



Best practices for building large GWT applications

Heiko Braun <hbraun@redhat.com>

About me

- Heiko Braun
- Senior Software Engineer JBoss / Red Hat
- 4 years JBoss, 12 years industry
- Focus on SOA, BPM, GWT
- Contributor:
JBossWS, jBPM, Riftsaw, Errai, SAM, Savarra

> <http://jboss.org>

Topics

- What is GWT?
- Decomposing a large GWT application
- Introducing project Errai



What is GWT?

Google Web Toolkit



- Write AJAX apps in the Java language, then compile to optimized JavaScript
- SDK: API, Compiler, Hosted Mode Browser
- Edit Java code, then view changes immediately without re-compiling
- Step through live AJAX code with your Java debugger
- Compile and deploy optimized, cross-browser JavaScript

> <http://code.google.com/webtoolkit/>

GWT Features

- Communicate with your server through really simple RPC
- Reuse UI components across projects
- Use other JavaScript libraries and native JavaScript code
- Localize applications
- Choice of development tools
- Test your code with JUnit
- Open Source



Decomposing large GWT applications

Example: JBoss SOA tooling

The screenshot displays the JBoss BPM Console interface. At the top, the browser address bar shows the URL `http://localhost:8080/jbpm-console/app.html#errai_ToolSet_Processes;none`. The page title is "BPM Console". In the top right corner, the user "alex" is logged in, with a "Logout" button next to it.

The main interface is divided into several sections:

- Tasks:** A sidebar menu on the left with options for "Processes", "Reporting", "Runtime", and "Settings".
- Process Overview:** The central area, featuring a "Refresh" button, a filter dropdown set to "All", and action buttons for "Start", "Terminate", and "Delete".
- Process List:** A table listing various process types and their counts.
- Instance Table:** A table showing active process instances with columns for Instance, State, and Start Date.
- Execution details:** A section at the bottom right providing specific information for the selected instance, including "Process: Hql", "Instance ID: Hql.10006", "Key:", "State: RUNNING", and "Start Date: 2010-02-21 22:00:12". It also includes buttons for "Diagram" and "Instance Data".

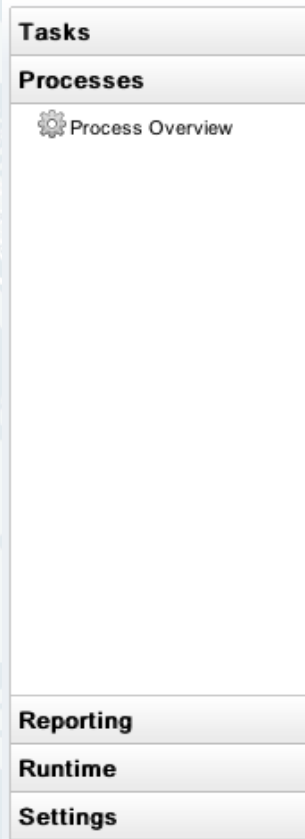
Process	v.
AsyncActivity	1
AsyncFork	1
conditionalSequenceFlow	1
Custom	1
EndMultiple	1
EndProcessInstance	1
EndState	1
Hql	1
inclusiveGateway	1
intermediateTimerCatch	1
noneStartEndEvent	1
Order	1

Instance	State	Start Date
Hql.10006	RUNNING	2010-02-21 22:00:12
Hql.10034	RUNNING	2010-02-21 22:03:12

Execution details

Process: Hql
Instance ID: Hql.10006
Key:
State: RUNNING
Start Date: 2010-02-21 22:00:12

Challenge #1: Feature Set



- Different features per:
 - Target runtime
 - Development stage
 - Project lifecycle
 - Target audience

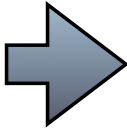
Solution #1: Compile-time composition

- Leverage Maven dependency sets
- Using Deferred Binding
 - Create and select a specific implementation of a class
 - Either using Replacement or Generators

Solution #1: Compile-time composition

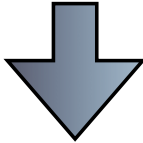
(1) 'mvn -Dconsole.profile=drools install'

```
<profile>
  <id>drools-console.profile</id>
  <activation>
    <property>
      <name>console.profile</name>
      <value>drools</value>
    </property>
  </activation>
  <dependencies>
    <dependency>
      <groupId>org.jboss.bpm</groupId>
      <artifactId>gwt-console-profile-drools</artifactId>
      <version>${version}</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>
```



(2) Properties file or annotations

```
workspace.cfg
org.jboss.bpm.console.client.task.TaskEditor
org.jboss.bpm.console.client.process.ProcessEditor
org.jboss.bpm.console.client.report.ReportEditor
org.jboss.bpm.console.client.SettingsEditor
```



(3) Deferred Binding Generator

```
private Workspace createWorkspace()
{
  Workspace workspace = new Workspace(menu, this);
  WorkspaceLauncher launcher = GWT.create(WorkspaceLauncher.class);
  launcher.launch(this, workspace); // calls Workspace.addEditor()
  return workspace;
}
```

Limitation #1: Component interplay

- Each plugin component isolated
- No interplay possible
- It would introduce dependencies
- Grouping by functionality vs. usability
- Conceptual split not necessarily technical split

Challenge #2: Coupling between components

BPM Console

http://localhost:8888/app.html?gwt.codesvr=192.168.0.10:9997#errai_ToolSet_Processes:none

alex Logout

Tasks

Processes

Process Overview

Process Overview

Process	v.
AsyncActivity	1
AsyncFork	1
BPMN2 Example process using tas	1
Custom	1
EndMultiple	1
EndProcessInstance	1
EndState	1
Hql	1
Order	1
Sql	1
StateChoice	1
StateSequence	1

Refresh All Start Terminate Delete

Instance	State	Start Date
Hql.50013	RUNNING	2010-01-20 20:36:29
Hql.70004	RUNNING	2010-01-22 14:39:27
Hql.90001	RUNNING	2010-01-26 16:51:00
Hql.130008	RUNNING	2010-02-02 12:46:10
Hql.150001	RUNNING	2010-02-05 14:49:13
Hql.170001	RUNNING	2010-02-16 16:39:14
Hql.170008	RUNNING	2010-02-16 16:39:17
Hql.170015		

process instances / process definition

Process Definition	Count
EndState-1	3
VacationRequest-1	7
EndProcessInstance-1	3
Hql-1	20
EndMultiple-1	1
vacationRequestProcess-1	2
AsyncFork-1	6
Order-1	1
StateChoice-1	1
AsyncActivity-1	1

Execution details

Process: Hql

Instance ID: Hql.170001

Key:



State: RUNNING

Start Date: 2010-02-16 16:39:14

Activity: n/a

Challenge #2: Coupling between components

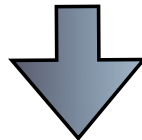
- Components “decorate” functionality
 - i.e. Process Management & Reporting
- Dependencies may come and go
 - Different feature set:
 - Maturity (CR vs. GA)
 - Environment (staging vs. production)
 - Profiles (custom composition)

Tasks
Processes
Reporting
Runtime
 Deployments
 Jobs

Solution #2: MVC

(1) Model changed

```
// refresh process definitions
controller.handleEvent(
    new Event(UpdateDefinitionsAction.ID, null)
);
```



(2) Update View

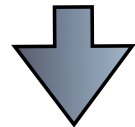
```
DefinitionListView view = (DefinitionListView)
    controller.getView(DefinitionListView.ID);
if(view!=null) // may not be initialized (lazy)
{
    JSONValue json = JSONParser.parse(response.getText());
    List<ProcessDefinitionRef> definitions =
        DTOParser.parseProcessDefinitions(json);
    view.update(definitions);
}
```

- Model-View-Controller ?
- Less coupled
- Still compile-time dependencies

Solution #2: Pub/Sub

(1) Publish messages

```
MessageBuilder.createMessage()  
    .toSubject("UserManagement")  
    .command(UserManagementCommands.Remove)  
    .noErrorHandling().sendNowWith(bus);
```



(2) Subscribe Listener

```
@Service  
@RequiresRoles({"admin"})  
public class UserManagement implements MessageCallback {  
  
    public void callback(Message message) {  
        // handle user management commands here  
    }  
}
```

- Messaging through publish / subscribe
- Messaging API only shared dependency
- Notion of “presence”

Limitation #2: Pub/Sub

- Decoupling through de-typed nature
- No compile-time checking
- Exchange protocol (contract) not “visible”
- Choreography validation?

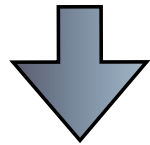
Challenge #3: UI Components coupled to services

- I.e. Email client requires SMTP service
- Services may come and go:
 - SOA promise
 - Different product versions
 - Target runtime derivations

Solution #3: Bootstrap

(1) Client UI starts, request server status

```
// bootstrap
controller.handleEvent (
    new com.mvc4g.client.Event (BootstrapAction.ID, null)
);
```



(2) PluginInfo (type, available)

```
public static ServerStatus parseStatus(JSONValue json)
{
    ConsoleLog.debug("parse " + json);

    ServerStatus status = new ServerStatus();

    JSONArray jsonArray = JSONWalk.on(json).next("plugins").asArray();
    for (int i = 0; i < jsonArray.size(); i++)
    {
        JSONValue item = jsonArray.get(i);
        String type = JSONWalk.on(item).next("type").asString();
        boolean avail = JSONWalk.on(item).next("available").asBool();
        status.getPlugins().add( new PluginInfo(type, avail) );
    }

    return status;
}
```

- Bootstrap: “Give me a list of capabilities”
- Usually RPC call when app starts
- Problem: Fixed initialization point
- Lazy Components?

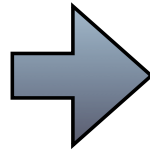
Solution #3: Messaging w. Presence

- Presence: “Need a plumber. Please call XYZ”
 - Relies on messaging bus behind the scenes
 - Async, independent, durable

(1) Client: Seek capability

```
// engage mail sender service
shoutbox.engageOffer(ID, "demo.mailSender",
    new ShoutboxCallback()
    {
        public void offerSubmitted(String provider)
        {
            status.setHTML("MailSender available");
        }

        public void offerRetracted(String provider)
        {
            status.setHTML("<strike>MailSender unavailable<strike>");
        }
    }
);
```



(2) Provider: Offer capability

```
shoutbox.submitOffer(ID, "demo.mailSender");
```



Introducing Project Errai

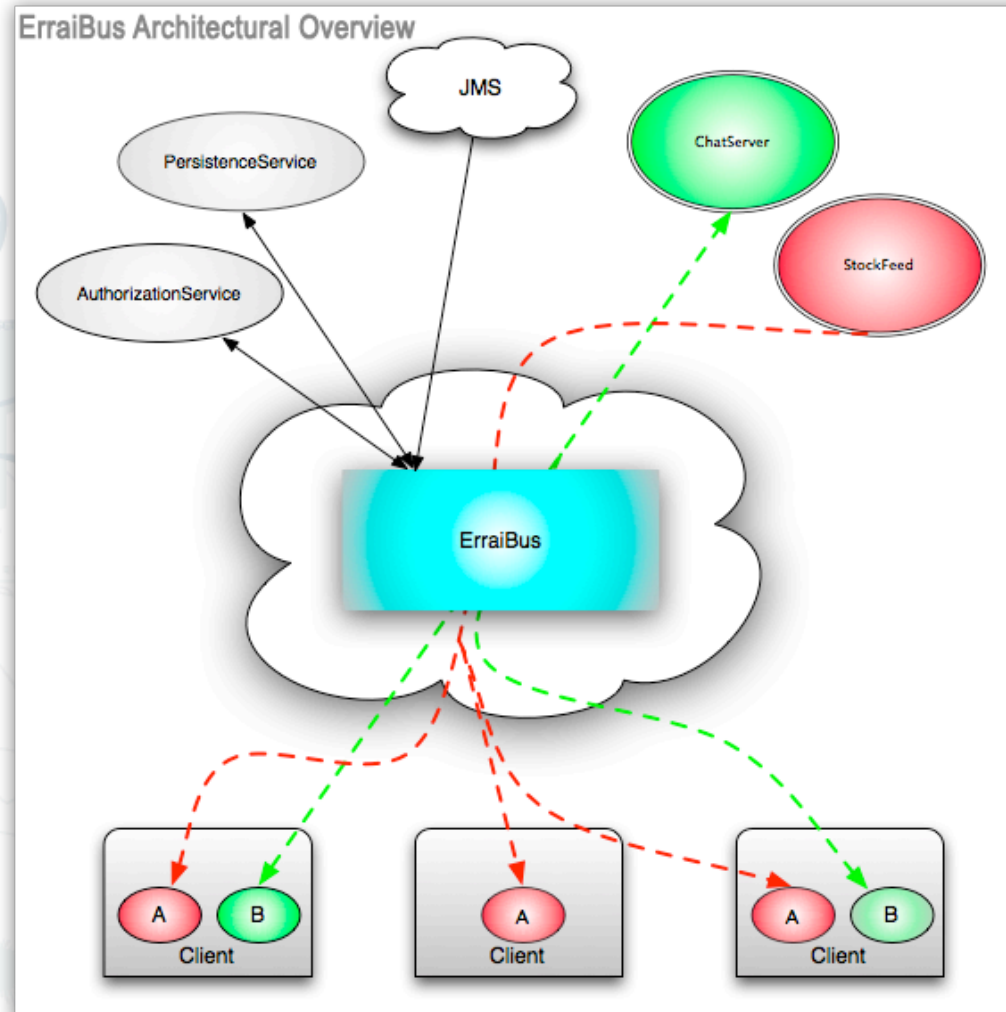
Project Errai

- Consolidates JBoss GWT efforts
- Tackles the problems described earlier
- Both R&D and actual product development
- Main components:
 - Message Bus, Workspace framework, Widget library

> <http://jboss.org/errai>

Errai-Bus

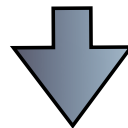
- Backbone to application design
- Common architecture across client&server
- Enables federated architecture
- Asynchronous messaging (pub/sub)
- Conversational
- Both GWT and Javascript API (OpenHub Spec)



Errai-Bus API: Common to client & server

(1) Client: Publish

```
MessageBuilder.createMessage()  
    .toSubject("ChatServer")  
    .command(EmployeeChatCommands.SendMessage)  
    .with(EmployeeChatParts.User, nickname)  
    .with(EmployeeChatParts.ToUser, name)  
    .with(EmployeeChatParts.MessageText, textEntry.getText())  
    .noErrorHandling().sendNowWith(bus);
```



(2) Server: Subscribe

```
@Service("ChatServer")  
public class EmployeeChatServer implements MessageCallback {  
  
    public void callback(Message message) {  
        // handle chat message here  
    }  
}
```


Pub/Sub roles vs. tiers

- client-client across server (chat server)
- client-client w/o server (inter component)
- client-server (client send)
- server-client (server push)

Workspace framework

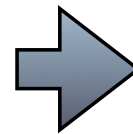
- UI environment for which to deploy your console
- Provides development infrastructure, documentation and examples:
 - Tear down barriers, ease of use
- Common, shared services, i.e:
 - Authentication & Authorization
 - Logging & Exception handling
- Allows toolset composition at various stages:
 - Sandbox, Project, Product

Workspaces API

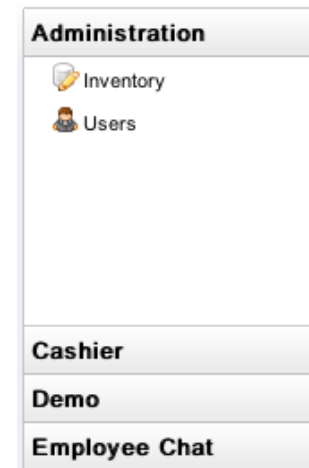
- Handles loading, initialization and access to tools
- Uses Deferred Binding as well

(1) Component declaration

```
@LoadTool(  
    name = "Users",  
    icon = "userGrayIcon",  
    group = "Administration"  
)  
@RequireRoles({"admin"})  
public class UserManagement implements WidgetProvider {
```



(2) Automatic workspace assembly



Errai Widgets

- Complements OSS offering (i.e. Mosaic)

The screenshot displays a web application interface with a top navigation bar containing tabs for 'TimeDisplay', 'Inventory', 'Users', and 'MailSender'. Below the navigation bar is a 'User Management' section with 'Add' and 'Delete' buttons, and a 'MailSender available' notification. A table lists user information:

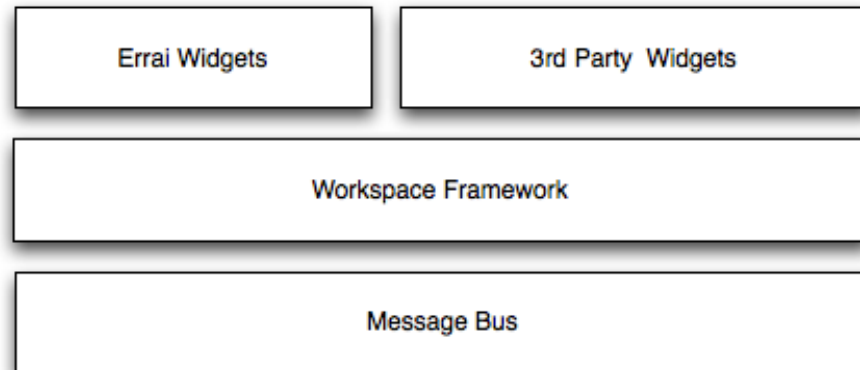
Employee ID	Name	Date Started	Position
002	Mike Brock	Sep 15, 2006	Assistant Manager
001	Lillian Angel	May 15, 2005	Cashier
003	Rodney Russ	Jun 07, 2008	Manager

Two dialog boxes are overlaid on the interface:

- Alert!**: A dialog box with a red flag icon, containing the text 'A panel is already open for 'Inventory'. What do you want to do?' and two buttons: 'Open New' and 'Goto'.
- Security Challenge**: A dialog box with a key icon, containing 'User:' and 'Password:' input fields and a 'Login' button.

Putting it all together

- Baseline for JBoss SOA tooling
- Free composition of console components
- Different projects provide management tools
- Mix and match with 3rd party elements





Demo Applications



Q&A

- > <http://jboss.org/errai>
- > <http://errai-blog.blogspot.com/>