GWT 2.0

Framework zur Erstellung von browserbasierten Applikationen

Themen

- Einführung
- Features
- Entwicklungs-Zyklus
- UiBinder
- Vorteile / Nachteile

- MVP-Architektur (falls Zeit)

Historie

- GWT 1.0 25. Mai 2006
- GWT 1.3 18. Januar 2007
- GWT 1.7 13. Juni 2009
- GWT 2.0 8. Dezember 2009
- GWT 2.0.4 2. Juli 2010
- GWT 2.1 tbd

Was ist GWT?

- GWT = Google Web Toolkit
- Framework für AJAX Applikationen, welche in Java geschrieben und debugged, aber als hochoptimiertes Javascript in verschiedenen Browsern laufen
  - kein Javascript Knowhow notwendig
- Architektur: Client – Server
  - Client ist Javascript Applikation im Browser

Features 1/2

- Am Ende ist es einfach Javascript
- XmlHttpRequest? Da war doch was…
- Sprache der Backend Implementierung frei (Kommunikation über XML / JSON / GWT-RPC)
- Einbinden von Javascript (Bibliotheken) möglich
- Unterstützung des Back-Buttons im Browser
- Entwicklung / Optimierungen mit Standard Java-Tools möglich (z.B. Ecplise, IntelliJ, Netbeans, JProfiler)
- Unterstützung von Mehrsprachigkeit

Features 2/2

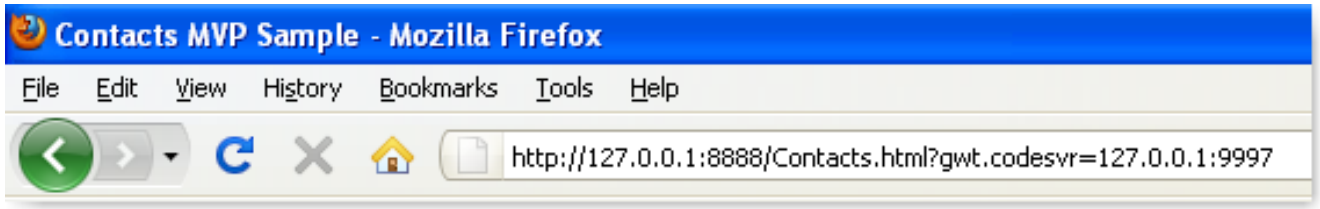
- In Browser-Development (<-> hosted mode)
- Code Splitting – Optimierung des Downloads
- UiBinder – Deklarative User-Interfaces
- ClientBundle – optimierte Ressourcen-Verwaltung
- Unittesting mit JUnit (GWTTestCase)

Entwicklungs-Vorgehen

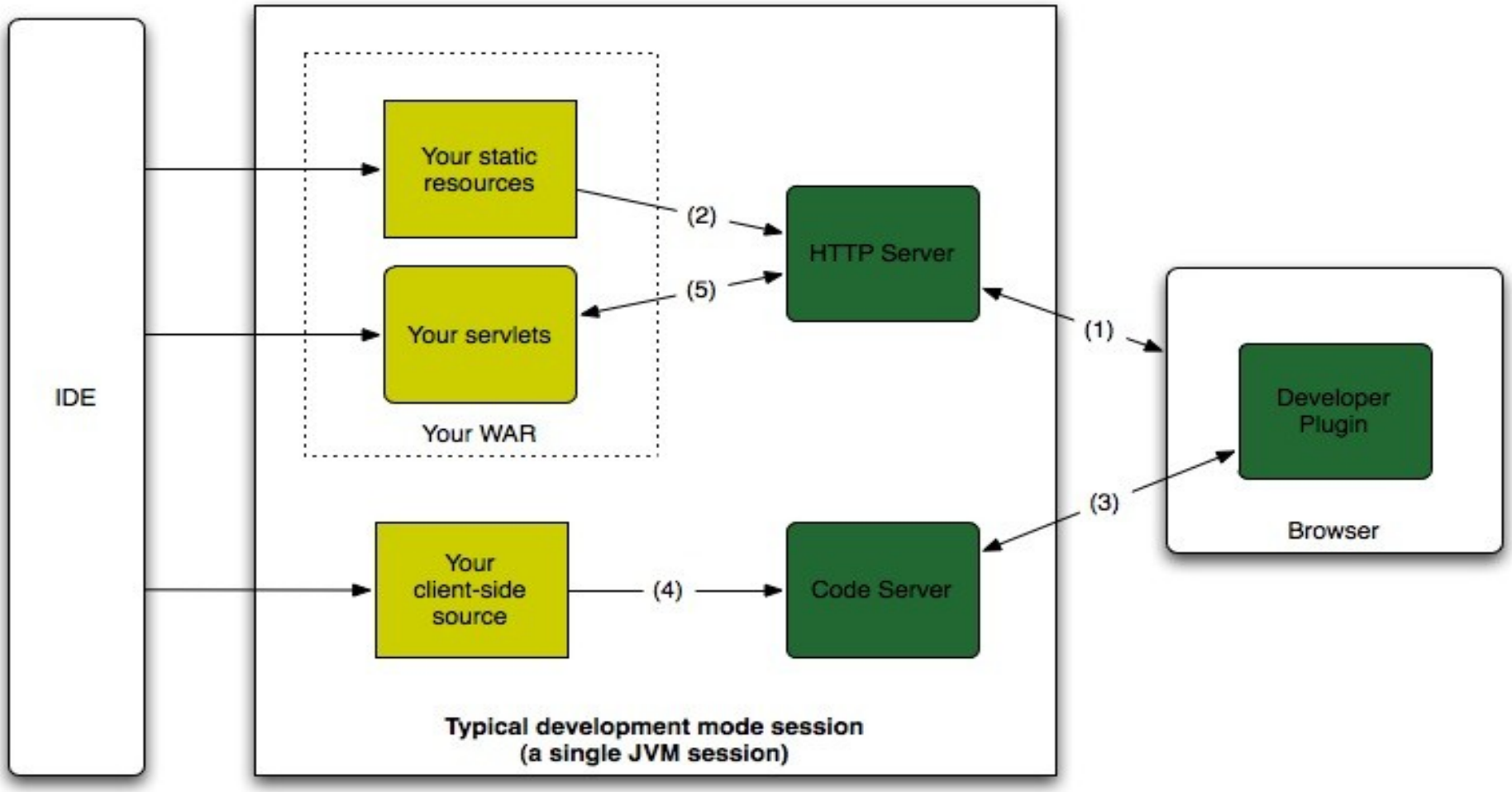
- Applikation schreiben
- im Developmentmode debuggen / testen
- optimieren
- Compilieren und Deployment auf einen Webserver (production mode)

Development Mode

- Entwickeln im Development Mode
  - Java Code wird in Bytecode kompiliert und läuft in einer lokalen JVM
  - Debugging / Break points in der IDE
  - Anzeige in einem Browser mit dem GWT developer plugin (IE, Firefox, Google Chrome), welches über TCP/IP mit der JVM kommuniziert
  - Browser Plugins nutzbar für Analysen im Browser (z.B. Firebug)



Architektur Development Mode

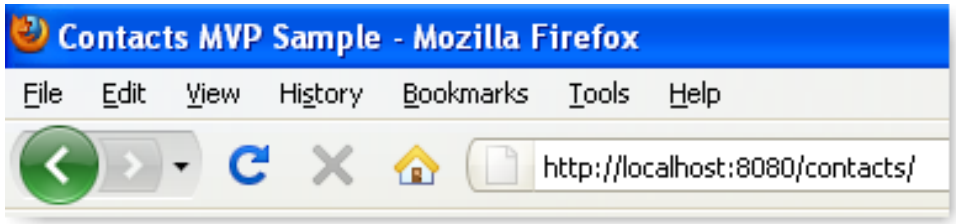


Optimieren

- GWT liefert zwei Werkzeuge für Optimierungen mit
  - Java to Javascript compiler
    - Method Inlining + devirtualization
    - Entfernung von ungebrauchtem Code
    - String-Optimierungen
  - Speedtracer
    - Performance-Probleme im Browser finden
    - Browser Layout-Operationen und CSS

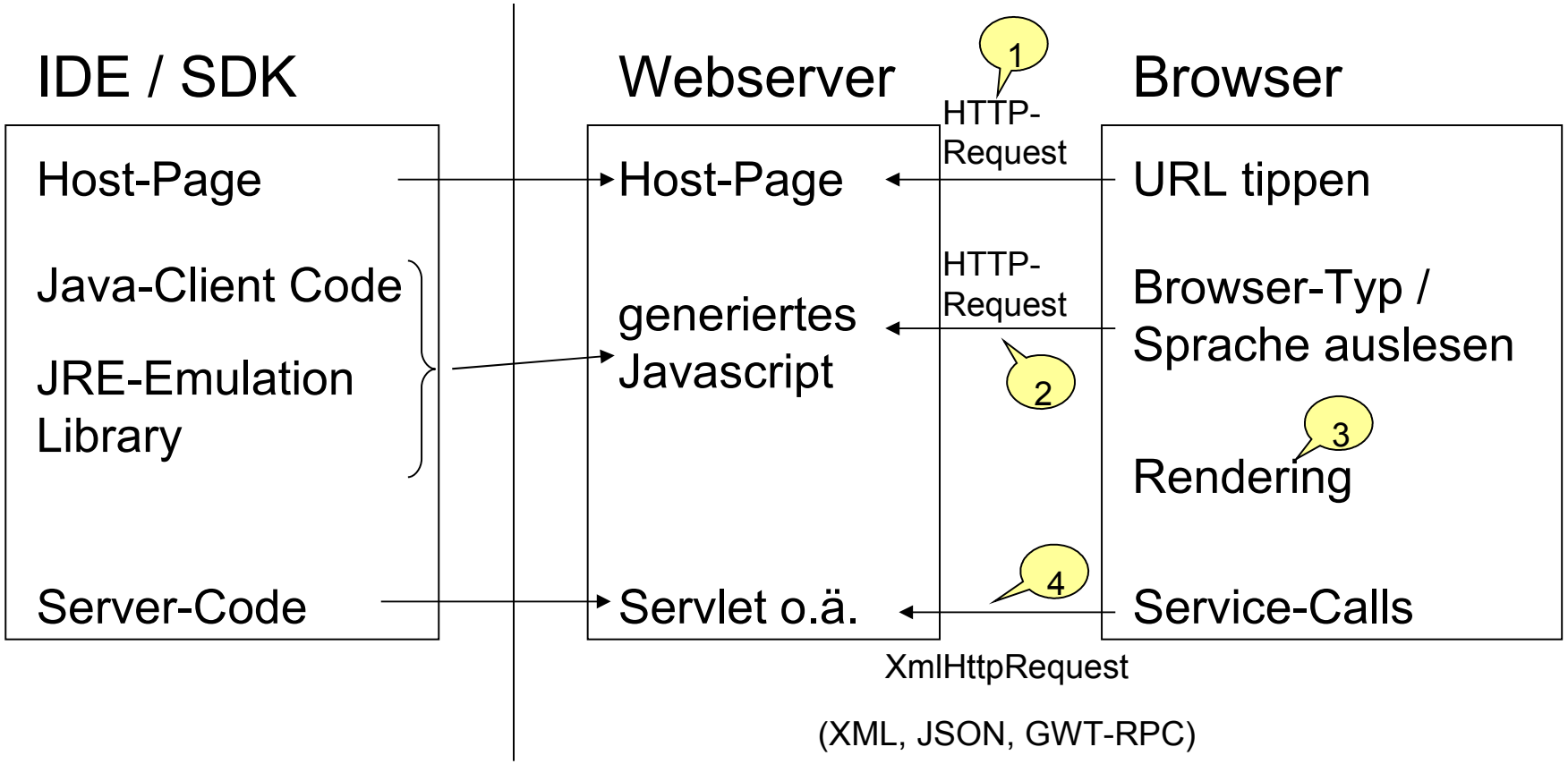
Production Mode

- Produktives Deployment im Production Mode
  - Java Code wird nach Javascript compiliert
  - pro Browser wird in jeder Sprache eine optimierte Variante erstellt
  - Hilfs-Skript, welches auswählt, welche Variante angezogen werden soll -> nur die passende Variante wird geladen
  - Applikationen laufen in allen bekannteren Browsern sowie mobilen Browsern für Android und das iPhone



Architektur production mode

- Produktion



Ohne UiBinder: „Swing“-Stil

```
private final TextBox firstName;
private final FlexTable detailsTable;
…
public EditContactView() {
  VerticalPanel contentDetailsPanel = new VerticalPanel();
  contentDetailsPanel.setWidth("100%");

  detailsTable = new FlexTable();
  detailsTable.setCellSpacing(0);
  detailsTable.setWidth("100%");
  firstName = new TextBox();
  …
  detailsTable.setWidget(0, 0, new Label("Firstname"));
  detailsTable.setWidget(0, 1, firstName);
  …
  contentDetailsPanel.add(detailsTable);
}
```

UiBinder: deklarativ

```
<?xml version="1.0" encoding="UTF-8"?>
<ui:UiBinder xmlns:ui='urn:ui:com.google.gwt.uibinder'
    xmlns:g='urn:import:com.google.gwt.user.client.ui'
    ui:generateFormat='com.google.gwt.i18n.rebind.format.PropertiesFormat'
    ui:generateKeys="com.google.gwt.i18n.rebind.keygen.MD5KeyGenerator"
    ui:generateLocales="default">
  <g:HTMLPanel>
    <table>
      <tr>
        <td><ui:msg description="firstName">Firstname</ui:msg></td>
        <td><g:TextBox ui:field="firstName" /></td>
      </tr>
      ...
      <tr>
        <td colspan="2"><g:Button ui:field="saveButton">
            <ui:msg description="saveButton">Save</ui:msg>
          </g:Button>
          ...</td>
      </tr>
    </table>
  </g:HTMLPanel>
</ui:UiBinder>
```

UiBinder: deklarativ

```
public class EditContactView extends Composite implements
    EditContactPresenter.Display {

  // @UiTemplate("EditContactView.ui.xml")
  interface MyUiBinder extends UiBinder<Widget, EditContactView>{}

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

  @UiField TextBox firstName;
  @UiField Button saveButton;
  ...

  public EditContactView() {
    initWidget(uiBinder.createAndBindUi(this));
  }
```

Vorteile

- Sehr gut dokumentiert
- Browser-Abhängigkeiten weitgehend eliminiert durch generiertes Javascript
- Browser „Back“ Button kann unterstützt werden
- Es können trotz Webapplikation sehr performante GUIs entwickelt werden
- Einfach anzuwenden für Java-Entwickler
- Klare Trennung von GUI- und Server-Code
- Open Source

Nachteile

- reine GWT-Applikationen sind nicht sichtbar für Such-Maschinen (kein Html, nur Javascript) -> Mischform notwendig
- Im schlimmsten Fall muss generiertes Javascript debuggt werden, um browserspezifische Probleme eliminieren zu können

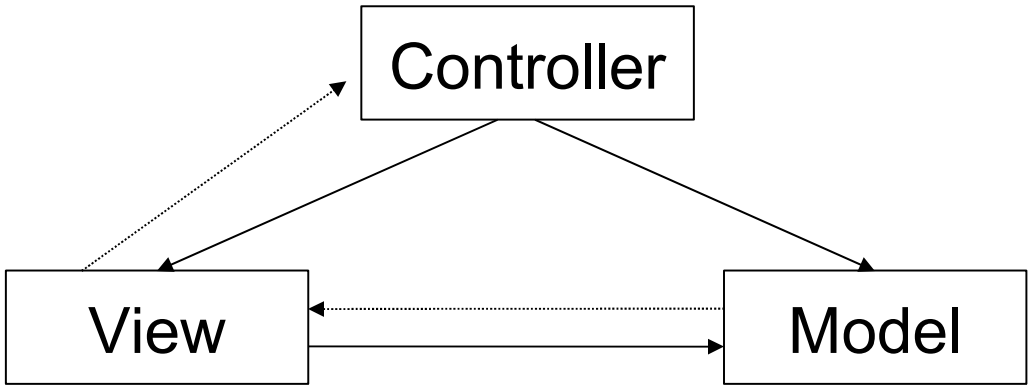
Referenzen

- http://code.google.com/webtoolkit/ - die GWT-eigene Seite
- http://www.vogella.de/articles/GWT/article.html - separates GWT-Tutorial
- http://dl.google.com/io/2009/pres/Th_0200_GoogleWebToolkitArchitecture-BestPracticesForArchitecting - ausführliche Beschreibung von best practices
- http://dl.google.com/io/2009/pres/W_1230_MeasureinMilliseconds-PerformanceTipsforGoogleWebToolk - Highspeed GWT Applikationen
- http://www.fh-htwchur.ch/uploads/media/Vortrag_02.pdf - Präsentation über GWT

Anhang

Model View Presenter

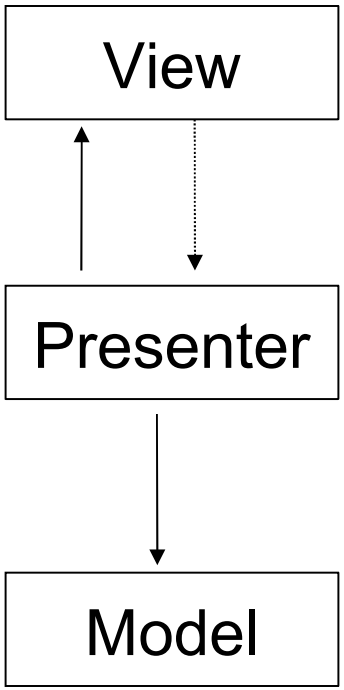
- Model View Controller



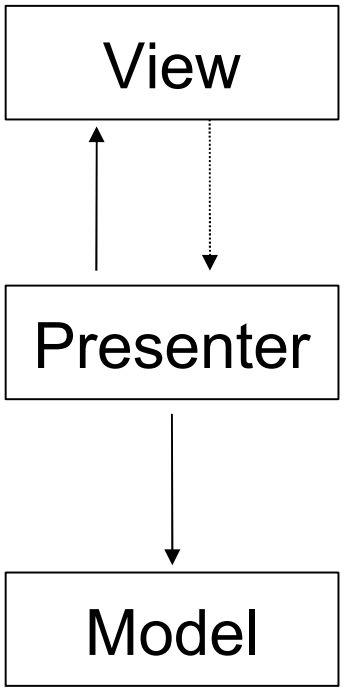
- Grösster Nachteil: immer noch viele Abhängigkeiten zwischen den Komponenten

M odel V iew P resenter

- Model View Presenter versucht, die Abhängigkeiten zu minimieren



M odel V iew P resenter




```
• public class EditContactPresenter
  implements Presenter{
•   public interface Display {
•     HasClickHandlers getSaveButton();
```

```
public class Contact implements Serializable {
  public String id;
  public String firstName;
  public String lastName;
  public String emailAddress;
```

M odel V iew P resenter

- Presenter beinhaltet Interface, welches von der View implementiert werden muss

```
public class EditContactPresenter implements
    Presenter{

  public interface Display {
    HasClickHandlers getSaveButton();
    HasClickHandlers getCancelButton();
    HasValue<String> getFirstName();
    HasValue<String> getLastName();
    HasValue<String> getEmailAddress();
    Widget asWidget();
  }
```

M odel V iew P resenter

- View-Klasse implementiert das Interface
- Kaum Funktionalität -> einfach zu „mocken“

```
public class EditContactView extends Composite
    implements EditContactPresenter.Display {

@UiField TextBox firstName;
@UiField Button saveButton;

public HasValue<String> getFirstName() {
    return firstName;
}

public HasClickHandlers getSaveButton() {
    return saveButton;
}
```

M odel V iew P resenter

- Presenter registriert sich für Events der View

```
private void bind() {
    this.display.getSaveButton().addClickHandler(
      new ClickHandler() {
        public void onClick(ClickEvent event) {
          doSave();
        }
    });

    this.display.getCancelButton().addClickHandler(
      new ClickHandler() {
        public void onClick(ClickEvent event) {
        eventBus.fireEvent(new EditContactCancelledEvent());
      }
    });
  }
```

M odel V iew P resenter

- Aufruf der EditContactView -> Vorbereiten der zu editierenden Daten

```
public void prepareEdit(Contact editContact) {
  contact = editContact;

  display.getFirstName().setValue(contact.getFirstName());
  display.getLastName().setValue(contact.getLastName());
  display.getEmailAddress().setValue(contact.getEmailAddress());
}
```

M odel V iew P resenter

- Abspeichern der Änderungen (Update Model)

```
private void doSave() {
  contact.setFirstName(display.getFirstName().getValue());
  contact.setLastName(display.getLastName().getValue());
  contact.setEmailAddress(display.getEmailAddress().getValue());

  rpcService.updateContact(contact, new AsyncCallback<Contact>() {
    public void onSuccess(Contact result) {
      eventBus.fireEvent(new ContactUpdatedEvent(result));
    }
    public void onFailure(Throwable caught) {
      Window.alert("Error updating contact");
    }
  });
}
```