units of work on the client.

Correlating seemingly separate measurements with a particular application activity, task, or phase is tricky. The more complex the client/server relationship, especially when it involves concurrent interactions, the harder it becomes for measurement and analysis tools to perform that correlation properly.

4.3. RIAs: What and Where to Measure

The previous section described two major complications introduced by RIAs—measurements may become harder to specify and more difficult to correlate. We now discuss four areas in which existing measurement approaches break down altogether, and must be changed to accommodate the RIA behavior model. These changes deal with the issues of what, and where to measure.

4.3.1. RIAs: What to Measure?

What do we measure in an RIA? The short answer is not individual Web pages. The asynchronous behavior of RIAs undermines two current assumptions about measuring Web applications:

- We can no longer think of a Web application as a series of Web pages.
- We can no longer assume that the time it takes to complete a page download corresponds to something a user perceives as important.

In fact, when a client-side engine can be programmed to continually download new content, or a server-side engine can keep pushing new content to the browser over a connection that never closes (see later discussion of Comet), some page downloads may never complete. Therefore, to report useful measurements of the user experience of response times, instead of relying on the definition of physical Web pages to drive the subdivision of application response times, analysts must break the application into logical pages. To do this, a measurement tool must recognize meaningful application milestones or markers that signal logical boundaries of interest for reporting, and thus subdivide the application so that tools can identify and report response times by logical page.

Because (as noted earlier) it is usually much harder to correlate seemingly separate pieces of measurement data after the fact, these milestones will ideally be identified before measurement takes place. One of two methods can be used to create milestones:

- Identifying and recognizing events that already occur naturally within the application.

- Adding instrumentation, by inserting simple monitoring calls like those used by the ARM (Application Response Measurement) standard, an existing method for measuring distributed applications. In this approach, developers imbed simple instrumentation calls at logical boundaries in the application's flow, to enable its subsequent measurement.

The former method places the burden on those setting up measurements to first identify application-specific milestones. The latter frees the measurement tool from the need to know anything about the application, but places the burden on application developers to instrument their code by inserting markers at key points.

4.3.2. Comet: Pushing the RIA Model Even Further

Most discussions of Flash and Ajax technology focus on how asynchronous application behaviors can be implemented by client-side engines, which can be programmed to communicate with server(s), independent of a user's actions. In general, a user action within a Rich Internet Application can trigger zero, one, or many server requests.

Nevertheless, those requests still involve synchronous communication between browser and server. That is, communication is always initiated by the browser (or the client-side engine operating as an extension of the browser), and follows the standard synchronous HTTP request/response protocol.

But recently Alex Russell proposed the name Comet to describe ways to exploit HTTP persistent connections to implement a push model of communication. Using these techniques, illustrated in Figure 4, a server uses long-lived persistent connections to send updates to many clients, without even receiving requests (polls) from clients. For details, see Russell's presentation to *ETech 2006*.

Comet is a label for a push communication style implemented using persistent HTTP connections.

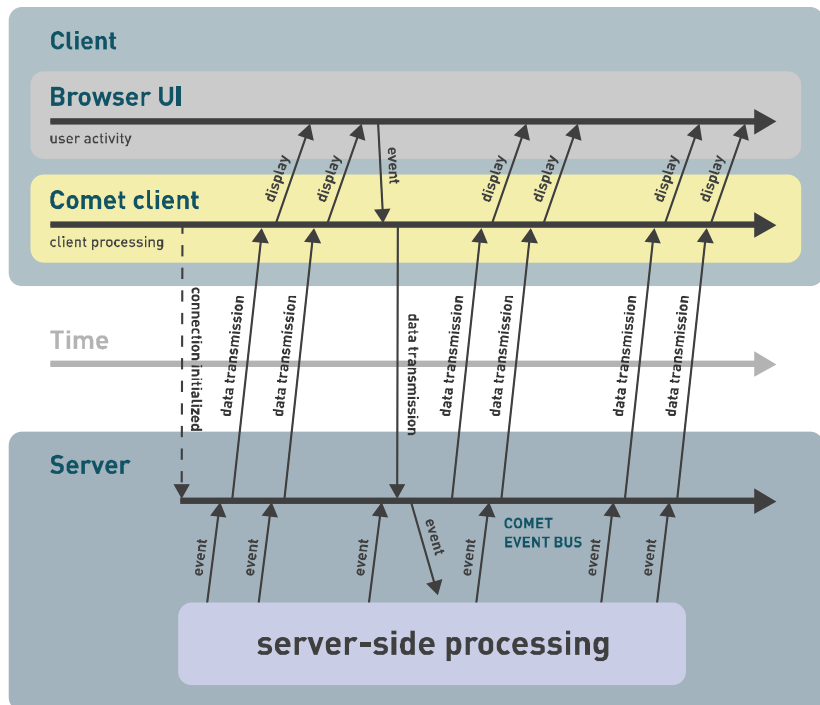


Figure 4: Push communication using the Comet RIA model

RIA push models (like Comet) will require the invention of new metrics to measure Web site effectiveness.

The RIA's client-side engine breaks apart the traditional cycle of communication between browser and server, creating two separate cycles that may operate independently.

Passive monitoring of server requests will be insufficient to determine a user's perception of RIA responsiveness.

The complexity of a push architecture is justified only for applications that manage highly volatile shared data. But when it is implemented, it creates an entirely new set of measurement challenges.

4.3.3. Measuring Push Technology

If information pushed to the client does help the user to work faster, its benefits will be reflected in normal measurements of application responsiveness. But normal responsiveness metrics cannot be used to evaluate a server-initiated activity that simply updates information a user is already working on.

New metrics must be devised. Depending on the application, one may need to measure the currency or staleness of information available to the user, or possibly the percentage of times a user's action is invalidated by a newly received context update from the server. This kind of "hiccup" metric would be conceptually similar to the frequency of rebuffering incidents, a typical measure of streaming quality.

Server capacity will also be a major issue requiring careful design and load testing. These SLM challenges must be faced by anyone considering the Comet approach.

4.3.4. RIAs: Where to Measure?

We noted above that active measurement tools must necessarily reside at the client; whereas today's passive tools that use packet sniffing technology can measure Web application traffic anywhere on the network between the client and the server. However the RIA architecture limits the flexibility of the packet sniffing approach.

Reviewing Figure 3a, we can characterize the response time of a traditional Web application as the time to complete the synchronous round trip of:

Click(C) => Browser(B) => Request(Q) => Server(S) => Response(R) => Display(D)

However, as we see in Figure. 3b, the client-side engine breaks apart this cycle into two separate cycles operating asynchronously—a user/client-engine cycle, and a client-engine/server cycle:

Click(C) => Engine(E) => Display(D)

Request(Q) => Server(S) => Response(R)

Another way of describing these two cycles might be as foreground (C-E-D) and background (Q-S-R). Both cycles are important, because neither stands alone; it is their relationship that defines application behavior. But that relationship depends only on the application design, which cannot (in general) be inferred by a measurement tool, especially one that can observe only one of the two cycles.

Therefore to measure RIA responsiveness, tools must reside on the client machine, where they can see both the level of responsiveness experienced by the browser or engine (the Q-S-R cycle) and the user's experience of responsiveness (the C-E-D cycle). An approach comprising only passive listening on the server side will be insufficient to measure RIA responsiveness. Although traditional packet sniffing methods can still

Active measurement tools must simulate user actions, not just engine actions, to measure RIA responsiveness.

Active measurement tools must simulate user think times during the measurement process, to reflect a user's experience accurately.

monitor Q-S-R cycles, doing so will permit only limited inferences about C-E-D cycles, which are separately managed by the client-side engine.

4.4 RIAs: How to Measure Responsiveness

We have seen that RIAs will affect where a passive measurement tool can be used. Active measurement tools, because their modus operandi is to simulate the user's behavior, are not affected by this issue. Because they mimic the client, they naturally reside there. For these measurement tools, the issue raised by RIAs is how closely a user's behavior needs to be emulated.

4.4.1. Emulating the User's Actions

Using the terminology introduced above, active measurement tools must drive the C-E-D cycle, not the Q-S-R cycle, because RIAs can generate back-end traffic in response to any user action, and not only when the user clicks. For example, the *Google maps* application can trigger preloading of adjacent map segments based on the direction of a user's cursor movement.

Therefore to measure RIA responsiveness, an active measurement tool must simulate the user's actions, not the engine's actions. The former involves driving the client-side engine to obtain its backend behavior; the latter would require the tool user to supply a complete script of the engine's behavior to the active measurement tool, which would not normally be practical.

4.4.2. Emulating User Think Times

A client-side engine may be doing useful work in the background, while a user is reading the previous response or deciding what to do next. It may, for example, be pre-fetching content in anticipation of the user's next action. Therefore the time a user devotes to these activities—labeled *think time* in the RIA Behavior Model—may affect the user's perception of the application's responsiveness.

For passive measurements of real user traffic, this is not a problem, because the data includes the effects of the users' actual think times. And active measurement tools have not needed to simulate think times for traditional Web apps (by pausing between simulated user actions) because introducing such delays would not have altered the result. When measuring an RIA however, setting think times to zero (as is typically done today) minimizes the possibility of any background preloading activity, and so maximizes the measured response times of later user actions.

Therefore to reflect a user's experience, active measurement tools will need to evolve to simulate user think times during the measurement process. Inserting realistic think times between user actions, as the best load testing tools do already, will produce the most realistic measures of the response times users actually perceive.

4.5 Implications for Measurement Tools

Table 1 below summarizes the implications of the main conclusions of Part 2 of the paper, showing how the need to measure Rich Internet Applications will affect ten different activities in the field of performance management or SLM, and the tools used to perform those activities.

Id	Measurement Class	Purpose of Measurement Activity	Measurement method(s) available	Tools must emulate user think-time behavior?	Tools must drive or emulate engine behavior?	Tools must measure response time at client interface?
1	Measurement	Verifying that an application meets its service level objectives	Active or passive	Yes, if active measurement	Yes, if active measurement	Yes
2	Measurement	Detecting abnormal application behavior (typically, slow response)	Active or passive	Yes, if active measurement	Yes, if active measurement	Yes
3	Measurement	Identifying bottlenecks in application components	Active or passive	Yes, if active measurement	Yes, if active measurement	No
4	Measurement	Comparing builds, versions, or releases of an application	Active or passive	Yes, if active measurement	Yes, if active measurement	Yes
5	Measurement	Comparing two applications that perform comparable functions	Active or passive	Yes, if active measurement	Yes, if active measurement	Yes
6	Testing	Verifying that a system will be able to handle the projected traffic	Active only	Yes	Yes	Yes
7	Testing	Determining how many users a server environment can handle	Active only	Yes	Yes	Yes
8	Testing	Predicting bottleneck components as workload levels grow	Active only	Yes	Yes	No
9	Testing	Comparing builds, versions, or releases of an application	Active only	Yes	Yes	Yes
10	Testing	Identifying components that fail after extended use	Active only	Yes	Yes	No

5. Designing and Managing Rich Internet Applications

To satisfy its users and meet the objectives of its creators, a Web site or application must fulfill four distinct needs—*availability*, *responsiveness*, *clarity and utility*, each of which is essential to the success of any application. Figure 5 presents these four essential qualities in the order of their significance, at least for first-time visitors to a site.

We will use this simple framework to evaluate the challenges enterprises face as they seek to exploit RIA technologies to create engaging, responsive, and ultimately successful Web applications.

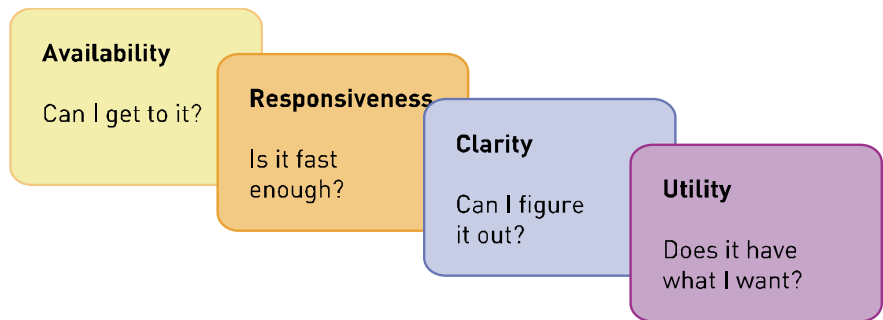


Figure 5. Four Dimensions of Application Usability

“Creating an Ajax application from scratch is like having to build a brick wall but first having to figure out how to create the bricks.”
—Todd Spangler

Making tradeoffs to deliver some responses faster may slow down others. It is essential to test a wide spectrum of usage scenarios.

Users who abandon before completing transactions tie up server resources unnecessarily

5.1. Availability: A Measure of Overall Quality

Application availability is the first quality encountered by customers, but the last one to be determined by the development process, because it depends on the quality of everything else that happens during application design, development, and testing. A chain is only as strong as its weakest link, and developing RIAs will uncover any weaknesses in an enterprise’s development and SLM processes.

RIAs inevitably involve running complex code on the client, and use more complex and application-specific strategies to manage the communication between browser and server. Making such complex code efficient and bug-free is not a task to be taken lightly. Better tools are needed to help us meet the challenges of creating and managing robust applications; today, while the Adobe suite of Flash tools is a little more mature, Ajax development and testing tools are still in their infancy. The Open Ajax initiative may help to address this issue.

Therefore deploying an RIA successfully will demand more resources—skills, tools, process, time, and (of course) money—than would be required to deploy a more traditional Web-based application in which all application logic is executed on the server. Enterprises that decide to employ the technology must be prepared to invest more in design, development, testing and management to make it successful.

5.2. Responsiveness: Achievable, but not Guaranteed

According to their proponents, RIAs will be more responsive applications. Web server requests will be limited to the bare minimum required to do the job, reducing the size and frequency of data exchanges between browser and server. The reality, however, is less straightforward, as we discuss in the following sections.

5.2.1. Improving Responsiveness Involves Tradeoffs

RIA design can be complex, magnifying the risk of failure should the designer miscalculate. Generally, optimizing any design to speed up one class of work always makes some other workload slower. Unless you consider and test for a wide range of use cases, what seemed to be a clever way to reduce network traffic may sometimes turn out to be slower than the older and simpler design. A simple design applied well beats a clever design misapplied.

Even if the application serves some users quickly, those whose usage patterns do not match the profile the application designer had in mind will probably not receive good service, and in the worst case, may abandon transactions before completing them. Apart from the lost opportunity to serve a customer, abandonment also wastes scarce server resources, because the allocations earmarked for now-abandoned transactions languish unused until finally freed by some type of timeout mechanism.

5.2.2. Will Prefetched Content Be Used?

For example, a common method of accelerating client-side response is to anticipate the user’s next action(s) and program the engine to preload (or prefetch) some content while the user decides what to do next. Depending on the think time, when the user does act, part or all of the desired response can be available on the client. This technique has long been used

Prefetching content improves application responsiveness for those users who follow the anticipated path, but slows down the application for all others.

RIAs that implement “chatty” client/server communications will not deliver a responsive user experience.

Processing scripts on the client takes much longer than running the same code on a server.

To compare the performance of a RIA with a traditional Web application you must measure equally important activities within each.

in client/server systems to improve client responsiveness; a version (Link Prefetching) is implemented by Mozilla browsers.

Preloading will certainly create a more responsive experience if the user follows the anticipated path through the application. But what if the user doesn't? Then the client engine has made unnecessary server requests, and still must react to the user's input with yet more server requests. So the design has placed extra load on the servers for no benefit.

The extra load imposed by these background requests, on behalf of what may be hundreds or even thousands of clients running the preloading application, can delay a server's responses to requests for which users are waiting. Slower responses lengthen transaction times, driving up the number of concurrent users, adding to server congestion, and further slowing down responses.

5.2.3. Unintended Communication Overheads

Having more design and implementation options also creates new opportunities for developers to make performance-related mistakes. Accidentally or deliberately, developers can implement “chatty” client/server communication styles that perform extremely slowly under some workload conditions.

Even with thorough testing, some of these problems may remain undiscovered until after the application is deployed unless applications are subjected to a systematic SLM process that includes measurement activities to identify, investigate, and fix them.

5.2.4. Will Offloaded Processing Be Faster?

Some RIA advocates also argue that since much of the processing has moved to the client, Web server processing time will be saved, making the server more responsive. However, any interpreted script will probably take several orders of magnitude longer to run on the client than it would on a server. So, whether a benefit exists will depend very much on the nature of the application.

5.2.5. Making Meaningful Comparisons

When setting out to measure an RIA, you must think carefully about the purpose of the measurement. Table 1 reminds us the reason for measuring applications is often to make comparisons. If an RIA is replacing a traditional Web application, or being compared with one, it is important not let the traditional Web approach to a business task determine what measurements are taken.

In this situation, we cannot measure only at the level of single Web pages or server interactions. Finding that it takes so many milliseconds to get a server response for request-type X is meaningless if that client/server interaction does not occur in both versions of the application.

Aleksandar Šušnjar writing of his experience developing a Rich Internet Application, wrote: “A typical example concerned how long it took to upload or download a document. Those metrics were sometimes irrelevant, depending on the application context. To make really useful performance comparisons, we had to approach the problem at a higher level—for example, ‘how long does it take to move a document from folder

“We can’t compare apples and oranges by measuring an apple or an orange. We must approach measurement as if we were comparing applications built by different developers—one a traditional Web application and the other a client/server application with a completely different UI.”
—Aleks Šušnjar

“Those who cannot remember the past are condemned to repeat it.”
—George Santayana

A to folder B?’ In a traditional Web app that task would likely require multiple clicks on separate pages, whereas with an RIA/Ajax implementation, we could do it with a single drag and drop.

So to make valid comparisons, we had to measure and compare the net effect of two rather different aspects of performance—one concerning only the computer (how many operations of type X can a machine perform per hour), the other involving human performance (how many documents can an employee move per hour). But both affected the overall conclusion. Generalizing, I would say that:

- The server that has to generate additional HTML for multiple responses in a traditional Web app will likely use many more processor cycles than the one supporting an RIA/Ajax implementation, where all the user interactions are handled on the client and the server just has to provide a service at the end.
- If the designer takes care to avoid ‘chatty’ client/server communications, network utilization will probably also be significantly lower in the second case, further improving server performance.
- Finally, if the client-side interface is well designed, a RIA should allow users to think and work faster.

In the final analysis, all these components of application performance must be weighed.” For more background, see Aleksandar Šušnjar’s Wikipedia page about RIA and Ajax.

5.3 Clarity Requires Distributed Application Design

The site must be simple and natural—it must be easy to learn, predictable and consistent. RIA technology may indeed allow developers to create more natural interfaces, but it will not guarantee a better customer experience than a traditional Web application. To quote Garrett, “the more power we have, the more caution we must use in exercising it. We must be careful to use Ajax to enhance the user experience of our applications, not degrade it”.

This advice might seem obvious, but the computing world has a tendency to forget its history, reinvent the wheel, repeat yesterday’s mistakes, and trip over previously solved problems. Rich Internet Applications are a case in point. Mainframe (thin client) computing was replaced by the client/server (fat client) model, which in turn was displaced by Web-based (thin client) applications. Now the emergence of RIAs signals a return to the fat client model again.

The difference between the client/server and RIA models is that “sneakernets” and static installation protocols have been replaced by Internet and Web technologies, which provide an almost universally available platform for distributing functions to client desktops dynamically. But the overoptimistic claims being made today for RIA technology resemble those of 15 to 20 years ago, when client/server enthusiasts predicted the mainframe’s imminent demise, which, by the way, did not happen.

As broadband penetration increases, the richer user interaction models of RIAs will inevitably take hold. Customers expect Web sites to keep up with

“Ajax development is still hard; you can’t crank out an Ajax application quickly.”
—Scott Dietzen

Monitoring RIAs in production, understanding the customer’s experience, and detecting and diagnosing problems will be harder, requiring more careful planning.

the prevailing user interface models, and penalize those that fall behind. To stay current, enterprises must not underestimate the effort needed to use the newest technologies effectively.

Scott Dietzen, President and CTO of Zimbra speaking about Ajax at a recent MIT/Stanford VLAB meeting observed that “Ajax is still hard ... you need a solid background in distributed applications.” To demonstrate the difference in the engineering skills needed to build traditional Web applications and RIAs, he revealed that Zimbra has had to interview 40 JavaScript developers for every one hired.

5.4 Utility Depends on Everyone’s Contribution

Does a site deliver the information or service customers want? While it is doubtful that switching to an RIA design would ever lessen the utility of a site, neither does it assure increased utility. The synchronous browser communication model may have been a simple one, but its simplicity has not prevented most companies from doing business successfully over the Web using a standard browser.

A few applications such as online gaming and some types of online trading required users to download and install proprietary client software, but these examples were the minority. However, a Web site’s utility diminishes over time as customer needs and expectations change. RIA technology continues to evolve, and as fast broadband connections become the norm, new interfaces and behaviors will become popular because they deliver value. The recent emergence of the Comet push model, discussed earlier, is an example.

5.4.1. The Need for Agile Processes

Consider again the RIA Behavior Model shown in Figure 2. Application *utility* is tied to user behavior, which is driven by the *clarity* of the site, the design of the client-side engine, and the *responsiveness* of interactions between client and server components. The design and implementation of those components will ultimately also determine application *availability*. Now imagine trying to focus on any one of the four aspects of usability without paying attention to the others. It can’t be done.

Since the four dimensions of usability are so intertwined, enterprises must adopt agile software development processes that keep all four usability goals in focus throughout the life cycle. Agile methods use an iterative approach to design, development and testing that involves all of the stakeholders, including marketers or business owners, Web designers, application developers, and IT staff who understand what it takes to manage a site in production. All parties must communicate and share their areas of expertise throughout the process.

These ideas did not emerge for the first time with the advent of the Rich Internet Application—they are merely descriptions of good development and service level management practices. But the added complexity of the RIA model makes it essential that we put these kinds of methods into practice.

6. Resources

Introductions

Wikipedia on Ajax: <http://en.wikipedia.org/wiki/AJAX>

Wikipedia on Rich Internet Applications:

http://en.wikipedia.org/wiki/Rich_Internet_Application

Wikipedia on Web 2.0: http://en.wikipedia.org/wiki/Web_2

Aleksandar Šušnjar's Wikipedia page about RIA and AJAX:

http://en.wikipedia.org/wiki/User:Aleksandar_%C5%A0u%C5%A1njar/RIA_and_AJAX

IDC White Paper introducing Rich Internet Applications:

http://www.macromedia.com/platform/whitepapers/idc_impact_of_rias.pdf

Flash

Papers about Flash and RIAs are available from the Adobe (formerly Macromedia)

Web site: <http://www.macromedia.com/software/flashremoting/whitepapers/>

Aflax: A JavaScript library for the Flash platform: <http://www.aflax.org/>

Developing Rich Internet Applications with Macromedia MX:

http://www.macromedia.com/devnet/studio/whitepapers/rich_internet_apps.pdf

Flash is Ajax?: http://radar.oreilly.com/archives/2005/05/flash_is_ajax_o.html

Ajax

Seminal Ajax paper by Jesse James Garrett:

<http://www.adaptivepath.com/publications/essays/archives/000385.php>

Google maps application: <http://maps.google.com/>

The Open Ajax Initiative: http://www.zimbra.com/community/open_ajax.html

Todd Spangler, Baseline Magazine: <http://www.baselinemag.com/article2/0,1540,1887817,00.asp>

MIT/Stanford VLAB meeting on Ajax:

<http://www.vlab.org/Htdocs/204.cfm?eventID=61>

Zimbra: <http://www.zimbra.com/>

Comet

"Comet: Low Latency Data for the Browser" by Alex Russell. Techniques that implement a push model of communication: <http://alex.dojotoolkit.org/?p=545>

Russell's presentation slides from the O'Reilly ETech Conference:

<http://alex.dojotoolkit.org/wp-content/LowLatencyData.pdf>

Phil Windley's write-up of Russell's talk:

http://www.windley.com/archives/2006/03/alex_russell_on.shtml

ARM

The Open Group: <http://www.opengroup.org/tech/management/arm/>

Paper describing the ARM (Application Response Measurement) standard:

http://www.opengroup.org/tech/management/arm/uploads/40/6357/ARM_4.0_paper.pdf

Software Development

Agile Software Development Methods:

http://en.wikipedia.org/wiki/Agile_software_development