1.– 4. September 2014
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

JSF trifft JavaScript

JSF-Komponenten mit JavaScript für Enterprise-Anwendungen

Sven Kölpin
open knowledge GmbH

JSF trifft JavaScript

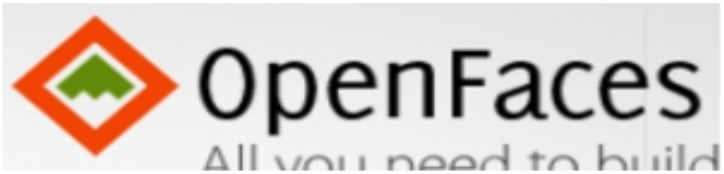
JSF-Komponenten mit JavaScript erstellen

Sven Kölpin

open knowledge **GmbH**

RichFaces

ICEfaces



Probleme aus der Praxis

- Kundenanforderungen an die GUI
  - Individuelle Probleme
  - Erweiterbarkeit
    - Frameworks sind Blackboxes
- Third-Party-Libraries
  - Boilerplate-Code
  - Bugs

Roadmap

1. **JSF Composite Components & Behavior API**
2. JSF und JavaScript: Grundlagen & Pattern
3. JSF und JavaScript: JSON-Kommunikation

JSF-Composite Components

- JSF-Code in Komponenten auslagern
  - Parametrisierbar
  - Wiederverwendbar
  - Ziel: deklarative Kapselung von GUI-Logik

- Seit JSF 2.0

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ok="http://java.sun.com/jsf/composite/ok">

<ok:labelTextBox
      value="#{createCustomerBean.firstName}"
      name="Firstname">
</ok:labelTextBox>
```

JSF Composite Components

- webapp/resources/**ok**/labelTextBox.xhtml
  → xmlns:ok="http://java.sun.com/jsf/composite/**ok**"

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:composite="http://java.sun.com/jsf/composite"
      xmlns:jsf="http://xmlns.jcp.org/jsf">

<composite:interface>
    <composite:attribute name="name" />
    <composite:attribute name="value"/>
</composite:interface>

<composite:implementation>
    <label jsf:id="lbl" jsf:for="inputComponent" jsf:value="#{cc.attrs.name}"/>
    <input  type="text" jsf:id="inputComponent" jsf:value="#{cc.attrs.value}"/>
</composite:implementation>
```

JSF Composite Components

- #{cc}-Attribut

```
<composite:interface>
   <composite:attribute name="name" />
   <composite:attribute name="value"/>
</composite:interface>

<composite:implementation>
   <label jsf:id="lbl" jsf:for="inputComponent" jsf:value="#{cc.attrs.name}"/>
   <input  type="text" jsf:id="inputComponent" jsf:value="#{cc.attrs.value}"/>
</composite:implementation>
```

JSF Composite Components

- #{cc}-Attribut
  - Jede Component hat UIComponent-Backingbean
    → Zugriff über EL via #{cc.*}
  - Eigene Backingbeans möglich
- Nützliche Eigenschaften
  - cc.attrs.* (Parameter-Map)
  - cc.clientId

Client Behavior

- Idee: Komponenten um clientseitige Funktionalität erweitern
- Einfache Erweiterung von Komponenten um JavaScript-Funktionalität
  - Spätere Autoren müssen keine Kenntnis von Javascript haben
- Mittel: Deklarative Kapselung von JavaScript-Code in JSF-Behavior
  - Ähnlich zu <f:ajax/> - Tag

Client Behavior

- Deklarative Kapselung von JavaScript

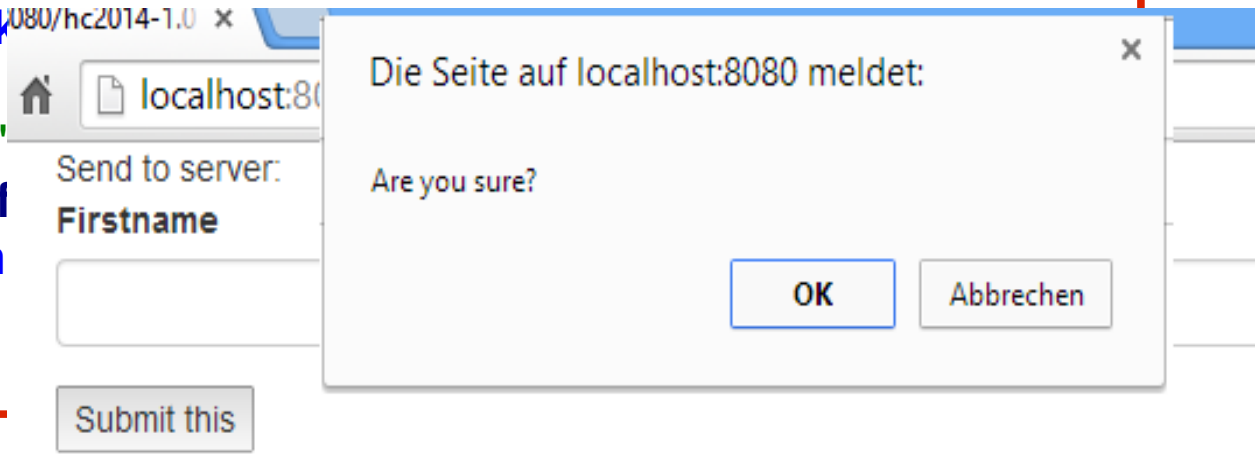
```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:okb="http://openknowledge.de/behavior" ...>

<input type="submit" jsf:id="submit" value="Submit this">
   <okb:confirm message="Are you sure?"/>
   <f:ajax render="parentForm"/>
</input>
```

Client Behavior

- Deklarative Kapselung von JavaScript

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ok
<input type="
   <okb:conf
   <f:ajax ren
</input>
```



Client Behavior

- Was braucht man?
  - Behavior-Klasse
    - extends ClientBehaviorBase
    - @FacesBehavior(“de.my.behavior“)
  - taglib.xml
    - Konfiguration der Komponente
  - JavaScript-Code
    - Das eigentliche Herzstück

Behavior-Klasse

```
@FacesBehavior("de.openknowledge.hc2014.SimpleBehavior")
public class SimpleBehavior extends ClientBehaviorBase {
  private String message;

  @Override
  public String getScript(ClientBehaviorContext behaviorContext) {
    return "return confirm('“+getMessage()+“')";
  }

  public String getMessage() {
    return message;
  }

  public void setMessage(String msg) {
    this.message = msg;
  }
...
```

Behavior-Klasse

- Never do JavaScript in Java!

```
@FacesBehavior("de.openknowledge.hc2014.SimpleBehavior")
@ResourceDependencies(value = {
    @ResourceDependency(name = "ok/confirm.js")
})
public class SimpleBehavior extends ClientBehaviorBase {
  private String message;

  @Override
  public String getScript(ClientBehaviorContext behaviorContext) {
    return "ok.confirmDialog.show();";
  }
 ...
}
```

Taglib.xml

- META-INF/ok.taglib.xml

```
<facelet-taglib version="2.2"  … >
   <namespace>http://openknowledge.de/behavior</namespace>
    <tag>
       <tag-name>confirm</tag-name>
       <behavior>
           <behavior-id>
                de.openknowledge.hc2014.SimpleBehavior
           </behavior-id>
       </behavior>
        <attribute>
          <name>message</name>
       </attribute>
     </tag> ...
```

Was passiert da?

- JSF:

```
<input type="submit" jsf:id="submit" value="Submit this">
   <okb:confirm message="Are you sure?"/>
   <f:ajax render="parentForm"/>
</input>
```

- HTML:

```
<input id="parentForm:submit"
      onclick="jsf.util.chain(this,event,
         'return confirm(\'Are you sure?\')',
         'mojarra.ab(this,event,\'action\',0,\'parentForm\')');"/>
```

Roadmap

1. JSF Composite Components & Behavior API

2. **JSF und JavaScript: Grundlagen & Pattern**

3. JSF und JavaScript: JSON-Kommunikation

Enterprise JavaScript


```
function a(){
    alert("im starting");
    document.getElementById("content").innerHTML = "IM BLINKING";
}
function doBlink() {
    var blink = document.all.tags("div")
    for (var i=0; i < blink.length; i++)
        blink[i].style.visibility = blink[i].style.visibility == "" ? "hidden" : ""
}
function startBlink() {
    a();
    setInterval("doBlink()",500)
}
window.onload = startBlink;
```

Enterprise JavaScript

Enterprise JavaScript

- Globale Variablen und Funktionen vermeiden
  → Namespace Pattern
- Module und Vererbung verwenden
  → Module Pattern
  → Information Hiding
- Build-Systeme verwenden
  → GruntJS
  → GulpJS

- Be unobstrusive!

Enterprise JavaScript - Namespaces

- Variablen außerhalb von Funktionen sind global!

```
var myglobal = "hello"; // antipattern
console.log(myglobal); // "hello"
console.log(window.myglobal); // "hello"
console.log(window["myglobal"]); // "hello"
console.log(this.myglobal); // "hello"
```

- Kollisionen vorprogrammiert
  - Mit Libraries
  - In verschiedenen JS-Dateien
  - Wartbarkeit?

Enterprise JavaScript - Namespaces

- Namespaces verhindern Namenskollisionen
  - Ähnlich zu Java Packages
  - Namespaces sind JavaScript-Objekte

```
var Application = {};
Application.SubModul = {};
Application.SubModul.SomeClass = function()...;
Application.SubModul = {
    SomeClass : function(){...}
};
```

Enterprise JavaScript – Module Pattern

- Wird von nahezu allen JS-Bibliotheken verwendet
  - Strukturiert Code
  - Information Hiding
  - Einfache Vererbung
- Herzstück
  - Self-Invoking Functions
  - Function-Scope von JavaScript

Enterprise JavaScript – Module Pattern

```
//self invoking function als Modul
var Module = (function () {

})();
//öffentliche API zurückgeben
var Module = (function () {
    var privateMethod = function(){...};
    return {
        publicMethod: function (arg1, arg2) {
            //access private methods...
        }
    };
})();
Module.publicMethod("A", "b");
```

GruntJS & GulpJS

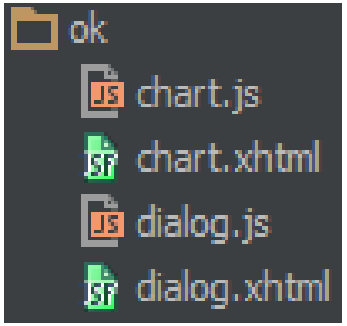
- Build-Systeme für JavaScript
  → „Maven für JavaScript“
- Verschiedene Tasks im Build-Lifecycle
  - JSHint (extrem wichtig!)
  - Minify JS, Minify CSS
  - UnitTests
  - …
- Ohne JS-Build-System keine enterprisefähige JS-Entwicklung!

JSF und JavaScript

JSF und JavaScript

- Know the DOM!
  - JSF-Komponenten in HTML
    - , <input> & <a> statt <h:panelGroup>, <h:commandButton> & <h:commandLink>
      → **HTML-Friendly Markup, JSF 2.2**
  - JSF-Ids (parentForm:j_idt4:filter)
    - → **jQuery-selectors (parentForm\\:j_idt4\\:filter)**

- JS in Komponenten:
  - Be unobstrusive! → JS der Komponente in eigene Datei!
  - Initialisierung des JS-Codes über Inline-Script-Tag
    - Möglichkeit zur Übergabe von cc.attrs.* Parametern



JSF-Components und JavaScript

```
<composite:implementation>

   <input type="text" jsf:id="filter" placeholder="Filter"/>
   <h:dataTable value="#{cc.attrs.value}" var="customer">
      ...
   </h:dataTable>

   <script type="text/javascript">
      ok.SimpleTableComponent.create({
         clientId: "#{cc.clientId}"
      });
   </script>

</composite:implementation>
```

JSF-Components und JavaScript

- JS in Komponenten: #{cc.clientId} als Grundlage

```
<input id="parentForm:j_idt4:filter" type="text"/>
<table style="width: 100%;">
...
</table>

<script type="text/javascript">
ok.SimpleTableComponent.create({
    clientId: "parentForm:j_idt4"
});
</script>
```

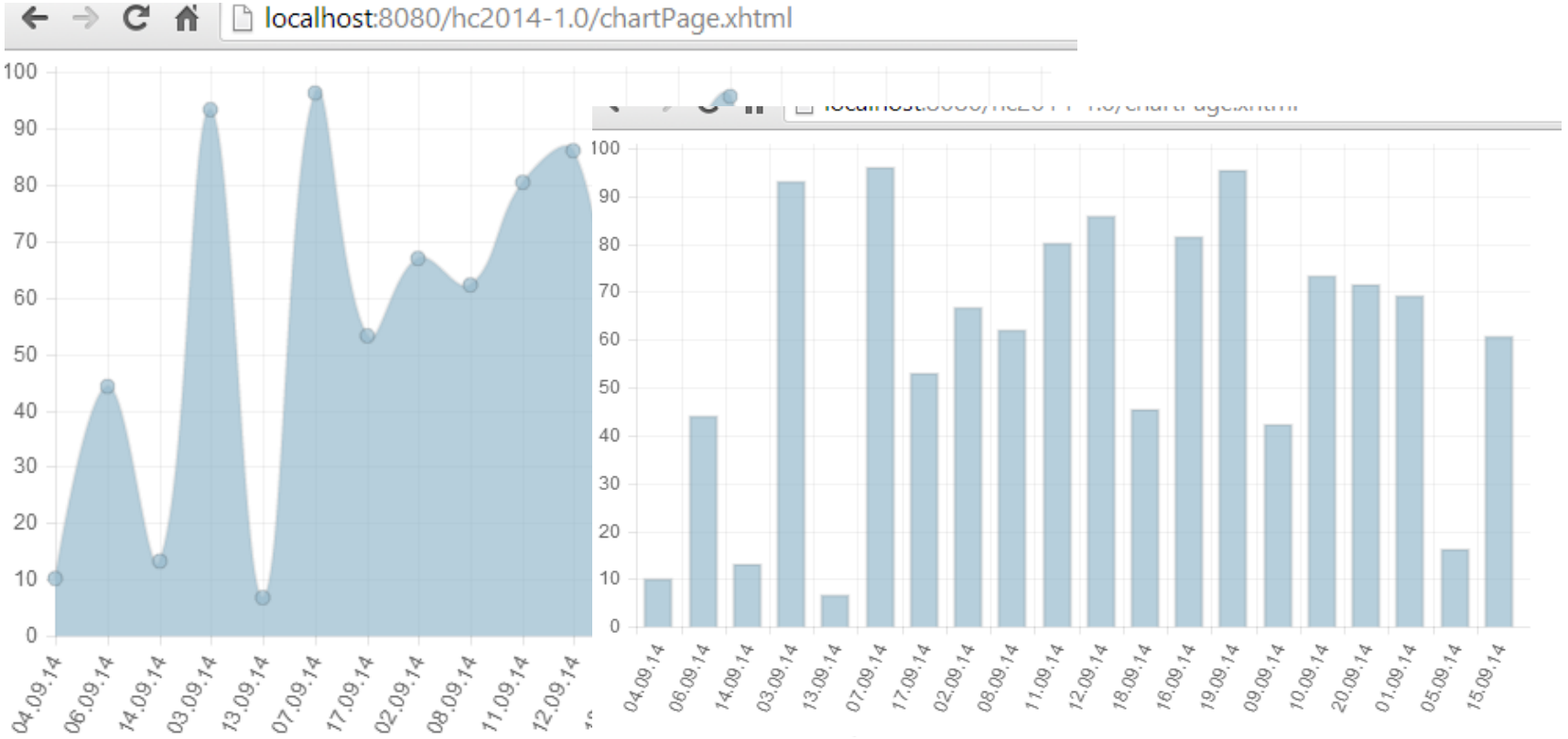
JSF-Components und JavaScript

- Und das JavaScript?

```
…
getJQueryElement : function (id) {
    return $("#" + (this.options.clientId + ":" + id).replace(/:/g, "\\:"));
},
init : function () {
    var searchField = this.getJQueryElement("filter").focus();
    searchField.on("keyup", function () {
        ….
    });
},
create : function (options) {      //{clientId : “#{cc.clientId}“}
    this.options = options;
    this.init();
    return this;
},…
```

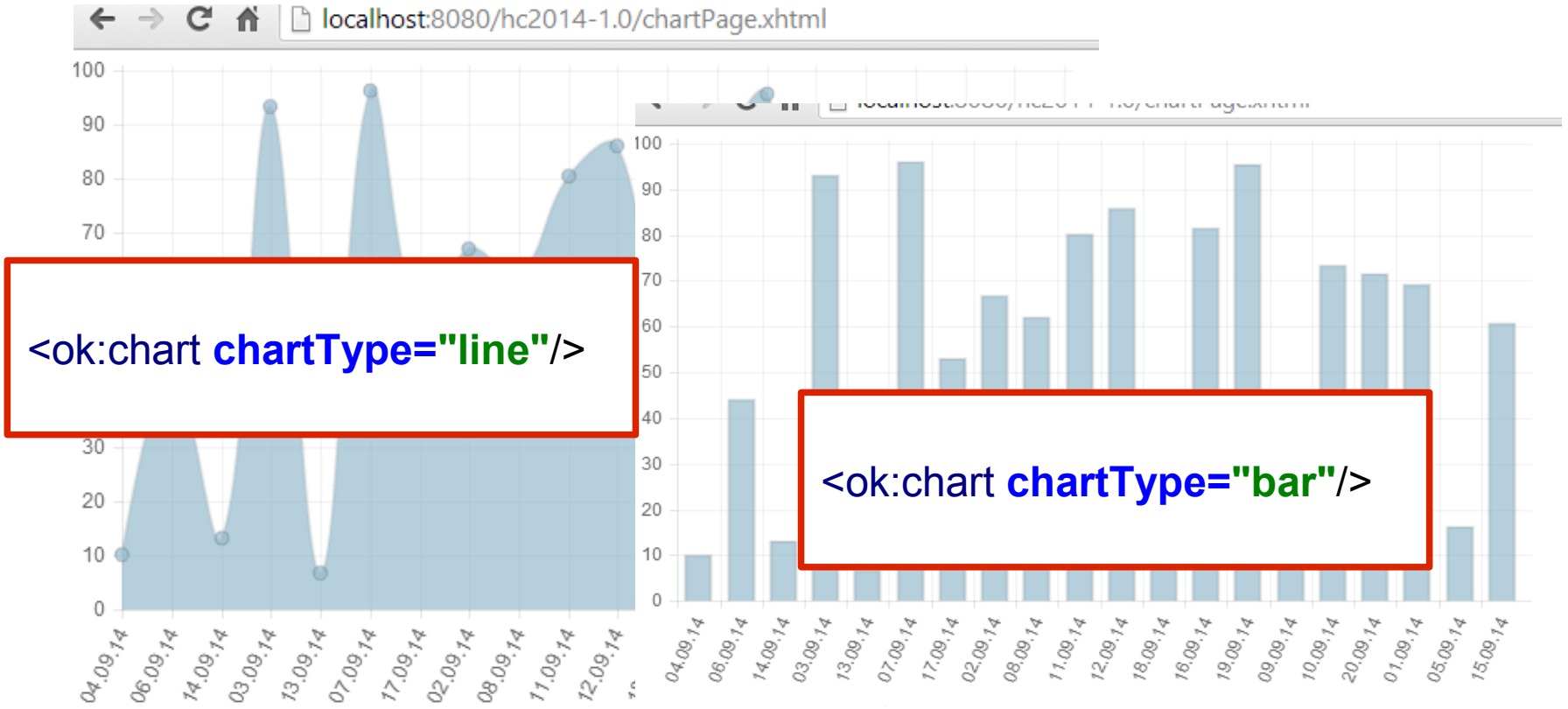
JSF-Components und JavaScript

- Deklarative Konfiguration der Komponente



JSF-Components und JavaScript

- Deklarative Konfiguration der Komponente



JSF-Components und JavaScript

- Deklarative Konfiguration der Komponente

```
<composite:interface>
   ...
   <composite:attribute name="chartType" type="java.lang.String"/>
</composite:interface>
<composite:implementation>
 ...
<script type="text/javascript">
   ok.ChartComponent.create({
      clientId: "#{cc.clientId}",
      chartType: "#{cc.attrs.chartType}"
   });

</script>
</composite:implementation>
```

JSF-Components und JavaScript

- Deklarative Konfiguration der Komponente

```
ok.ChartComponent = (function (okComponent) {
    var init = function () {
        if (this.options.chartType === "line") {
            createLineChart.call(this, dataSeries);
        }
        ...
    };
     ...
})(ok.CComponent);
```

JSF-Components und JavaScript

- Zusammenfassung:
  - Be unobstrusive
    - Jede Komponente hat eigene JS-Datei
  - Inline-JS nur zur Komponenteninitialisierung
    - #{cc.clientId} also Pflichtparameter
    - Deklarative Parameter bei der Initialisierung übergeben

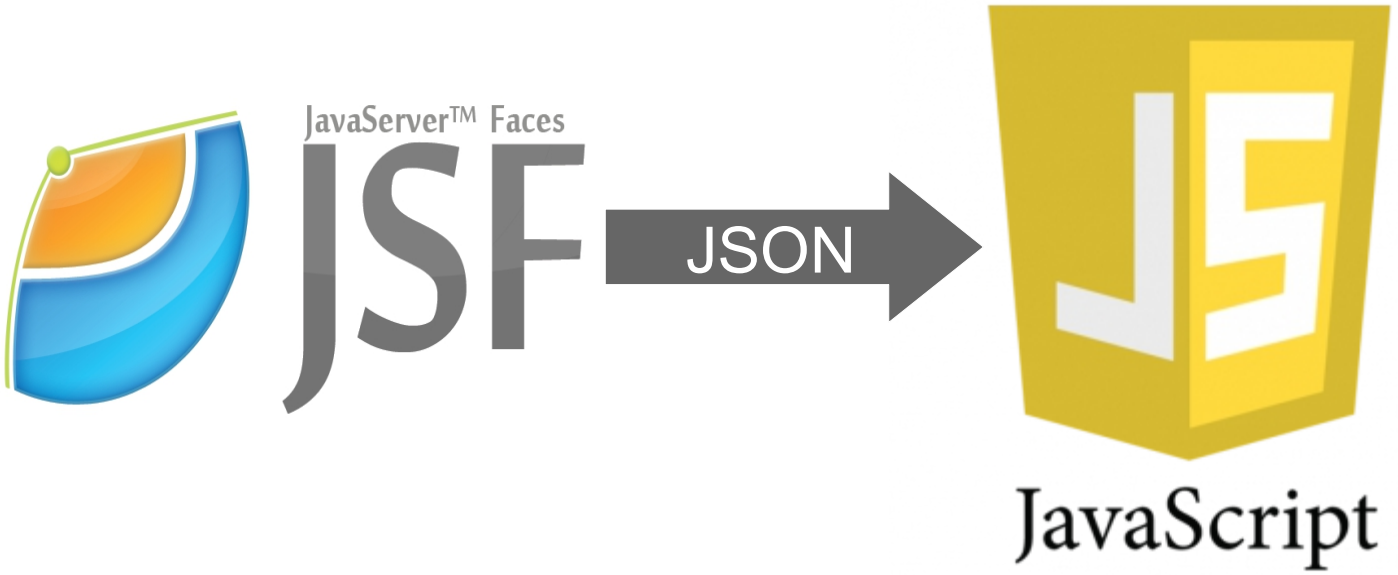
Roadmap

1. JSF Composite Components & Behavior API

2. JSF und JavaScript: Grundlagen & Pattern

3. JSF und JavaScript: JSON-Kommunikation

JSF → JavaScript



JSF → JavaScript

- JSF (Ajax) Kommunikation basiert auf XML
  - Übertragung semantischer Daten

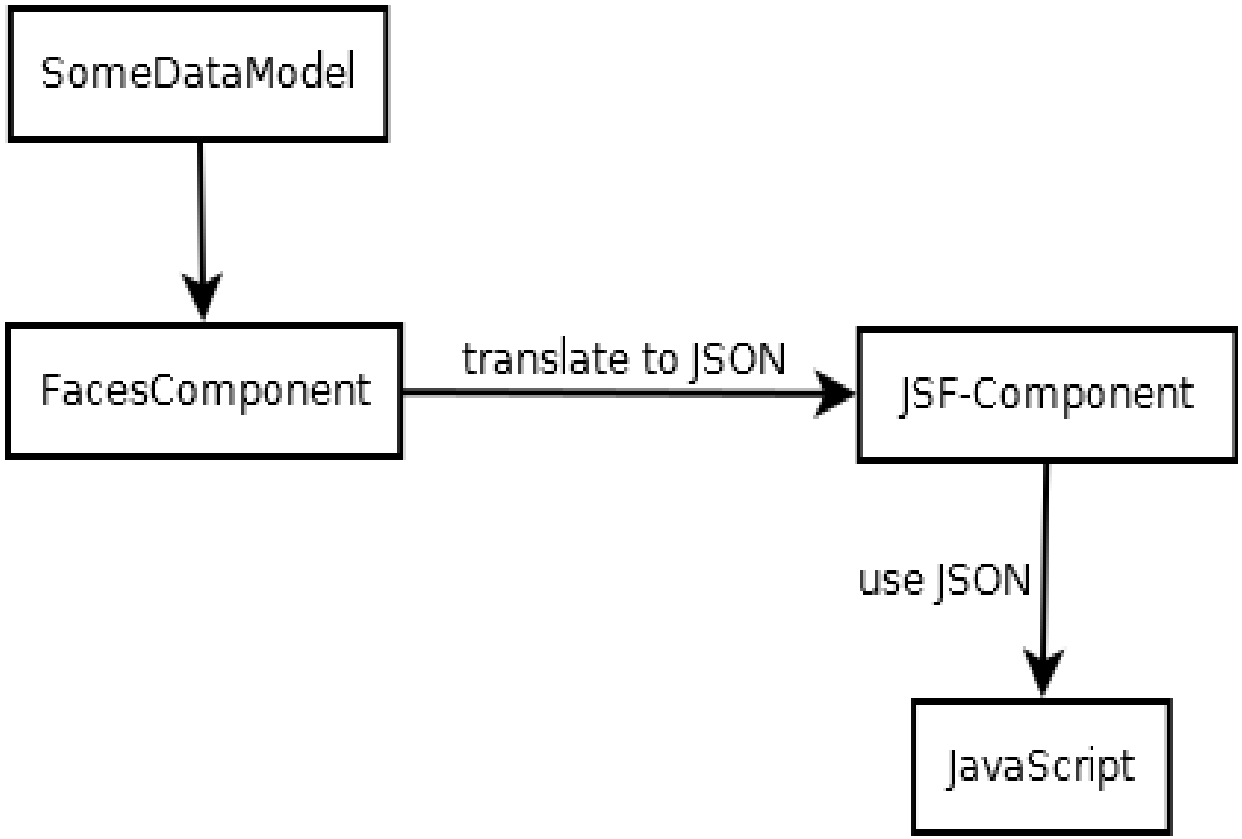
```
<partial-response id="j_id1"><changes><update id="parentForm"><!
[CDATA[
<form id="parentForm" method="post" class="container">
<input type="hidden" name="parentForm" value="parentForm" />
```

- JavaScript-Komponenten nutzen i. d. R. JSON
  - Übertragung „reiner“ Daten nötig

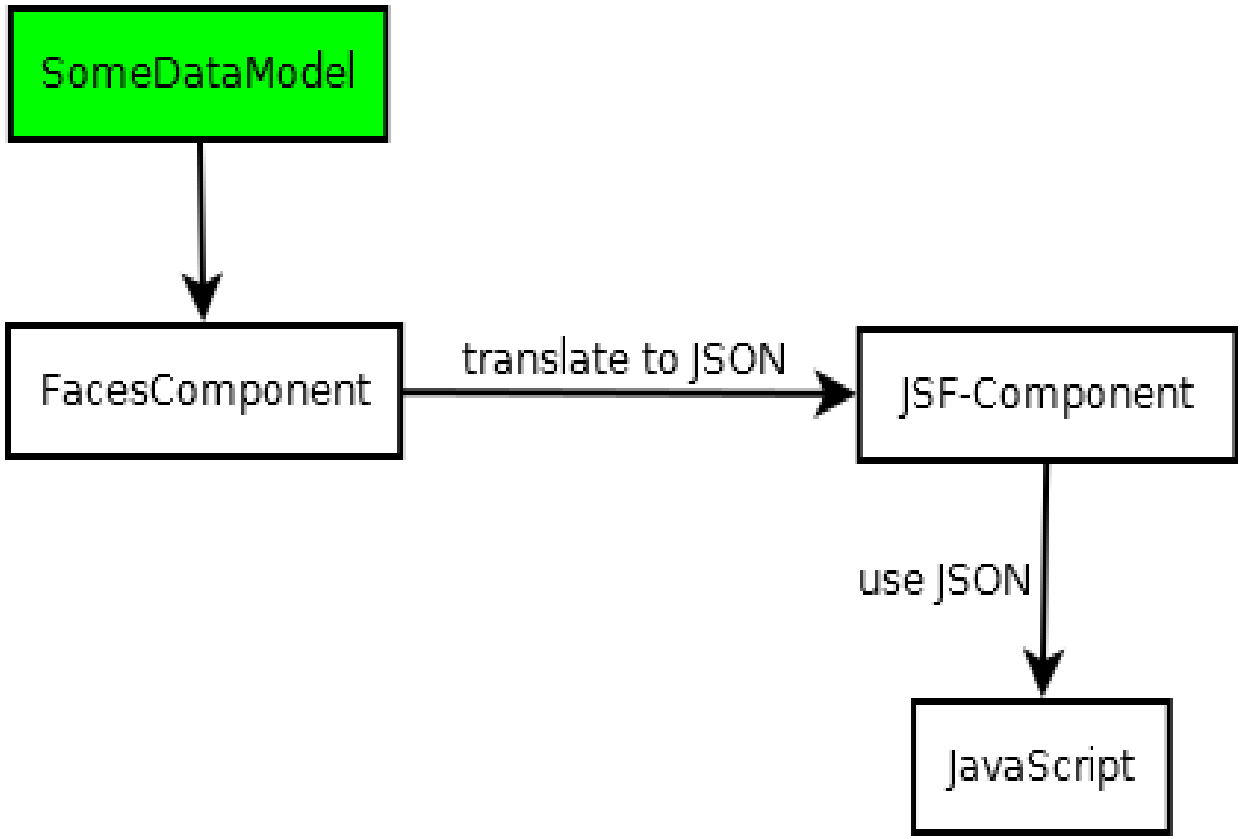
JSF → JavaScript

- Idee: Übertragung des JSON in Hidden-Input-Fields
  - Nutzung von JSF im Standard
  - Keine Libraries oder Workarounds nötig
  - Minimaler Overhead

JSF → JavaScript



JSF → JavaScript



JSF → JavaScript

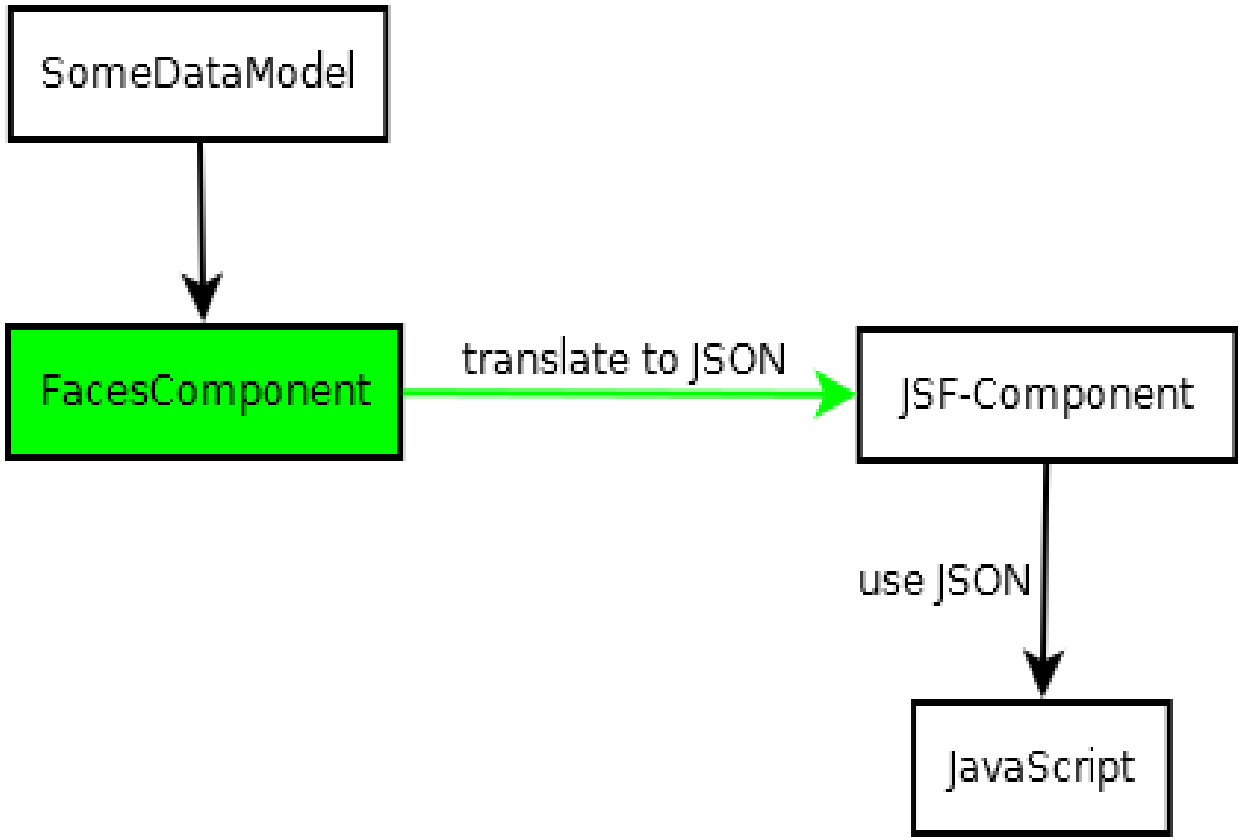
- Jede Komponente hat ein geeignetes Datenmodell
  - z.B. Chart - Menge aus x und y Werten
  - z.B. Map - Menge aus Lat. und. Lon. Werten

```
public abstract class ChartDataModel<T, X, Y> {
  private Map<X, Y> dataSeries;

  public Map<X, Y> getDataSeries() {
    return dataSeries;
  }
  protected abstract X getXValue(T t);

  protected abstract Y getYValue(T t);
}
```

JSF → JavaScript



JSF → JavaScript

- Komponenten-Klasse:
  - @FacesComponent
  - extends UINamingContainer → Reicht i.d.R.

```
<composite:interface
    componentType="de.openknowledge.ChartComponent">
```

- Datenmodell in der Komponenten-Klasse transformieren
  - JSON-Konvertierung Java EE 7 im Standard möglich

JSF → JavaScript

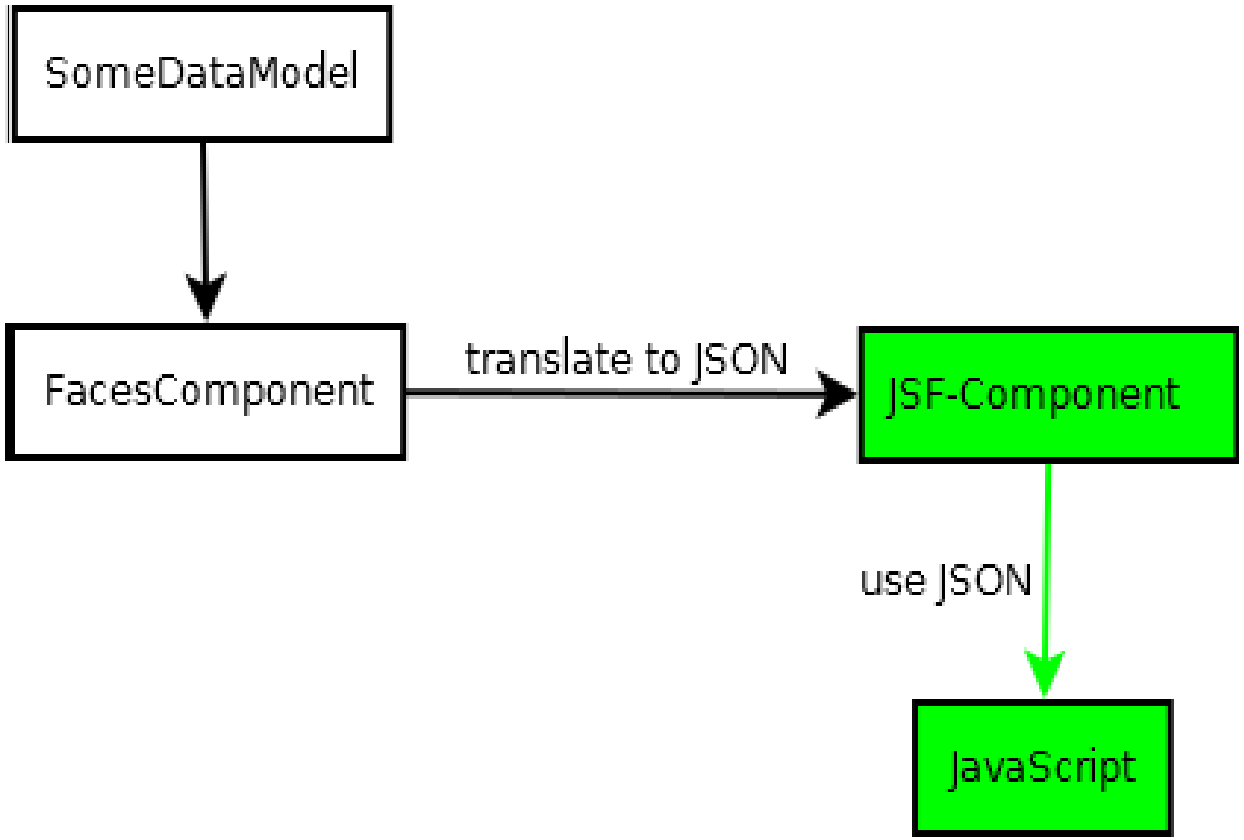
```
@FacesComponent(value = "de.openknowledge.ChartComponent")
public class ChartComponent extends UINamingContainer {

 ...

  public String convertToJson(Map dataSeries) {
    JsonArrayBuilder xAxesArray = Json.createArrayBuilder();
    JsonArrayBuilder yAxesArray = Json.createArrayBuilder();

    for (Object key : dataSeries.keySet()) {
      xAxesArray.add(key.toString());
      ...
    }
  }
...
```

JSF → JavaScript



JSF → JavaScript

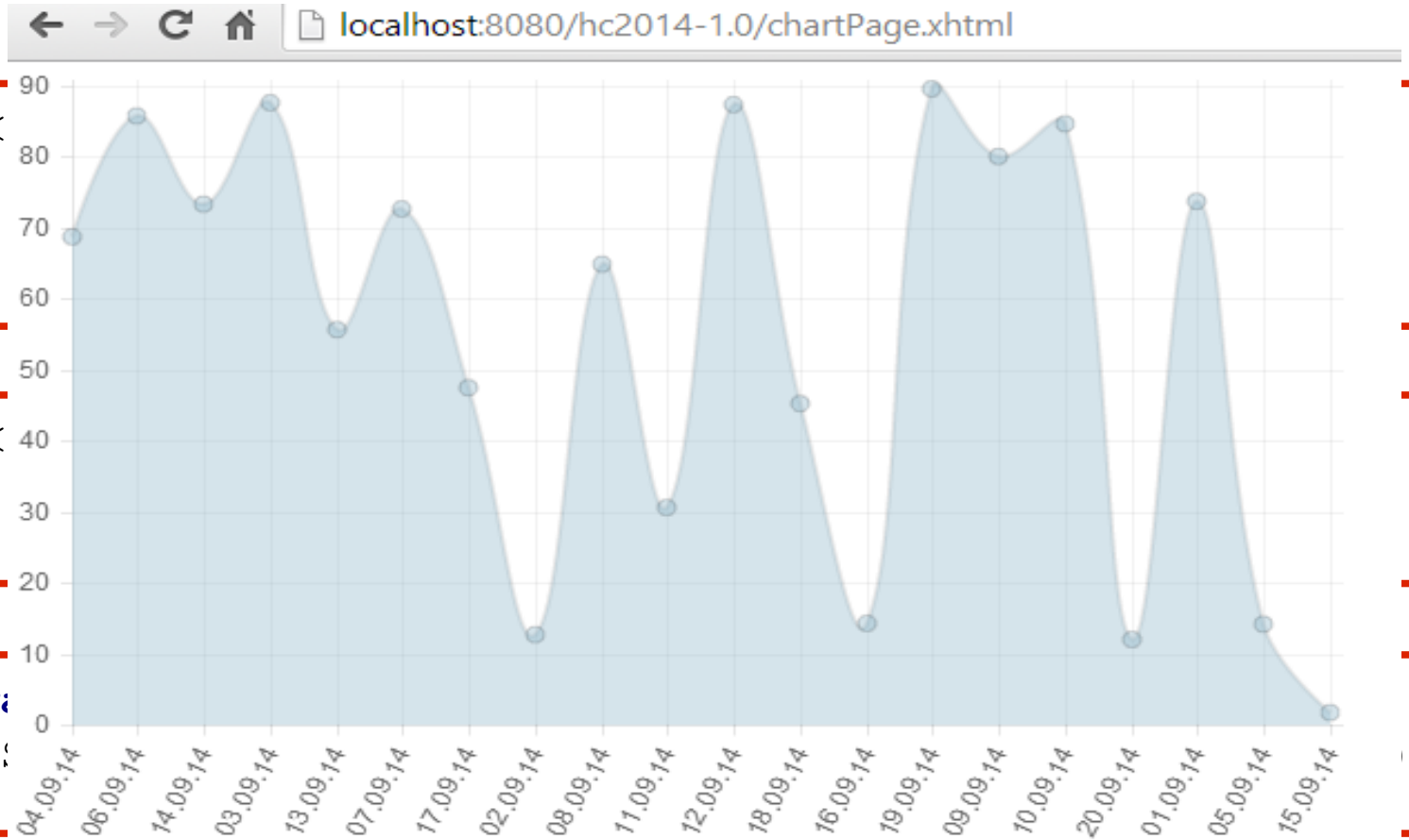
- Hidden-Input-Field mit JSON „befüllen“

```
<input
        type="hidden"
        jsf:id="jsonHelper"
        jsf:value="#{cc.createJsonData()}"/>
```

```
<input id="..." type="hidden"
        value="{"xAxes":["04.09.14","06.09.14"],
         "yAxes":["8.74","85.77"]}"/>
```

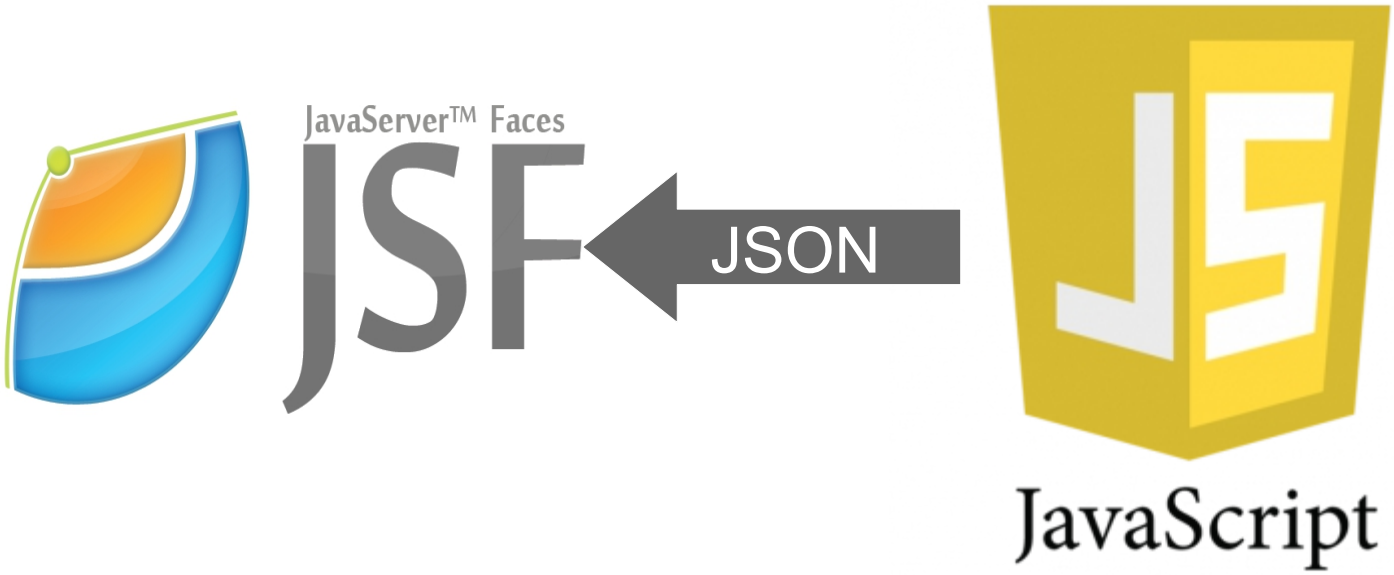
```
var chartDataJson =
JSON.parse(this.getJQueryElement("jsonHelper").val());
```

JSF → JavaScript



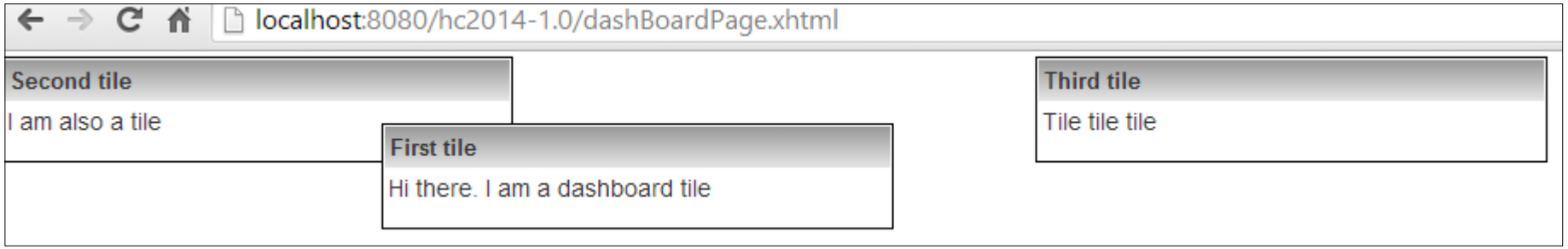
Und andersrum?

JSF ← JavaScript

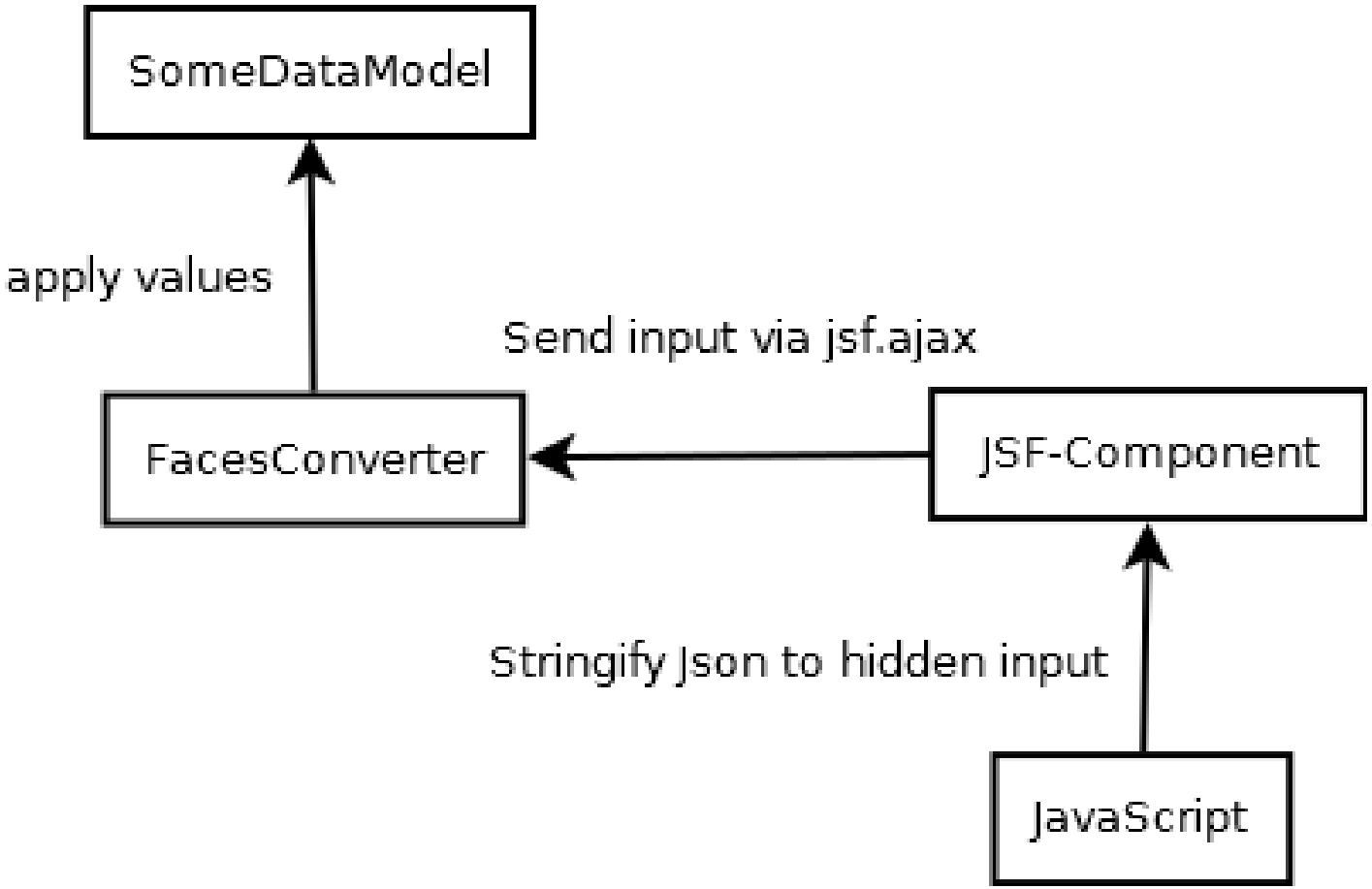


JSF ← JavaScript

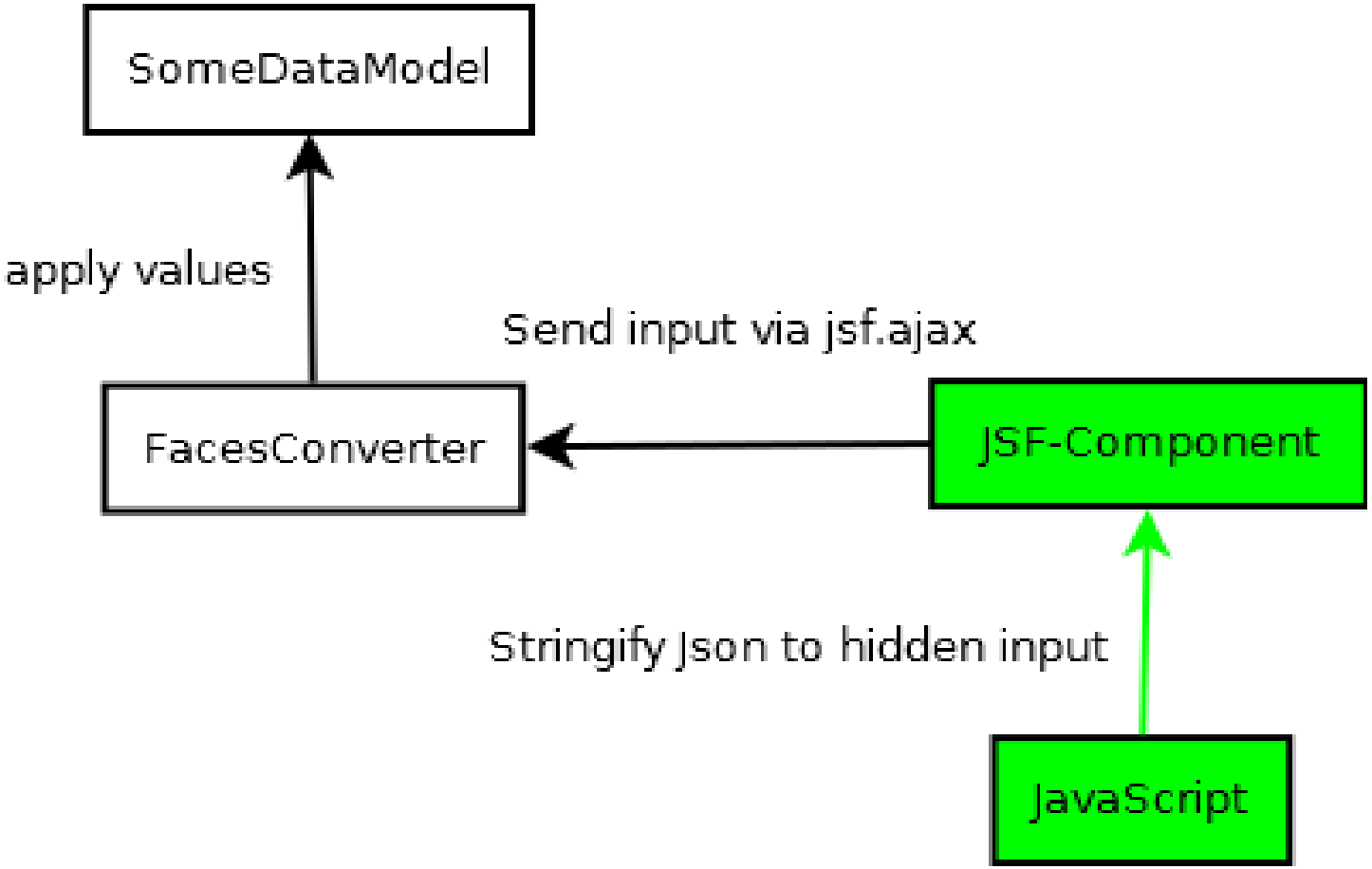
- Datenstrukturen der JS-APIs i.d.R in JSON
  - z.B. jQueries' sortable
- Idee: Hidden-Input-Field
  - JSON in Hidden-Field schreiben
  - Hidden-Field via jsf.ajax-API abschicken
  - JSF-Converter zur „Übersetzung“ nutzen



JSF ← JavaScript



JSF ← JavaScript



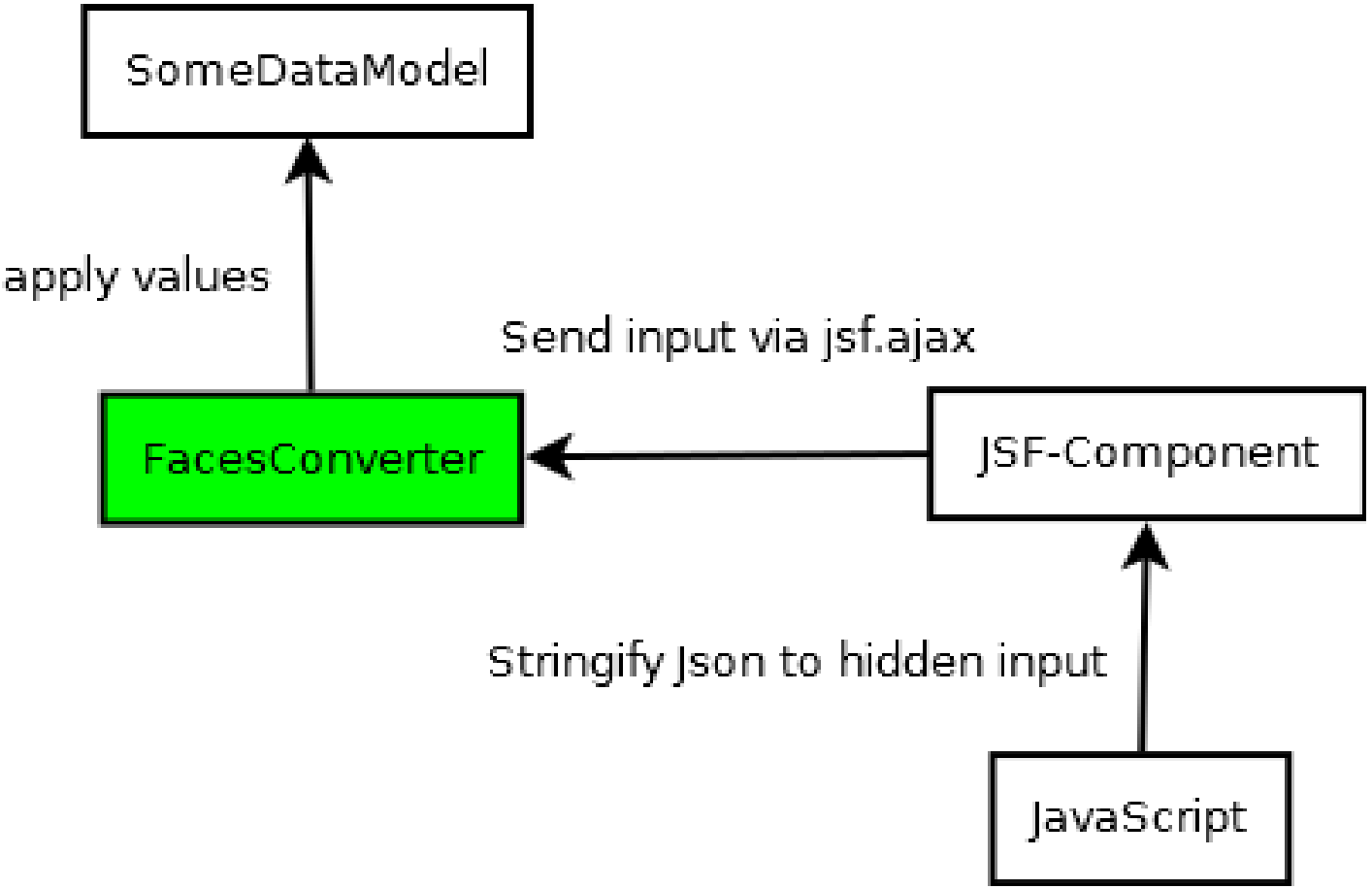
JSF ← JavaScript

```
<input type="hidden" jsf:id="jsonHelper"
       jsf:value="#{cc.attrs.value.tileOrder}">
  <f:converter
       converterId="de.openknowledge.OrderConverter"/>
</input>
```

```
var jsonHelper = this.getJQueryElement("jsonHelper");
jsonHelper
    .val(JSON.stringify(dashBoard.sortable("toArray")));
```

```
jsf.ajax.request(jsonHelper.get(0), null, {
    execute: "@this"
});
```

JSF ← JavaScript



JSF ← JavaScript

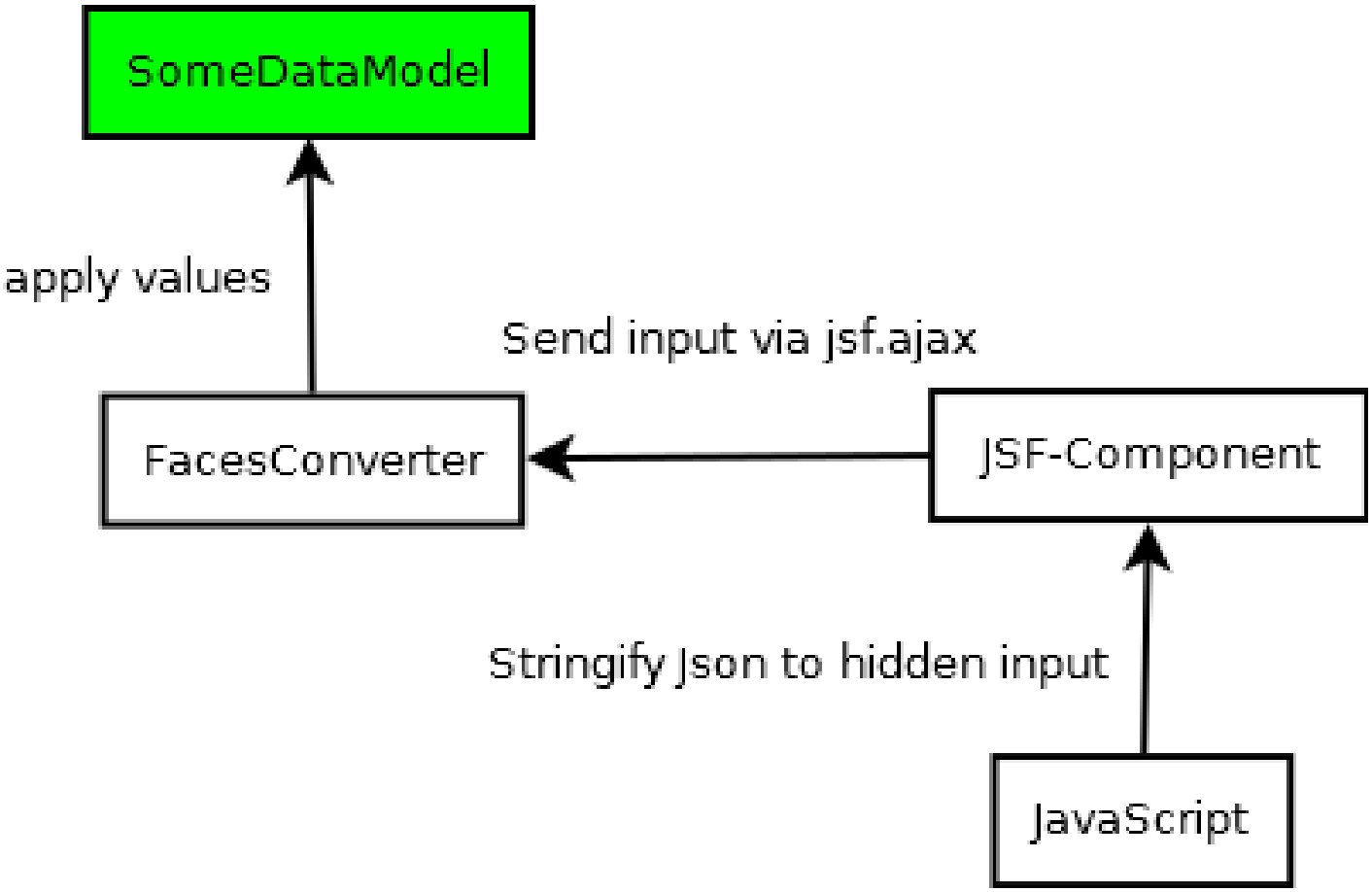
- JSON-Java-Converter

```
@FacesConverter(value =
"de.openknowledge.hc2014.OrderConverter")
public class TileOrderConverter implements Converter {

  @Override
  public Object getAsObject(..., String value) {
    List<String> ids = new ArrayList<>();

    Json.createReader(new StringReader(value)).readArray()
        .forEach(jsonValue ->
            ids.add(((JsonString)jsonValue).getString()));
    return ids;
  }
  ...
```

JSF ← JavaScript



JSF ← JavaScript

- Datenmodell

```
public class DashBoardDataModel {
  private List<String> tileOrder = new LinkedList<>();

  public void setTileOrder(List<String> tileOrder) {
    this.tileOrder = tileOrder;
  }
  ...
```

Zusammenfassung und Fazit

- Erstellung eigener Komponenten sehr einfach
  → JavaScript-Kenntnisse essentiell!
- Eigene Komponenten bringen enorme Flexibilität
  → initialer Aufwand höher
- JSON-Kommunikation mit JavaEE 7 Boardmitteln
- Einsatz von 3rd-Party Libraries abwägen
  → 0 8 15 vs. fancy web

Vielen Dank!

Sven Kölpin

open knowledge **GmbH**

open knowledge **GmbH**

- unabhängiges, inhabergeführtes Softwareentwicklungs- und Beratungsunternehmen aus Oldenburg
- Kernkompetenzen im Enterprise- und Mobile-Computing

- Sven Kölpin
- Enterprise Developer

facebook.com/openknowledge

twitter.com/_openknowledge