# The Server-side Architecture Behind OpenLaszlo Applications

**Geert Bevin**
gbevin@uwyn.com
http://www.uwyn.com
http://www.rifers.org

# Agenda

- What are Rich Internet Applications?
- Why use OpenLaszlo?
- Architecture comparison with regular web MVC
- Designing a multi-purpose RIA server-side solution
- Implications for the client-side
- Making your RIA applications maintainable
- Q&A

# Who am I?

- **Geert Bevin**
- **CEO of Uwyn, a small custom application development company** (http://uwyn.com)
- **founder of the RIFE Java web application framework** (http://rifers.org)
- **official contributor to OpenLaszlo**
- **creator of Bla-bla List, open-source RIA to-do list tracker in OpenLaszlo** (http://blablalist.com)

# What are Rich Internet Applications?

- full-featured web-based interactive GUI applications
- launch without any installation
- run in a secure sandbox to protect the local machine
- information is stored on the server
- the application can be used from anywhere
- platform independent

# Examples

- **Amazon Store**

  **This RIA interpretation of an Amazon music store demonstrates presenting a range of functionality all in one window, drag & drop between windows, a dynamic shopping cart, and more, including pulling live data from Amazon's back-end database.**

  **http://www.laszlosystems.com/partners/support/demos/**

# Examples

- **Bla-bla List**

  **Bla-bla List is a free, secure, simple and sharable to-do list service. It's open-source and written to explore the world of rich internet applications. The first implementation uses OpenLaszlo, and the plan is to implement the same features in other RIA technologies**

  **http://www.blablalist.com**

# Examples

- **Pandora**

  **Pandora is an intelligent radio station that automatically suggests the songs that are played according to your musical preference and taste.**

  **http://www.pandora.com**

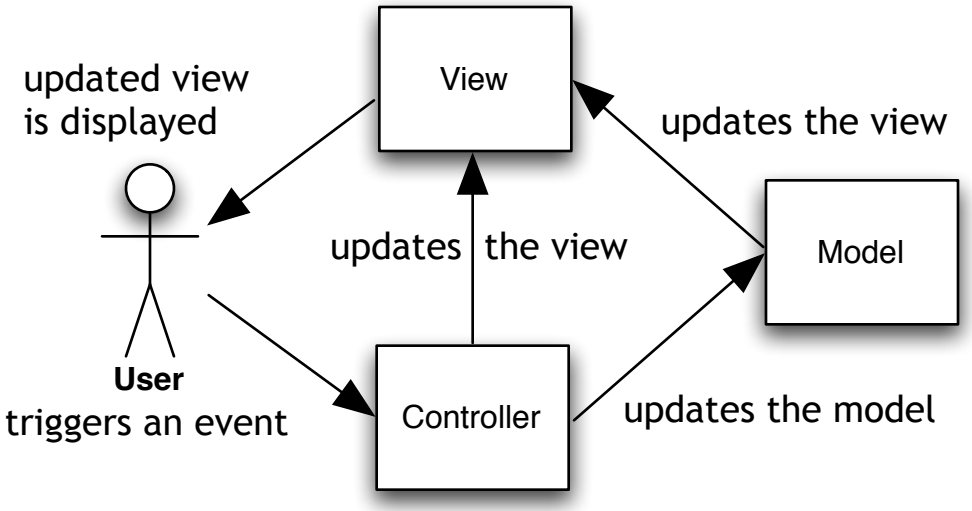
# Why use OpenLaszlo?

- open-source, stable and well-documented
- runtime-independent development platform
- powerful object-oriented component-based language
- extends the RIA focus to multi-media capabilities
- currently targets Flash which has 97% market penetration
- no browser compatibility hell

# Comparison with regular web MVC 1/5

## MVC stands for model-view-controller

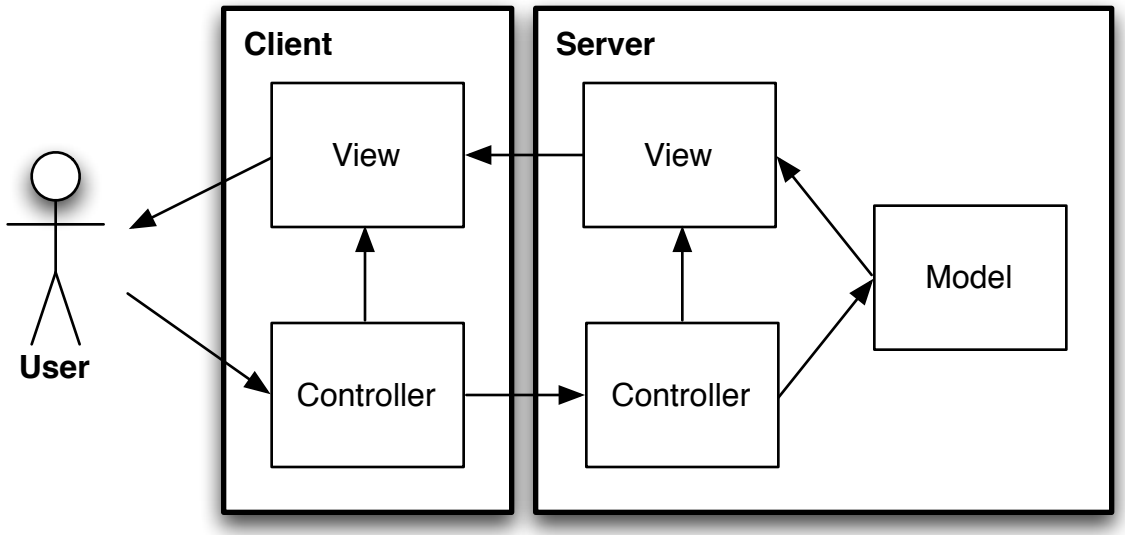
standard pattern for separating concerns in a data-driven GUI application



# Comparison with regular web MVC 2/5

**Server-side web MVC puts all the burden on the server with only primitive interaction possibilities in the client.**

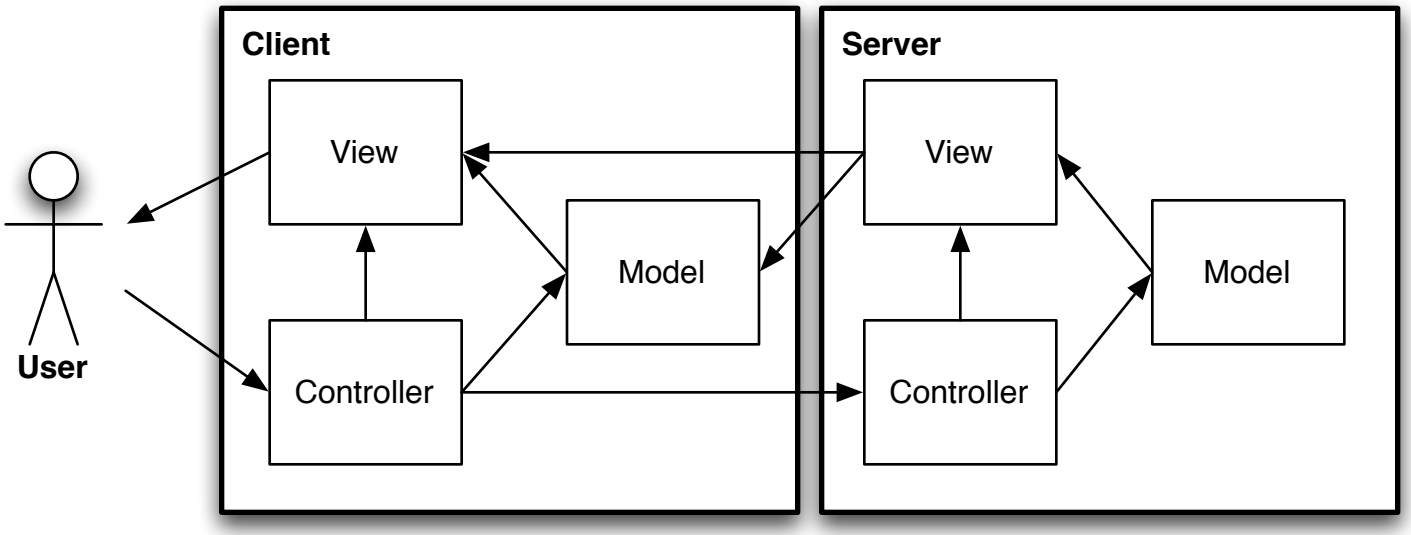
- each client interaction makes the server view generate instructions for the client's view: the HTML document that the browser interprets.
- the client's controller doesn't interact with the server controller directly: the browser handles input event which are summarized in HTTP requests.



# Comparison with regular web MVC 3/5

**This architecture has been extended with more intelligent features like client-side validation, at the expense of adding more complexity.**

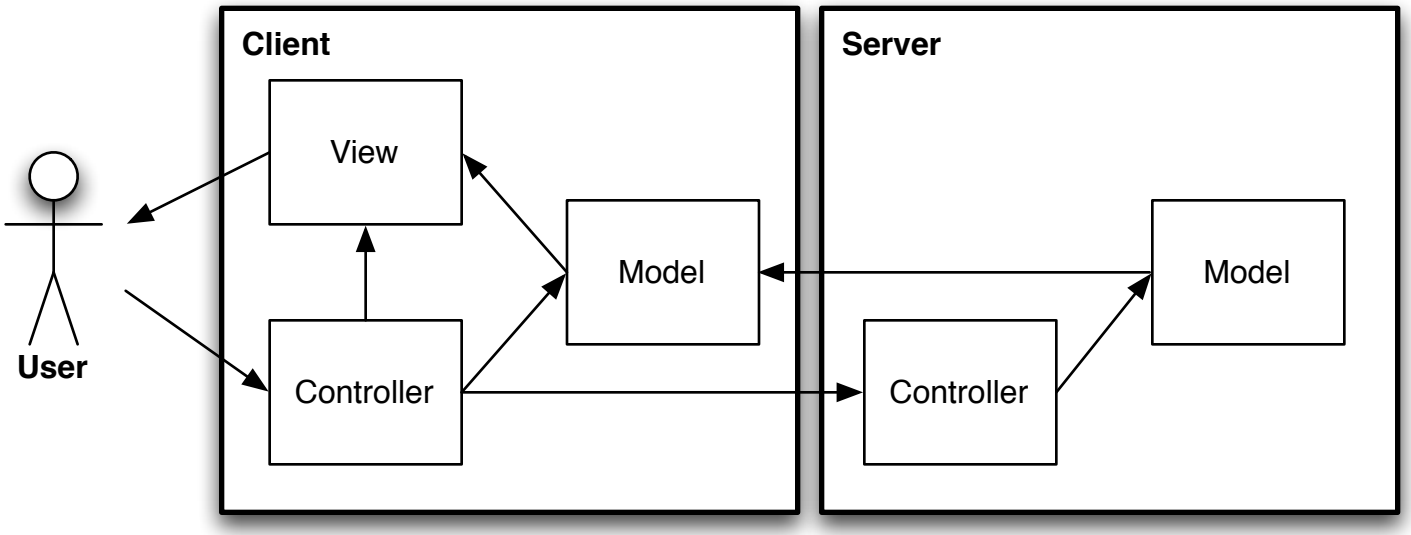
- the client needs to receive parts of the model to be able to perform the more interactive functionalities
- the client will contain logic to be able to work with the model parts it received



# Comparison with regular web MVC 4/5

**Rich Internet Applications are able to handle all view-related functionalities, which eliminates the need for the server-side view.**

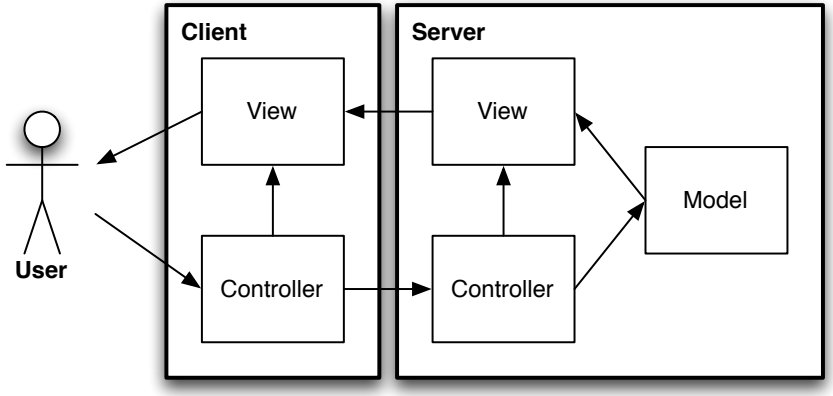
- the client has become thick and is able to provide a rich user-interface experience
- the server contains only the core logic that is driven by the controller, parts of the model are provided to the client



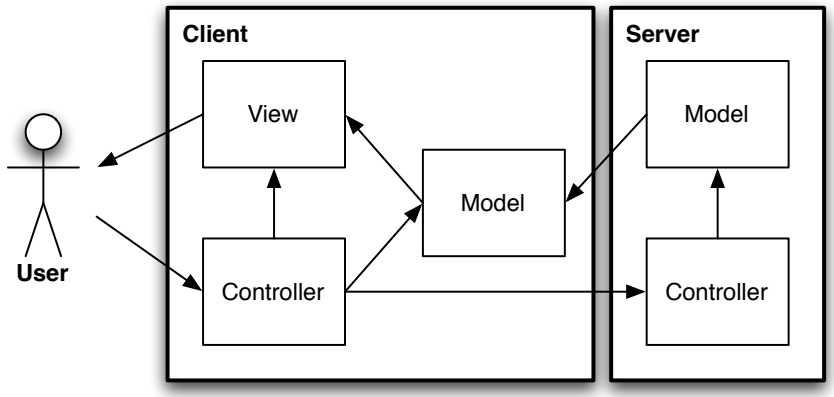
# Comparison with regular web MVC 5/5

**Overview of the architectural 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



# A multi-purpose RIA server-side solution 1/7

- **turn the server into an generic API**
- **provide RESTful web-services**
- **open up the application for other clients**

# A multi-purpose RIA server-side solution 2/7

## 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

# A multi-purpose RIA server-side solution 3/7

## RESTful web-service POST example

**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 |

**example XML response**

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

# A multi-purpose RIA server-side solution 4/7

## RESTful web-service GET example

**example request to get the info of a to-do list while being logged in**

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

**example XML response**

```
<list authid="622c895dec2d96cf127f0d557785d200"
          id="23" name="Things I need to do this weekend" shortname="this_weekend"
          public="true" listurl="http://blablalist.com/list/johnsmith/this_weekend"
          feedurl="http://blablalist.com/feed/johnsmith/this_weekend"
          printurl="http://blablalist.com/printablelist?id=23"
          privateshares="1"
          count="2" isowner="true">
    <description>It&apos;s really time that I finally get all this done!</description>

    <entry id="140" name="Fix the electricity" done="false" priority="0"/>
    <entry id="141" name="Put in the new lightbulbs" done="false" priority="1"/>
    <entry id="142" name="Wash the car" done="true" priority="2"/>
</list>
```

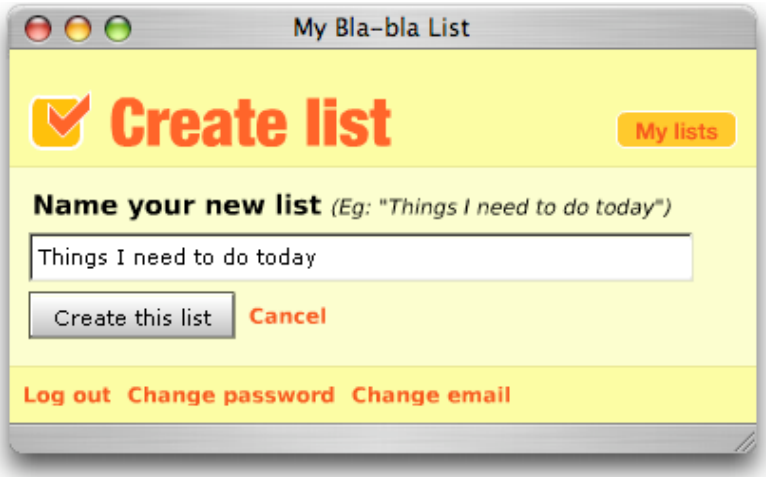
# A multi-purpose RIA server-side solution 5/7

## What are the advantages of an open API?

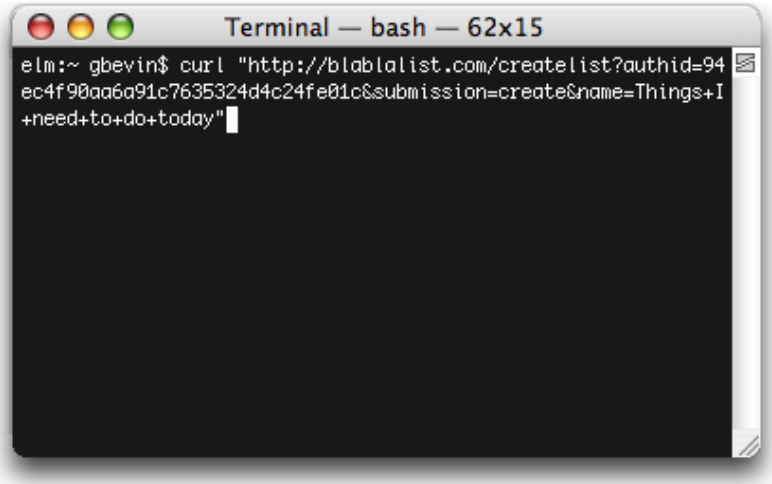
- development of other GUIs
- integration with other tools
- scriptability and automation
- easier to develop clients with different capabilities

# A multi-purpose RIA server-side solution 6/7

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



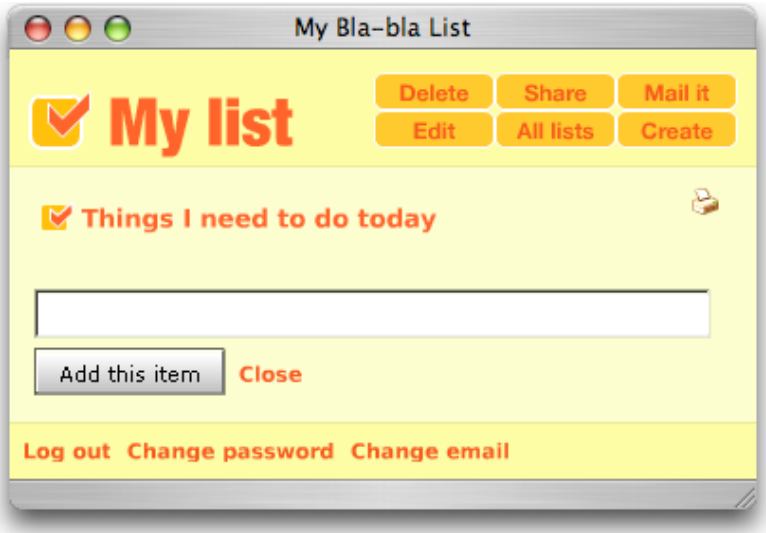

RIA OpenLaszlo

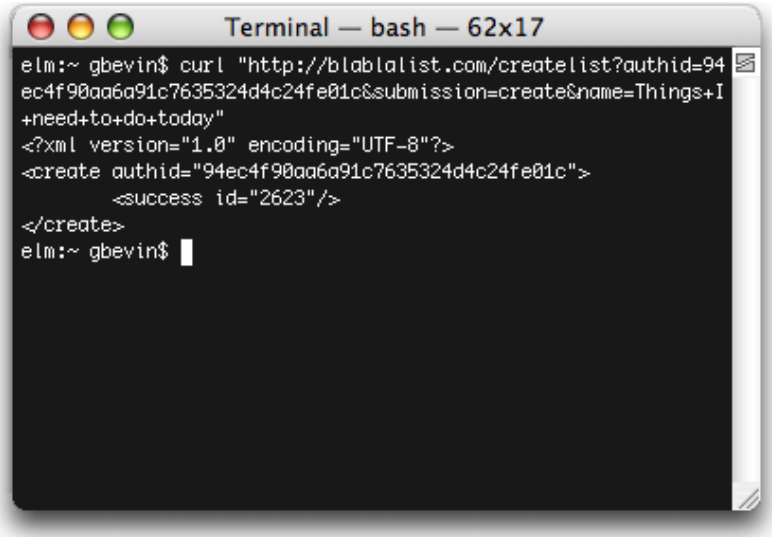



command-line Curl

# A multi-purpose RIA server-side solution 7/7

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




RIA OpenLaszlo

command-line Curl

# Implications for the client-side



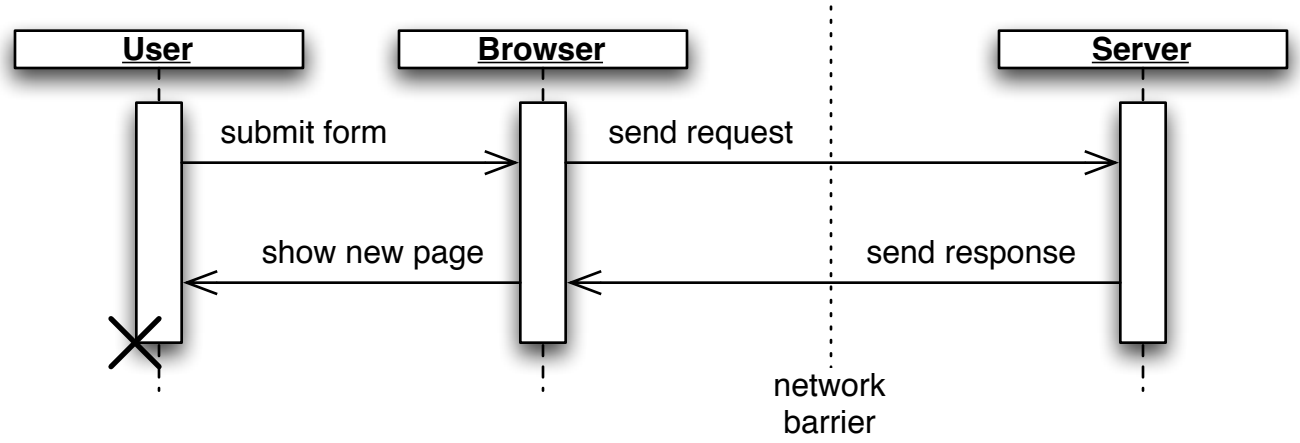
- **usability suffers from latency**
- **individual steps and panes aren't implicitly accessible through a dedicated URL**
- **client-side performance is limited**
- **create alternate printable views**

# Implications for the client-side



## Network latency was not a real problem before

- traditional web applications have a render step at each action
- occasional slowness accepted because the entire page is refreshed
- the applications don't resemble desktop applications

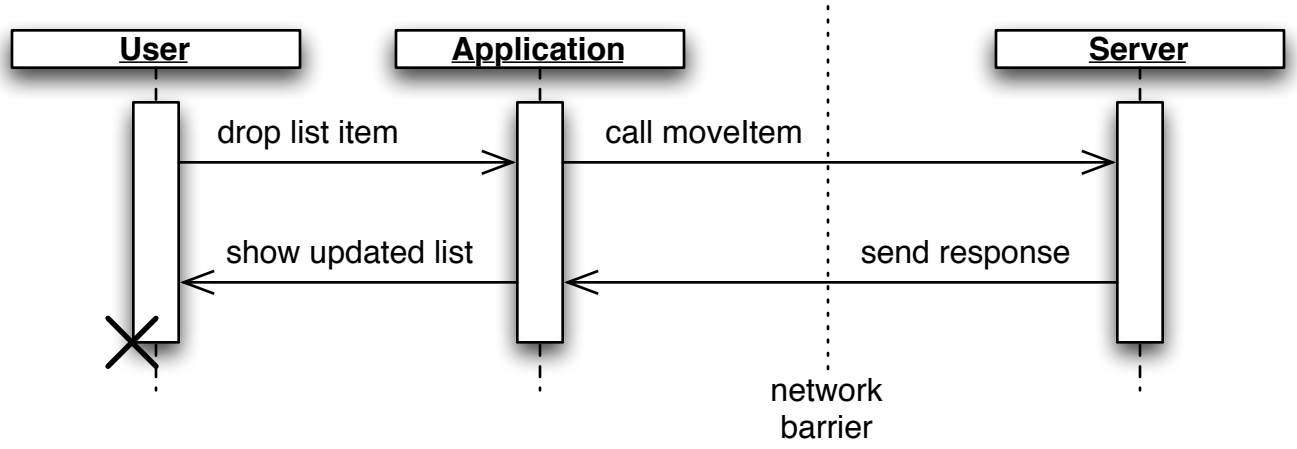


# Implications for the client-side



**For RIAs, network latency can give the impression of a sluggish application**

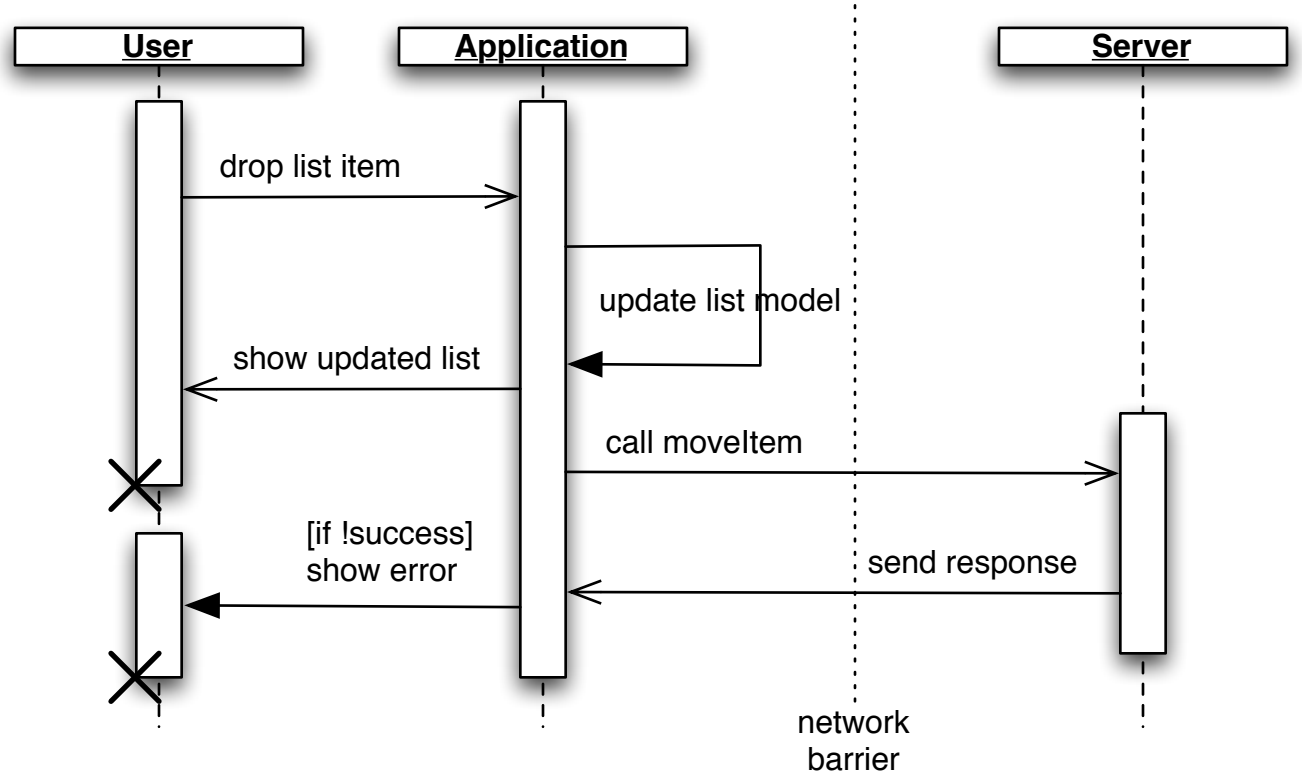
- **only modified parts of the interface are updated**
- **fine-grained actions (like drag & drop) should respond instantly**
- **desktop applications respond immediately and RIA resemble them**



# Implications for the client-side



**The solution is to update the internal application model and asynchronously send a request to the server**

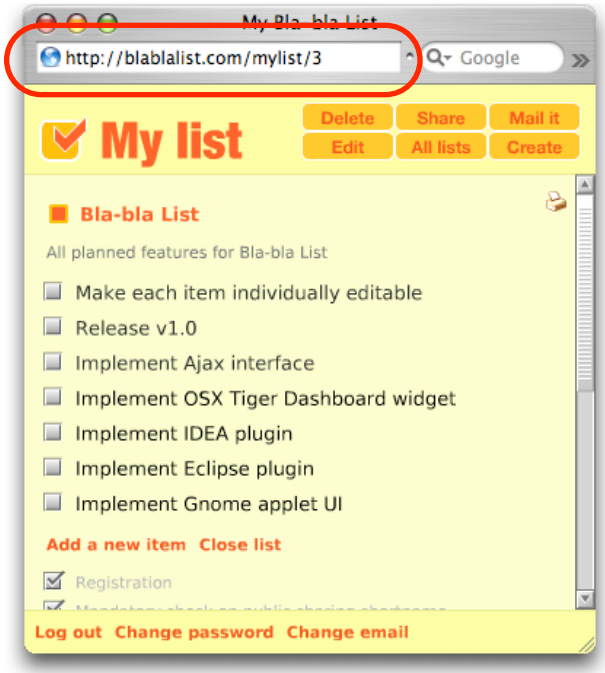


# Implications for the client-side



## Full render functionality is also needed

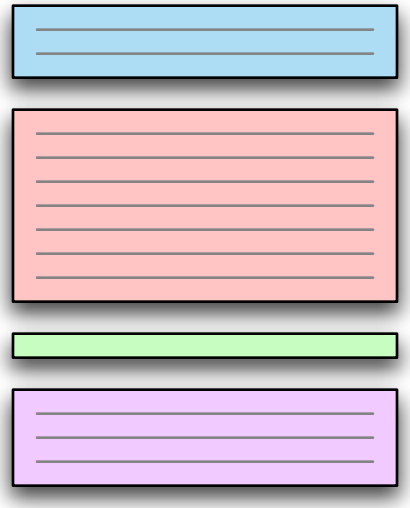
- your application panels aren't populated when the application is first accessed after startup
- invisible views should not be updated in the background for performance reasons
- it's still a web application, people want to be able to access URLs to retrieve information directly

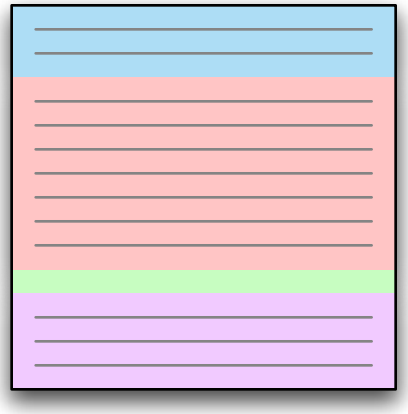


# Implications for the client-side 6/9

## Development is more complex for the GUI of RIAs

- fine-grained incremental UI and model updates need to be supported
- complete UI render steps of the same information is needed too





# Implications for the client-side 7/9

## Be careful about the logic you move to the client side

- Flash's garbage collector has trouble with lots of small objects
- OpenLaszlo's JavaScript implementation isn't fully Ecma compliant
- Flash executes byte-code slowly
- OpenLaszlo's JavaScript can be up to 400 times slower than the JavaScript interpreter of your browser

## Solutions

- deploying for Flash version 7 and 8 improve performance a lot
- OpenLaszlo v3 let's you communicate with browser JavaScript through its `LzBrowser.loadJS` method and benefit from the speed increase and complete Ecmascript functionalities

# Implications for the client-side



## OpenLaszlo has no native printing support

- relies on the print functionality of the browser
- non visible elements can't be printed
- impossible to make a print-version layout

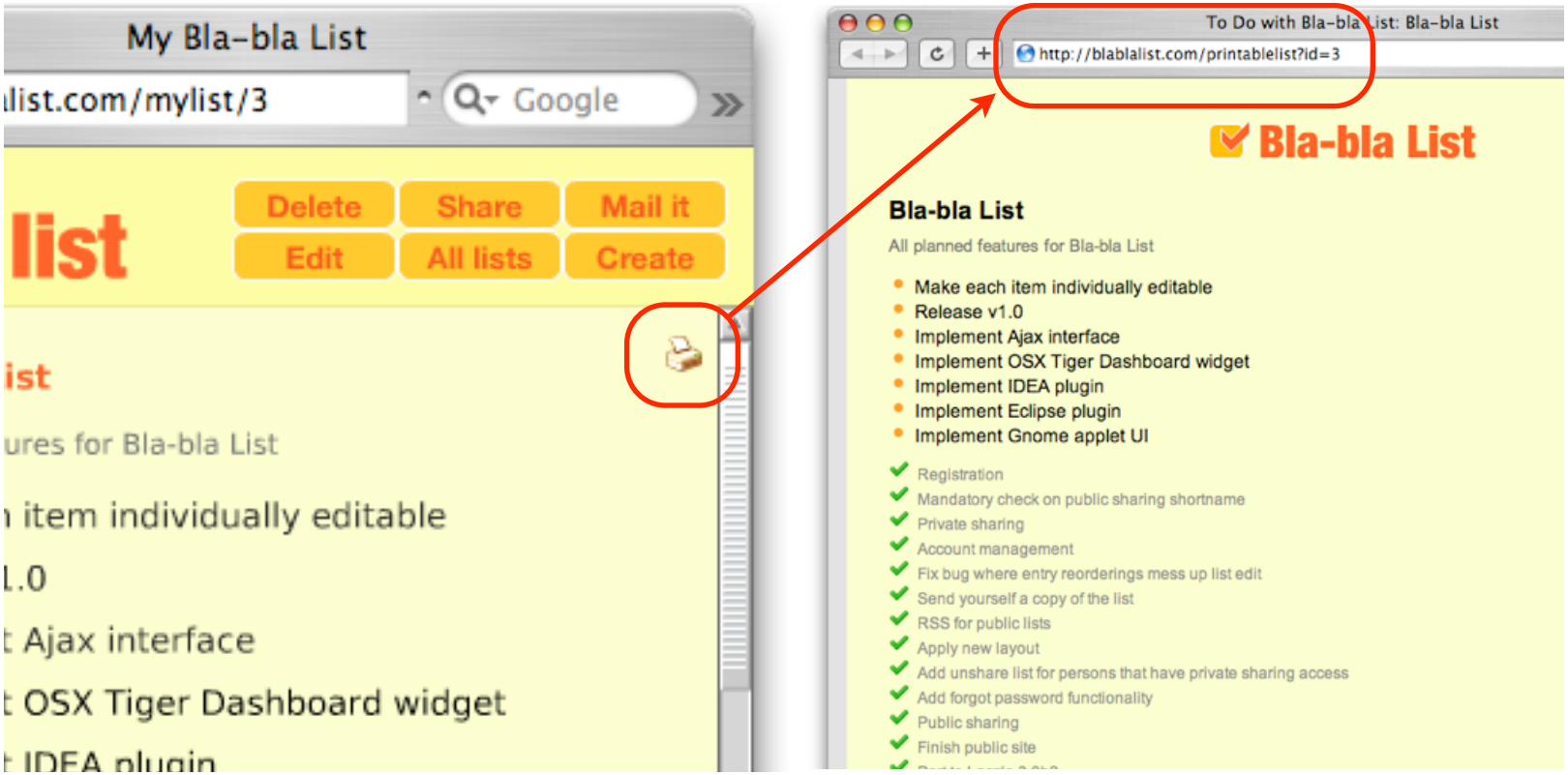
## Solution

- create printer-specific regular XHTML pages
- provide links to these pages from the RIA UI where the print functionality should be available
- the REST XML output makes it possible to use templating and transformation solutions like XSLT to layout the data without having to re-implement the back-end logic

# Implications for the client-side



## Example of printing support



# Maintainable RIA applications 1/3

- **traditional web applications typically have a separate entry point for each application page**
- **RIA typically have one main entrance that loads the entire application**
  - **panel switches happen immediately without complete page reloads**
  - **similar to desktop applications and welcomed by users**
  - **as the application becomes larger, more actions are needed to get to the location where development happens after a recompilation**
  - **problematic for developers**

# Maintainable RIA applications 2/3

## OpenLaszlo's modularization to the rescue

- **put each component, screen, panel or module in the application into a library:**

*helloworld.lzx*

```
<library>
    <window x="20" y="20" width="200" height="250"
            title="Hello Window" resizable="true">
        <text>Hello World.</text>
    </window>
</library>
```

- **create a main wrapper canvas for each such library**

*helloworld_wrapper.lzx*

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

# Maintainable RIA applications 3/3

## Benefits

- each wrapper can be accessed individually to 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
- all libraries can be included in the main canvas and used to create the full-blown application
- every part of the application is already modularized and ready for when dynamic libraries are needed

# Audience Response

Questions?