OpenLaszlo: From RIA to Ajax and Mobile

Geert Bevin

CEO, Uwyn bvba/sprl, http://www.uwyn.com

Developer, Terracotta Inc., http://terracotta.org

TS-6725

Goal of This Talk

Learn how to use OpenLaszlo for building rich Internet applications that execute in Flash, DHTML, and even on mobile

Agenda

Overview of OpenLaszlo

Introducing LZX

Deployment

Tips and Caveats

Agenda

Overview of OpenLaszlo

Introducing LZX

Deployment

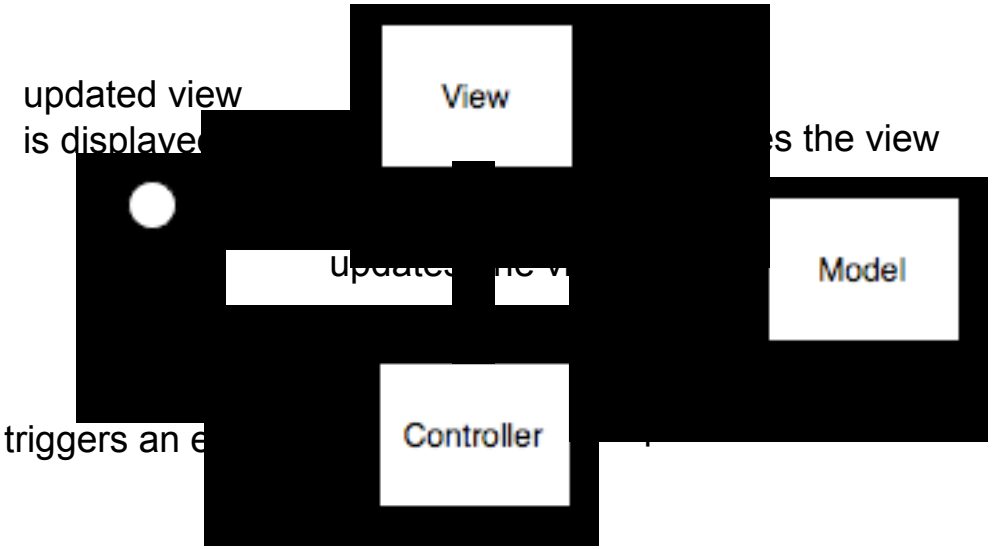
Tips and Caveats

What Is OpenLaszlo?

- Creation of zero-install web applications
- Open-source, stable, and well documented
- Integrates perfectly with the Java™ platform
- Runtime-independent development platform, supporting Flash, DHTML and Java Platform, Micro Edition (Java ME platform) as Project Orbit (https://orbit.dev.java.net)
- Powerful object-oriented, component-based language
- Declarative and functional programming styles
- Extends the RIA focus to multimedia capabilities

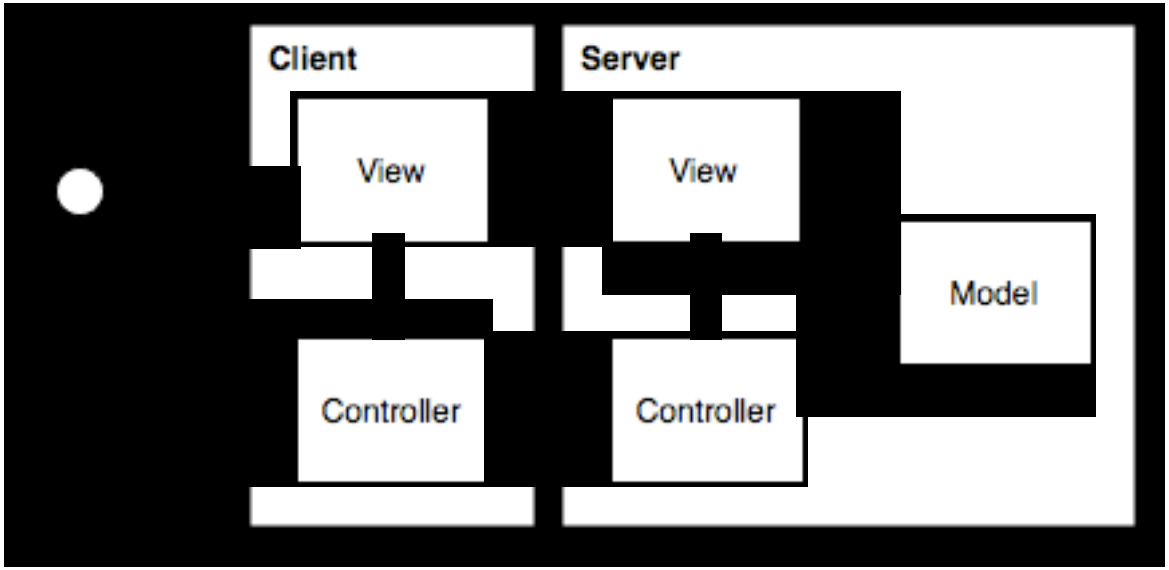
MVC Comparison 1/5

- MVC stands for model-view-controller
  - Separation of concerns in data-driven GUI applications
  - Concise overview



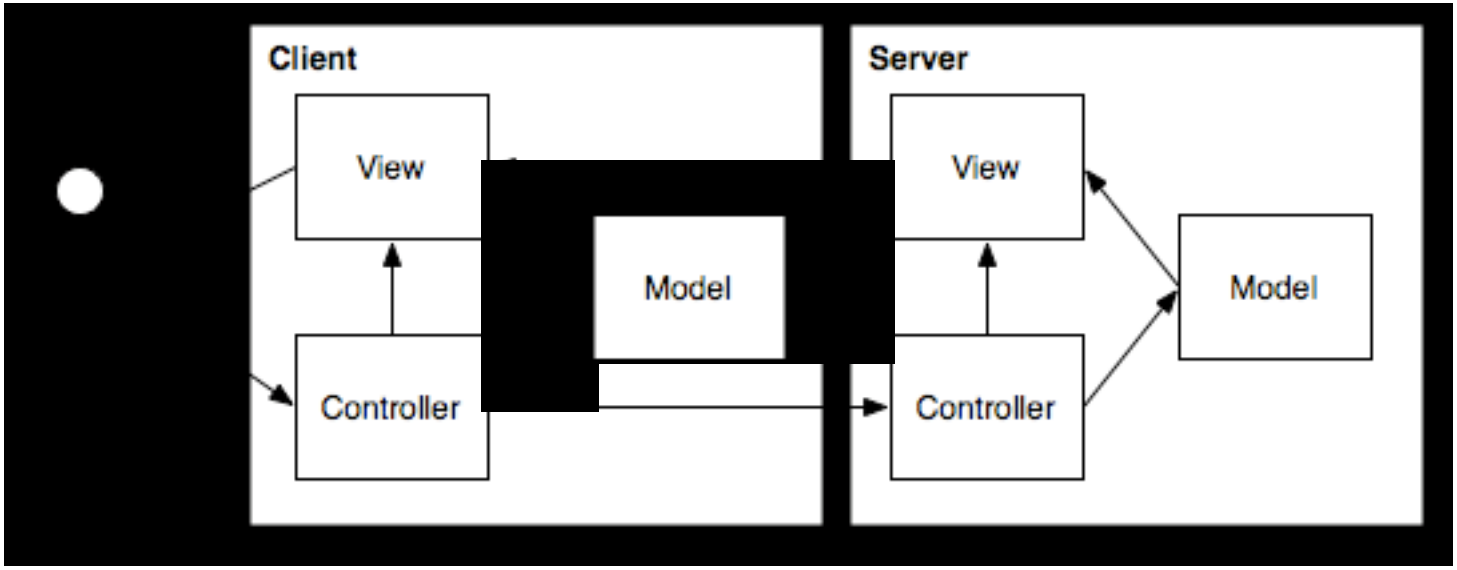
MVC Comparison 2/5

- Server-side web MVC puts burden on server
  - Primitive interaction possibilities in the client
  - Each request generates new HTML view
  - Browser input events summarized as HTTP requests



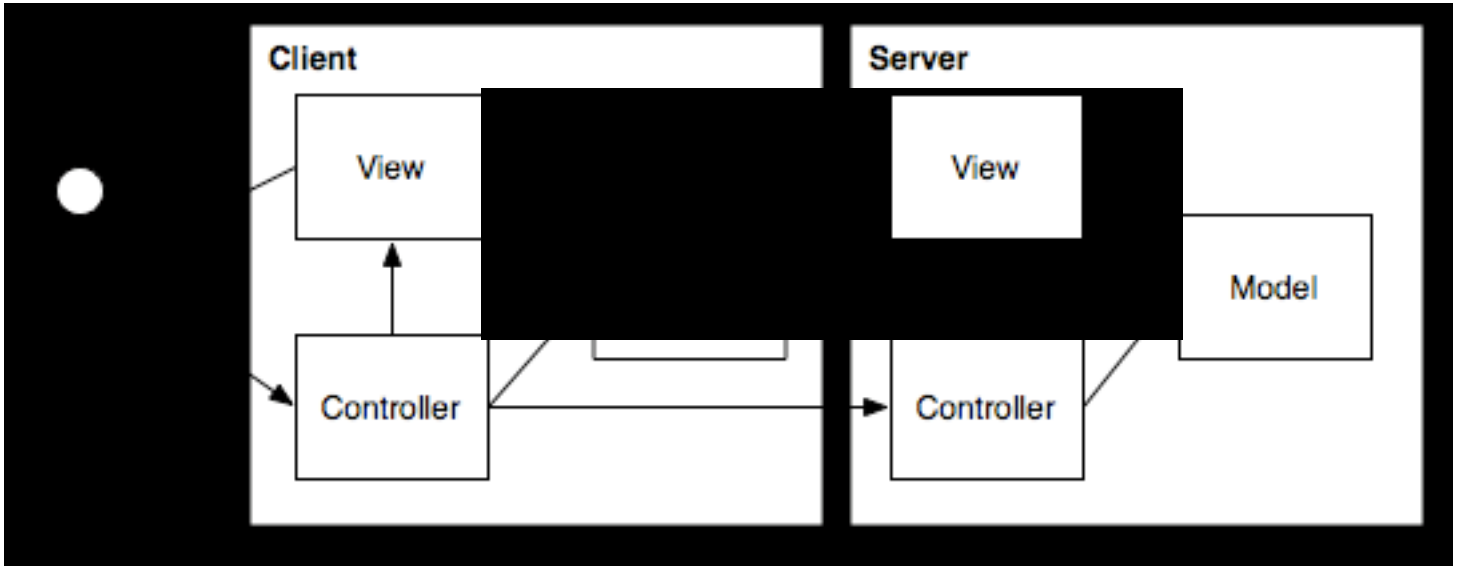
MVC Comparison 3/5

- Extended with Ajax features
  - Client receives parts of the model
  - Client contains dedicated logic



MVC Comparison 4/5

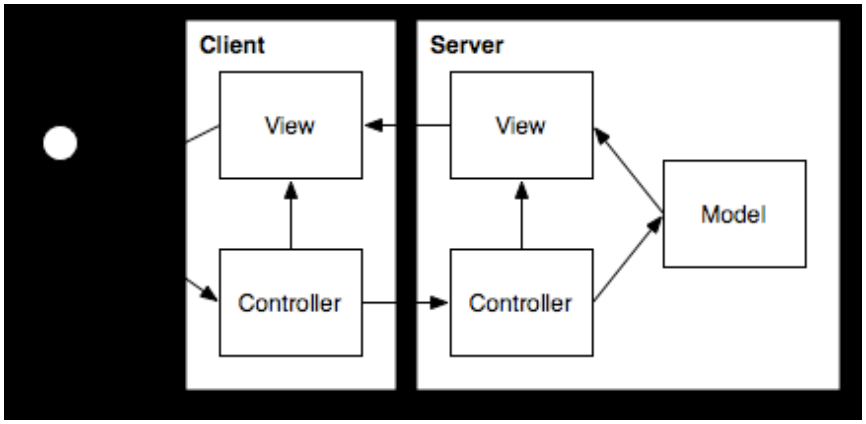
- RIAs eliminate the need for the server-side view
  - Thick client provides a rich user interface experience
  - Core logic in the server that is driven by the controller
  - Parts of the model are provided to the client



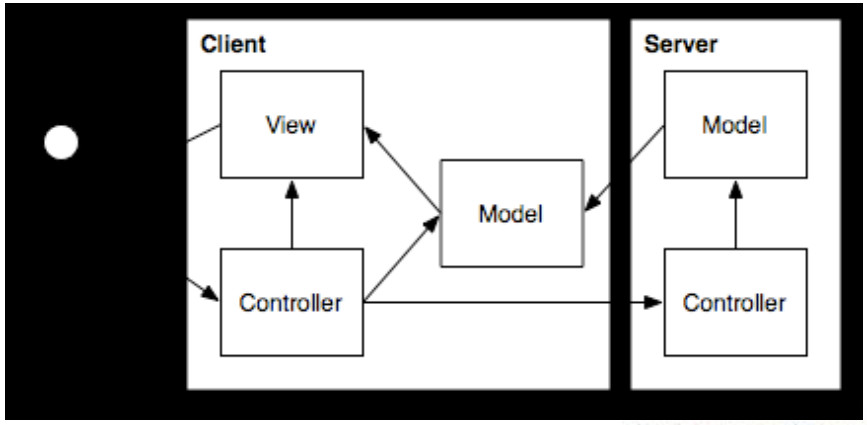
MVC Overview 5/5

Shift from server-side web MVC towards RIA

- Double view handling
- Complete request/response cycle
- Limited client functionalities
- Heavy burden on server



- Single view at the client
- Targeted request/response cycle
- Rich client functionalities
- Thick client and light server load



Agenda

Overview of OpenLaszlo

Introducing LZX

Deployment

Tips and Caveats

Agenda

Overview of OpenLaszlo

Introducing LZX

Deployment

Tips and Caveats

What Is LZX?

- Declarative language
- Tags to build up a user interface
- Feels similar to HTML

```
<canvas title="Laszlo Example 1">
    <text>Hello World!</text>
</canvas>
```

How Do I Try This Out?

- Download OpenLaszlo: http://www.openlaszlo.org
- Install the platform, it will auto-launch
- Afterwards, start the server with provided shortcut
- Open ‘My Apps’ folder in installation directory
- Create your file there, for instance ex1.lzx
- Use your browser to visit: http://127.0.0.1:8080/lps-4.0.0/my-apps/ex1.lzx
- Just reload to see source modifications

DEMO

Running Example 1

LZX Is Event-Driven

- JavaScript™ programming language used to tie functionality to events
- Several styles of event handlers
- Automatic events for attribute changes

```
<canvas title="Laszlo Example 2">
  <button id="btn" text="Press me"
    onclick="msg.setAttribute('visible', true)"/>
  <text id="msg" x="85" y="4" visible="false" text="Hi!">
    <handler name="onvisible" args="v">
      btn.setAttribute("enabled", !v);
    </handler>
  </text>
</canvas>
```

DEMO

See the Event Handling of Example 2

LZX and JavaScript Programming Language

- LZX and JavaScript programming language complement each other
  - LZX for permanent declarative programming
  - JavaScript programming language for dynamic functional programming
- Access LZX from within JavaScript programming language
- Similar to the HTML/JavaScript programming language duo
- Existing web developers will feel right at home

LZX Constraints

- Constraints create relationships between attributes
- Easy dependencies between layout elements that are automatically evaluated

```
<canvas title="Laszlo Example 3">
  <view id="v1" height="50" bgcolor="#000066"
      width="${sl.value}"/>
  <view id="v2" height="50" bgcolor="#999999"
      x="${v1.width}" width="${sl.width - v1.width}"/>
  <slider id="sl" width="300"
      y="${v1.height + thumbheight}" maxvalue="${width}"/>
</canvas>
```

DEMO

See the Automatic Updating of Constraints for Example 3

LZX's Layout Managers

- Better solution for relative layouts than relying on absolute coordinates
- Similar to layout management in GUI toolkits

```
<canvas title="Laszlo Example 4">
  <simplelayout axis="y" spacing="10" inset="10"/>
  <view>
    <simplelayout axis="x" spacing="10" inset="10"/>
    <text>First name</text> <edittext/>
  </view>
  <view>
    <simplelayout axis="x" spacing="10" inset="10"/>
    <text>Last name</text> <edittext/>
  </view>
</canvas>
```

DEMO

See the Layout Management of Example 4

LZX Classes

- Create new tags from existing declarative code
- Quick prototyping and easy componentization
- First, get it working, then make it reusable

```
<canvas title="Laszlo Example 5">
  <class name="insetlayout" extends="simplelayout"
    spacing="10" inset="10"/>
  <class name="forminput">
    <insetlayout axis="x"/>
    <attribute name="label" type="string"/>
    <text text="${classroot.label}"/> <edittext/>
  </class>
  <insetlayout axis="y"/>
  <forminput label="First name"/>
  <forminput label="Last name"/>
</canvas>
```

DEMO

Show That Example 5 Produces Exactly the Same Result as Example 4

LZX Animation

- Animates modifications of attribute values

```
<class name="forminput" onmouseover="bg.flash.doStart()">
  <attribute name="label" type="string"/>
  <view name="bg" bgcolor="#ffcccc" opacity="0"
        width="${parent.width+2}" height="${parent.height}">
    <animatorgroup name="flash" process="sequential"
                   attribute="opacity" start="false">
      <animator from="0" to="1" duration="500"/>
      <animator from="1" to="0" duration="2000"/>
    </animatorgroup>
  </view>
  <view name="field">
    <insetlayout axis="x"/>
    <text text="${classroot.label}"/> <edittext/>
  </view>
</class>
```

DEMO

Show the Flashing Fields of Example 6

LZX Data-Binding

- Binds views to XML data
- Uses XPath to address data elements
- Supports in-lined XML, included, and remote data
- Internal DOM model of the provided data
- DOM modifications are reflected through bindings

LZX Data-Binding

```
<canvas title="Laszlo Example 7">
  <dataset name="ds">
    <people>
      <person firstname="Homer" lastname="Simpson"/>
      <person firstname="Marge" lastname="Simpson"/>
      <person firstname="Montgomery" lastname="Burns"/>
    </people>
  </dataset>
  <class name="forminput" extends="hbox" spacing="10">
    <attribute name="label" type="string"/>
    <text text="${classroot.label}"/>
    <edittext text="${classroot.data}"/>
  </class>
  <simplelayout axis="y" spacing="15"/>
  <vbox datapath="ds:/people[1]/person" spacing="5">
    <forminput label="First name" datapath="@firstname"/>
    <forminput label="Last name" datapath="@lastname"/>
  </vbox>
</canvas>
```

DEMO

Show the Data-Binding of Example 7

Agenda

Overview of OpenLaszlo

Introducing LZX

Deployment

Tips and Caveats

Agenda

Overview of OpenLaszlo

Introducing LZX

Deployment

Tips and Caveats

Two Main Execution Modes

- Proxied
  - Requires OpenLaszlo server to be running
  - Provides additional capabilities
    - Media types other than SWF, JPG, or MP3
    - Persistent connection
    - SOAP, XML-RPC
    - HTTP response headers in XML requests
- Solo
  - Can be deployed to any web server
  - Requires pre-compilation
  - Lacks some of the capabilities of proxied applications

Two Main Execution Environments

- Flash
  - Supports multimedia features
  - Runs on all browsers that support Flash 7 (95% of the users)
  - Doesn't look entirely native due to Flash's font rendering, for instance
- DHTML
  - Doesn't need a plug-in, but supports less UI features
  - Only runs on Firefox 2, Internet Explorer 6, and Safari 2
  - First stable release that still has some issues
  - Uses the browser's font rendering and widgets

Pre-Compiling SOLO Applications

- SOLO applications can be pre-compiled
  - Use the development console
  - Use the lzc command line utility
  - Use OpenLaszlo's Java platform main class
    - With Apache Ant
    - Integrated inside your Java application

DEMO

Show the Different Executions Modes and Environments, as Well as the Development Console an SOLO Pre-Compilation Options

Agenda

Overview of OpenLaszlo

Introducing LZX

Deployment

Tips and Caveats

Agenda

Overview of OpenLaszlo

Introducing LZX

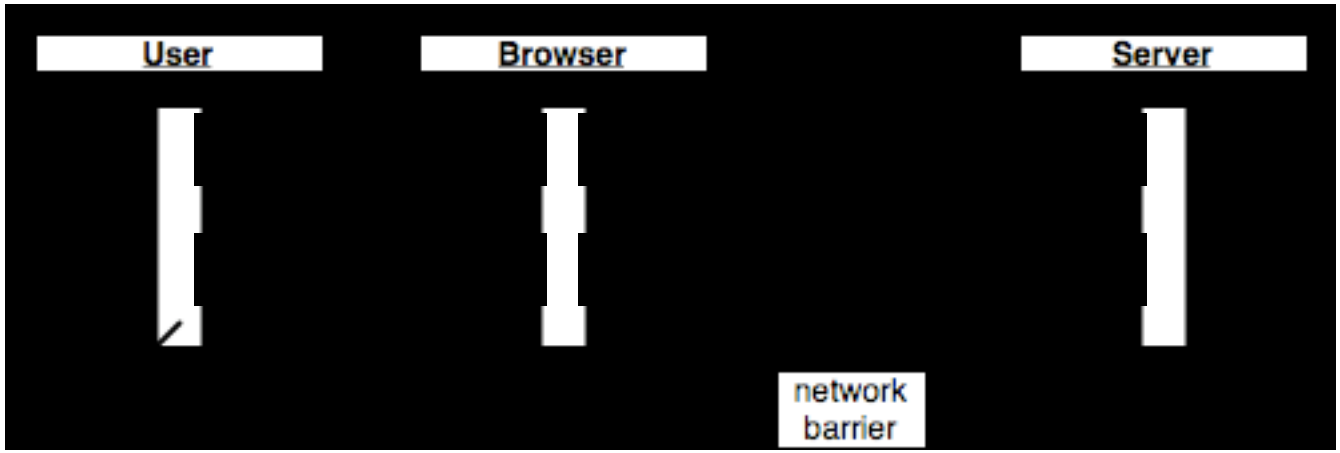
Deployment

Tips and Caveats

Reduce Latency to Improve Usability

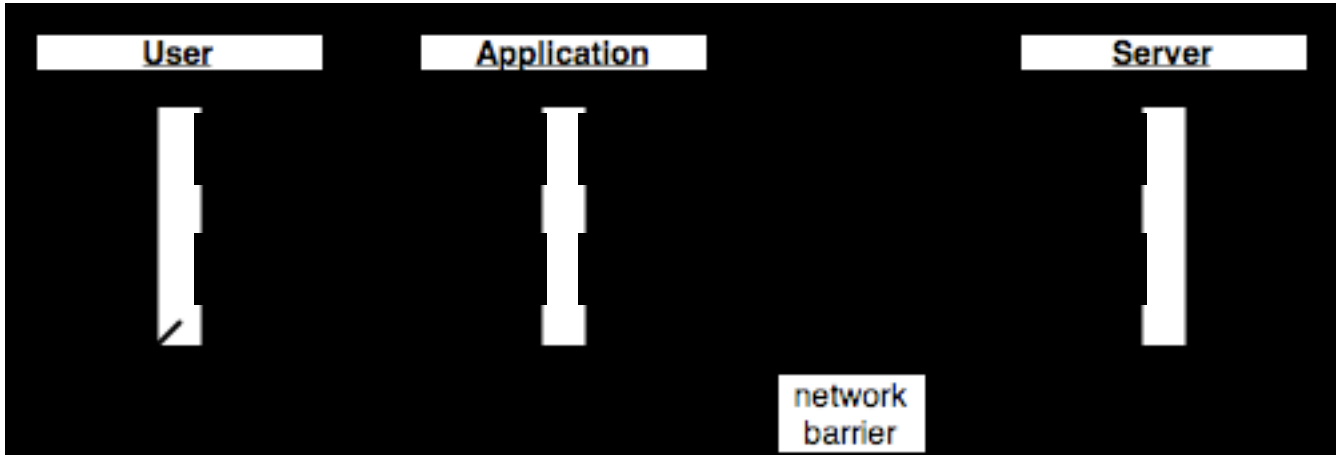
Reduce Latency to Improve Usability

- Network latency was not a real problem before
  - Traditional web apps render the page at each request
  - Occasional slowness accepted due to page refresh
  - Applications don’t resemble desktop applications



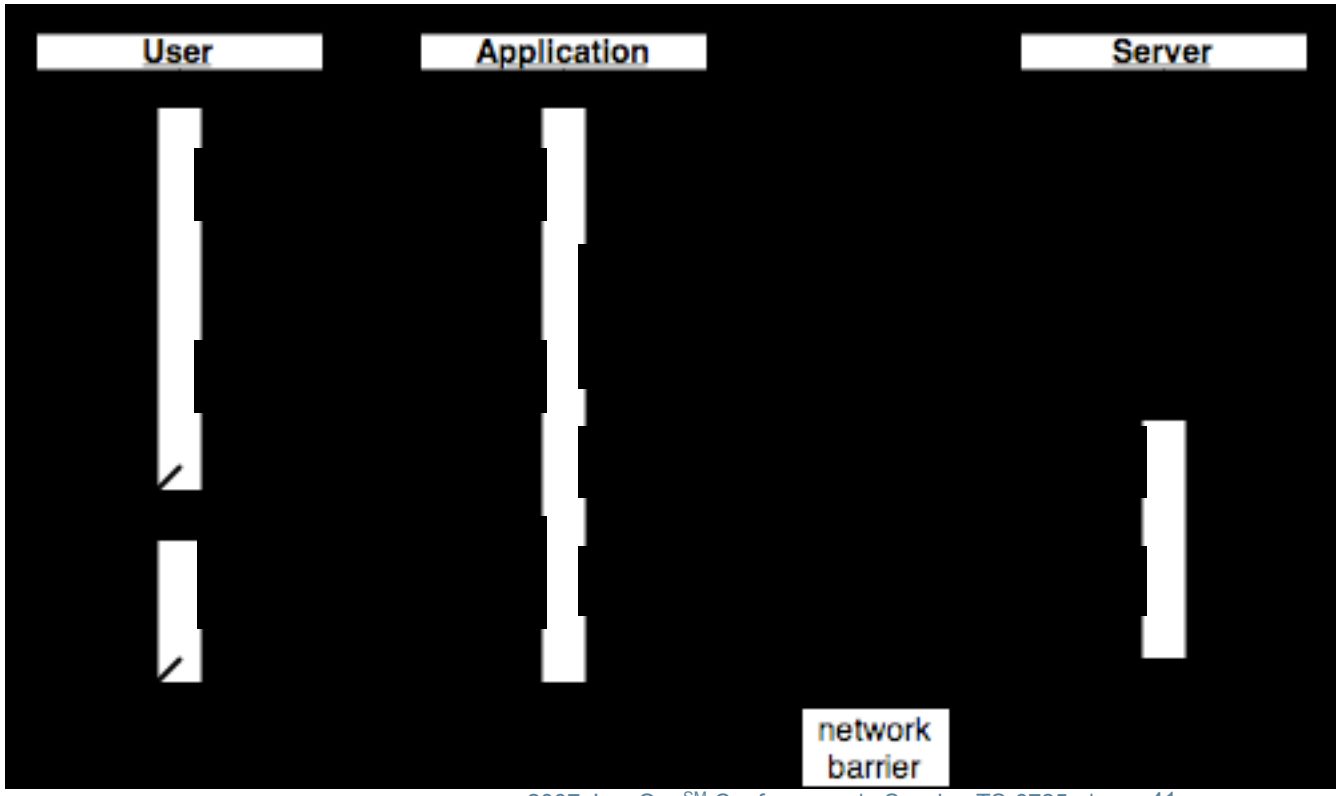
Reduce Latency to Improve Usability

- For RIAs, network latency can give the impression of a sluggish application
  - Only modified parts of the interface are updated
  - Fine-grained actions should respond instantly
  - Desktop applications respond immediately and RIA resemble them



Reduce Latency to Improve Usability

- Update the internal application model
- Asynchronously send a request to the server



Create a Multi-Purpose, Server-Side Solution

Multi-Purpose, Server-Side Solution

- Turn the server into an generic API
- Provide RESTful web-services
- Open up the application for other clients

Multi-Purpose, Server-Side Solution

- What are RESTful web-services?
  - Standard HTTP requests with clean URLs and parameters
  - Use the POST method for modifications
  - Use the GET method for idempotent actions
  - Responses are XML representations of the model

Multi-Purpose, Server-Side Solution

- RESTful web-service POST example
  - Request to create a new to-do list while being logged in

```
http://blablalist.com/createlist
```

POST parameters:

```
authid      622c895dec2d96cf127f0d557785d200
submission  create
name        My new list
```

Multi-Purpose, Server-Side Solution

- RESTful web-service POST example
  - Request to create a new to-do list while being logged in
  - Example XML response

```
<create authid="622c895dec2d96cf127f0d557785d200">
     <success id="37"/>
</create>
```

Multi-Purpose, Server-Side Solution

- RESTful web-service GET example
  - Request to get to-do list info while being logged in

```
http://blablalist.com/getlist?
    authid=622c895dec2d96cf127f0d557785d200&
    id=23
```

Multi-Purpose, Server-Side Solution

- RESTful web-service GET example
  - Request to get to-do list info while being logged in
  - Example XML response

```
<list authid="622c895dec2d96cf127f0d557785d200"
    id="23" name="Things I need to do this weekend"
    shortname="this_weekend" public="true" count="2"
    listurl="http://blablalist.com/list/johnsmith/this_we"
    feedurl="http://blablalist.com/feed/johnsmith/this_we">

  <description>It&apos;s time to do this!</description>

  <entry id="140" name="Thing1" done="false" priority="0"/>
  <entry id="141" name="Thing2" done="false" priority="1"/>
  <entry id="142" name="Thing3" done="true" priority="2"/>
</list>
```

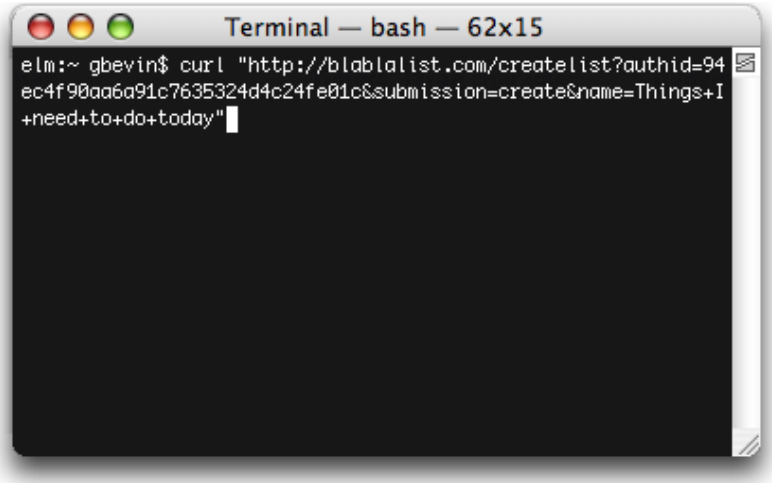
Multi-Purpose, Server-Side Solution

- What are the advantages of an open API?
  - Development of other GUIs
  - Integration with other tools
  - Scriptability and automation
  - Easier to develop clients with different capabilities
  - Mashups!

Multi-Purpose, Server-Side Solution

- Two clients that use the same server services (request)

RIA OpenLaszlo

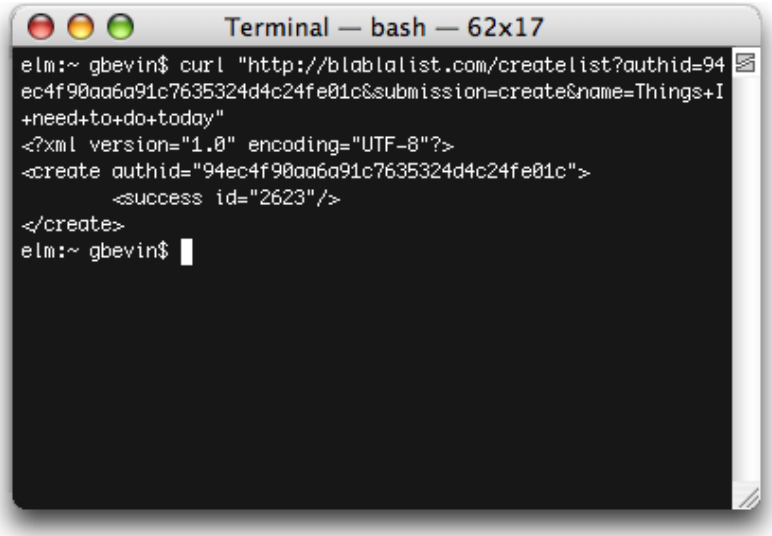


Command-Line Curl

Multi-Purpose, Server-Side Solution

- Two clients that use the same server services (response)

RIA OpenLaszlo



Command-Line Curl

Structure for Maintainability

Structure for Maintainability

- Traditional web apps have separate entry points for each application page
- RIA have one main entrance
  - Panel switches happen immediately
  - No complete page reloads
  - Similar to desktop applications and welcomed by users
- As the application becomes larger
  - More events are needed reach desired location after a recompilation
  - Problematic for developers

Structure for Maintainability

- OpenLaszlo's modularization to the rescue
  - Put each component, screen, panel or module in the application into a library, for example helloworld.lzx:

```
<library>
  <window title="Hello Window" resizable="true">
     <text>Hello World.</text>
  </window>
</library>
```

  - Create a main wrapper canvas for each such library, for example helloworld_wrapper.lzx:

```
<canvas width="100%" height="100%">
  <include href="helloworld.lzx"/>
</canvas>
```

Structure for Maintainability

- Benefits
  - Each wrapper can be accessed individually
  - Focus on the development of that particular library
  - Initialization variables can be setup in the wrapper to test different situations or to setup a context
  - Include libraries in canvas to create main application
  - Application is already modularized
  - Ready when dynamic libraries are needed

Q&A

OpenLaszlo: From RIA to Ajax and Mobile

Geert Bevin

CEO, Uwyn bvba/sprl, http://www.uwyn.com

Developer, Terracotta Inc., http://terracotta.org

TS-6725