RIA with the Ajax Framework ZK

Processwide AG
Daniel Seiler

Agenda

processwide

9:10	Introduction
9:30	ZK part 1 – Introduction
10:10	20 Minutes break
10:30	ZK part 2 – Basics (Building an application)
12:00	60 Minutes lunch break
13:00	Exercise 1
13:30	ZK part 3 – Advanced (Building a component)
15:00	20 Minutes break
15:20	Exercise 2
16:15	ZK and the others
16:45	Wrap up

Introduction

processwide

- Goals
- What is the problem we try to solve?
- The big picture
- The right technology for the right job
- Why Ajax
- Different levels of Ajax
- Ajax tools

Goals

processwide

Infect you with the ZK virus

You are able to explain the position of ZK in the current RIA Landscape

You are able to build simple applications from scratch with ZK

What is the problem we try to solve?

processwide

To build rich, interactive, fast and scalable, distributed business applications ...

... we need a framework and technology that ...

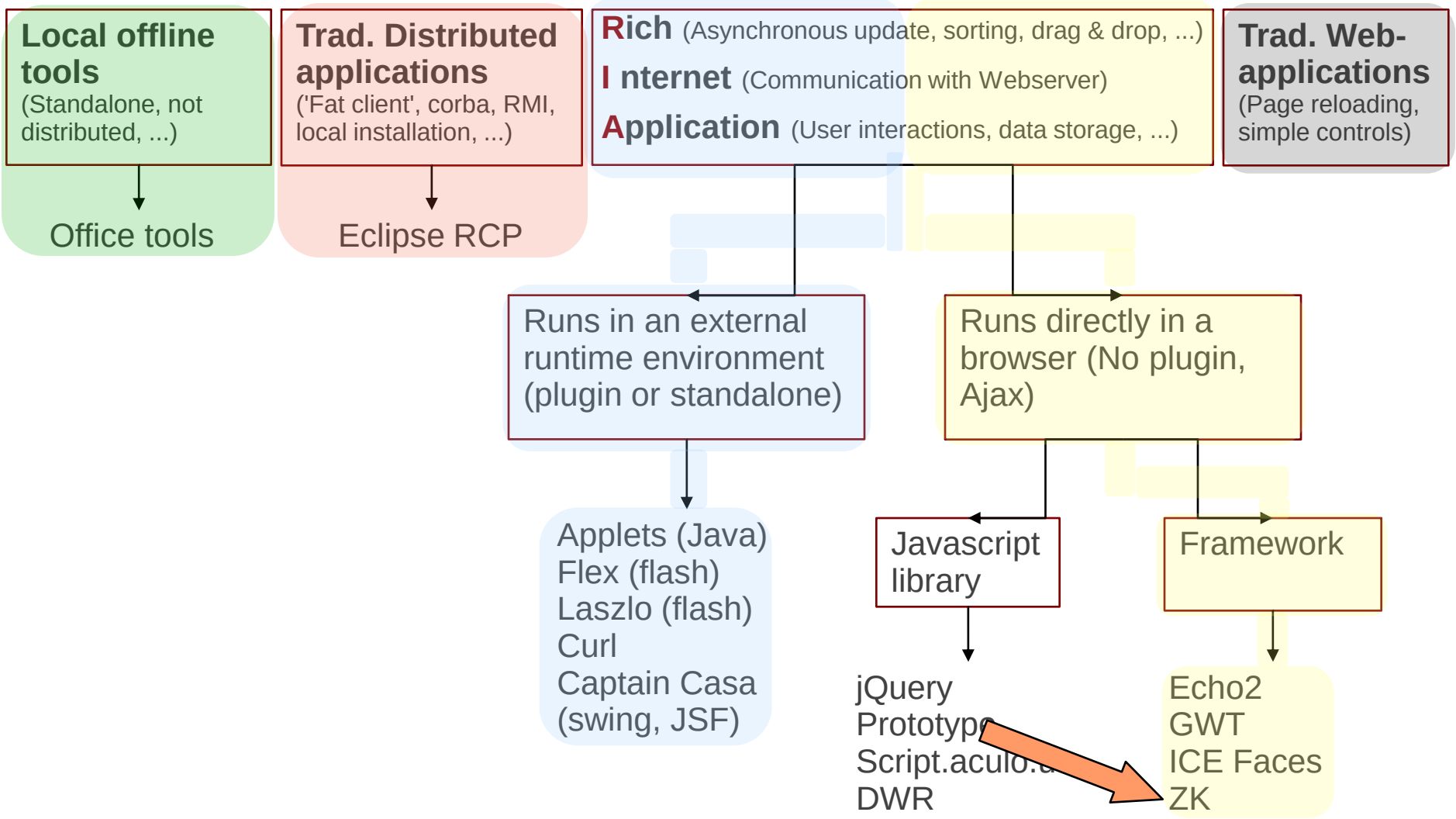
... maximizes our productivity by abstracting and hiding much of the complexity

... provides a rich set of prebuilt components and features

... is easy to extend

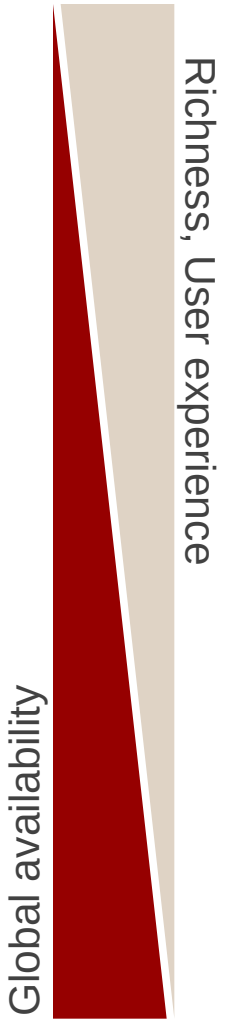
The big picture

processwide



The right technology for the right job

processwide



Local offline tools

+ Rich UI
- Local, no central access
- Application distribution (e.g. Java Webstart etc...)

Trad. distributed applications (Fat)

+ Rich UI
+ Proven development model, tool support
- Application distribution (e.g. Java Webstart etc...)
- Complex to build distributed applications

RIA External RTE

+ Rich UI
+ Easy application distribution
- Needs installation of RTE
- Communication with server not fully out of the box

RIA Ajax

+ Quite rich UI
+ Distributed by nature
- Slower user experience due to slow Javascript
- Needs lot of resources on the server

Trad. web-apps

+ Everywhere available through standard browser
- Poor UI
- Complex technology stack (HTML, CSS, Javascript, ...)

Why Ajax (asynchronous JavaScript and XML) ?

processwide

Basic idea:

breaking up the concept of the webpage to allow dynamic updates of certain areas within one page to increase the overall user experience

Typical technology stack:

- ***XHTML and CSS for presentation***
- ***the Document Object Model for dynamic display of and interaction with data***
- ***XML and XSLT for the interchange and manipulation of data, respectively***
- ***the XMLHttpRequest object for asynchronous communication***
- ***JavaScript to bring these technologies together***

Disadvantages:

- ***Traditional webconcepts like back button and bookmarks need to be rethought***
- ***Indexing through search engines is not guaranteed***
- ***Depends on Javascript switched on in the browser***
- ***Javascript is quite slow. There is hope: (http://www.google.com/chrome)***

Different levels of Ajax

processwide

1. **Snippet:** At this level, you add bits of JavaScript to an existing application to achieve minor client-side behavior without rearchitecting the application. Basic client-side validation of user input fields is an example of snippet-level Ajax.

(-) low level javascript coding, browser incompatibilities, only small improvements possible

2. **Widget:** At this level, basic browser UI controls are augmented with Ajax-enabled controls that incorporate more interactive features. Examples of Ajax-enabled controls include menus, pop-up panels, trees, etc. Again, widget-level development can be achieved without significant re-architecting of the application, but interactive features are confined on a widget-by-widget basis.

(-) Inter widget communication

3. **Framework:** At this level, a JavaScript framework provides features beyond a set of widgets for developing the UI in a client-centric manner. The framework provides basic infrastructure that supports greater interaction between widgets. For example, the framework might provide an event API where widgets can register to receive information from other widgets, and react to changing states within the UI. This level of Ajax requires a significant rearchitecting of the application, essentially a complete rewrite.

(-) Coding needed for integration of server side logic, rearchitecting of existing application

4. **Enhanced Framework:** At this level, the client-side framework is augmented with a server-side framework to deliver an end-to-end solution. This is the first level that addresses server-side requirements, but leads to fragmentation of the application where the data model is server-resident, but the UI business logic is client-resident. It suffers from the same re-architecting issues that the framework level does, but does not leave server-side integration entirely as an exercise for the developer.

Ajax tools

processwide

Snippet
jQuery, prototype, script.aculo.us, DWR

Widget
Yahoo UI Library, jbossrichfaces former Ajax4JSF, Dojo Toolkit

Framework
Backbase, ZK, Echo2, GWT, ICE Faces

ZK Part 1 - Introduction

processwide

- What is ZK?
- Server centric vs. Client centric
- Selling points
- ZK's Ajax solution
- Architecture
- Components

What is ZK?

processwide

Web: http://www.zkoss.org

Developed by: **Potix Corporation**

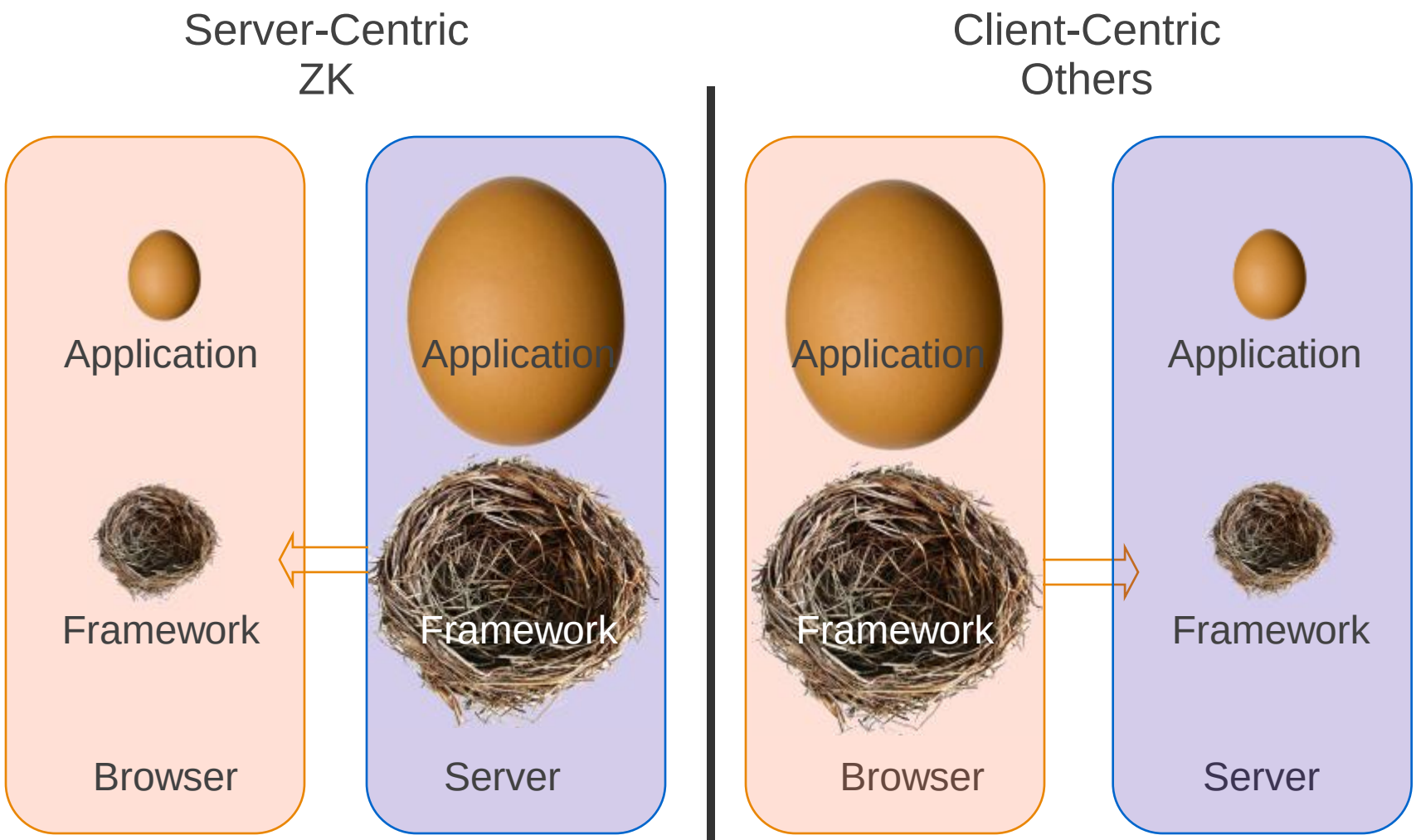
11F-2, No.87, Zhengzhou Road
Taipei, 103
Taiwan, Republic of China
+886-2-2552-1002

According to wikipedia: http://en.wikipedia.org/wiki/ZK_Framework

ZK is an **open-source Ajax Web application framework**, written in Java, that enables creation of **rich graphical user interfaces** for Web applications with **no JavaScript** and little programming knowledge.

ZK takes the so called **server-centric approach** that the content synchronization of components and the event pipelining between clients and servers are automatically done by the engine and **Ajax plumbing codes are completely transparent** to web application developers.

Server centric vs. Client centric



Why server centric?

processwide

- Avoid browser incompatibility
- Easy to use!
- Robustness & Security
- Lower maintenance costs
- Incremental extensibility

Selling points

processwide

- *Open Source: ZK is the leading open source Ajax + Mobile framework. ZK developer community is extremely active with 20+ translations, 100+ articles/blogs, and 100,000+ lines of codes, 700,000+ downloads, from 190+ countries.*
- *Rich User Experience: 170+ off-the-shelf state-of-art XUL/HTML-complaint Ajax components. Numerous third party widgets: JFreeChart, JasperReports, Google Maps, FCKeditor, Timeline, Timeplot, ExtJS, Dojo and so on*
- *Standards-based: ZK is a standard-compliant solution.*
- *Extensibility and Customizability: ZK is fully customizable and extensible with modular and plug-and-play architecture.*
- *Mobile Access: ZK extends the reach of enterprise Internet applications to 1+ billion mobile devices at minimal cost. ZK supports Java Mobile, Android, and various mobile browsers.*
- *Security: ZK is designed from the ground up to be secure.*

Sun is using it for their virtual platform management suite:

http://www.openxvm.org/xvmsui.html

ZK's Ajax solution

processwide

Many Documents

Simplify

One ZUL

```
<window title="Context Menu and Right
    Click" border="normal"
    width="360px">

<label value="Move Mouse Over Me!"
    tooltip="edit"/>
</window>
```

ZK Ajax Framework

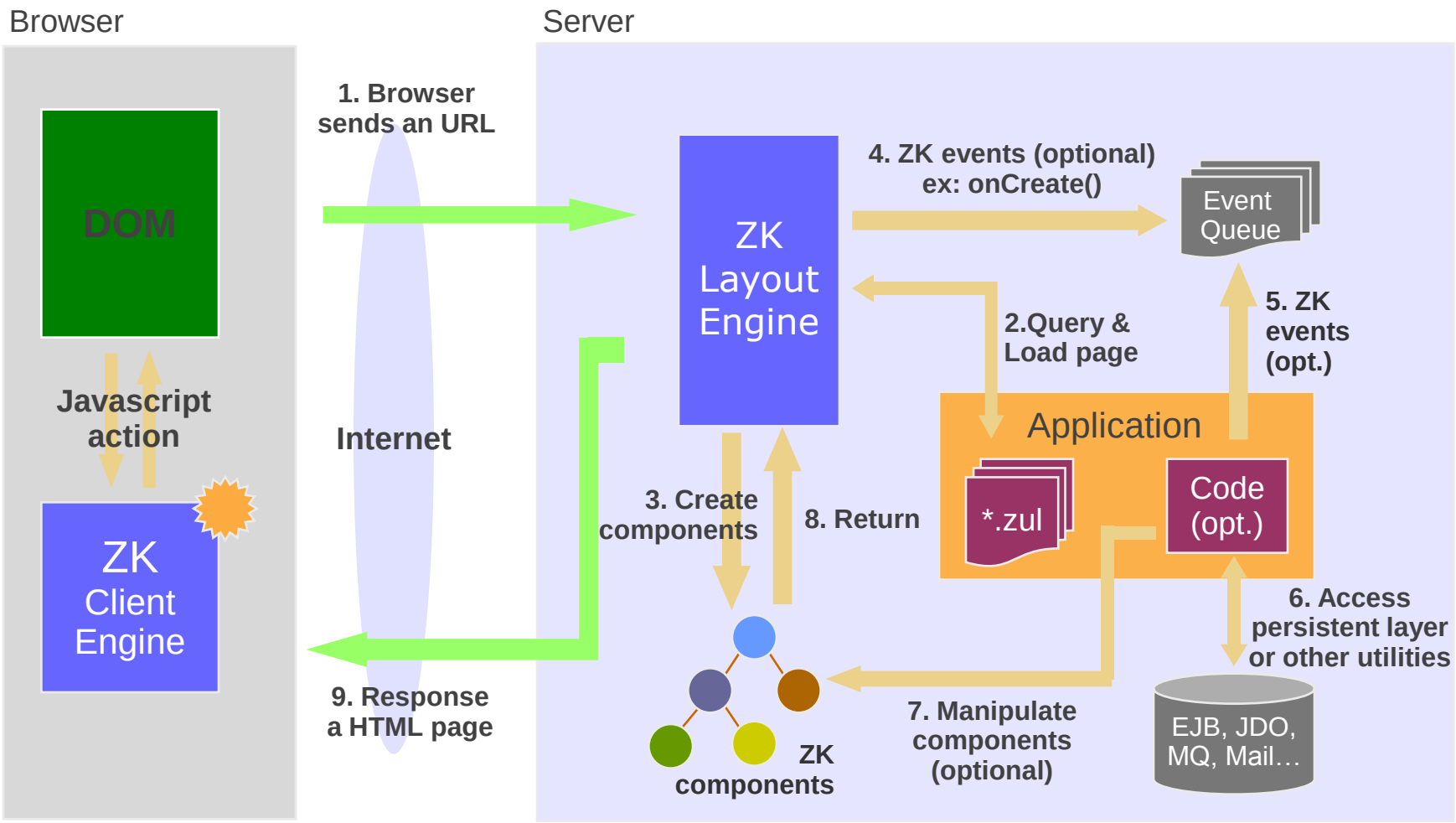
IE6 | Firefox

One Browser

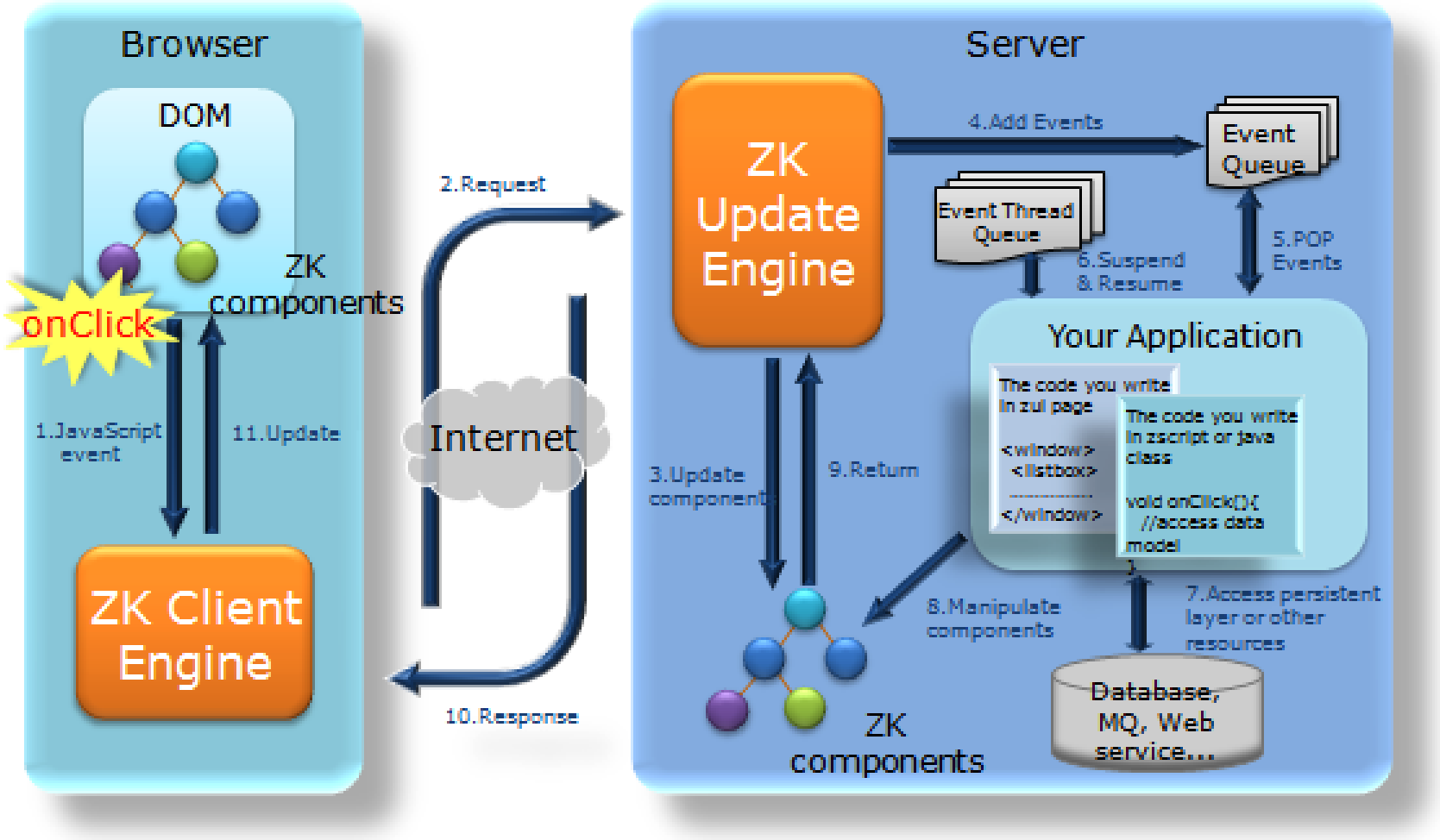
Opera | ... | Firefox | IE7

Many Browsers

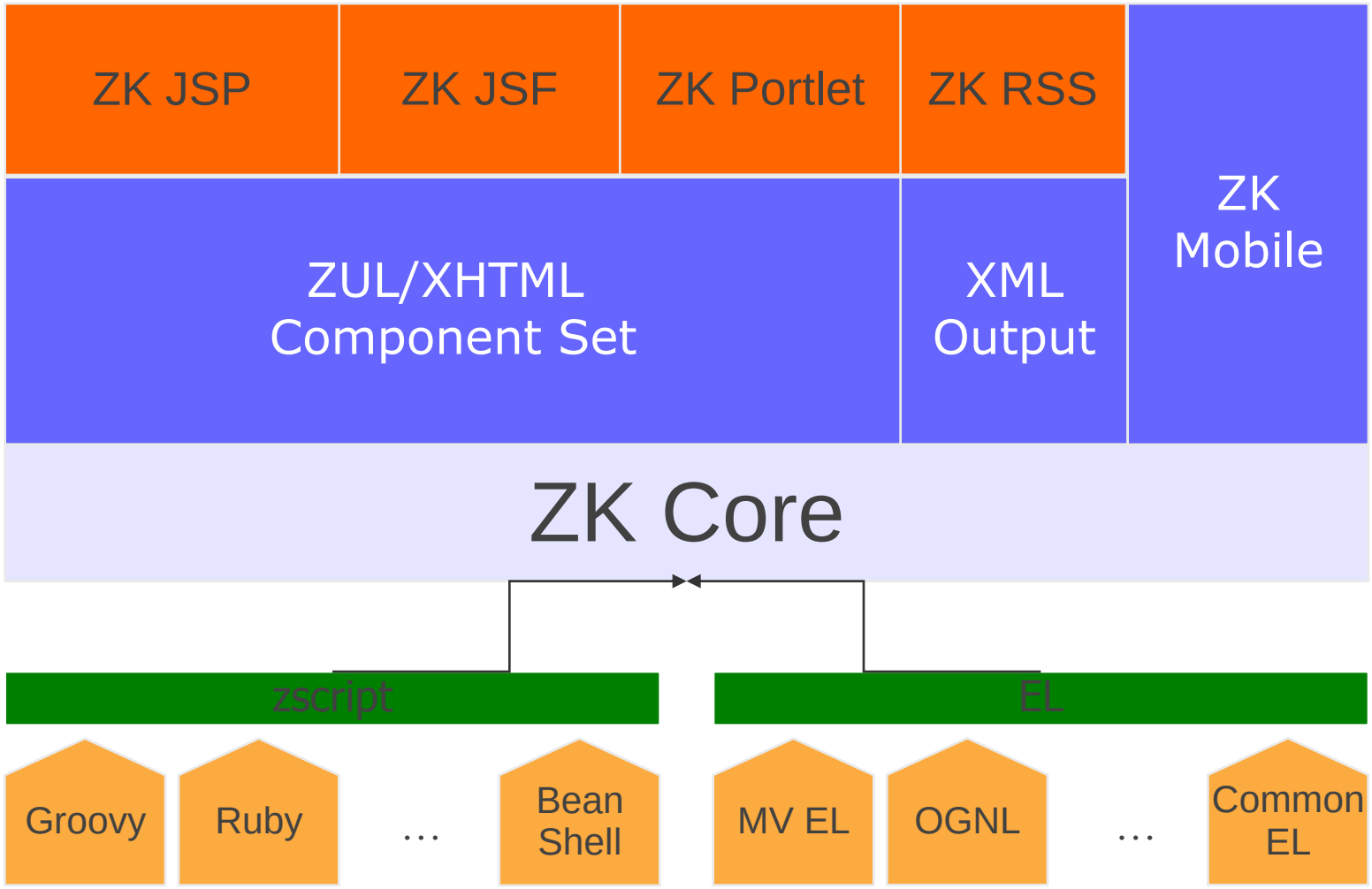
Generating an HTML page



ZK – Doing Ajax



ZK – Overall picture



ZK Components

processwide

ZK Components: http://www.zkoss.org/zkdemo/userguide/

- XUL (Mozilla) compliant
- 170+ off-the-shelf components
  - grid, tabbox, listbox, combobox, chart, splitter, slider, groupbox…
  - Context Menu
  - Drag and Drop
  - Live data (listbox and grid)
  - Auto-completion (combobox)
  - Customizable sorting (listbox and grid)

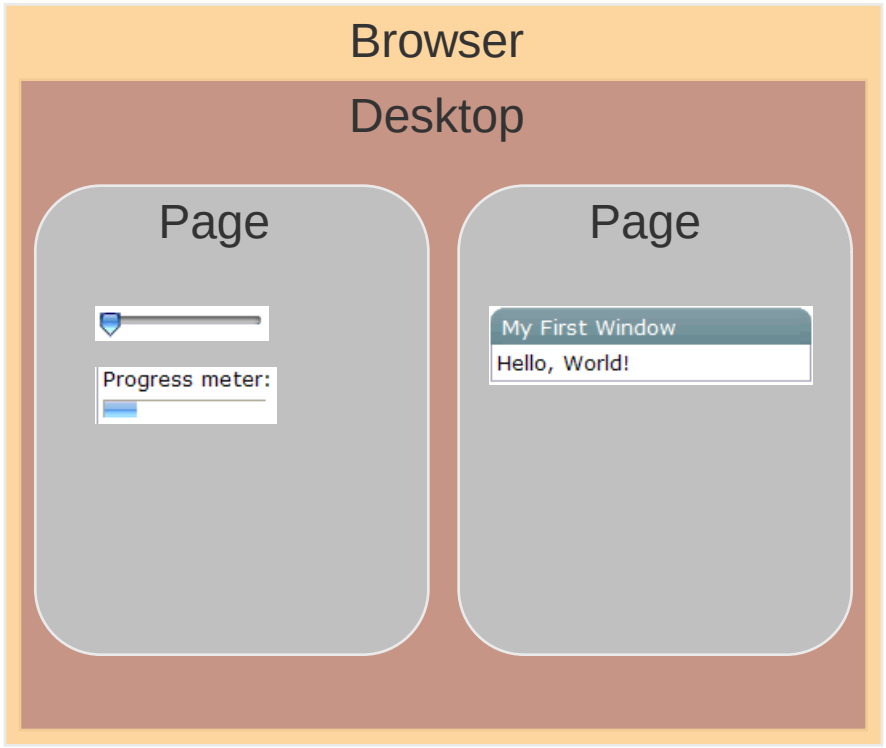
ZK Part 2 - Basics

processwide

- Desktop, page and components
- ZUL and zscript
  - If, unless, forEach
- Life Cycle of Page loading
- ID Space
- First application step by step
  - MVC pattern applied
  - Setup dev environment in VM

Desktop, Page and Components

- A desktop is a browser window. It has at least one page.
- A page is a collection of components, which are displayed in a certain portion of the browser.
- A **component** is an UI object, e.g., window, button…



ZUL and zscript

processwide

```
<window title="fileupload demo" border="normal">
  <button label="Upload">
    <attribute name="onClick">
      {
        Object media = Fileupload.get();
        if (media instanceof org.zkoss.image.Image) {
          Image image = new Image();
          image.setContent(media);
          image.setParent(pics);
        } else if (media != null)
          Messagebox.show("Not an image:" + media, "Error",
Messagebox.OK, Messagebox.ERROR);
      }
    </attribute>
  </button>
  <vbox id="pics" />
</window>
```

Try it out on the ZK components Demo page ...

ZUL and zscript

processwide

What is the result? Why?

```
<window border="normal">
  <label id="l" value="hi label"/>
  <zscript>
    l.value = "hi zscript";
  </zscript>
  ${l.value}
</window>
```

Result: hi zscript hi zscript

What is the result? Why?

```
<window border="normal">
  <label value="${l.value}"/>
  <label id="l" value="hi label"/>
</window>
```

Result: hi label

if, unless and forEach

```
<window>
  <zscript>
    newBtn = true;
    contacts = new String[] {"Monday", "Tuesday","Wednesday"};
  </zscript>
  <button label="New" if="${newBtn}">
    <attribute name="onClick">
      alert("I am a new Button!");
    </attribute>
  </button>
  <button label="Old" unless="${newBtn}" />
  <separator />
  <listbox width="100px">
    <listitem label="${each}" forEach="${contacts}" />
  </listbox>
</window>
```

Try it out on the ZK components Demo page ...

The life cycle of loading pages

processwide

1) The page initial phase

2) The component creation phase

3) The event processing phase

4) The rendering phase

The page initial phase

processwide

1. The page initial phase
2. The component creation phase
3. The event processing phase
4. The rendering phase

init processing instruction gets processes (if defined) by calling the doInit method of a class or by calling a zscript.
In this phase the page is not yet attached to the desktop!

```
<?xml version="1.0" encoding="UTF-8"?>
<?page id="userGuide" title="ZK Live Demo"?>
<?init class="MyInit"?> or <?init zscript="myinit.zs"?>
<zk>
  <window id="win" border="normal" width="200px"
   sizable="true">
    <caption image="/img/inet.png" label="Hi there!"/>
    <checkbox label="Hello, Wolrd!"/>
    <button label="center"
          onCreate="win.setBtn(self)"/>
  </window>
</zk>
```

The component creation phase

processwide

1. The page initial phase
2. The component creation phase
3. The event processing phase
4. The rendering phase

In this phase, ZK loader interprets a ZUML page. It creates and initializes components accordingly. It takes several steps as described on the next slide...

```
<?xml version="1.0" encoding="UTF-8"?>
<?page id="userGuide" title="ZK Live Demo"?>
<?init class="MyInit"?> or <?init zscript="myinit.zs"?>
<zk>
  <window id="win" border="normal" width="200px"
   sizable="true">
    <caption image="/img/inet.png" label="Hi there!"/>
    <checkbox label="Hello, Wolrd!"/>
    <button label="center"
          onCreate="win.setBtn(self)"/>
  </window>
</zk>
```

The component creation phase

processwide

1) For each element, it examines the **if and unless attribute** to decide whether it is effective. If not, the element and all of its child elements are ignored.
2) If the **forEach attribute** is specified with a collection of items, ZK repeats the following steps for each item in the collection.
3) **Creates the component** based on the element name, or by use of the class specified in the use attribute, if any.
4) **Initializes the members** one-by-one based on the order that attributes are specified in the ZUML page.
5) **Interprets the nested elements** and repeat the whole procedure.
6) **Invokes the afterCompose method** if the component implements the org.zkoss.zk.ui.ext.AfterCompose interface.
7) After all children are created, the **onCreate event is sent** to this component, such that application could initialize the content of some elements later. Notice that the onCreate events are posted for child components first.

Note: a developer can perform some application-specific initialization by listening to the onCreate event or implementing AfterCompose. AfterCompose is called in the Component Creation Phase, while the onCreate event is handled by an event listener.

An event listener is free to suspend and resume the execution (such as creating modal dialogs), while AfterCompose is a bit faster since no need to fork another thread

The event processing phase

processwide

1. The page initial phase
2. The component creation phase
3. The event processing phase
4. The rendering phase

In this phase, ZK invokes each listener for each event queued for this desktop one-by-one. An independent thread is started to invoke each listener, so it could be suspended without affecting the processing of other events. During the processing, an event listener might fire other events.

```
<?xml version="1.0" encoding="UTF-8"?>
<?page id="userGuide" title="ZK Live Demo"?>
<?init class="MyInit"?> or <?init zscript="myinit.zs"?>
<zk>
  <window id="win" border="normal" width="200px"
   sizable="true">
    <caption image="/img/inet.png" label="Hi there!"/>
    <checkbox label="Hello, Wolrd!"/>
    <button label="center"
          onCreate="win.setBtn(self)"/>
  </window>
</zk>
```

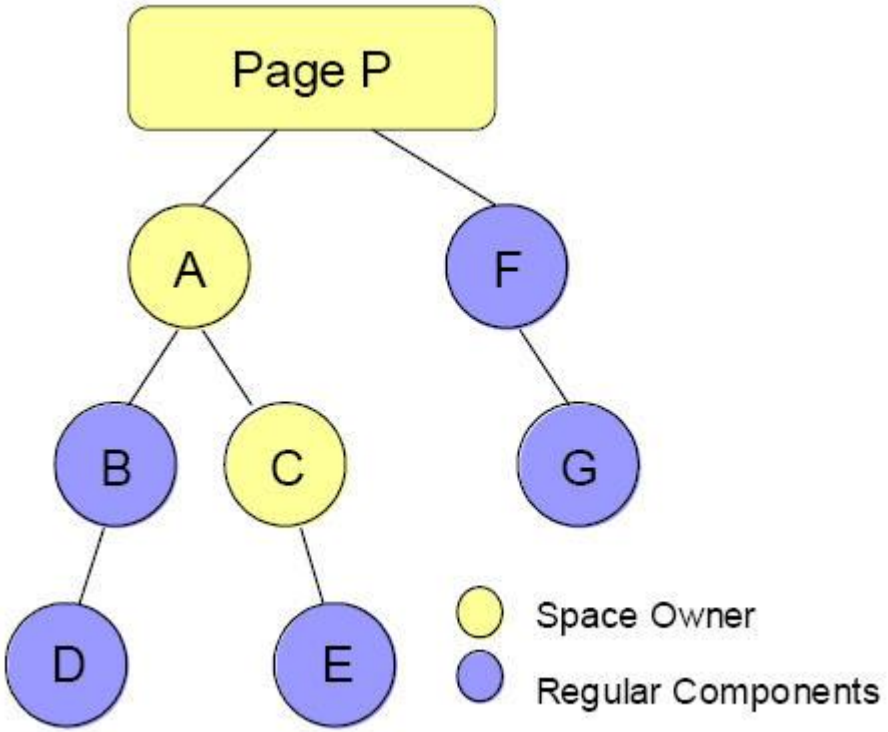
The rendering phase

1. The page initial phase
2. The component creation phase
3. The event processing phase
4. The rendering phase

After all events are processed, ZK renders these components into a regular HTML page and sends this page to the browser.
To render a component, the redraw method is called. The implementation of a component shall not alter any content of the component in this method.

```
<?xml version="1.0" encoding="UTF-8"?>
<?page id="userGuide" title="ZK Live Demo"?>
<?init cla
<zk>
  <wind
   sizab
    <ca
    <ch
    <bu

  </window>
</zk>
```

ID Space



Definition:
An ID space is a subset of components of a desktop. The uniqueness of a components id is guaranteed only in the scope of an ID space.

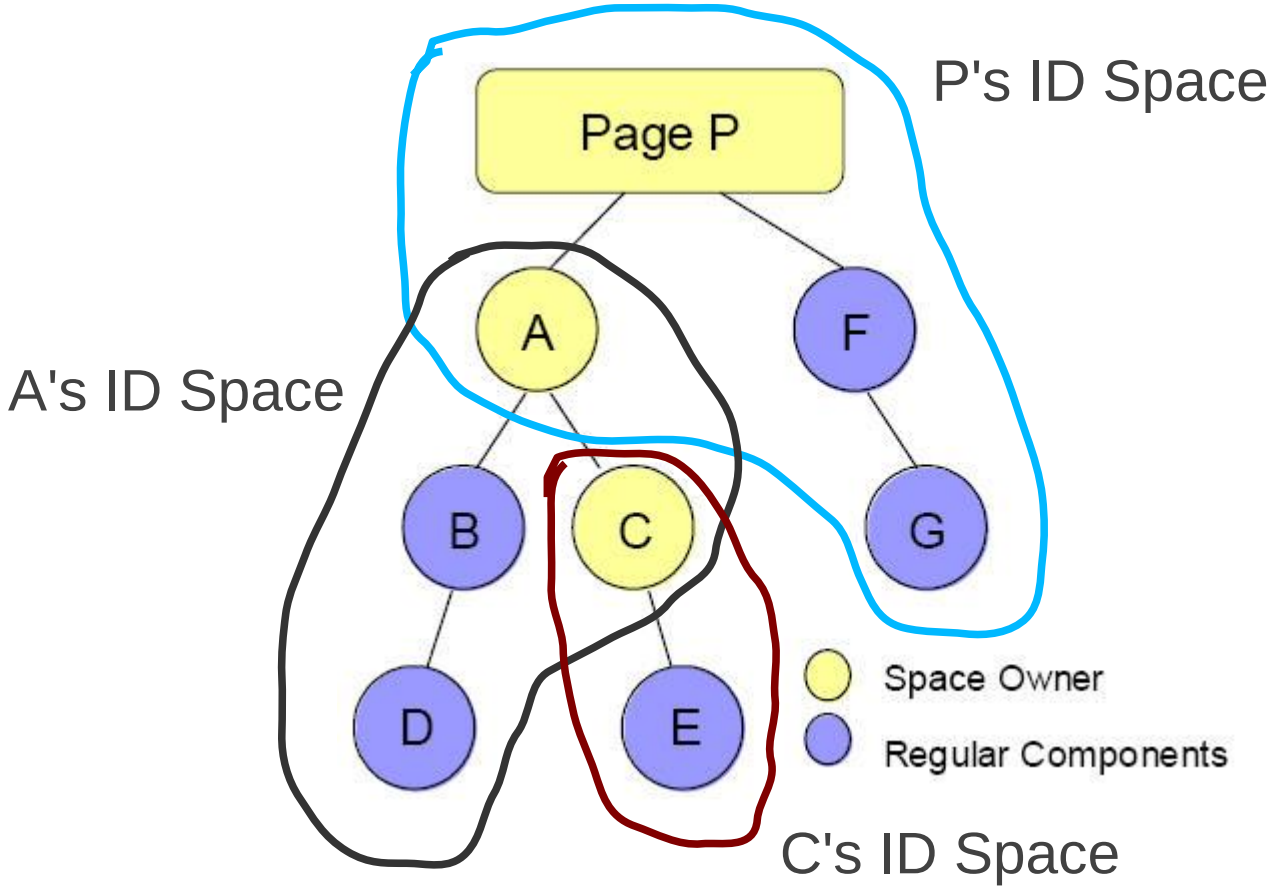
Implementation:
Any component that implements the org.zkoss.zk.ui.IdSpace interface

org.zkoss.zk.ui.Page
org.zkoss.zul.Window

The topmost component of an ID space is called the owner of the ID space, which could be retrieved by the *getSpaceOwner* method of the Component interface.

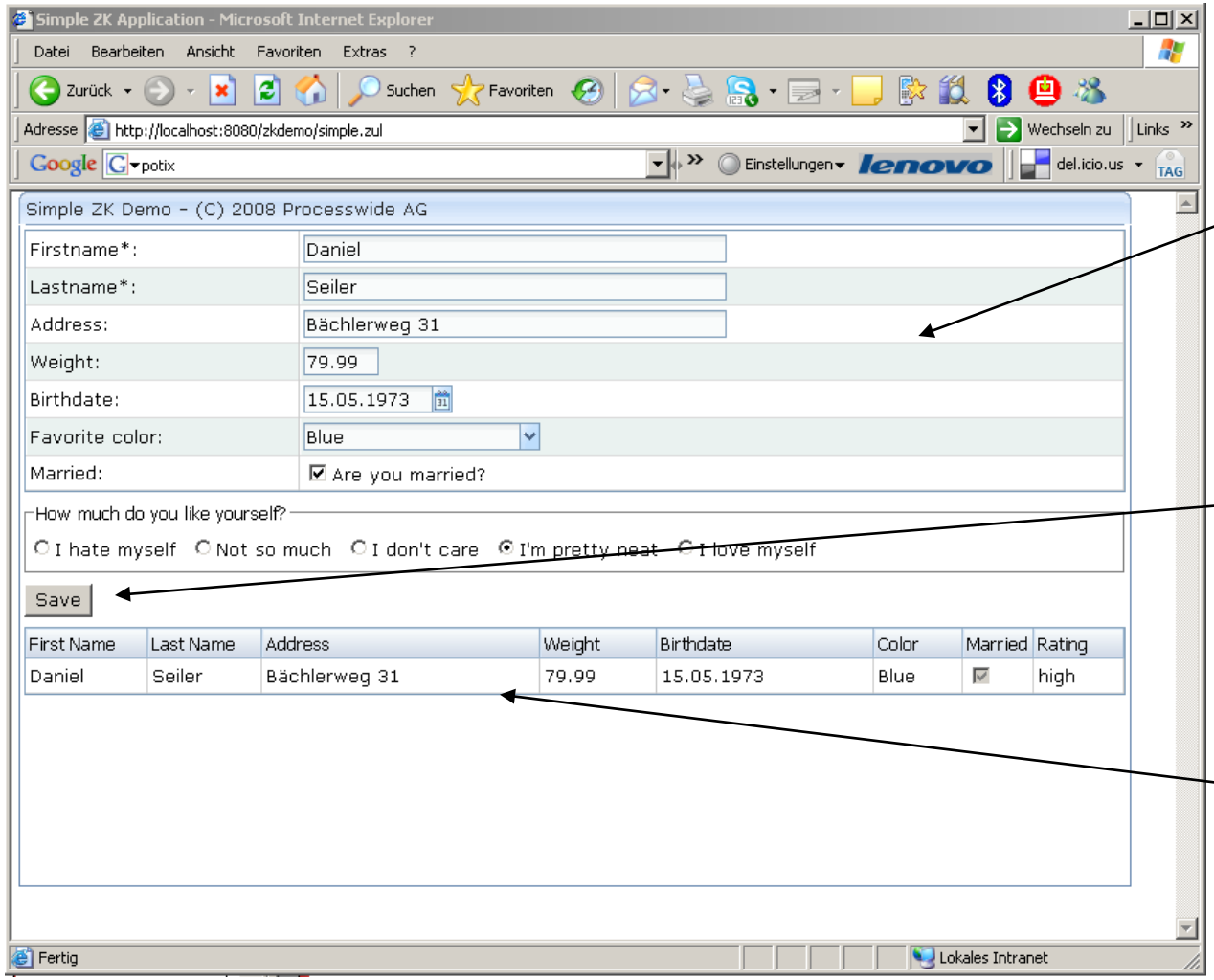
```
Component E = Path.getComponent("A/C/E");
```

ID Space - Fellows



Component D = A.getFellow(„D“);

First application step by step

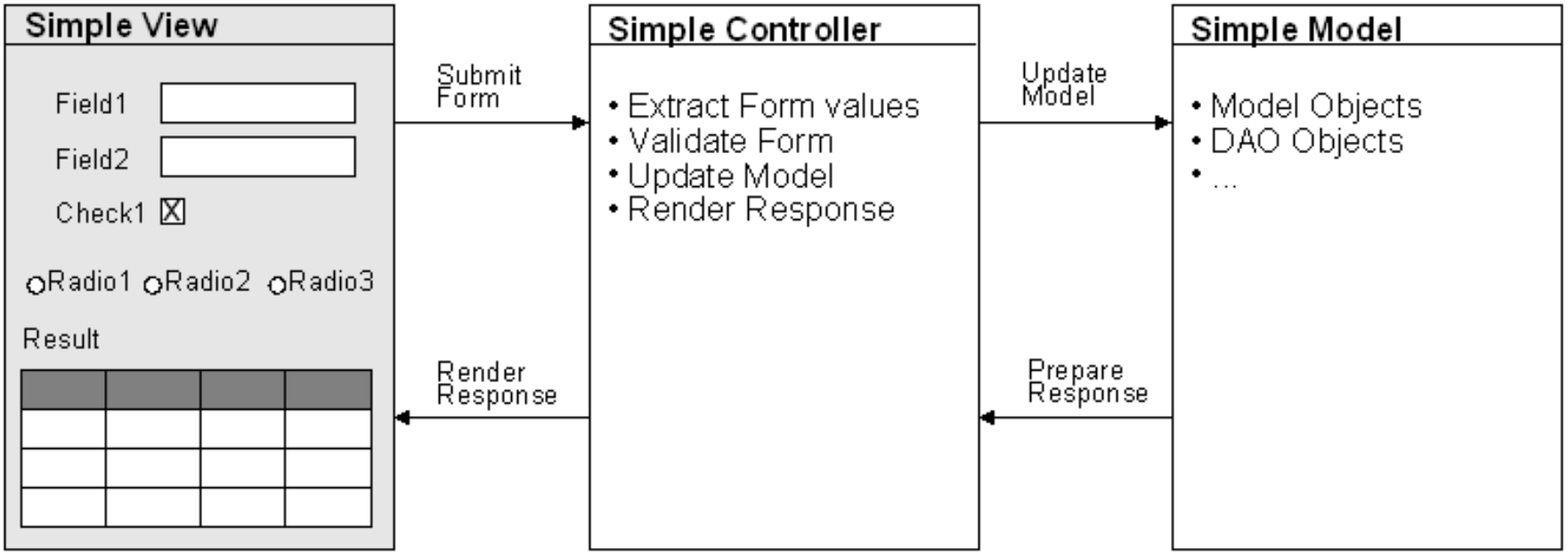


1) User enters his personal data

2) Data gets sent to the server and stored in a DB

3) Table gets updated with the new user

First application - architecture



First application - simple.zul

```
<?page id="simplePage" title="Simple ZK Application"?>
<zk>
<window id="simpleWindow" use="com.processwide.demo.zk.window.SimpleWindow"
      title="Simple ZK Demo" border="normal" width="800px" height="500px">
      <caption label="${simpleWindow.myCaption}" />
      <grid>
            <rows>
                  <row>Firstname*:<textbox id="firstname" width="300px" /></row>
                  <row>Lastname*:<textbox id="lastname" width="300px" /></row>
                  ...
                  <row>Birthdate:<datebox id="birthdate" format="dd.MM.yyyy" /></row>
                  <row>Favorite color:
                        <combobox id="color">
                              <comboitem label="Red" description="Red means Red ;-)" />
                              <comboitem label="Blue" />
                        </combobox>
                  </row>
                  <row>Married:<checkbox id="married" label="Are you married?" /></row>
            </rows>
      </grid>
      ...
      <button label="Save" onClick="simpleWindow.addPerson()" />
      <separator />
      <grid id="personsGrid">
      ...
      </grid>
</window>
</zk>
```

Declaration of external Java class as view handler

Expression language, default: common-el

Scripting code, default: BeanShell

Placeholder for dynamically updated result table

First application - SimpleWindow.java

processwide

```
public class SimpleWindow extends Window {
     public static SimpleDateFormat sdf = new SimpleDateFormat("dd.MM.yyyy");
     private Grid personsGrid;

     public void onCreate() {
          createPersonsGrid();
     }

     public String getMyCaption() {
          return "(C) 2008 by Processwide AG";
     }

     private void createPersonsGrid() {
          personsGrid = (Grid)Path.getComponent("/simpleWindow/personsGrid");
          // add all the persons
          List<Person> persons = PersonDAO.getInstance().getAllPersons();
          for (Iterator<Person> it = persons.iterator(); it.hasNext();) {
               Person person = it.next();
               addPersonRecord(person);
          }
     }
     ...
```

First application - SimpleWindow.java

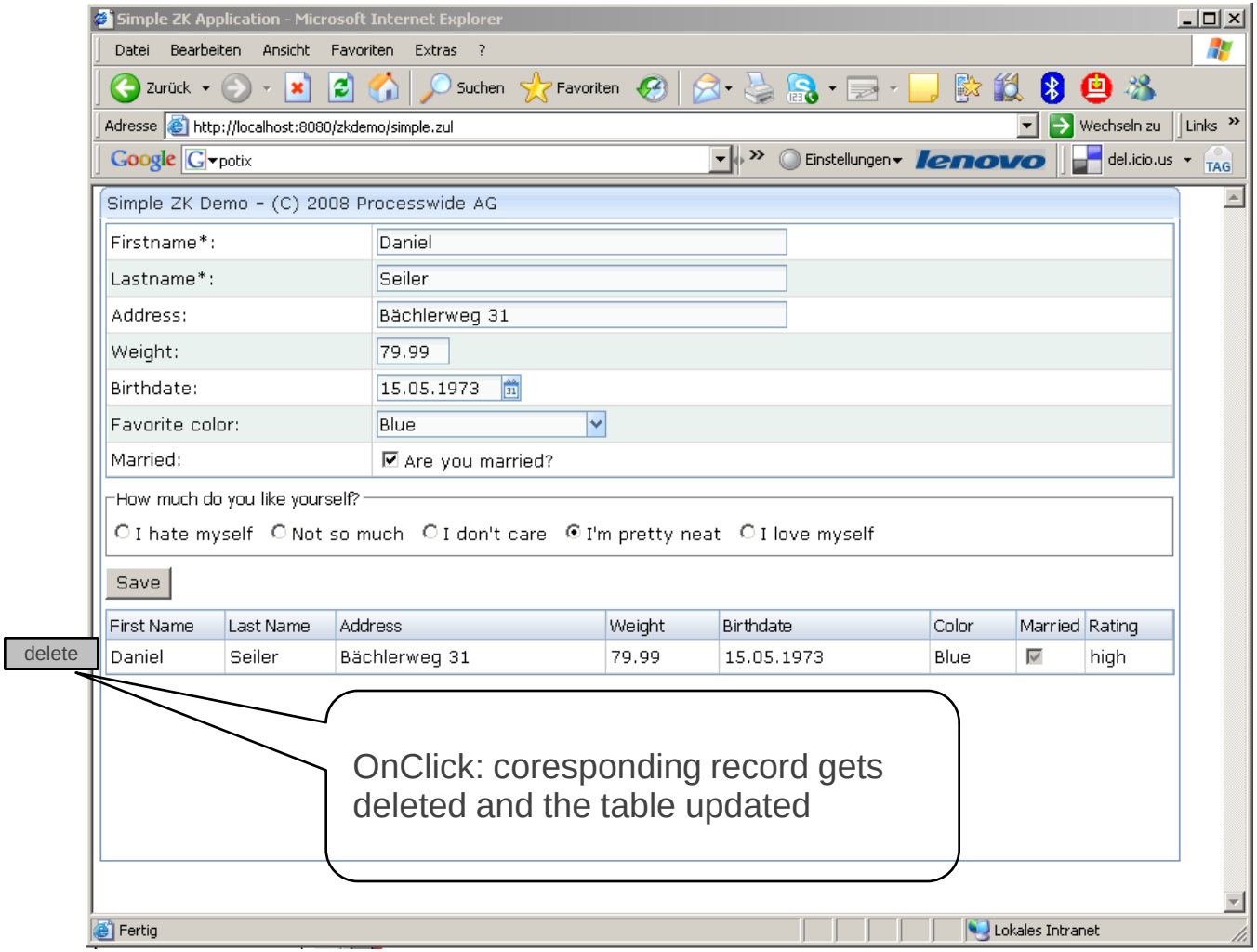
processwide

```
...
public void addPerson() {
	// extracting the person data
	Textbox firstName = (Textbox) Path.getComponent("/simpleWindow/firstname");
	Combobox color = (Combobox) Path.getComponent("/simpleWindow/color");
	Checkbox married = (Checkbox) Path.getComponent("/simpleWindow/married");
	...
	Person personBean = new Person();
	personBean.setFirstname(firstName.getValue());
	...
	personBean.setColor(color.getValue());
	personBean.setMarried(married.isChecked());
	...
	PersonDAO.getInstance().addPerson(personBean);

	addPersonRecord(personBean);
}

public void addPersonRecord(Person person) {
	Row row = new Row();
	Rows rows = personsGrid.getRows();
	rows.appendChild(row);
	row.appendChild(new Label(person.getFirstname()));
	row.appendChild(new Label(person.getColor()));
	...
}
}
```

First application – get it running

processwide

Exercise 1: Extend first application



Tasks:

1) Paging
Enhance the result table, so that after 5 records a proper paging appears.

2) Delete
For each record add a delete button with which the coresponding record can be removed

Exercise 1: Solution

processwide

1) Add the delete button to every record and register an event listener

```
public void addPersonRecord(final Person person) {
      final Row row = new Row();
      final Rows rows = personsGrid.getRows();
      rows.appendChild(row);
      // create the delete button
      Button deleteBtn = new Button("delete");
      deleteBtn.addEventListener("onClick", new EventListener() {
            public void onEvent(final Event arg0) throws Exception {
                  // delete the person from the database
                  PersonDAO.getInstance().deletePerson(person.getId());
                  // update the table
                  rows.removeChild(row);
                  }
            });
      row.appendChild(deleteBtn);
      row.appendChild(new Label(person.getFirstname()));
      ...
```

2) Add an empty column for the delete button to the personsGrid and add paging support

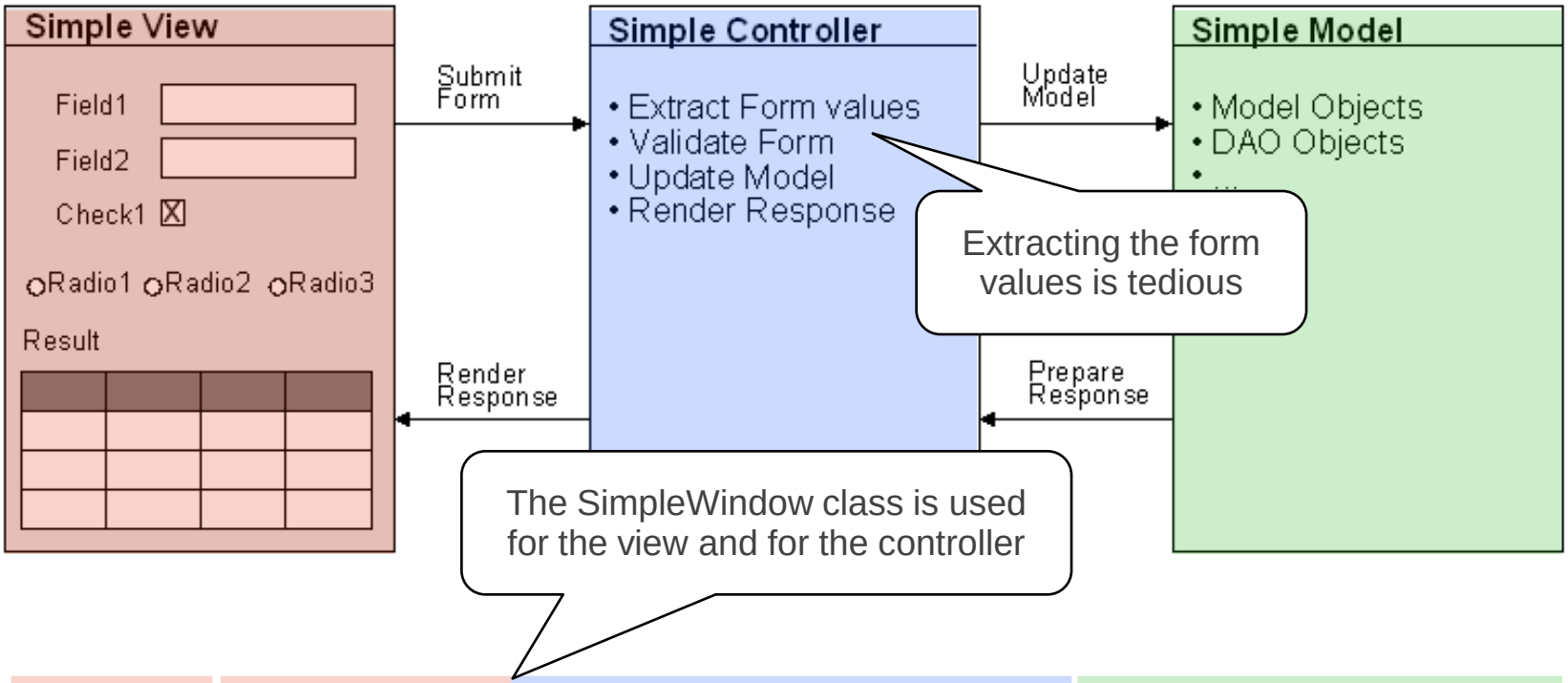
```
<grid id="personsGrid" mold="paging" pageSize="5">
<columns sizable="true">
            <column label="" />
            <column label="First Name" />
...
```

ZK Part 3 - Advanced

processwide

- Improving Simple App
  - MVC Applied
  - Databinding
- Components
  - Macro components
  - Components by extending existing components
  - Component from scratch
- We build a DoubleCombo component
  - Pure Ajax Solution
  - ZK Solution

Improved application – what's wrong?



As is:	simple.zul	SimpleWindow.java	Person.java, PersonDAO.java
To be:	simple2.zul, SimpleWindow2.java	SimpleComposer.java	Person.java, PersonDAO.java

Improved application - simple2.zul

```
<?init class="com.processwide.demo.zk.controller.SimpleWindowInit" ?>
<?page id="simplePage" title="Simple ZK Application"?>
<zk>
<window id="simpleWindow2" use="com.processwide.demo.zk.window.SimpleWindow2"
      apply="com.processwide.demo.zk.controller.SimpleComposer"
      title="Improved Simple ZK Demo" border="normal" width="800px" height="500px">
      <caption label="${simpleWindow2.myCaption}" />
      <grid>
            <rows>
                  <row>Firstname*:<textbox id="firstname" value="@{person.firstname}" width="300px" /></row>
                  <row>Lastname*:<textbox id="lastname" value="@{person.lastname}" width="300px" /></row>
                  ...
                  <row>Birthdate:<datebox id="birthdate" value="@{person.birthdate}" format="dd.MM.yyyy" /></row>
                  <row>Favorite color:
                        <combobox id="color" value="@{person.color}">
                            <comboitem label="Red" description="Red means Red ;-)" />
                            <comboitem label="Blue" />
                        </combobox>
                  </row>
                  <row>Married:<checkbox id="married" checked="@{person.married}" label="Are you married?" /></row>
            </rows>
      </grid>
      ...
      <button id="saveBtn" label="Save" onClick="simpleWindow.addPerson()" />
      ...
</window>
</zk>
```

Do some initialization for the data binding

Apply a proper controller class

Properties are bound to the 'person' variable

The event handler is defined implicitly in the conroller. The button needs an 'id'

Improved - SimpleWindow2.java

processwide

```
public class SimpleWindow2 extends Window {

	public static SimpleDateFormat sdf = new SimpleDateFormat("dd.MM.yyyy");

	private Grid personsGrid;

	public String getMyCaption() {
		return "(C) 2008 by Processwide AG";
	}

	public void createPersonsGrid(List<Person> persons) {
		personsGrid = (Grid)Path.getComponent("/simpleWindow2/personsGrid");
		// add all the persons
		for (Iterator<Person> it = persons.iterator(); it.hasNext();) {
			Person person = it.next();
			addPersonRecord(person);
		}
	}

	public void addPersonRecord(Person person) {
		...
	}
}
```

- much leaner window class
- only view related functionality
- no access to the data layer

Improved - SimpleComposer.java

processwide

```
public class SimpleComposer extends GenericForwardComposer {

    Radiogroup ratingRadiogroup;

    public void onCreate$simpleWindow2(Event evt) {
        // query all the persons
        List<Person> persons = PersonDAO.getInstance().getAllPersons();
        ((SimpleWindow2)self).createPersonsGrid(persons);
    }

    public void onClick$saveBtn(Event evt) {
        Person person = (Person)page.getVariable("person");
        // for radiogroups the databinding is not supported
        person.setRating(ratingRadiogroup.getSelectedItem().getValue());
        // store the person in the db
        PersonDAO.getInstance().addPerson(person);
        // update the view
        ((SimpleWindow2)self).addPersonRecord(person);
    }
}
```

Improved - SimpleWindowInit.java

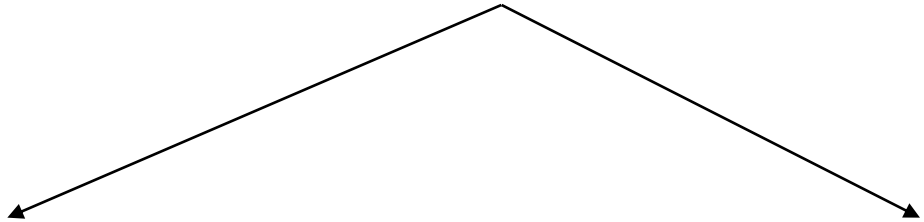
processwide

```
public class SimpleWindowInit extends
    org.zkoss.zkplus.databind.AnnotateDataBinderInit {

    //AnnotateDataBinder binder = new AnnotateDataBinder(simpleWindow);
    //binder.bindBean("person", person);
    //binder.loadAll();

     //override this
     public void doAfterCompose(Page page) {
          //prepare the person object
          Person person = new Person();

          //bind Person to id "person"
          page.setVariable("person", person);

          //remember to call the super
          super.doAfterCompose(page);
     }
}
```

ZK Custom Components

The ability to implement custom components easily and intuitively is one of the reasons for the productivity gain ZK brings compared to other frameworks

Create a custom component by ...



Extending existing Components

Creating class deriving from org.zkoss.zk.ui. AbstractComponent

Custom component by-macro

processwide

```
<?component name="myName" macroURI="/mypath/my.zul"
   [inline="true|false"] [class="myPackage.myClass"]
   [prop1="value1"] [prop2="value2"]...?>
```

Content of the macro-component: /header.zul

```
<hbox height="30px">
    This is a header with the title ${arg.myTitle}
</hbox>
```

Usage of the macro-component:

```
<?component name="header" macro-uri="/header.zul"
   myTitle="Components Demo Page" ?>
...
<header/>
...
```

Custom component by-class

processwide

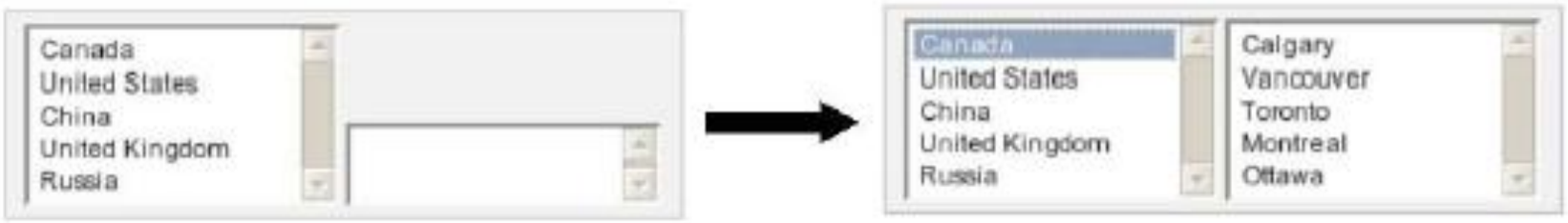
The by-class Format

```
<?component name="myName" [class="myPackage.myClass"]
   [extends="existentName"] [moldName="myMoldName"]
   [moldURI="/myMoldURI"] [prop1="value1"]
   [prop2="value2"]...?>
```

you can override properties of existent components by specifying extends="existentName". In other words, if extends is specified, the definition of the specified component is loaded as the default value and then override only properties that are specified in this directive.

--> DoubleCombo Example

Custom component: Double Combo



Client Solution

All the possible values are loaded at page loading to the client and changes are handled on the client through Javascript

\+ Once loaded it is fast

\- For large data sets the initial loading becomes inpractical

Server Solution (see http://www.ch.ch/karte/index.html?lang=de)

Each time the selection in the first combo changes, the whole page gets reloaded with the corresponding values filled into the second combo

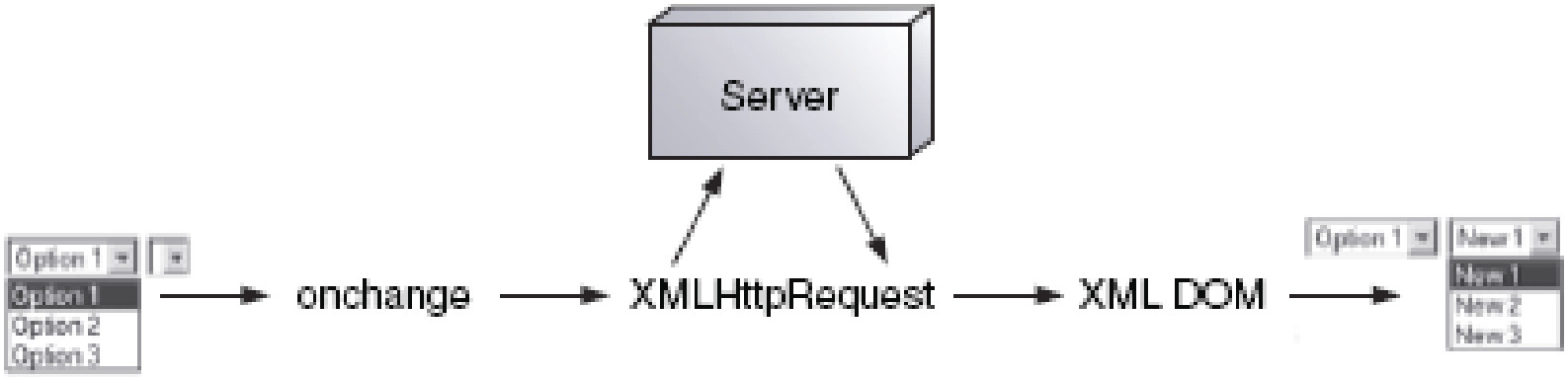
\+ Can handle large data sets

\- The user experience, response time is bad due to reloading of the whole page

Ajax Solution

Each time the selection changes in combo1 an Ajax request is made and the combo2 gets updated without reloading the whole page

Double Combo: Ajax Solution



+ can handle large data sets
+ user experience and response time is good due to partial update of the page
- complex to implement without a framework like ZK !!

On chapter 9 of the book 'Ajax in Action' (online available at: http://www.manning-source.com/books/crane/crane_ch09.pdf)
it takes around 16 pages to explain the implementation of this simple component!

Double Combo – ZK Implementation

processwide

Declaration:

```
<?component name="doubleCombo" extends="div"
    class="com.processwide.demo.zk.components.DoubleComboComponent"?>
```

Usage:

```
<doubleCombo
     orientation="horizontal"
     rows="5,5"
     dataSourceHandler="${componentsWindow.dataSourceHandler}"
     titleCombo1="Kantone in der Schweiz"
     titleCombo2="Gemeinden"
/>
```

Java:

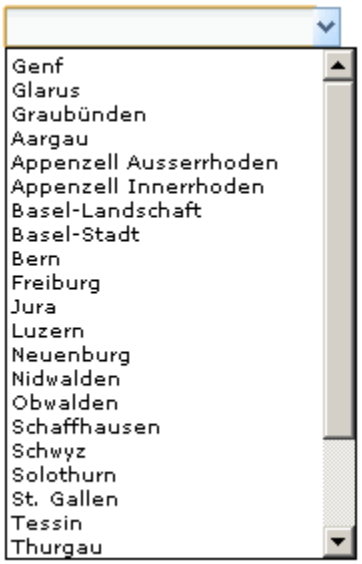
```
package com.processwide.demo.zk.components;
public class DoubleComboComponent extends Div {
...
```

Exercise 2: Autocompletion Box

processwide

Implement an autocompletion component that updates the available items dynamically according to the entered characters:

```
<?component name="autocompletionCombo" extends="combobox"
    class="com.processwide.demo.zk.components.AutocompletionCombobox"?>

<autocompletionCombo allItems="${componentsWindow.comboItems}"/>
```



ZK and the others – ICE Faces

processwide

Technology

ICEFaces is a JavaServer Faces implementation enriched by AJAX functionality: "The ICEfaces Component Suite provides a complete set of enhanced standard and custom JavaServer Faces (JSF) components".

Pros and Cons

(+) Good choice for adding Ajax to existing JSF applications

(+) Good choice for JSF experts that want to stick with JSF

(+) Rich and pretty set of UI components

(-) 3rd party components don't mix with ICEFaces components on the same page

(-) Being a JSF implementations, ICEFaces faces the inherent JSF limitations and complexity. (e.g. validation, complicated JSF lifecycle, configuration issues, JSP's or Facelets, ...)

Conclusion: Simplicity matters!

Compared to ZK, ICEfaces is more like a JSF components set than an Ajax framework.

ZK and the Google Web Toolkit (GWT)

processwide

Technology

GWT is a client side framework: develop in Java, compile to Javascript and run in the browser on the client. Each RPC call to the server needs to be explicitly programmed.

Pros and Cons

(+) Webapps can be developed like Swing applications without directly getting in contact with Javascript programming

(+) Open Source and developed by Google

(-) Limited set of Java classes available for the Java to Javascript compiler

(-) Server side part of the application needs to be integrated manualy

(-) Limited set of widgets and difficult to extend

(-) Difficult to debug because of client centric nature

Conclusion

GWT follows an interesting path in creating client centric Ajax applications by compiling Java classes to Javascript

If the requirements in terms of richness and responsiveness of the UI are very demanding, a client centric solution should be considered. Due to the limitations of GWT, I wonder if other client centric technologies like Laszlo, Curl or Flex should be preferred.

ZK and the others - Echo2

processwide

Technology

Echo2 is a server centric framework similar to ZK.

Pros and Cons

(+) Rich set of components

(-) Documentation

(-) Community support

(-) Integration of server side

Conclusion

From a technology point of view Echo2 and ZK are very similar. In terms of productivity, extensability, community support and documentation ZK has big advantages.

Wrap up

processwide

Among the AJAX Frameworks ZK is outstanding in ...

- Productivity
- Component library
- Building custom components
- Support and development activity of the provider
- Documentation
- Integration with other technologies (JSF, JSP, RSS, GMaps, Dojo, FCKEditor, Timeline, PayPal Service, Portlets, JFreeChart, JasperReports, ...

Links

processwide

ZK Main page: http://www.zkoss.org
ZK Article on Wikipedia: http://en.wikipedia.org/wiki/ZK_Framework
ZK vs. ICEFaces: http://www.zkoss.org/smalltalks/zkIcefaces/
ZK vs. GWT: http://www.zkoss.org/smalltalks/gwtZk/

Sun is using ZK: http://www.openxvm.org/xvmsui.html
Comparison of frameworks: http://www.theserverside.com/tt/articles/article.tss?l=ZKandAgile

Prototype: http://prototypejs.org/
Jquery:http://jquery.com/
Script.aculo.us: http://script.aculo.us/
DWR: http://directwebremoting.org/
Yahoo UI: http://developer.yahoo.com/yui/
Dojo: http://dojotoolkit.org/
JBoss Richfaces (former Ajax4JSF): http://www.jboss.org/jbossrichfaces/
Backbase: http://www.backbase.com/
Echo2/3: http://echo.nextapp.com/site/
GWT: http://code.google.com/webtoolkit/
ICEFaces: http://www.icefaces.org/