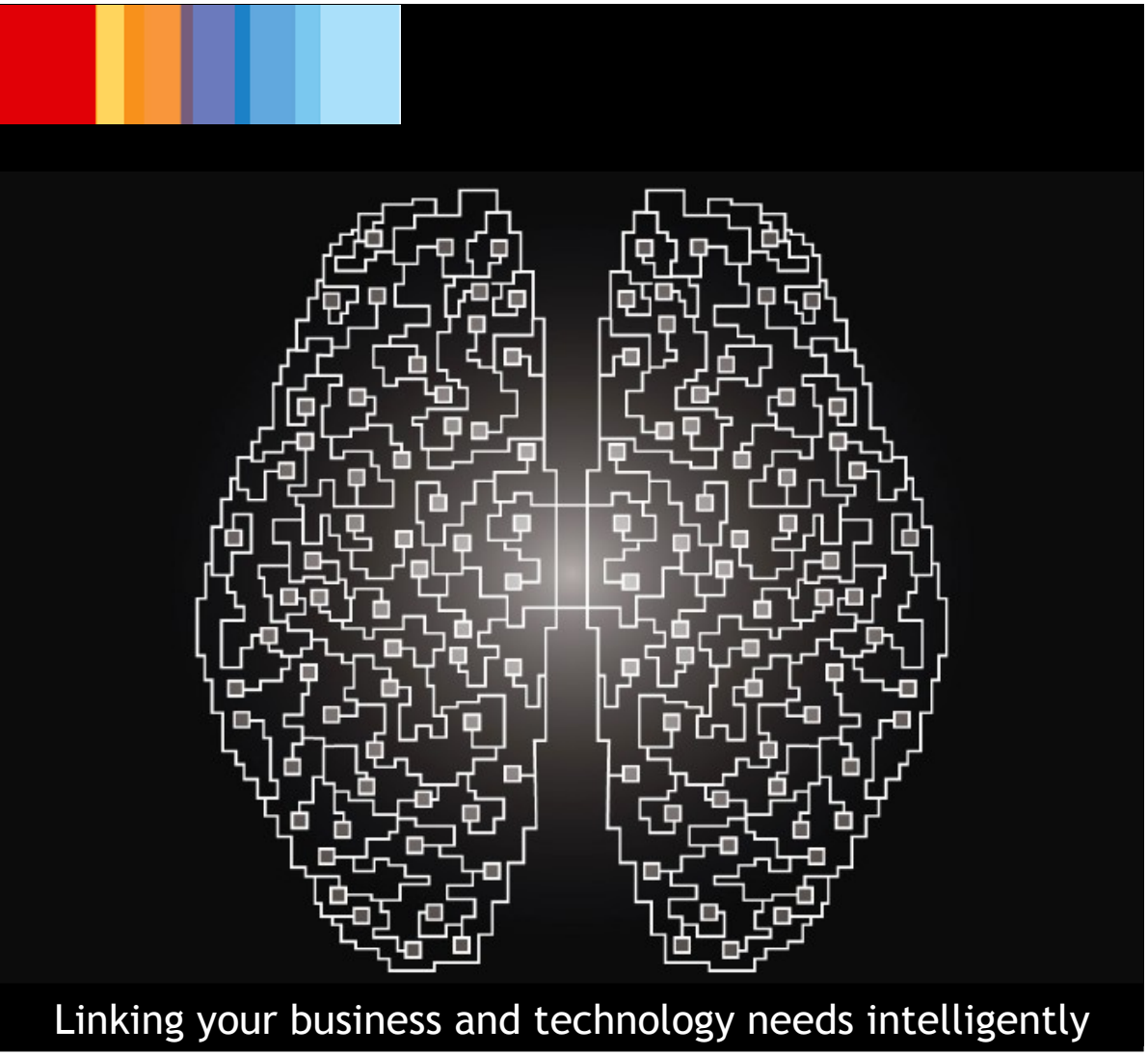


GWT Portlets Framework & GWT Generators & GWT 1.6+ Events

18th Aug 2009
David Tinker

Introduction

This talk will cover the following topics:

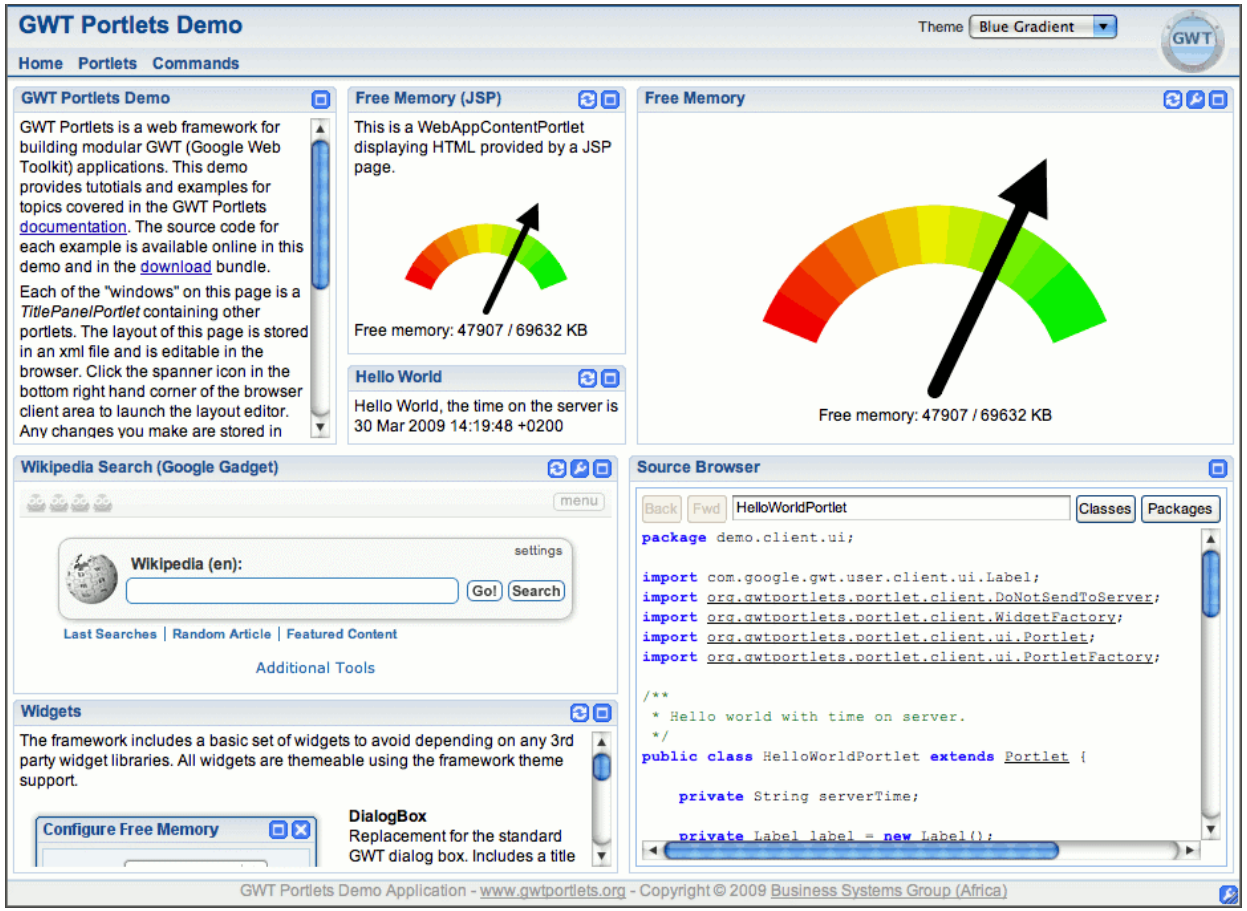
- GWT Portlets Open Source project update
- Using GWT generators
- The GWT 1.6+ event model
- New stuff in GWT 2.0

What is the Google Web Toolkit (GWT)?

- Java to Javascript compiler
- JRE emulation library (java.lang, java.util, obviously no IO etc.)
  - No reflection as only reachable code is compiled
  - Generated Javascript is very compact
- Widget library (wrap HTML DOM elements)
- Easy RPC mechanism with Serialization support
- Hosted mode for debugging client side code
- All state can be maintained on the client → stateless highly scalable server

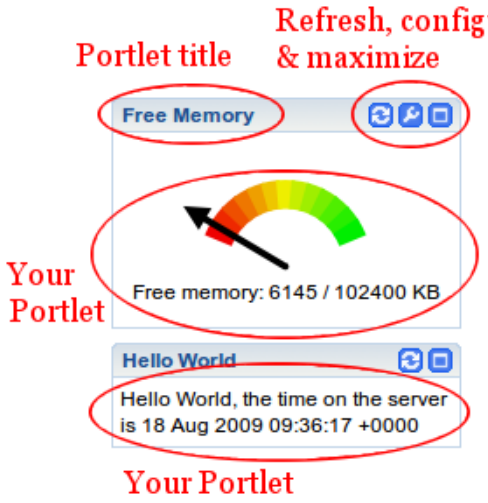
What is GWT Portlets?

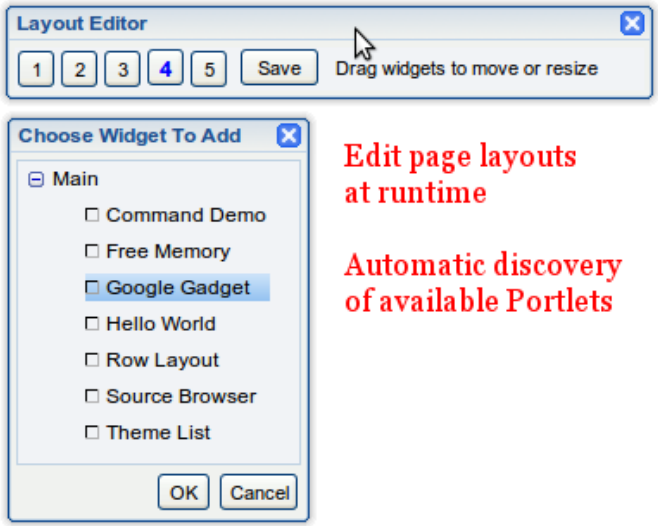
- Open Source (LGPL license) framework for building modular GWT applications
- Released by BSG as open source about 6 months ago
  - Now hosted on Google Code: http://code.google.com/p/gwtportlets/

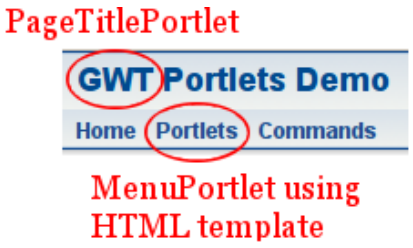


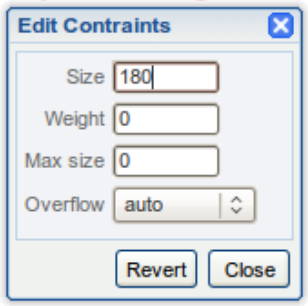
What does GWT Portlets give you?

- Very simple & productive, yet powerful programming model to build good looking, modular GWT applications









Why use GWT Portlets?

- Without a framework any application will develop problems:
  - Communication between components is tricky
  - Components tend to become closely coupled
  - Spaghetti like structure is hard to maintain
  - Modularization and customization is difficult due to tight coupling

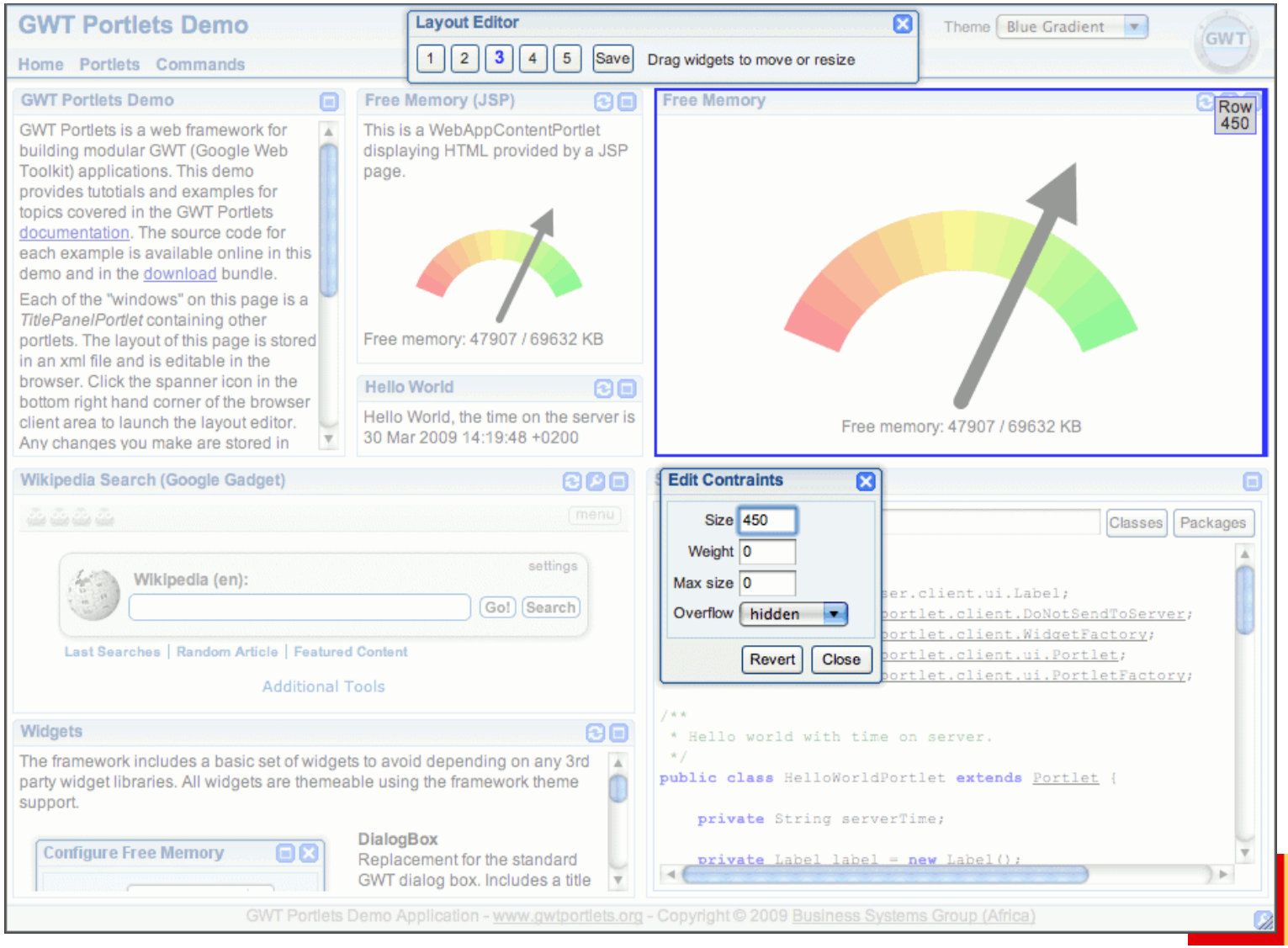
- GWT applications suffer from additional issues:
  - Explosion of RPC methods
    - Typically separate RPC calls for read, add/update and delete
    - Often all end up in a “kitchen sink” RPC interface resulting in tighter coupling
  - UI layout constructed in code is “hardcoded”
    - Cannot use the same binaries (Javascript files) for different deployments
    - Cannot modify at runtime
  - Lack of reflection makes solving these problems more difficult

- The developers on the project end up spending a lot of time writing their own framework to solve these issues

GWT Portlets Design

- The programming model is somewhat similar to writing JSR168 portlets for a portal server (Liferay, JBoss Portal etc.)
- The “portal” is your application
  - The GWT Portlets framework is used as a library
  - It provides the necessary components i.e. your application is not deployed into a portal: it “is” the portal and remains in control
- Application functionality is developed as loosely coupled Portlets each with an optional server side DataProvider – only 2 classes per piece of functionality
- Every Portlet knows how to externalize its state into a serializable PortletFactory subclass (momento / factory pattern) making important functionality possible
  - CRUD operations are handled by a single RPC for all Portlets
  - Trees of WidgetFactory's are marshalled to/from XML on the server to store GUI layouts in XML page files
- Pages can be edited in the browser at runtime (by developers and/or users)
- Portlets are positioned absolutely so can use scrolling regions
- Portlets are configurable, indicated when they are busy loading etc.

Page Editor Screenshot



Hello World Portlet

- main.client.ui.HelloWorldPortlet
  - Client side GUI
  - Factory static inner class (memento and DTO)
- main.server.HelloWorldDataProvider
  - Populates a HelloWorldPortlet.Factory with data on the server

Alt-tab to demo

- Hello world source files
- Overview of packages and classes in IntelliJ
- Page file
- Page editor
- Add “Free memory” Portlet to page

How the available Portlets list is put together is covered later in this presentation

Open Page Process

User navigates to a new 'page'
`http://host/gwtapp/#hello_world?foo=bar`
generating
History.onValueChange event

Client Side

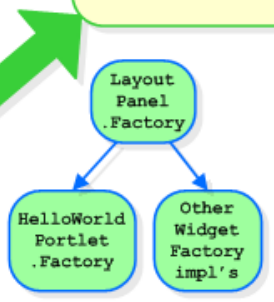
getPage(hello_world?foo=bar)
RPC call to server

The PageProvider uses the history token "hello_world?foo=bar" to choose an XML page file (hello_world.xml)

```
<LayoutPanel>
  <widgets>
    <HelloWorldPortlet
        styleName=""/>
    ...
  </widgets>
</LayoutPanel>
```

The XML is marshalled into a tree of WidgetFactory's by XStream

Each WidgetFactory knows how to create a Widget (e.g. Portlet) and initially contains data (e.g. styleName) from the XML page file



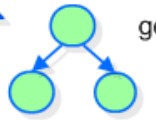
WidgetFactory tree

The tree is walked and WidgetDataProvider's are given a chance to refresh each WidgetFactory with additional data (e.g. from a DB) and history token parameters foo=bar

Server Side

HelloWorldPortlet .Factory

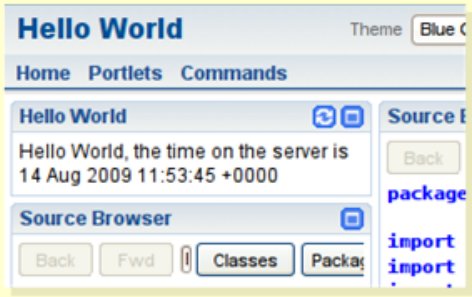
```
public class HelloWorldDataProvider …{
  public void refresh(
      HelloWorldPortlet.Factory f,
      PageRequest req) {
    f.serverTime = DATE_FORMAT.format(
          new Date());
...
```



getPage(hello_world?foo=bar)
RPC call returns
WidgetFactory tree

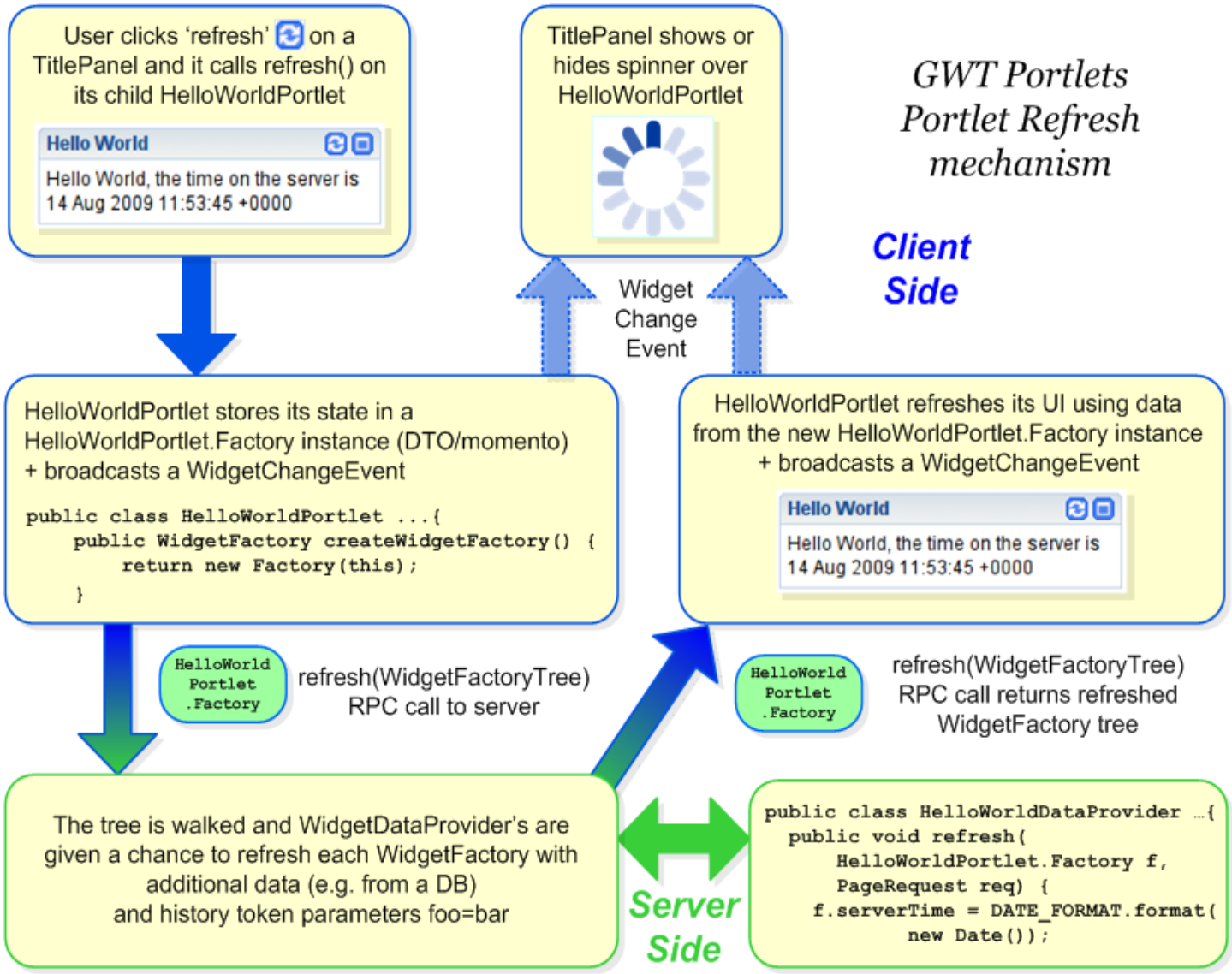
PagePortlet replaces its contents with a tree of Widgets created and refreshed by each WidgetFactory

```
WidgetFactory f = ...
Widget w = f.createWidget();
f.refresh(w);
```



GWT Portlets Open Page mechanism

Portlet Refresh Process



CRUD Portlet

- main.client.ui.SimpleCrudPortlet
  - Uses Portlet Factory for add/edit and delete as well as refresh
  - Avoids need to add RPC methods for this purpose
  - Portlet ‘automatically’ refreshed with new data after the operation
- main.server.SimpleCrudDataProvider
  - Populates a SimpleCrudPortlet.Factory with data on the server
  - Performs update and delete operations

Alt-tab to demo

- Simple CRUD source files
- Client side and server side of an update

Containers & Layouts

- Absolute positioning within browser viewport
- Widgets (e.g. Portlets) arranged using constraints and layout managers (like Swing, AWT etc.)
- Makes it possible to use scrolling regions
- The “contents” of Portlets are often positioned statically i.e. using normal browser layout

LayoutPanel is a general purpose container:

```
LayoutPanel panel = new LayoutPanel(); // defaults to RowLayout in a column
panel.add(chart); // use all free space and include scrollbars if needed
panel.add(label, 24); // 24 pixels high, no scrollbars (overflow is hidden)
panel.layout(); // adding widgets does not automatically redo the layout
```

Alt-tab to demo (Portlets | RowLayout)

- RowLayout can be used to construct “BorderLayout” and others

GWT Generators

- GWT has no support for reflection
  - All reachable code can be discovered at compile time to reduce the size of the generated Javascript
- It is easy to generate Java code that is compiled into Javascript at compile time to perform “reflection like” tasks e.g. list all classes implementing a given interface
- Create an interface with the operations you need:

```
public interface WidgetFactoryHelper {
 /** Get a list of all classes that implement WidgetFactory. */
 public WidgetFactoryMetaData[] getWidgetFactoryList();
```

- Create an implementation of the interface somewhere in client code:

```
private static WidgetFactoryHelper widgetFactoryHelper
 = GWT.create(WidgetFactoryHelper.class); // replaced by compile time generated impl
```

- Specify a com.google.gwt.core.ext.Generator subclass in GWT module (Portlets.gwt.xml):

```
<generate-with class="org.gwtportlets.portlet.rebind.WidgetFactoryHelperGenerator">
  <when-type-assignable class="org.gwtportlets.portlet.client.impl.WidgetFactoryHelper"/>
</generate-with>
```

GWT Generators (2)

- The generator class ends up “printing” Java code (not Javascript!) that is compiled into Javascript along with the rest of the application
- Use the “-gen” GWT compiler option to see generated Java code
- Same mechanism is used to create GWT serialization code

```
private static final String WIDGET_FACTORY_META_DATA =
        WidgetFactoryMetaData.class.getName();

for (JClassType t : typeOracle.getTypes()) {
    if (t.isClass() != null && !t.isAbstract()
            && t.isDefaultInstantiable()
            && t.isStatic()
            && implementsWidgetFactory(t)) {
        widgetFactoryMap.put(t, c++);
    }
}

sw.println("private static final " + WIDGET_FACTORY_META_DATA +
        "[] WIDGET_FACTORY_LIST = new " + WIDGET_FACTORY_META_DATA + "[]{");
sw.indent();
for (JClassType t : widgetFactoryMap.keySet()) { ...}
```

GWT 1.6+ Event Model

- GWT 1.5 and earlier used conventional addXXXListener & removeXXXListener model similar to Swing et al
- To provide “Click” event support a Widget would need to do the following:
  - Add a *ClickListenerCollection* field to track the listeners
  - Implement SourcesClickEvents interface
  - Add addClickListener and removeClickListener methods
  - Call *sink(Event.ONCLICK)* in the constructor to receive DOM click events
  - Override onBrowserEvent, detect a click event and fire it to listeners
- With GWT 1.6+ a Widget only has to:
  - Implement HasClickHandlers interface
  - Add addClickHandler method: call *addDomHandler(handler, ClickEvent.getType());*
- Applications using the new model call addClickHandler instead of addClickListener
- This returns a HandlerRegistration which must be stored if the handler needs to be removed at some point

New Stuff in GWT 2.0

- Hosted mode debugging using the native browser
- Compiler enhancements for speed and size
- Developer guided code splitting aka dynamic module loading
- ClientBundle: ImageBundle for arbitary resource types e.g. CSS files
- Faster RPC

Questions / Discussion + Snacks + Beer