



THE IT ARCHITECTS

Rich Internet Applications

Frameworks evaluation

Document Reference	TSL-SES-WP-0001
Date	4 January 2008
Issue	1
Revision	0
Status	Final

Document Change Log

Version	Pages	Date	Reason of Change
1.0 Draft	17	04/01/08	Initial version

Table of Contents

1	Introduction	1
1.1	Purpose	1
1.2	Scope.....	1
2	RIA vs Stand-alone Desktop applications	2
2.1	Benefits	3
2.2	Disadvantages and shortcomings	3
2.3	Decision criteria for choosing RIA technology.....	4
3	Frameworks Evaluation	6
3.1	Eclipse RAP (Rich Ajax Platform).....	6
3.2	JSF.....	7
3.2.1	<i>IBM RAD</i>	<i>8</i>
3.2.1.1	Web application creation	8
3.2.1.2	Portals and Portlets creation	8
3.2.2	<i>Netbeans Visual Web Editor.....</i>	<i>9</i>
3.2.3	<i>IceFaces Components</i>	<i>9</i>
3.3	OpenLaszlo.....	9
3.4	Nexaweb	9
3.5	Tibco General Interface	10
3.6	Google Web Toolkit (GWT)	10
3.7	Echo 2.....	10
3.8	Adobe Flex.....	11
	Conclusions.....	12

1 Introduction

1.1 Purpose

The purpose of this document is firstly to present RIA and to describe the advantages of RIA-based applications in comparison with the traditional web and desktop applications, and secondly to cover in the second part of the document an evaluation of different RIA technology frameworks.

1.2 Scope

This document pretends to give an overview of RIA and a brief evaluation of existing RIA frameworks from a generic point of view. When RIA adoption is feasible in most application cases, an evaluation of the different UI application requirements (widgets, responsiveness, etc...) and available products is required in order to find the right RIA approach.

2 RIA vs Stand-alone Desktop applications

Traditional Desktop applications characterise for a rich user experience and complex UI (menus, multi-window, multi-tabbed, etc...) that runs fast and are deployed locally on the end user platform. They start becoming problematic when the user community is not localised and the application has to be used across networks with different security constraints. Additionally, when the user community is large, the installation, maintenance and access flexibility becomes difficult as new factors as compatible hardware, OS, libraries, etc... might need to be taken into account per user.

Web applications, in the other hand, provided a way to overcome those problems introducing a thin client-server architecture based on internet standards and protocols. A simple web browser replace the presentation layer in the application and the rest runs centralised on a server, so with connection to the network is enough for anybody to access the "application" from anywhere. No need for installations and updates on the client system. The major problems with traditional web applications are that they are very static, all application interaction must pass through the server, requiring data to be sent to the server and the server to respond and they lack a rich UI user.

Rich Internet Applications try to bridge the two worlds, bringing the best of each one of them, introducing an additional client layer (client engine) technology that allows the execution of code on the client side to provide better performance and a rich UI user experience.

In short, the major justification points for RIA are:

- No need for installation/updates. Updating and distributing the application is an instant, automatically handled process. This is a big advantage when deploying an application where flexible and extensive access is required.
- End users can use the application from any computer with a network connection and access to the server, and usually regardless of what OS that computer is running.
- Applications user community can be easily extended even through security-managed areas as standard protocols are used.
- Cross browser and cross-platform.

2.1 Benefits

The main benefits of RIA, already drafted above, compared with traditional desktop applications are:

- *Easy to deploy and maintain.* Being a web application, a web browser with the client engine is enough to access the application. Thus, the updates are intrinsically automatic.
- *They are richer.* They can offer close to desktop applications user-interface behaviours. This richer functionality may include anything that can be implemented in the technology being used on the client side, including multi-window, multi-tab, drag and drop, themes, calculations on the client and many more.
- *More responsive.* The behaviour of the interface is much more responsive than the traditional web and can be very close to the traditional desktop applications:
 - *Asynchronous communication.* It allows to the client to asynchronously and without user intervention to interact with the server (e.g. prefetching needed data).
 - *Network usage optimisation.* The network traffic is also significantly reduced as the required data exchange between the client and the server are decreased to the strictly needed.
- *Secure.* Normally, RIA client engines run on a sandbox, restricting the harm that the applications can do compared with installed applications.

2.2 Disadvantages and shortcomings

Not a single technology provides the perfect solution for all scenarios, nor RIA. The technology, while already quite mature, is still evolving presenting the following disadvantages:

- *UI Richness.* While RIA is meant to provide a better UI user experience it normally cannot compete with traditional desktop applications in all fronts (mostly in the native graphical area).
- *Sandbox.* One of the limitations of running in a sandbox is that they have restricted access to system resources.
- *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. This is not an issue with compiled client languages like Java, where performance is comparable to that of traditional compiled languages, or with Flash, in which the bulk of the operations are performed by the native code of the Flash player.

- *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 from the network, currently the typical RIA requires network connectivity.

2.3 Decision criteria for choosing RIA technology

The typical decision tree when choosing the right RIA approach for a given application is depicted below:

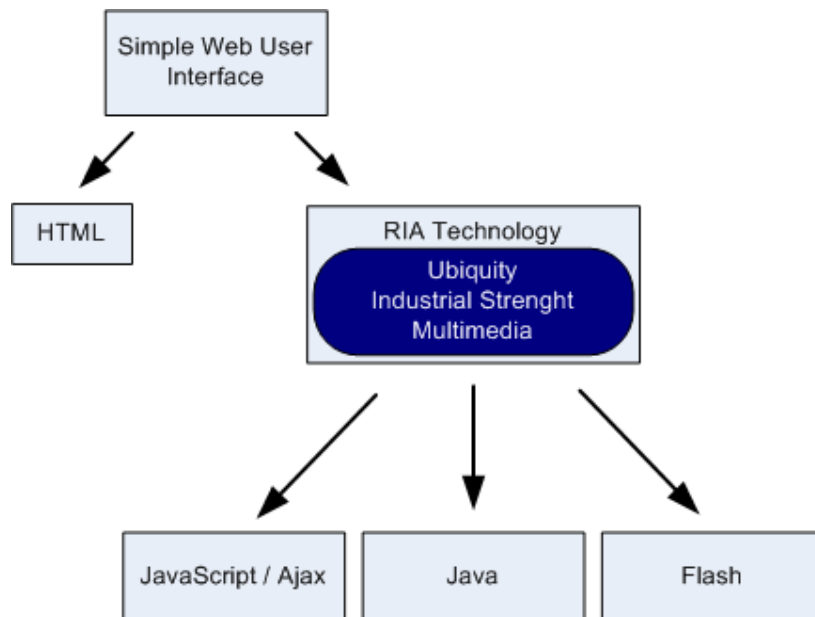


Figure 1. RIA Technologies Decision tree

In the present document we focus on JavaScript/Ajax and Flash-type frameworks. Additionally, depending on the amount of logic that is executed in the client, the different solutions separate in Fat and Thin RIA solutions as depicted in the diagram below:

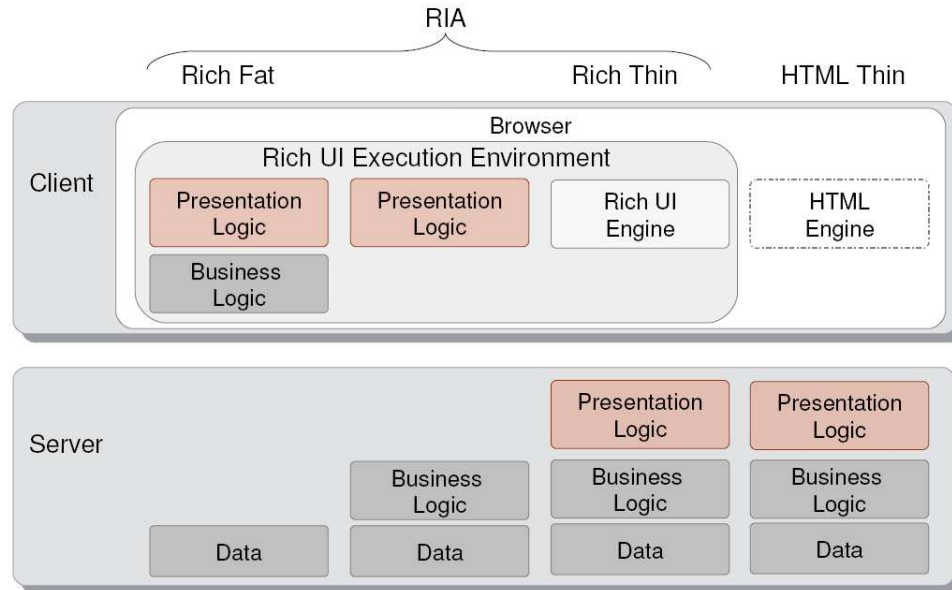


Figure 2. RIA Technologies architectures

3 Frameworks Evaluation

In this section is presented the list of evaluated frameworks describing advantages and disadvantages of each one of them.

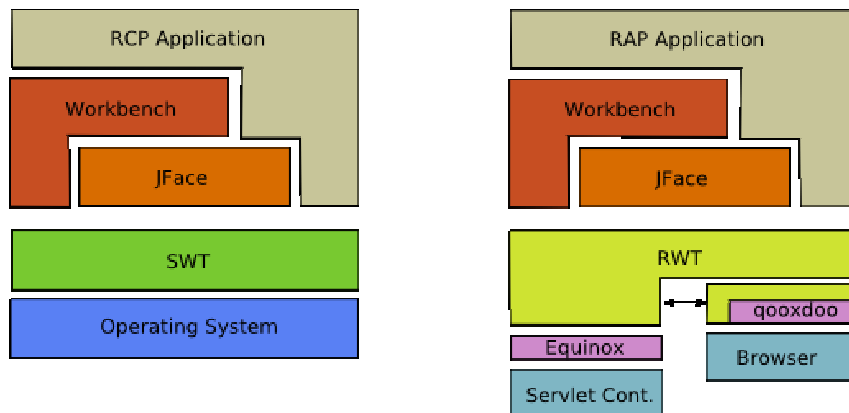
3.1 Eclipse RAP (Rich Ajax Platform)

Eclipse RAP is the platform for RIA of the eclipse foundation that permits the development of SWT/Jfaces interfaces on the Web (HTML + Javascript + Ajax).

In the area of enterprise software RCP is already firmly established as a client application platform. Replacing or extending existing Rich-Client-Applications with web front-ends has traditionally required a significant investment. RAP provides a fast-path bridge between the application development with RCP and the increasingly need for Rich web applications.

RAP does that allowing the developers to keep the same programming paradigm they know for RCP, doing the abstraction of the web dependencies and details.

The diagram below shows the difference between the RCP and RAP architectures.



As it can be noted, the same architecture is kept on top of the graphical API provided for each of them, SWT for RCP and RWT for RAP. RCP plug-in architecture and extension points are therefore still available. The workbench and Jface in RAP are customised versions of the ones existing for RCP.

The Widget API in RAP, RWT (RAP Widget Toolkit), keeps the same widget hierarchy that SWT but implementing a subset of SWT.

1. RWT renders the widgets by pushing instructions to the browsers, where they are interpreted by qooxdoo, a specialized JavaScript library. By executing on the browser, qooxdoo draws the UI, responds to events, and propagates events back to the server.
2. RAP utilizes the basic components of Eclipse in order to execute RCP code in a servlet container. Equinox, the Eclipse OSGi implementation supports the execution in a servlet Container.
 - As the main difference resolves to different rendering techniques, it results that the RCP code already written can be almost without change be migrated to a RAP application.
 - The only main difference is that the developer must think in terms of a multi-user environment, as this concept is introduced intrinsically when moving to Web applications (e.g. Application singletons might need to be moved to session singletons).

The current major limitations of Eclipse RAP are:

- RWT only offers a subset of SWT components. Some components are currently not migrated for obvious limitations, e.g. direct canvas drawing, keystroke event handling.
- It is not possible to use and RWT editor integrated with Eclipse due to libraries conflicts, but it is possible to design with the SWT in a different instance of Eclipse and later on import the generated code in the RAP instance.

3.2 JSF

Java Server Faces is a new standard created by Sun for the development of component-based Web applications based on the MVC pattern.

This is the most noticeable advantage. As it defines a standard, multiple providers have created custom components, guaranteeing a broader support for this technology. There exist a big number of OpenSource and proprietary components, a comparative of the different libraries of JSF Ajax can be found in:

<http://www.jsfmatrix.net/>

The major disadvantages of JSF are:

- It has a long learning curve.
- The development of custom components is very costly, as the defined standard identifies five phases for the components that facilitates the interoperability and flexibility but complicates the development.
- The components are targeted in their majority to Web pages, not to more Swing-type applications.

3.2.1 IBM RAD

The IBM RAD environment provides a framework for the creation of web applications and portals. Among others, the main functionalities are:

- HTML, Javascript and JSP Edition
- Visual support for JSF components
- Portal creation support
- Portlets creation support

3.2.1.1 Web application creation

The support for the creation of Web applications is based on a visual HTML and JSPs editor with specific JSF components support. Although the interface is visual is not intuitive and the JSF components offered and limited and more basic that those offered by other libraries. Moreover, none of the Faces RAD components supports Ajax, but RAD v7 provides of a mechanism to refresh components, as complete html pieces. More information in the following url:

http://www.ibm.com/developerworks/rational/library/06/1205_kats_rad2/

3.2.1.2 Portals and Portlets creation

The tool to create portals is divided in two parts:

- **Portal creation:** The tools allows, using templates previously defined in HTML, create a portal skeleton to integrate portlets. It is possible to align the portlets in columns and rows, as well as in combinations of them.
- **Portlets creation:** The visual tool permits the creation of portlets, being basically the same tool as the one described in section 3.2.1.1, not adding any specific support.

3.2.2 Netbeans Visual Web Editor

This editor, integrated within Netbeans is more powerful and flexible than IBM RAD, facilitating the complete integration of advance components coming with IceFaces (covered next).

3.2.3 IceFaces Components

Icefaces is a JSF framework that provides advanced Ajax components. It differentiates from the rest of JSF-AJAX components libraries that it permits partial renderisation of the pages, therefore allowing for instance the creation of components based on events. It also allows the integration of Portlets, so the integration with portals is simplified.

IceFaces is currently the most advanced component library but the frameworks are evolving really fast and it is recommended to evaluate at decision time the status of the rest of libraries.

3.3 OpenLaszlo

OpenLaszlo is a very powerful open-source platform for rich Internet applications that introduces an additional level of abstraction, where applications are written in the generic XML language (LZX). That language can be then compiled to several runtime targets, including both flash-type and Ajax type applications.

This framework is an example of an architecture that can deploy different models of RIA (thin or fat) depending of the necessities of the applications.

The available components list is complete and additional development is being performed in this area. However, the WYSIWYG designing tool developed as an Eclipse plug-in (IDEIaszlo) has been archived and no further development is being performed.

3.4 Nexaweb

Similar but more powerful than OpenLaszlo, Nexaweb is a commercial platform to build RIA applications that uses a specific XML language (XAL) and POJOs to build the applications, abstracting from the underlying technologies. In this way, Nexaweb also presents a Unified Client Framework that allows application deployment using different technologies like Ajax, Java, Flash,... depending of the requirements.

The Ajax implementation uses Dojo to display the widgets.

Additionally, the components base is quite rich and there exist a complete design tool (Nexaweb Studio) integrated with Eclipse to build applications.

3.5 Tibco General Interface

Tibco General Interface is a Open Source (BSD) frameworks for the creation of RIA applications. It comes with a very powerful interface editor that shows the power of the components that it includes.

The development interface is very intuitive allowing the fast creation of prototypes.

It must be noted that the framework is a thick RIA solution, and everything is executed in the client, communicating with the services via Web Services, HTTP or JSON.

The major disadvantage is that the event programming and component iteration must be done in Javascript, although it provides a set of utilities to program with a higher level of abstraction.

3.6 Google Web Toolkit (GWT)

GWT is the RIA framework developed by Google. The core of the framework is a Java-to-Javascript compiler, which allows the development of application in Java to later be translated to Javascript.

Currently, the library doesn't contain many components and those included are quite basic. However, more advanced libraries are appearing in the market covering the lack of advanced components of GWT (eg: <http://mygwt.net/> or <http://code.google.com/p/gwt-ext>), so it is expected that the evolution to a more complete framework in a short/mid term.

Worth to mention is the existence of a Eclipse plugin to visually design interfaces with GWT. The editor is called GWT Designed (<http://www.instantiations.com/gwtDesigner>), but at this moment is currently immature to be used in normal development.

3.7 Echo 2

Echo 2 is yet another Ajax framework to develop component oriented Web applications.

It differentiates with GWT in the interaction with JavaScript. GWT implements a subset of the Java API in Javascript, while in Echo 2 the components are controlled from the

server. The server generates Javascript and DHTML to renderise the components based on events.

Even if the technology is robust, it has not received much support from the developers community as GWT or Eclipse RAP.

3.8 Adobe Flex

Adobe Flex is an example commercial RIA framework based on proprietary Flash technology.

Initially, Flex required a server J2EE component but the most recent versions support the creation of static files that are compiled and can be distributed inline without server license. This type of files can be used with multiple server-side technologies, as it communicates using standard protocols and transport mechanisms (e.g. XML, WS or JSON).

Flex is on of the new generation frameworks for the web. These frameworks drop completely HTML and Javascript technologies to use more powerful but proprietary technologies. Another examples in this area are:

- **Microsoft Silverlight:** Microsoft solution for multimedia-type RIA interfaces.
- **Sun JavaFX:** Sun answer for this technology that provides an evolution of Java to Web interfaces.
- **OpenLazlo**, covered in section 3.3.

The evaluation of this commercial framework and its designed tool (Flex Builder, based on Eclipse) has shown to be very mature, easy to use and powerful. Moreover, the creation of new components is much more simple than the rest of platforms.

Conclusions

As internet standards become the preferred delivery mechanism for dynamic data driven applications and content management systems, there is a growing demand to migrate traditional desktop application to a more flexible, dynamic and rich content web applications to provide a more effective and efficient way to present and process complex visual information.

Currently, we are in a moment of transition of traditional web Technologies (HTML, JavaScript, CSS) that have shown to be insufficient to the necessities of the market. The new technologies provide a better support and higher level of abstraction for web applications that can in many cases replace, the traditional desktop applications, with the advantages that this brings along.

RIA applications are a reality in many business areas. There are a variety of options (Ajax, Java, Flash) to implement RIA applications depending of the requirements of the application, with some technologies already mature and with a fast evolution.

A. Other AJAX Frameworks

There are a high number of Ajax frameworks that either are not well known or are difficult to use. For reference reasons, they are listed in this section, as some of them might evolve and be more mature in the near future.

In the following URL there is a complete list of available AJAX Frameworks:

<http://ajaxpatterns.org/wiki/index.php?title=AJAXFrameworks>

- **Thin Wire (OpenSource)**

It is programmed using Java and the events and translated into Javascript/Html dynamically (like Echo2).

<http://www.thinwire.com/>

- **ExtJS (OpenSource)**

It is a Javascript framework that extends JQuery, Prototype and YUI. It provides more advanced components, but requires a high level of Javascript knowledge.

<http://extjs.com>

- **Qooxdoo (OpenSource)**

It is a javascript frameworks. It is used by Eclipse RAP to renderize the components in the browser side.

<http://qooxdoo.org>

- **Java2Script (OpenSource)**

It compiles Eclipse RCP applications into Javascript. It is not as evolved as Eclipse RAP and it is known to be slower than RAP.

<http://j2s.sourceforge.net/>

- **Bindows (Commercial)**

Provides advanced components as Table, Tree and TreeTable. It supports OLAP style data representation. It uses a mixture of XML and Javascript programming, using a special compiler to process those XML and Javascript files into the final Javascript.

<http://bindows.net>