

## Rich Internet Application

### Rich Internet Application

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.

RIAs typically do the following:

- Run in a web browser, or do not require software installation.
- Run locally in a secure environment called a sandbox.

### History

The term "Rich Internet Application" was introduced in a white paper of March 2002 by Macromedia (now Adobe), though the concept had existed for a number of years earlier under names such as:

- Remote Scripting, by Microsoft, circa 1998
- X Internet, by Forrester Research in October 2000
- Rich (web) clients
- Rich web application

Traditional web applications centered all activity around client-server architecture with a thin client. Under this system all processing is done on the server, and the client is only used to display static (in this case HTML) content. The biggest drawback with this system is that all interaction with the application must pass through the server, which requires data to be sent to the server, the server to respond, and the page to be reloaded on the client with the response. By using a client side technology which can execute instructions on the client's computer, RIAs can circumvent this slow and synchronous loop for many user interactions. This difference is somewhat analogous to the difference between "terminal and mainframe" and Client-server/Fat client approaches.

Internet standards have evolved slowly and continually over time to accommodate these techniques, so it is hard to draw a strict line between what constitutes a RIA and what does not. But all RIAs share one characteristic: they introduce an intermediate layer of code, often called a client engine, between the user and the server. This client engine is usually downloaded as part of the instantiation of the application, and may be supplemented by further code downloads as use of the application progresses. The client engine acts as an extension of the browser, and usually takes over responsibility for rendering the application's user interface and for server communication.

What can be done in a RIA may be limited by the capabilities of the system used on the client. But in general, the client engine is programmed to perform application functions that its designer believes will enhance some aspect of the user interface, or improve its responsiveness when handling certain user interactions, compared to a standard Web browser implementation. Also, while simply adding a client engine does not force an application to depart from the normal synchronous pattern of interactions between browser and server, in most RIAs the client engine performs additional asynchronous communications with servers.

Today's client/server architecture is different from that of a years back, some of those differences are:

- Client applications are no longer accessing databases directly. Instead they access data services embodied in the ethos of Web 2.0.
- Client/server applications do not run directly on the operating systems, but on an intermediary platform called a runtime. This makes portability easier -- a lesson learned in large part from Java.
- With broadband, it's no longer an issue to download an application that is a couple of MB or more. Distribution technologies even allow in-place updates of software right at the client machines, so distribution has become less of an issue than it was 10 years ago.

## Rich Internet Application

### Benefits

Although developing applications to run in a web browser is a much more limiting, difficult, and intricate process than developing a regular desktop application, the efforts are often justified because:

- installation footprint is smaller -- overhead for updating and distributing the application is trivial, or significantly reduced compared to a desktop or OS native application
- updates/upgrades to new versions can be automatic or transparent to the end user.
- users can use the application from any computer with an internet connection.
- many tools exist to allow off-line use of applications, such as Adobe AIR, Google Gears, and other technologies
- most RIA technologies allow the user experience to be consistent, regardless of what operating system the client.
- web-based applications are generally less prone to viral infection than running an actual executable

Because RIAs employ a client engine to interact with the user, they are:

- Richer. They can offer user-interface behaviors not obtainable using only the HTML widgets available to standard browser-based Web applications. This richer functionality may include anything that can be implemented in the technology being used on the client side, including drag and drop, using a slider to change data, calculations performed only by the client and which do not need to be sent back to the server, for example, a mortgage calculator.
- More responsive. The interface behaviors are typically much more responsive than those of a standard Web browser that must always interact with a remote server.

The most sophisticated examples of RIAs exhibit a look and feel approaching that of a desktop environment. Using a client engine can also produce other performance benefits:

- Client/Server balance. The demand for client and server computing resources is better balanced, so that the Web server need not be the workhorse that it is with a traditional Web application. This frees server resources, allowing the same server hardware to handle more client sessions concurrently.
- Asynchronous communication. The client engine can interact with the server without waiting for the user to perform an interface action such as clicking on a button or link. This allows the user to view and interact with the page asynchronously from the client engine's communication with the server. This option allows RIA designers to move data between the client and the server without making the user wait. Perhaps the most common application of this is pre-fetching data, in which an application anticipates a future need for certain data and downloads it to the client before the user requests it, thereby speeding up a subsequent response. Google Maps uses this technique to load adjacent map segments to the client before the user scrolls them into view.
- Network efficiency. The network traffic may also be significantly reduced because an application-specific client engine can be more intelligent than a standard Web browser when deciding what data needs to be exchanged with servers. This can speed up individual requests or responses because less data is being transferred for each interaction, and overall network load is reduced. However, over-use of asynchronous calls and pre-fetching techniques can neutralize or even reverse this potential benefit. Because the code cannot anticipate exactly what every user will do next, it is common for such techniques to download extra data, not all of which is actually needed, to many or all clients.

### Shortcomings

Shortcomings and restrictions associated with RIAs are:

- Sandboxing. Because RIAs run within a sandbox, they have restricted access to system resources. If assumptions about access to resources are incorrect, RIAs may fail to operate correctly.

## Rich Internet Application

- Enabled scripting. JavaScript or another scripting language is often required. If the user has disabled active scripting in their browser, the RIA may not function properly, if at all.
- Client processing speed. To achieve platform independence, some RIAs use client-side scripts written in interpreted languages such as JavaScript, with a consequential loss of performance (a serious issue with mobile devices). This is not an issue with compiled client languages such as Java, where performance is comparable to that of traditional compiled languages, or with Flash, Curl, or Silverlight, in which the compiled code is run by a Flash, Curl or Silverlight plugin.
- Script download time. Although it does not have to be installed, the additional client-side intelligence (or client engine) of RIA applications needs to be delivered by the server to the client. While much of this is usually automatically cached it needs to be transferred at least once. Depending on the size and type of delivery, script download time may be unpleasantly long. RIA developers can lessen the impact of this delay by compressing the scripts, and by staging their delivery over multiple pages of an application.
- Loss of integrity. If the application-base is X/HTML, conflicts arise between the goal of an application (which naturally wants to be in control of its presentation and behaviour) and the goals of X/HTML (which naturally wants to give away control). The DOM interface for X/HTML makes it possible to create RIAs, but by doing so makes it impossible to guarantee correct function. Because a RIA client can modify the RIA's basic structure and override presentation and behaviour, it can cause failure of the application to work properly on the client side. Eventually, this problem could be solved by new client-side mechanisms that granted a RIA client more limited permission to modify only those resources within the scope of its application. (Standard software running natively does not have this problem because by definition a program automatically possesses all rights to all its allocated resources).
- Loss of visibility to search engines. Search engines may not be able to index the text content of the application.
- Dependence on an Internet connection. While the ideal network-enabled replacement for a desktop application would allow users to be "occasionally connected", wandering in and out of hot-spots or from office to office, special platforms (e.g. Adobe AIR, Google Gears) are required to allow off-line functionality for RIA applications.
- Accessibility. There are a lot of known Web accessibility issues in RIAs, most notably the fact that screen readers have a hard time detecting dynamic changes (caused by JavaScript) in HTML content. The WAI-ARIA suite provides a solution for this problem (via Live Regions); as well as providing a way of adding critical role, state and property information to DHTML based user interfaces.
- No deployment. Rich internet applications cannot be deployed the way traditional desktop applications can be deployed. Except for Adobe AIR technology.

### Software development complications

The advent of RIA technologies has introduced considerable additional complexity into Web applications. Traditional Web applications built using only standard HTML, having relatively simple software architecture and being constructed using a limited set of development options, is relatively easy to design and manage. For the person or organization using RIA technologies to deliver a Web application, their additional complexity makes them harder to design, test, measure, and support.

Use of RIA technology poses several new service level management (SLM) challenges, not all of which are completely solved today. SLM concerns are not always the focus of application developers, and are rarely if ever perceived by application users, but they are vital to the successful delivery of an online application. Aspects of the RIA architecture that complicate management processes are:

- Greater complexity makes development harder. The ability to move code to the client gives application designers and developers far more creative freedom. But this in turn makes development harder, increases the likelihood of defects (bugs) being introduced, and complicates software testing activities. These complications lengthen the software development process, regardless of the particular methodology or process being employed. Some of these issues may be mitigated through the use of a web application framework to standardize aspects of RIA design and development. However, increasing complexity in a software solution can complicate and lengthen the testing process, if it increases the number of use cases to be tested. Incomplete testing lowers the application's quality and its reliability during use.

## Rich Internet Application

One could argue that the above comment applies not specifically to RIA technology, but to complexity in general. For example, that exact same argument was used when Apple and Microsoft independently announced the GUI in the 1980s, and perhaps even when Ford announced the Model T. Nonetheless, humans have shown a remarkable ability to absorb technological advances for decades, if not centuries.

- RIA architecture breaks the Web page paradigm. Traditional Web applications can be viewed as a series of Web pages, each of which requires a distinct download, initiated by an HTTP GET request. This model has been characterized as the Web page paradigm. RIAs invalidate this model, introducing additional asynchronous server communications to support a more responsive user interface. In RIAs, the time to complete a page download may no longer correspond to something a user perceives as important, because (for example) the client engine may be prefetching some of the downloaded content for future use. New measurement techniques must be devised for RIAs, to permit reporting of response time quantities that reflect the user's experience. In the absence of standard tools that do this, RIA developers must instrument their application code to produce the measurement data needed for SLM.
- Asynchronous communication makes it harder to isolate performance problems. Paradoxically, actions taken to enhance application responsiveness also make it harder to measure, understand, report on, and manage responsiveness. Some RIAs do not issue any further HTTP GET requests from the browser after their first page, using asynchronous requests from the client engine to initiate all subsequent downloads. The RIA client engine may be programmed to continually download new content and refresh the display, or (in applications using the Comet approach) a server-side engine can keep pushing new content to the browser over a connection that never closes. In these cases, the concept of a "page download" is no longer applicable. These applications are commonly known as refreshless. These complications make it harder to measure and subdivide application response times, a fundamental requirement for problem isolation and service level management. Tools designed to measure traditional Web applications may -- depending on the details of the application and the tool -- report such applications either as a single Web page per HTTP request, or as an unrelated collection of server activities. Neither description reflects what is really happening at the application level.
- The client engine makes it harder to measure response time. For traditional Web applications, 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 packet sniffer can read and interpret packet-level data, and infer the user's experience of response time by tracking HTTP messages and the times of underlying TCP packets and acknowledgments. But the RIA architecture reduces the power of the packet sniffing approach, because the client engine breaks the communication between user and server into two separate cycles operating asynchronously -- a foreground (user-to-engine) cycle, and a background (engine-to-server) cycle. 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 the most complete RIA measurements can only be obtained using tools that reside on the client and observe both cycles.

### Current status of development

RIAs are still in the early stages of development and user adoption. There are a number of restrictions and requirements that remain:

- Browser adoption: Many RIAs require modern web browsers in order to run. Advanced JavaScript engines must be present in the browser as RIAs use techniques such as XMLHttpRequest for client-server communication, and DOM Scripting and advanced CSS techniques to enable the rich user interface
- Web standards: Differences between web browsers can make it difficult to write an RIA that will run across all major browsers. The consistency of the Java platform, particularly after Java 1.1, makes this task much simpler for RIAs written as Java applets.
- Development tools: Some Ajax Frameworks and products such as Curl, Adobe Flex and Microsoft Silverlight provide an integrated environment in which to build RIAs.

## Rich Internet Application

- Accessibility concerns: Additional interactivity may require technical approaches that limit applications' accessibility.
- User adoption: Users expecting standard web applications may find that some accepted browser functionality (such as the "Back" button) may have somewhat different or even undesired behaviour?

### List of RIA Platforms / Approaches

This section does not cite any references or sources. (May 2008)

Please help improve this section by adding citations to reliable sources. Unverifiable material may be challenged and removed.

This article or section is written like an advertisement.

Please help rewrite this article from a neutral point of view.

For blatant advertising that would require a fundamental rewrite to become encyclopedic, use `{{db-spam}}` to mark for speedy deletion. (May 2008)

### Adobe Flash, Adobe Flex and Adobe AIR

Adobe Flash is another way to build rich internet applications. This technology is cross-platform and quite powerful to create an application UI.

Adobe Flex is a framework that provides the option for a developer to build user interfaces by compiling MXML, an XML based interface description language. This Adobe Flex framework is compiled and turned into a SWF file to be run in the Adobe **Flash player**.

Adobe has also released Adobe AIR (Adobe Integrated Runtime), which is a runtime platform that is independent to the hosting operating system. Adobe AIR allows for Flash Player and Ajax applications to be deployed/installed onto a user's desktop as one would a desktop application.

### Backbase

Backbase is an AJAX-based RIA framework for standards-based RIA development. The client framework runs in all common browsers and can be used in combination with existing presentation tier technologies using Java, PHP, .NET, Perl and XML/XSLT. Backbase Visual Ajax Builder is a WYSIWYG visual RIA editor and is available as an Eclipse plug-in. Backbase offers a JSF Edition that enables standards based RIA development for Java Server Faces (JSF).

### Curl

Curl began as a research project at MIT in the late 1990s. It was commercialized by Curl, Inc., which made a first release in 2000. The current release is version 6.0, available for Windows, Linux, and Mac clients. There is no required server side component; any web server can be used. Curl provides a wealth of features, but in an easily accessible form that allows users with a variety of backgrounds to work at different levels of complexity, from simple HTML-like formatting to sophisticated object-oriented programming. The free Curl plugin is on the order of 10 MB in size, but can be installed quickly by broadband users. One advantage of the plugin architecture is that Curl applets work the same on every platform, with any browser, with the exception of a few platform specific APIs. Curl applets are compiled to machine code for fast execution, and various caching mechanisms speed up the loading of applets. Curl is free to use for non-commercial and some commercial uses (see licensing). A Pro version is available which provides additional enterprise class capabilities.

Curl supports the software engineering of large complex applications, and may be more efficient in terms of the amount of code needed to write applications. A comparative study by Sonata found that Curl applications required about a third less code than Adobe Flex and Ajax.

## Rich Internet Application

Curl has had a feature of "detached applets" for several years, which is a web deployed applet which can run independently of a browser window, similarly to Adobe AIR. Curl applets can also be written so that they will run when disconnected from the network. In fact, the Curl IDE is an application written in Curl.

### The Dojo Toolkit

The Dojo Toolkit is a modular open source JavaScript toolkit (or library), designed to ease the rapid development of JavaScript- or Ajax-based applications. It was started by Alex Russell in 2004.

Open source development tools for Dojo include the Aptana IDE and the WaveMaker Visual Ajax Studio.

The Dojo Foundation is a non-profit organization founded to help open source projects. Its primary goals are to aid in adoption by companies, and encourage projects in the foundation to collaborate with one another.

### Google's GWT framework

Google released the 'Google Web Toolkit' or GWT in 2006 which allows the development and testing of JavaScript-based AJAX RIAs using the Java language. The GWT programming paradigm centers around coding user interface logic in Java (similar to the Swing/AWT model), and then executing the GWT compiler to translate this logic into cross-browser-compatible JavaScript. Designed specifically for Java developers, GWT enables Java programming, refactoring, debugging and unit testing of RIAs using existing tools (e.g. Eclipse), without requiring knowledge of JavaScript or specific browser DOM irregularities (although hand-written JavaScript can still be used with GWT if desired).

### Java applets

Java applets run in standard HTML pages and generally start automatically when their web page is opened with a modern web browser. Java applets have access to the screen (inside an area designated in its page's HTML), as well as the speakers, keyboard and mouse of any computer their web page is opened on, as well as access to the Internet, and provide a sophisticated environment capable of real time applications.

### Java applications

Java based RIAs can be launched from within the browser or as free standing applications via Java Web Start which integrate with the desktop. Java RIAs can take advantage of the full power of the Java platform to deliver rich functionality, 2D & 3D graphics, and off-line capabilities.

Java is widely adopted and there is a vast range of both commercial and open source libraries available for the platform, making it possible to include support for virtually any system, including native applications via JNI or JNA. When it comes to RIAs, Java's main weakness is its multimedia support. Java 6 Update N improves some features that have hindered the use of Java for RIAs including startup time and download size, and Sun may even include new multimedia support in this release (due Q2,2008).

Numerous frameworks for Java RIAs exist, including XML-based frameworks such as Swixml, or Canoo's, UltraLightClient.

### JavaFX

Sun Microsystems has announced JavaFX, a family of products based on Java technology designed to provide a consistent experience across a wide variety of devices including desktops, (as applets and stand-alone clients) set-top boxes, mobile devices, and Blu-Ray players. The JavaFX platform will initially comprise JavaFX Script and JavaFX Mobile. Invented by Sun Software Engineer Chris Oliver as a skunk works project, JavaFX Script enables rapid development of rich 2D interfaces using a declarative syntax similar to SVG. Sun plans to release JavaFX Script as an open source project, but JavaFX Mobile will be a commercial product available through an OEM license to carriers and handset manufacturers.

## Rich Internet Application

### JavaScript / Ajax

The first major client side language and technology available with the ability to run code and installed on a majority of web clients was JavaScript. Although its uses were relatively limited at first, combined with layers and other developments in DHTML it has become possible to piece together an RIA system without the use of a unified client-side solution. Ajax is a new term coined to refer to this combination of techniques and has recently been used most prominently by Google for projects such as Gmail and Google Maps. However, creating a large application in this framework is very difficult, as many different technologies must interact to make it work, and browser compatibility requires a lot of effort. In order to make the process easier, several open source Ajax Frameworks have been developed, as well as commercial frameworks.

### LeeBeLLuL

LeeBeLLuL is a RIA style application that uses IronPython in the browser and InfoPath in the form design.

### Microsoft ActiveX controls

Embedding ActiveX controls into HTML is a very powerful way to develop rich Internet applications. However they are only guaranteed to run properly in Internet Explorer, since no other web browser at this time supports ActiveX controls. In addition, ActiveX controls are not executed in sandbox. Therefore, they are potential targets for computer viruses and malware making them high security risks.

At the time of this writing, the Adobe Flash Player for Internet Explorer is implemented as an ActiveX control for Microsoft environments, as well as in multi-platform Netscape Plugin wrappers for the wider world. Only if corporations have standardized on using Internet Explorer as the primary web browser, is ActiveX per se a good choice for building corporate applications.

### Microsoft Silverlight

Microsoft Silverlight, which can be considered a subset of Windows Presentation Foundation (WPF) allows developers to develop RIA. Like Windows Presentation Foundation, Silverlight uses XAML.

Client machines need to install a small (about 2MB) plug-in (Silverlight Runtime) in order to be able to play Silverlight contents. At this time, Silverlight client for Windows and Mac OS X is available from Microsoft. A third-party open-source plug-in called Moonlight is also available for Linux. Microsoft has also promised to broaden the range of supported clients.

During the opening keynote at MIX08 conference in Las Vegas, the first beta of Silverlight 2 was shown running on a Nokia S60 platform as well as Microsoft Windows Mobile 6 devices.

### Mozilla Prism

Mozilla Prism is a product in development which integrates web applications with the desktop, allowing web applications to be launched from the desktop and configured independently of the default web browser

### OpenLaszlo

OpenLaszlo is an open source rich Internet application framework developed by Laszlo Systems Inc.. The OpenLaszlo server compiles programs written in the LZX language (a mixture of XML tags and JavaScript) into either DHTML (commonly known as AJAX now) or Adobe Flash bytecode, currently supporting Flash7 and Flash8. The server - which originally was a proprietary software - was open sourced in October 2004 under the Common Public License. OpenLaszlo is the only rich Internet application platform which is capable of compiling into two different runtimes from the same code

## Rich Internet Application

base (although GWT will compile into browser-specific java script from Java).

### REBOL 2.6 and Seaside for Smalltalk

Available alternatives to Java for RIA include abstract machines for REBOL and Smalltalk programming languages. REBOL does not require a browser and Seaside for Smalltalk uses a minor extension to Smalltalk to provide a much richer web experience. Both are far more mature than more familiar options and as old or older than Java and the JVM.

### reBOX

reBOX is a web application API based on JavaScript developed by iCUBE Network Solutions. It allows the user to create web applications in the browser that look like desktop applications. The API offers different controls (Button, Tree, TableView, etc.) and the possibility for server communication by using AJAX and accessing Web Services. An example which uses the reBOX API can be found at Browser OS.

### Seam

Seam is a powerful open source development platform for building rich Internet applications in Java. Seam integrates technologies such as Asynchronous JavaScript and XML (AJAX), JavaServer Faces (JSF), Java Persistence (JPA), Enterprise Java Beans (EJB 3.0) and Business Process Management (BPM) into a unified full-stack solution, complete with sophisticated tooling. Seam has been designed from the ground up to eliminate complexity at both architecture and API levels. It enables developers to assemble complex web applications with simple annotated plain Java classes, a rich set of UI components, and very little XML. Seam's unique support for conversations and declarative state management eliminates a whole class of bugs common in traditional web applications.

## RIA Related Topics

### RIA with real-time push

Traditionally, web pages have been delivered to the client only when the client requested for it. For every client request, the browser initiates an HTTP connection to the web server, which then returns the data and the connection is closed. The drawback of this approach was that the page displayed was updated only when the user explicitly refreshes the page or moves to a new page. Since transferring entire pages can take a long time, refreshing pages can introduce a long latency.

### Demand for localized usage of RIA

With the increasing adoption and improvement in broadband technologies, fewer users experience poor performance caused by remote latency. Furthermore one of the critical reasons for using an RIA is that many developers are looking for a language to serve up desktop applications that is not only desktop OS neutral but also installation and system issue free.

RIA running in the ubiquitous web browser is a potential candidate even when used standalone or over a LAN, with the required webserver functionalities hosted locally.

Client-side functionalities and development tools for RIA needed

With client-side functionalities like JavaScript and DHTML, RIA can operate on top of a range of OS and webserver functionalities.

## Rich Internet Application

### User interface languages

Instead of HTML/XHTML, new user interface markup languages can be used in RIAs. For instance, the Mozilla Foundation's XML-based user interface markup language XUL - this could be used in RIAs though it would be restricted to Mozilla-based browsers, since it is not a de facto or de jure standard. The W3C's Rich Web Clients Activity has initiated a Web Application Formats Working Group whose mission includes the development of such standards. The original DARPA project at MIT which resulted in the W3C also resulted in the web content/programming language Curl which is now in version 6.0.

RIA's user interfaces can also become richer through the use of scriptable scalable vector graphics (though not all browsers can render those natively yet) as well as Synchronized Multimedia Integration Language (SMIL).

### Other techniques

RIAs could use XForms to enhance their functionality.

Using XML and XSLT along with some XHTML, CSS and JavaScript can also be used to generate richer client side UI components like data tables that can be resorted locally on the client without going back to the server. Mozilla and Internet Explorer browsers both support this.

The Omnis Web Client is an ActiveX control or Netscape plug-in which can be embedded into an HTML page providing a rich application interface in the end-user's web browser.

Appcelerator is an open source platform for developing rich Internet applications using a service-oriented architecture and standards such as HTML, CSS and JavaScript. Appcelerator applications can integrate automatically with several different integration points on the service tier using Java, PHP, Python, .NET, Perl and Ruby on Rails. Appcelerator applications can use pre-built widgets to assemble high quality RIAs. Appcelerator is licensed under the GNU GPL version 2 License.