RAP vs GWT
Which AJAX technology is for you?

by Mark Russell and Dan Rubel
Overview

• Introduction
  Who we are
• Demo
  Example apps, UI elements, Styles and Graphing
• Development
  Architecture and Debugging
• Communication
  Browser interaction and RPC
• Build and Deployment
  Automation, process and server infrastructure
• Performance
  Measuring and tuning hosted apps
• Unit Testing
  Development and performance comparison
• Summary
Introduction

- Mark Russell
  - Co-architect of the Instantiations continuous build and deployment system
  - Build master with 14 years experience developing large and small build systems for large corporate systems and small open source projects
  - Developer on several Instantiations products such as CodePro and RCPDeveloper
  - Eclipse committer

- Dan Rubel
  - Chief Technology Officer for Instantiations
  - Developer & co-architect of several Instantiations products such as CodePro, RCPDeveloper and WindowTester
  - Author of several Eclipse articles for ACM Queue, Eclipse Review and WebSphere Advisor
  - Co-author of Eclipse: Building Commercial Quality Plug-ins
  - Eclipse committer
Instantiations

• Founded in 1997, Headquarters in Portland, OR

• Leading edge development tools for professional Java developers
  ♦ VA Assist Enterprise (Smalltalk & Java)
  ♦ CodePro Product Line (AnalytiX, Profiler, PlusPak, etc.)
  ♦ WindowBuilder Pro (SWT, Swing & GWT Designer)
  ♦ RCP Developer (Help Composer, RCP Packager)
  ♦ WindowTester Pro (SWT & Swing)

• Extensive Eclipse experience
  ♦ One of first IBM partners briefed on Eclipse in 1999
  ♦ Technical development since January 2000 (>8 years)
  ♦ First commercial Eclipse & WSAD add-on (Nov. 2001)
  ♦ First product certified as “Ready for WebSphere Studio”
  ♦ Eclipse Foundation member & major contributor
  ♦ Eclipse Projects: EPP, KOI (Collaboration), Pollinate (Beehive)
Similarities

• Rich browser based UI based on AJAX
• Developers write Java code (not JavaScript)

... but that’s where the similarities end
Demo – RAP application in action

Main “window” can be dragged within browser

Pull down menus

Panels are resizable within main window

Popup “windows” restricted to browser content area

Looks and feels like an Eclipse application embedded in a browser
RAP – UI Components

Most SWT widgets have RWT* counterparts, except:
• StyledText (under investigation)
• FormText
• GC (often used with custom widgets… under investigation)

Much of JFace is implemented such as tables, content and label providers, actions, contributions, wizard framework, many dialogs, layouts, colors, fonts, etc.

Also
• Drag and drop (planned)
• Custom Widgets are possible

*RWT = RAP Widget Toolkit
RAP – Styles

- no CSS files… “theme” XML and properties files instead

Register theme in plug-in manifest

```
<extension point="org.eclipse.rap.swt.themes">
  <theme id="my.application.aquablue" name="Aqua Blue Test Theme" file="aqua-blue.properties" />
</extension>
```

Associate theme in plug-in manifest

```
<extension point="org.eclipse.rap.ui.branding">
  <branding id="my.application.aquabranding" servletName="aqua"
            themeld="my.application.aquablue"
            defaultEntrypointId="my.application.entrypoint1">

  </branding>
</extension>
```

```
# My Aquablue Theme

# Frame border for group boxes
group.frame.border: 2 dotted #56a0ea

# Font for title bar of Shells
shell.title.font: bold 15 "Trebuchet MS", Arial, Helvetica, sans-serif
```

RAP – Supported Browsers

Wrappers the Qooxdoo JavaScript library, supporting the same set of browsers...

• Microsoft IE 5.5 + (decent performance with 7.0 + )
• Firefox 1.0 +
• Opera 8 +
• Any Gecko-based browser with Gecko 1.7 +
• Safari 2.0 (limited support)
• Safari 3.0

http://qooxdoo.org/documentation/0.7/requirements
Demo – GWT / MyGWT application in action

Main “window” fixed within browser

Pull down menus

Panels are resizable within main window

Popup “windows” restricted to browser content area
GWT – UI Components

GWT

- CheckBox and several types of buttons
- Several types of text based fields including SuggestBox
- List, Table, Tree, TabBar, Hyperlink
- MenuBar, DialogBox and various panels
- Custom widgets are possible
- Call “raw” JavaScript using JavaScript Native Interface (JSNI)
GWT – API

MyGWT – UI Components

MyGWT

• New layouts, tables, trees, windows, lists, menus, toolbars
• Windows, dialogs, draggable, resizable, effects such as fade in fade out
• Very integrated look and feel

http://mygwt.net/deploy/dev/explorer/
GWT – Styles

• Style based on standard CSS style sheets
• Standard CSS provided for consistent look and feel
• Can modify global styles for your app
• Can extend or override on individual widget styles

```
.mail-item-detail {
  background-color: #E8EAEC;
  border-bottom: 1px solid silver;
}

.mail-item-detail h1 {
  font-weight: bold;
  font-size: 14px;
  padding: 5 0 2 5px;
}

.mail-item-detail h2 {
  font-size: 12px;
  font-weight: normal;
  padding: 0 5 0 5px;
}
```
GWT – Supported Browsers

“most recent versions of Internet Explorer, Firefox, and Safari. (Opera, too, most of the time.)”

MyGWT (based on Ext JS – http://extjs.com/)
• Internet Explorer 6+
• Firefox 1.5+ (PC, Mac)
• Safari 2+
• Opera 9+ (PC, Mac)

MyGWT and Ext JS license: Open Source LGPL 3.0

Graphing

Widgets are great… but what about graphs?

Some options include:
• Graphing widgets
• Integrating “raw” JavaScript graphs
• Graphing on the server for display on the client
GWT – Graphing Widgets

GWT-Ext

• An alternate add-on widget library for GWT
• Similar to MyGWT
• Some graphing built in

http://gwt-ext.com/demo/#chartGenerator
Integrating “raw” JavaScript graphs

Many JavaScript (and Flash, PHP, …) graphing options, but you must integrate them yourselves

Interactive graphs

• “Visio-like” interactive graphing

http://www.mxgraph.com/demo.html
Graphing on the server

• Render graph on server using Java based graphics library (e.g. JFreeChart) into a file

• Reference file in HTML page
History Management

GWT provides functionality for programmatically adding state to the browser's back button history.

RAP does not currently have any framework for providing “browser back button” support. Qooxdoo offers functionality for browser back and browser history but RAP currently does not use those features.

This is an important feature missing from RAP… or is it?

http://google-web-toolkit.googlecode.com/svn/javadoc/1.4/com/google/gwt/user/client/History.html
Development

How do you develop applications with these frameworks?
RAP – Architecture

• Very similar to developing an RCP application

• Can use UI development tools such as WindowBuilder Pro
RAP – Architecture

- Java code executed entirely on the server
- Static JavaScript library deployed to client
RAP – Debugging

One launch configuration for client and server

Port 8090 – makes it easy to serve up both RAP and GWT
GWT – Architecture

Write Java code, some portion of which is compiled into JavaScript

Not all of the underlying Java runtime library functionality can be translated to JavaScript

- syntax of Java regular expressions is similar, but not identical
- long are mapped onto JavaScript double-precision floating point values
- does not support subsequent dynamic loading of classes
- does not support object finalization during garbage collection

GWT – Debugging

Launch server
then GWT client in hosted mode.
Click “Compile/Browse”
to launch in browser.
Communication

How do these frameworks communicate with the server?
GWT – Communication

Page Request → Display Page → Process Actions → Render Changes → Lookup Page

Browser

Server

http://...

XMLHttpRequest

Lookup Data
GWT – Simplified RPC (XMLHttpRequest)

IsSerializable (not java.io.Serializable)
Java serialization relies on a few mechanisms that are not available in compiled JavaScript, such as dynamic class loading and reflection

http://code.google.com/support/bin/answer.py?answer=55196&topic=10210
GWT – Simplified RPC Example

**Client:** (compiled to JavaScript)

```java
public interface MailCommandsAsync {
    public void getMailItems(AsyncCallback callback);
    public void getMailItemBody(AsyncCallback callback);
}
```

```java
instance = (…) GWT.create(MailCommands.class);
ServiceDefTarget target = (ServiceDefTarget) instance;
target.setServiceEntryPoint(... + "MailCommands");
```

```java
instance.getMailItems(
    new AsyncCallback() {
        public void onSuccess(Object result) {
            List items = (List) result;
            ... display items ...
        }
        public void onFailure(Throwable caught) {
            ... notify user of exception ...
        }
    }
);
```

**Server:**

```java
public interface MailCommands extends RemoteService {
    public List getMailItems();
    public String getMailItemBody();
}
```

```java
public class MailCommandsImpl
    extends RemoteServiceServlet
    implements MailCommands {
    public List getMailItems() {
        ... gather items
        return items;
    }
    ... etc ...
}
```

**XMLHttpRequest**
RAP and GWT – Communication

RAP

• Easy… no extra work… all communication is handled for you

GWT

• More flexible…
   Optionally cache information on the client
   Optionally pre-fetch information in background

• … but more work
  so what more is required for GWT ?
Build and Deployment

How are apps built with these frameworks deployed?
Build Automation

Scripted build process
• Ant
• Maven2
• Roll your own

Automation is a must
• Manual process tends to be error prone over time
• Scripted build process is very repeatable

Consider Continuous Integration
RAP – Plug-in Deployment

Target Servers
• Easily deployed as OSGi bundles with included http service
• Deployed to traditional J2EE Application server using the “Servlet Bridge”
• Cannot be deployed to a plain web server

Plug-in based Build and Deploy Process
• All plug-ins should be contained in a feature
  • Make the PDE build process easier
• Run PDE build to generate bundles
  • Reformat code to PDE format
  • Call PDE build
  • Unzip and reformat the PDE build result
• Upload build result to application server
• Run deployment scripts on application server

GWT – Plug-in Deployment

Target Servers

- Can be deployed to an OSGi based server using plug-ins
- Can be deployed to a more traditional J2EE Application server as WAR file
- Easily deploy simple apps (e.g. financial calculator… no RPC) to plain web server

Plug-in based Build and Deploy Process

- Switch GWT library jar file (gwt-user.jar ----> gwt-servlet.jar)
- Call the GWT compiler to compile Java to JavaScript
  - Generated JavaScript placed into a plug-in
- All plug-ins should be contained in a feature
  - Make the PDE build process easier
- Run PDE build to generate bundles
  - Reformat code to PDE format
  - Call PDE build
  - Unzip and reformat the PDE build result
- Upload build result to application server
- Run deployment scripts on application server
GWT – Traditional J2EE Server

Target Servers
• Can be deployed to an OSGi based server using plug-ins
  - Can be deployed to a more traditional J2EE Application server as WAR file
• Easily deploy simple apps (e.g. financial calculator… no RPC) to plain web server

Traditional J2EE based Build and Deploy Process
• Switch GWT library jar file (gwt-user.jar ----> gwt-servlet.jar)
• Call the GWT compiler to compile client-side Java to JavaScript
  - Generated JavaScript placed into a plug-in
• Call javac to compile server-side Java code
• Create WAR file containing client-side JavaScript and server-side Java
• Upload build result to application server
• Run deployment scripts on application server
Build and Deploy Summary

Build
• Automation is a must
• Consider Continuous Integration

Deployment
• Easier to deploy RAP than GWT to an OSGi based server.
• May be easier to deploy GWT than RAP to a more traditional application server.
• Construct build scripts such that apps can be deployed both on the local machine and on the server
Performance

How fast are apps built with these frameworks?
## Performance – Loading Pages

Various load times for our test application…

<table>
<thead>
<tr>
<th>Seconds to load</th>
<th>RAP</th>
<th>GWT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cached</td>
<td>Cached</td>
</tr>
<tr>
<td>Load Page</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Home</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>EclipseCon</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Load Folder</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Home</td>
<td>3</td>
<td>n/a</td>
</tr>
<tr>
<td>EclipseCon</td>
<td>¾</td>
<td>n/a</td>
</tr>
<tr>
<td>Sort List</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Home</td>
<td>2</td>
<td>n/a</td>
</tr>
<tr>
<td>EclipseCon</td>
<td>½</td>
<td>n/a</td>
</tr>
</tbody>
</table>

GWT is slightly faster… why?  
Does this make a drastic difference from an end user’s standpoint?
Performance – Image Bundling

Look at the whole stack, don’t just look at the application
Faster to load one bigger file than many smaller files

GWT provides functionality to combine multiple small images into a single large image for efficient transport to the browser

MyImageBundle imageBundle
   = (MyImageBundle) GWT.create(MyImageBundle.class)
imageBundle.printerIcon().createImage();
imageBundle.notesIcon().createImage();
imageBundle.editIcon().createImage();
imageBundle.graphIcon().createImage();
...

This is planned for RAP. Something comparable to image-bundles is just finding its way into Qooxdoo 0.8, and the RAP team will investigate once that is in place.

Incorporating this functionality in RAP would reduce the time to load a page.

Performance – Communication

In general, RAP pushes more work off the client onto the server, thus communicating more information and more frequently with the server.
Performance – Packets

GWT is faster sorting because more information is cached on the client and more processing is kept on the client…

<table>
<thead>
<tr>
<th></th>
<th>RAP</th>
<th></th>
<th>GWT</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Packets</td>
<td>Bytes</td>
<td>Packets</td>
<td>Bytes</td>
</tr>
<tr>
<td>Sort List(1)</td>
<td>63</td>
<td>25390</td>
<td>11</td>
<td>4661</td>
</tr>
<tr>
<td>Load Folder</td>
<td>8</td>
<td>2900</td>
<td>8</td>
<td>3827</td>
</tr>
<tr>
<td>Load Again(2)</td>
<td>6</td>
<td>2602</td>
<td>4</td>
<td>349</td>
</tr>
</tbody>
</table>

… but interesting that in our application, RAP is slightly more efficient loading a folder the first time… why?

(1) Sort List 6 times by clicking on columns 1, 2, 3, 1, 2, 3
(2) Load a different folder then load the original folder again
Performance – Virtual Tables

RAP passes only the information necessary to display the visible rows in the table.

In our GWT app, the entire table content is loaded the first time it is needed and cached on the client. We could have loaded only what we needed to reduce the refresh time.

Bottom line: GWT is more flexible and potentially faster, but by how much and at what cost (complexity) to the developer?
Unit Testing

Both RAP and GWT are based on Java, so…
• JUnit can be used to test the server-side code
• But how to test the client-side?
RAP – Unit Testing

• Non-UI code can be tested using standard JUnit
• UI code can be tested by
  ❖ Switching from RWT to SWT/JFace
  ❖ Using some type of JUnit based UI testing framework
    ▪ TPTP AGR
    ▪ Abbott SWT
    ▪ WindowTester
    ▪ Roll your own…

Switch from RWT to SWT and back for testing?  How?
RAP – as compared to RCP

The differences between RCP and RAP are small…

… only the underlying libraries are different but with the same API so that you can switch between
RAP – switching between RWT and SWT

Once you import and select your RAP target platform … … and import packages (not bundles) … … you can easily switch between SWT and RWT
GWT – Unit Testing

public class FooTest extends GWTTestCase {

    // Specifies a module to use when running this test case.
    public String getModuleName() {
        return "com.example.foo.Foo";
    }

    public void testStuff() {
        myService.doSomething(arguments, new AsyncCallback() {
            public void onSuccess(Object result) {
                finishTest();
            }
            public void onFailure(Throwable caught) {
                ... test failed ...
            }
        });
        // Set a delay period significantly longer than the event is expected to take.
        delayTestFinish(500);
    }
}

GWT includes a special GWTTestCase base class that provides JUnit integration

Must specify GWT module

Optional support for asynchronous test finish and specification of maximum execution time

AJAX Security Issues

There are many AJAX security issues and people *much* more qualified to talk about security than us.

Attack vectors include:
• cross-site scripting (XSS)
• manipulating client-side logic
• defeating logic protection techniques
• function hijacking (client-side code being changed)
• JavaScript Object Notation (JSON) hijacking
• denial of service attacks
• hacking Google Gears

Bottom line: Server should not implicitly trust *any* information inbound from a client


Video of security talk at GWT Conference:
ATF – AJAX Toolkit Framework

Eclipse.org project for AJAX tools in the Eclipse IDE

<table>
<thead>
<tr>
<th>ATF</th>
<th>RAP and GWT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Editing, debugging and deployment of JavaScript</td>
<td>Editing and debugging of Java while keeping JavaScript at “arms length”</td>
</tr>
</tbody>
</table>

http://www.eclipse.org/atf/
Thin Wire

Similar to GWT but using Swing/AWT

• http://www.thinwire.com/

Of note: last “news” entry dated 5-30-2007
More Information

• RAP
  - http://www.eclipse.org/rap/

• GWT
  - http://code.google.com/webtoolkit/
  - http://code.google.com/webtoolkit/examples/
  - http://www.asquare.net/gwt/