ISMLA Session 9 - GWT

Björn Rudzewitz

Tübingen University

December 18, 2017

Motivation

Reflecting on GWT development so far:

- What did you like/what worked well ?
- Which part did you find tedious/wher did you have to invest a lot of work ?
- How did the collaboration in teams work with GWT so far ?
  In which aspects ?

Plan

1. UiBinder
    - XML and HTML
    - Element binding
    - Event Handling
    - ui:style
    - Eclipse Plugin Components
2. Maven and GWT
3. Deployment

Motivation

- GWT development so far:
  - writing one client–side module (entry point) with programmatically created interface
  - writing server functions and proxies (RPC)
  - writing shared objects

Motivation

problems with approach so far:

- web page html normally consists of static parts
- writing code in a different language (Java) translated to another language (HTML) althoug you know how to write HTML
- redundancy in methods possible (e.g. creating a new heading object instead of updating)
- code not modularized, or only via functions
- separation of work difficult (function/frontend/style/. . .)

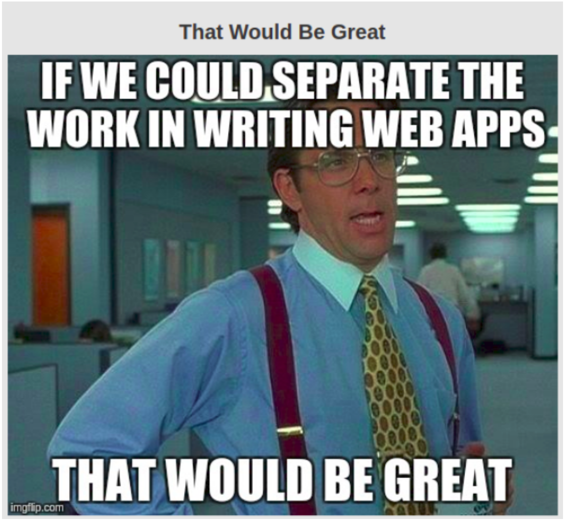
Motivation

problems with approach so far:

- web page html normally consists of static parts
- writing code in a different language (Java) translated to another language (HTML) althoug you know how to write HTML
- redundancy in methods possible (e.g. creating a new heading object instead of updating)
- code not modularized, or only via functions
- separation of work difficult (function/frontend/style/. . .)

⇒ solution: GWT UiBinder

Motivation



created with `https://imgflip.com/memegenerator` on Dec 18, 2017

UiBinder

- UiBinder to provide (certain) separation between function and frontend (elements & style)
- Ui**Binder**: bind HTML fields to Java fields
- i.e. define templates in HTML[1] and bind fields to Java instance fields
- designer can style the HTML template, Java developer can focus on the function behind it

[1] more precisely: in XML compiled to HTML

UiBinder

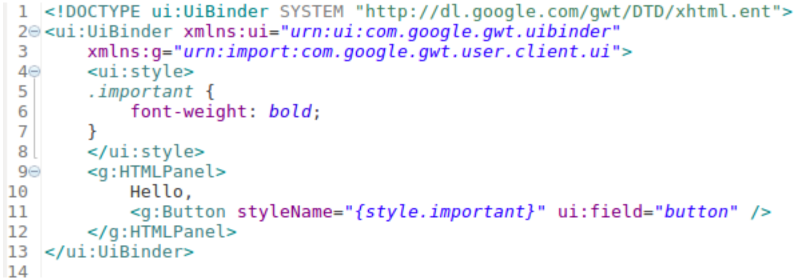
Generating a UiBinder widget:

- *New* $\rightarrow$ *Other* $\rightarrow$ *GWT* $\rightarrow$ *UiBinder*
- option to generate example content
- creates two files:
  1. MyWidget.java
  2. MyWidget.ui.xml

$\Rightarrow$ UiBinder creates **composite widgets**

UiBinder

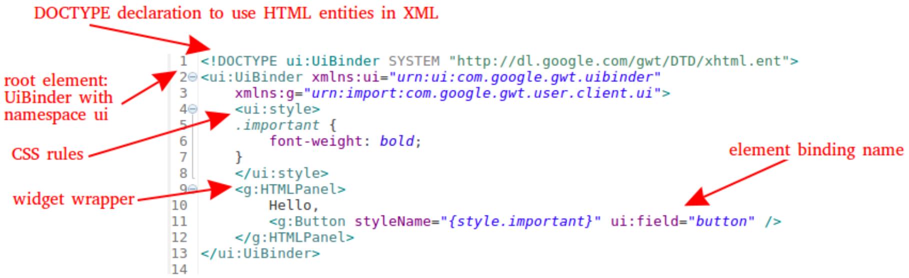
Result of example widget



```
<!DOCTYPE ui:UiBinder SYSTEM "http://dl.google.com/gwt/DTD/xhtml.ent">
<ui:UiBinder xmlns:ui="urn:ui:com.google.gwt.uibinder"
    xmlns:g="urn:import:com.google.gwt.user.client.ui">
    <ui:style>
    .important {
        font-weight: bold;
    }
    </ui:style>
    <g:HTMLPanel>
        Hello,
        <g:Button styleName="{style.important}" ui:field="button" />
    </g:HTMLPanel>
</ui:UiBinder>

```

UiBinder

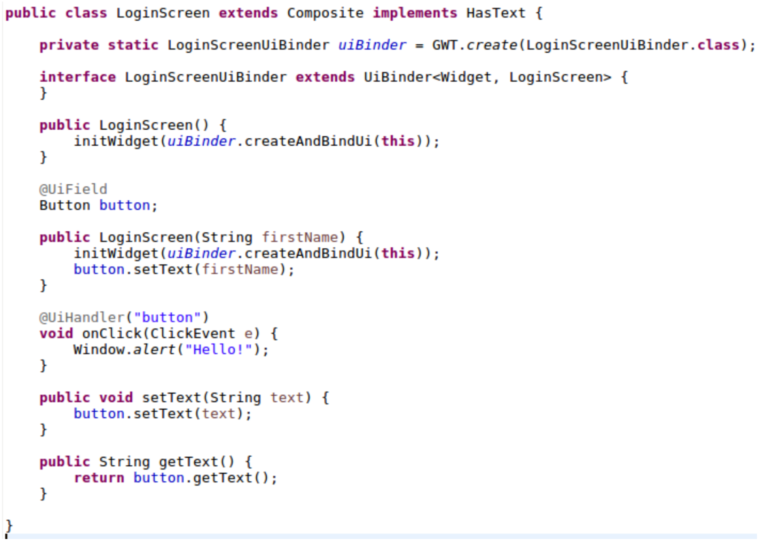
Result of example widget


DOCTYPE declaration to use HTML entities in XML
root element: UiBinder with namespace ui
CSS rules
widget wrapper
element binding name

```
<!DOCTYPE ui:UiBinder SYSTEM "http://dl.google.com/gwt/DTD/xhtml.ent">
<ui:UiBinder xmlns:ui="urn:ui:com.google.gwt.uibinder"
    xmlns:g="urn:import:com.google.gwt.user.client.ui">
    <ui:style>
    .important {
        font-weight: bold;
    }
    </ui:style>
    <g:HTMLPanel>
        Hello,
        <g:Button styleName="{style.important}" ui:field="button" />
    </g:HTMLPanel>
</ui:UiBinder>
```

UiBinder

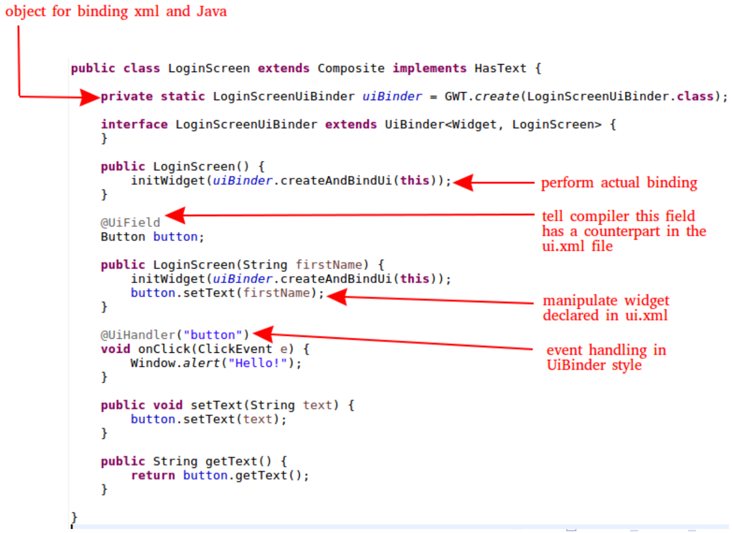
Result of example widget



```
public class LoginScreen extends Composite implements HasText {

    private static LoginScreenUiBinder uiBinder = GWT.create(LoginScreenUiBinder.class);

    interface LoginScreenUiBinder extends UiBinder<Widget, LoginScreen> {
    }

    public LoginScreen() {
        initWidget(uiBinder.createAndBindUi(this));
    }

    @UiField
    Button button;

    public LoginScreen(String firstName) {
        initWidget(uiBinder.createAndBindUi(this));
        button.setText(firstName);
    }

    @UiHandler("button")
    void onClick(ClickEvent e) {
        Window.alert("Hello!");
    }

    public void setText(String text) {
        button.setText(text);
    }

    public String getText() {
        return button.getText();
    }

}
```

UiBinder

Result of example widget


object for binding xml and Java
perform actual binding
tell compiler this field has a counterpart in the ui.xml file
manipulate widget declared in ui.xml
event handling in UiBinder style

```java
public class LoginScreen extends Composite implements HasText {
    private static LoginScreenUiBinder uiBinder = GWT.create(LoginScreenUiBinder.class);

    interface LoginScreenUiBinder extends UiBinder<Widget, LoginScreen> {
    }

    public LoginScreen() {
        initWidget(uiBinder.createAndBindUi(this));
    }

    @UiField
    Button button;

    public LoginScreen(String firstName) {
        initWidget(uiBinder.createAndBindUi(this));
        button.setText(firstName);
    }

    @UiHandler("button")
    void onClick(ClickEvent e) {
        Window.alert("Hello!");
    }

    public void setText(String text) {
        button.setText(text);
    }

    public String getText() {
        return button.getText();
    }

}
```

XML and HTML

- .ui.xml file contains XML-like representation of HTML
- XML syntax: every opening taq requires closing tag, etc.
- GWT/Google namespace required for binding elements
  `xmlns:g="urn:import:com.google.gwt.user.client.ui"`

Element binding

in MyWidget.ui.xml:

```
<g:Button ui:field="mybutton" />
```

in MyWidget.java

```
@UiField
Button mybutton;
```

⇒ elements are bound, element defined in XML can be used programmatically, e.g. `myButton.setText("Click");`

Event Handling

- event handlers can be bound to UiBinder elements via @UiHandler annotation and void method
- alternative: bind handler programmatically
- element name to which the event handler is bound defined via `ui:field="mybutton"`
- type of event given in function argument (widget must be able to fire this event type)

```java
@UiHandler("mybutton")
void onClick(ClickEvent e) {
	Window.alert("Hello!");
}
```

ui:style

- CSS rules for a UiBinder widget can be defined in `<ui:style>` element
- advantage: main CSS file doesn't get too confusing
- disadvantage: resolution of conflicting rules from multiple sources not always trivial
- CSS rules need to be accessed with special syntax: `class="{style.grey}"` or `styleName="{style.grey}"`
- alternative: assign class programmatically via `button.getElement().addClassName("myclass")`

ui:Style

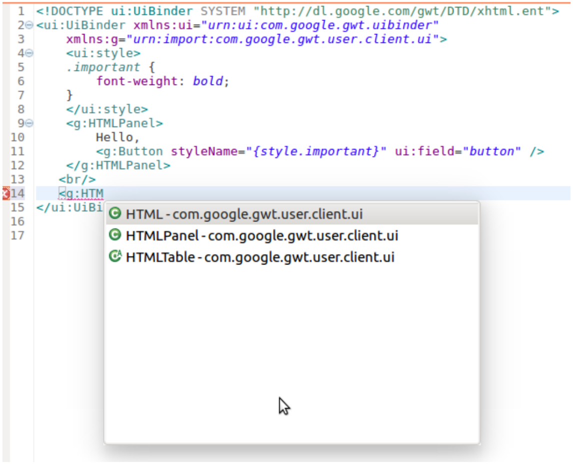
```
<ui:style>
       .important {
              font-weight: bold;
       }
</ui:style>

       ...

<g:Button styleName="{style.important}" ui:field="button" />
```

Eclipse Plugin Components

- autocompletion of elements in .ui.xml file
- error highlighting if binding can't be performed, e.g. spelling error in variable binding name

UiBinder

UiBinder Eclipse autocompletion

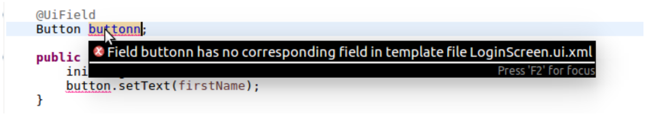


```
<!DOCTYPE ui:UiBinder SYSTEM "http://dl.google.com/gwt/DTD/xhtml.ent">
<ui:UiBinder xmlns:ui="urn:ui:com.google.gwt.uibinder"
    xmlns:g="urn:import:com.google.gwt.user.client.ui">
    <ui:style>
    .important {
        font-weight: bold;
    }
    </ui:style>
    <g:HTMLPanel>
        Hello,
        <g:Button styleName="{style.important}" ui:field="button" />
    </g:HTMLPanel>
   <br/>
   <g:HTM
</ui:UiBi
```

HTML - com.google.gwt.user.client.ui
HTMLPanel - com.google.gwt.user.client.ui
HTMLTable - com.google.gwt.user.client.ui

UiBinder

UiBinder Eclipse Error checks



```
@UiField
Button buttonn;

public ini
    ini
    button.setText(firstName);
}
```

Field buttonn has no corresponding field in template file LoginScreen.ui.xml

Press 'F2' for focus

Making use of UiBinder components

- UiBinder widgets can be used as Java objets and added to other widgets/RootPanel
- often necessary to define parametrized constructors to pass arguments to the UiBinder widget

UiBinder
Demonstration

- demonstration of how to create and use a UiBinder widget for a login page
- see project GWTUiBinderDemo for an working example

Maven and GWT

- so far in the seminar: Maven for dependency management, build cycle, . . .
- GWT projects using different structure (war directory, WEB-INF/lib)
- manually adding downloaded libraries to WEB-INF/lib
- ideally: combine the Maven project build power with the GWT architecture

Maven and GWT

- Maven archetype for GWT projects and GWT Maven plugin provided by Codehaus Mojo
- https://gwt-maven-plugin.github.io/gwt-maven-plugin/ (last access 2017-12-12)
- allows projects with 'normal' Maven structure (src/main/java, pom.xml, . . .) and with GWT compiler and setup

Maven and GWT

Step 1: Archetype

```
mvn archetype:generate
-DarchetypeGroupId=org.codehaus.mojo
-DarchetypeVersion=2.8.1
-DarchetypeArtifactId=gwt-maven-plugin
```

(follow interactive steps on command line)

Maven and GWT

Step 2: Import

Import project into Eclipse:
File $\rightarrow$ *Import* $\rightarrow$ *Maven* $\rightarrow$ *Maven* $\rightarrow$ *Existing Maven Project*

Maven and GWT

Step 3.1: BuildPath Configuration

Make the module recognizable: adapt the BuildPath such that NONE resource is excluded from src/main/resources (since module descriptor is in there and by default ignored !)

Maven and GWT

Step 3.2: Enable GWT Nature

If not enabled, enable the GWT project nature via *Right-click* → *Properties* → *Google* → *Web Application (use GWT, ensure module is selected)*

Deployment

- deployment: installing an application on a server
- i.e. putting a compiled version of the application onto a server that hosts the application
- different from development mode where application is only compiled on-demand
- compiled version on server significantly faster

Deployment
Compilation of project

- GWT comes with a compiler that builds *web archives* (war)
- *war* contains everything (code/resources) required to execute the application
- GWT allows to set the level of log in the compiler to detect potential problems

Deployment
Compilation of project

Compilation in a 'normal' GWT project:

Right-click $\rightarrow$ *Google* $\rightarrow$ *GWT Compile*

Compilation in a GWT Maven project:

Right-click $\rightarrow$ *Run as Maven build . . .* , specify the goal "package"

Deployment
Compilation of project

- result of steps on previous slide: compiled project
- for normal GWT project: package the complete content of the *war* directory in a *war* archive file
- for GWT Maven project: *.war* directly assembled in target directory

Deployment

- war file can be deployed to server, e.g. Apache Tomcat server
- Tomcat: hosts web applications, by default on port 8080
- applications deployed via Tomcat can be accessed by everyone who has access to the server

```
http://myserver.com:8080/MyApplication
http://kos.sfs.uni-tuebingen.de:8080/SpanishTrainer/
```

Deployment

- possibility to install Tomcat locally and host applications there[2]
- computer can be accessed via the ip address[3] and applications can be opened from other sources, e.g. via mobile phone
- useful for testing whether application runs outside development mode/resources are available

[2] see `http://tomcat.apache.org/`

[3] on same device: localhost:8080/MyApp

Deployment
Demonstration

- package and deploy GWT project on local Tomcat

Exercise

see handout

References

GWT books used: [Tacy et al., 2013], [Cooper and Collins, 2008], [Kereki, 2010]

excellent tutorial from which part of the structure of the presentation has been inspired: https://www.tutorialspoint.com/gwt/index.htm

Robert Cooper and Charlie Collins. *GWT in Practice*. Manning Publications Co., 2008.

Federico Kereki. *Essential GWT: building for the web with Google Web toolkit 2*. Pearson Education, 2010.

Adam Tacy, Robert Hanson, Jason Essington, and Anna Tokke. *GWT in Action*. Manning Publications Co., 2013.