potix

SIMPLY RICH

ZK

The Product Overview

January 2006

Potix Corporation

Last Modified
January 21, 2006

Executive Summary

As the Web continues to extend its reach into our daily lives, traditional page-based Web applications face a substantial challenge: the inability to visually represent the complexities in today's applications. The result are two folds: frustrated user experiences and excessive development costs.

Over a decade of evolution, Web applications evolved from static HTML pages, to Dynamic HTML pages, to Java applets and Flash, and, finally, to Ajax technologies (Asynchronous JavaScript and XML). Illustrated by Google Maps and Netflex, Ajax breathes new life into Web application by delivering the similar richness of desktop applications without special plug-in at the browser.

Enabling the rich user experiences by Ajax means adding more cost and risk to the already costly Web development. It includes more skill prerequisites for development, such as JavaScript and asynchronous programming. It includes more maintenance efforts, such as replication of business logic at thousands upon thousands of clients.

In response to this challenge, Potix has developed technologies and tools that enable Web applications to have both the rich user experiences and the simple programming model. The core of the Potix solution is ZK, which includes an Ajax-based event-driven engine to automate interactivity, a rich set of XUL and XHTML components to enrich usability, and a markup language to design user interfaces without programming.

With ZK, you represent your application in feature-rich off-the-shelf XUL and XHTML components, and manipulate them by listening to events triggered by users, as you did for years in desktop applications. All your application codes are running at the server, while the visual representations of components and user activities at the browsers are automatically synchronized by ZK.

Your users get the same engaged interactivity and responsiveness as using desktop applications, while your development remains the same simplicity as coding desktop applications. Meanwhile, your art designers, by use of a modern markup language, weave your fantastic user interfaces without programming as straight-forward as authoring Web pages with HTML.

The reach of your applications is boundless. With the standard edition, your applications could reach browsers that support HTML and JavaScript. With upcoming ZK for Mobile, your applications could reach any devices that support J2ME, such as PDA, mobiles and game consoles. Moreover, you don't need to modify your application at all[1].

To maximize ZK potential and to minimize the worry of vendor lock-in, Potix opens the source codes of ZK under GPL[2]. It got examined and testified from eyeballs all around the world. You got the freedom to add your value in, and to choose support and components from other parties.

1 For devices with small screen, you usually have to adjust the presentation pages.

2 http://www.fsf.org/licensing

The simplest way to make Web applications rich.

Introduction

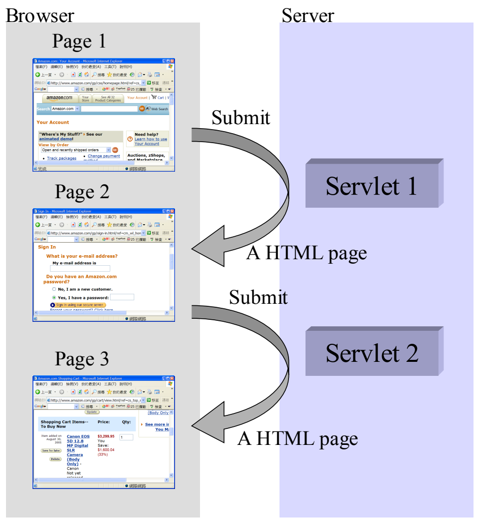
As the Web reaching our daily lives, the effectiveness of Web applications to communicate with users and the simplicity of developing Today's sophisticated applications becomes more important than ever. In response to this challenge, Potix has developed technologies and tools that enable Web applications to have both the rich user experiences and the simple programming model.

Traditional Web Applications

The Web has emerged as the default platform for application development, as the Web continues to extend its reach into offices, family and our daily lives. The adoption and usage of the Web has acted as the driving force behind Information Technology spending. It covers everything from filing expense claims to oversea collaboration, from shopping to sharing photos, from business to consumers, from technology to culture. According to IDC[3], Enterprise Information Portals (EIP), B2B and B2C applications, and e-Commerce spending are expected to grow at 41.2% ($3billitons), 20%, 57.2% ($5.7 trillions) CAGR in 2006, respectively.

As the Web continues to extend its ubiquitous reach and popularity, traditional Web applications face a substantial challenge: the inability to visually represent the complexities in today's applications. The limitations are inherent from the page-based and stateless-communication model.

In this model, a page is self-contained and the minimal unit to communicate between clients and servers. While simple and elegant in design and for exchanging documents, ironically the page-based model has become cumbersome and complex for developing modern applications.



For example, to give a customer a quotation, you might have to open another page to search his trading records, another page for the recent prices, and another page for current stocking. Users are forced to leave the page he is working on, and navigate among several pages. It is easy to get lost and confused, and the result is unhappy customers, lost sales and low productivities.

The challenge to develop a modern application upon this page-based model is also substantial. In this model, applications running at the server have to take care everything from parsing the request, rendering the response, routing processes that link users from one page to another, and handling versatile errors made by users. Tens of frameworks, such as Struct, Tapestry and JSF, are then emerged to simplify this

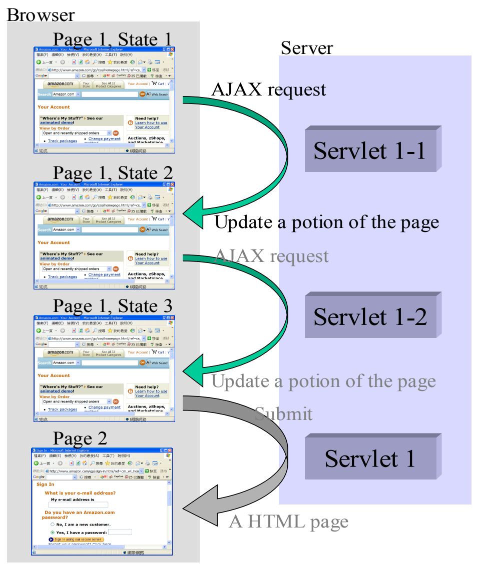
3 http://download.macromedia.com/pub/solutions/downloads/business/idc_impact_of_rias.pdf

development process. Due to the huge gap between the page-based model and the modern applications, learning and using these frameworks is never a pleasant process, not to mention intuition or simplicity.

Ad-hoc Ajax Applications

Over a decade of evolution, Web applications evolved from static HTML pages, to Dynamic HTML pages, to applets and Flash, and, finally, to Ajax[4] technologies (Asynchronous JavaScript and XML). Illustrated by Google Maps and Suggest, Ajax breaths new life into Web applications by delivering the same level of interactivity and responsiveness as desktop applications. Unlike applets or Flash, Ajax is based the standard browser and JavaScript and no proprietary plugin is required.

Ajax is a kind of new generation DHTML. Like DHTML, it heavily relies on JavaScript to listen events triggered by user's activity, and then manipulate visual representation of a page (aka. DOM) in the browser dynamically. Moreover, it takes a step further by enabling the communication with the server asynchronously without leaving or rendering the whole page again. It breaks the page-based model by introducing light-weight communication between clients and servers. With proper design, Ajax could bring rich components common to desktop applications to life in Web applications, and all of their content could be dynamically updated under the control of applications.



Challenges

When providing the interactivity that users demand, Ajax adds more complexities and skill prerequisites to the already costly development of Web applications.

1. Incompatible and Sophisticated JavaScript API

Manipulating DOM in the browser and communicating with servers to implement sophisticated components is not easy. Incomplete, even buggy, implementation of DOM API found in most browsers and incompatibility among different versions and browsers make the development process time-consuming and frustrating.

2. Replicating a Subset of the Application Data Model and Business Logic in the Browser

The use of Ajax is mainly to exchanges data between clients and servers, beside handling the visual representation. Such exchanging totally depends on the applications. It, like client/server,

4 Ajax is coined by Jesse James Garrett in Ajax: A New Approach to Web Applications.

has strong tendency to replicate business logic and even data to the client to simplify the chores of communication and to improve performance. It therefore introduces the significant cost of development and maintenance.

3. Synchronization between the Clients and the Servers

The Ajax clients communicate with the server asynchronous. From the server's viewpoint, Ajax requests are no different from regular HTTP requests and they are processed in parallel. It therefore causes some racing problems that application developers have to handle. For example, when user types a product number, an Ajax request is sent for asking the price. Meanwhile, user might click submit without waiting the price to come back. Then, the application is hard to decide whether a price is entered intentionally by users, or just not yet responsed back.

Current Solutions

In order to deliver a manageable Ajax solution, many frameworks or libraries are developed . They can be classified into three categories.

1. JavaScript Components and Libraries

The most straight forward way[5] is to provide ready-to-use JavaScript components so application developers need not to handle sophisticated user interface from scratch. However, application developers have to manipulate these components in JavaScript and develop a custom-made way to handle dynamic data exchanging between clients and servers, though some of them have a set of libraries to make such communication portable across different browsers.

2. Extending HTML with Special Tags

Some suppliers[6] eliminated the requirement of JavaScript programming for application developers by extending HTML with proprietary tags. A special engine (written in JavaScript) must run in the browser at first, and it then intercept all HTML content sent from the server to process these proprietary tags specially. No JavaScript, but exchanging dynamic data between clients and servers is required.

Like other declarative programming, the advantage of such extended tags are not difficult to learn for non-programmers. However, they might become cumbersome and complex if sophisticated logic, such as conditions and loops, is required. Multiple round trips to get a new page is another issue since the engine must run first.

3. Enhancing Existent Frameworks with Ajax

Extending existent Web frameworks[7] to embrace Ajax is another common approach. The results

5 Bindows, WebFX...

6 Backbase

7 AjaxFaces, Rubby on Trail...

highly depend on the original architecture. Most of them eliminated the requirement of JavaScript programming. Developing additional servlets to exchanging dynamic data between clients and servers is required.

After examining closely, these solutions, better or worse, are mainly solving the first challenge: eliminated the JavaScript hassles. The asynchronous communication between clients and servers remains, more or less, the duty of application developers.

ZK: Simple and Rich

In Potix, we believe Ajax is more of architecture than technology.

In 1994, we developed an infrastructure, inspired by zApp and OWL, for developing an accounting system for Windows. In 2000, we developed another infrastructure, inspired by Struct and WebWorks, for developing an ERP system for J2EE. After coaching and watching the development of these systems, we found that not only the Web version required much higher skills and prerequisites to develop, but also its total cost is four times more than the client/server one. Worse of all, the user-friendly reminded us the age of green terminals, though the look, after decorating with proper images and CSS, is modern and fresh.

We start wondering whether it is intrinsic, or the programming model is simply inadequate. Looking back the success of desktop applications in 1990s, the event-driven, component-based programming model is the corner stone of all excitements. Being blessed with the ease to learn and develop, it is the standard and best way to handle interactive and responsive user interfaces. Can we apply this model to Web applications?

In response to this challenge, Potix has developed technologies and tools that enable Web development to have both the same rich user experiences and the same simple programming model that desktop application developers enjoyed. The core of the Potix solution is ZK, which includes an Ajax-based engine to automate interactivity, a rich set of XUL and XHTML components to enrich usability, and a markup language to design user interfaces without programming.

With ZK, you represent your Web applications in feature-rich XUL[8] components, and manipulate them by listening to events triggered by users, as you did for years in desktop applications. Meanwhile, your art designers, by use of a markup language called ZUML[9], weave your fantastic user interfaces without programming as straight-forward as authoring Web pages with HTML.

At the heart, all your application codes are running at the server. Whatever events user trigger are automatically sent to your application running at the server. Whatever you alter components running at the server are automatically updated to the visual representation at the browser. It is just like you don't care how GDI communicates with the display card, when developing desktop applications.

8 http://xul.sourceforge.net/mozilla.html
9 ZK User-interface Markup Language.

No more terminal-like frustrated user interfaces. No more JavaScript headache. No more asynchronous hassles. No more replication of business logics at clients.

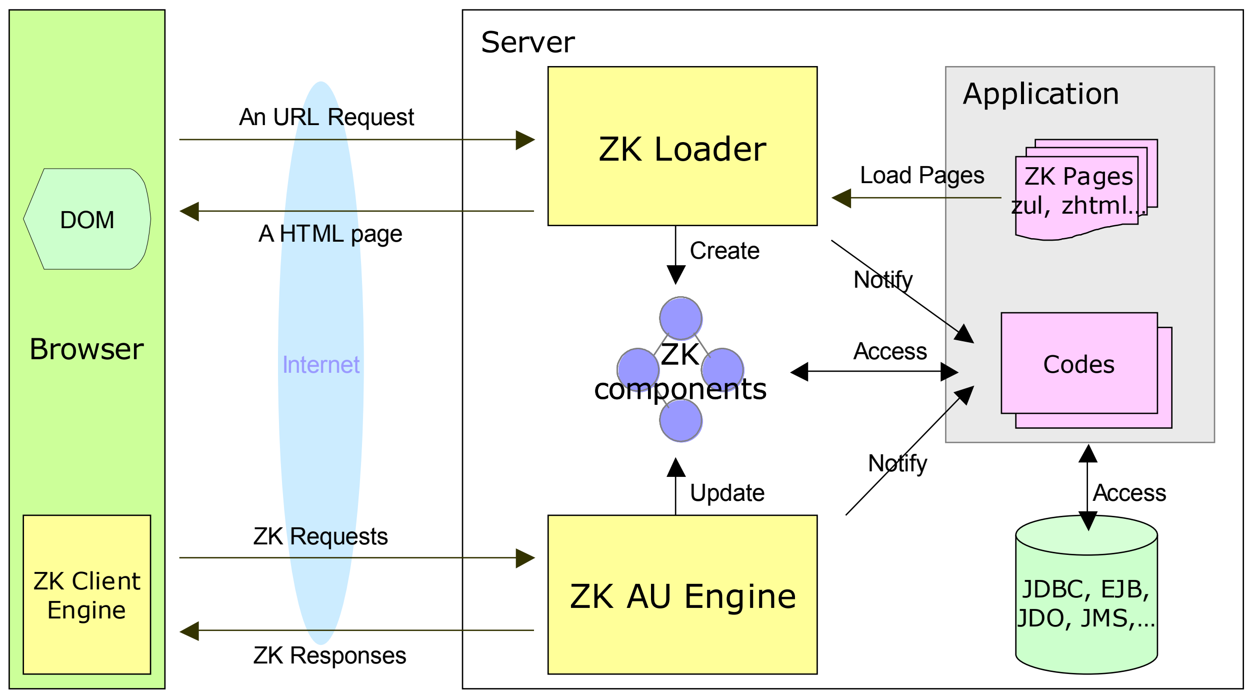
Seamlessly Evolution

As the rich user interface enabled by Ajax is changing the way we interact with the Web, how to preserve existent investments on Web applications has become more important than ever. To ensure to work with existent technologies, ZK is made to a pure presentation tier solution. Logic tier and data tier are both intact. Moreover, all codes of applications run at the server, and there is no need to apply RMI, RPC or Web Services, though not preventing from using them, either. All your middleware utilities work as they used to, such as JDBC, Hibernate, Java Mail or JMS.

ZK could co-exist with portals, JSP, dashboards or any other technologies. You could make any portions of your page highly interactive by simply including ZUML pages. ZUML page could include any kind of servlets and pages including another ZUML pages. You have the total control upon steps and speed in the process of making your applications communicating your customers more effectively.

If you preferred not to change the technology, say JSP, used to generated a HTML page, you could apply a ZK filter and add ZUML tags to it. Then, the filter will translate the dynamically generated page as if it is an static ZUML page.

Architecture Overview



The core of the Potix solution is ZK, which includes an Ajax-based mechanism to automate interactivity, a rich set of XUL-based components to enrich usability, and a markup language to simplify the development.

Ajax-based Mechanism

The Ajax-based engine consists of three parts: ZK Loader, ZK AU Engine and ZK Client Engine. The ZK Loader, loads ZUML pages[10] and render them into HTML pages in response to URL requests.

The ZK Client Engine is running at the browser, and the ZK AU Engine at the server. They act as pitcher and catcher. They deliver events happening in the browser to the application running at the server, and update the DOM tree at the browser based on how components are manipulated by the application.

- Client engine sits at the browser to detect any event triggered by user's activity such as moving mouse or changing a value. Once detected, it notifies AU Engine.
- Upon receiving the request from Client Engine, AU Engine updates the content of corresponding component, if necessary. And then, AU Engine notifies the applications by invoking the event handlers, if any.

10 A ZUML page is a file, an input stream or a string written in ZUML.

- If applications choose to change content of components, remove, add or move components, AU Engine send the new content of altered components to Client Engine. Then, Client Engine update the DOM tree accordingly.

Simple Thread Model

An Ajax request might be sent before the previous one has been processed. It created synchronization issue for ad-hoc Ajax applications. With ZK, requests (and events) are pipelined into an independent queue for each page , and event handlers is then invoked sequentially one-event-by-another. Thus, there is no concurrent access ever made to event handlers and components. Meanwhile, ZK processes Ajax requests for different pages simultaneously to maximize the performance.

The advantage is that the application developers need not to worry about threads, synchronization and other concurrency issues. Of course, it doesn't prevent developers from using threads. For example, it is common to use a thread to execute a long operation, while another UI thread to show the progress bar and a button for aborting the operation.

Suspend, Resume, and Modal Dialogs

When processing a request, an application sometimes needs to suspend the processing and wait for a condition being satisfied. For example, show a message and wait for user's confirmation. Due to request-and-response limitation, this is hard, if not impossible, to implement in Web applications.

With ZK event-driven model, developers could suspend an event handler any time and resume it later. Modal dialogs are a common example that utilizes this feature.

```
if (Messagebox.show("Are you ready?", "Ready",
Messagebox.YES|Messagebox.NO, Messagebox.QUESTION) == Messagebox.YES)
      go_ahead();
```

Minimized Network Traffics and Round-trips

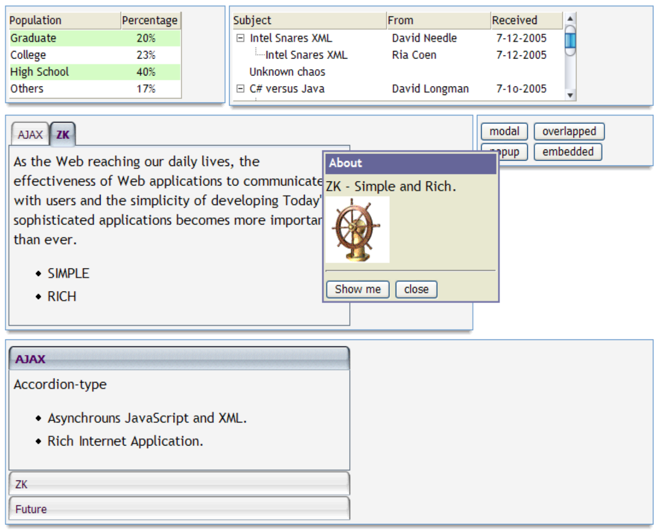
Client-side events are queued before they are really necessary, so multiple events could be sent in the same network packet. In additions, redundant events, such as changing the value of the same component twice, are eliminated.

XUL-based Components

Instead of inventing proprietary components, ZK provides a rich set of XUL-based components. XUL (XML User Interface Language) is a user interface markup language developed to support desktop applications like Mozilla Firefox and Mozilla Thunderbird. Unlike other XUL implementation, ZK components are tuned to have better performance across Internet.

Comprehensive Components

More than 60 components are supported. They include menus, tab boxes, list boxes, tree controls, sliders, group boxes, grids, date boxes and many other sophisticated components.



Population	Percentage
Graduate	20%
College	23%
High School	40%
Others	17%

Subject	From	Received
⊟ Intel Snares XML	David Needle	7-12-2005
Intel Snares XML	Ria Coen	7-12-2005
Unknown chaos		
⊟ C# versus Java	David Longman	7-1o-2005

As the Web reaching our daily lives, the effectiveness of Web applications to communicate with users and the simplicity of developing Today' sophisticated applications becomes more importar than ever.

- SIMPLE
- RICH

About

ZK - Simple and Rich.

AJAX

Accordion-type

- Asynchrouns JavaScript and XML.
- Rich Internet Application.

Component Garbage Collector

The life-cycle of components is managed automatically. The visual part of a component shown at the browser is created and removed automatically, if it is attached to and detached from a page. If it is no longer in use, the garbage collector releases its memory automatically. Managing ZK components is as simple as regular Java objects.

ZK User Interface Markup Language (ZUML)

ZUML is a XML-based markup language for describing the presentation of user interfaces. It is designed to be component-independent, such that developers could use different set of tags, such as XUL and HTML, in the same ZUML page.

Flexible User Interface Description

The simplest way is to use a file to describe an user interface in ZUML. After you copy the file to the proper location, users could use the corresponding URL to access the user face.

The file is called a ZUML page, which is interpreted at the run time when a user requests for it. It is re-interpreted automatically if the file has been modified.

```
<window title="My First XUL">
      Hello World!
      <button label="Say Hi" onClick="Messagebox.show(&quot;Hi&quot;)"/>
</window>
```

Each ZK component is an instance of specific Java class. Like Swing, you could create and manipulate them directly in Java codes.

```
new Label("Hello World").setParent(window);
```

Another example to create components from another ZUML page is illustrated below.

```
<checkbox onCheck="Executions.createComponents(&quot;/dir/another.zul&quot;,
null, null)"/>
```

The ZUML page could be constructed dynamically, too.

```
String content = “<window title=\"Hi\">Hello World!</window>”;
Executions.createComponentsDirectly(content, null, null);
```

With proper authorizing tools, a user could customize a page to have a unique view he likes.

Powerful Script in Java

By leveraging BeanShell, you could embed Java codes into ZUML pages to do initialization or handling events. To access a component in Java codes or EL expressions, you simply use the component's ID you assigned.

```
<textbox id=“what”/>
<button onClick=“what.disabled = true”/>
```

EL Expression

Like JSP, you could use EL expressions in ZUML pages. EL expressions are easier to learn for non-programmers, and somehow more elegant.

```
<?taglib uri="/WEB-INF/tld/pat/core.tld" prefix="p" ?>
<window id=“main” title=“${p:l('app.name')}”><!--Locale-dependent string-->
      What is ${main.title}?
</window>
```

Prototyping and MVC

Embedding Java codes in ZUML pages are mainly for quick prototyping and customization. In production version, it is sometimes better to apply MVC (Model, View and Controller). You could associate a component with your class by the use attribute as follows. The class then acts as the controller to handle child components.

```
<window use=“MyClass”>
      <listbox id=“checks”/>
</window>
```

Event Handlers

To listen and process an event, you could declare the event-specific attributes, say onClick, and methods to specify what to execute when an event occurs. In additions, you could add and remove event handlers dynamically.

```
<textbox id="input" onChange="do_something()"/>
<zscript>
     input.addEventListener("onChange", new DoAnother());
</zscript>
```

Live Data and Separation of Data and View

Some components, like listbox, supports live data. First, it separates the visual representation and the data by an application-dependent renderer. Then, when an item becomes visible (caused by, say, user's scrolling), the renderer is invoked automatically to retrieve and render it dynamically. It makes the application more efficient if there is a lot of data to display.

```
<zscript>
      String[] data = {"A", "B", "C", "D", "E", "F", "G"};
      ListModel strset = new SimpleListModel(data);
</zscript>
<listbox id="list3" multiple="true" width="200px" model="${strset}">
      <listcols>
            <listcol label="Dynamic"/>
      </listcols>
</listbox>
```

Embedded by or Embedding HTML

Today's Web applications are often built on a collection of different technologies, such as portals, JSP and JSF. ZUML pages are therefore designed to work with other Web technologies seamlessly.

First, any HTML page or portlet could include any number of ZUML pages.

```
<jsp:include page="/my/welcome.zul"/>
```

Second, a ZUML page could include any number of servlets including JSP pages and ZUML pages.

```
<include src="/another/servlet"/>
```

If you want to embed HTML tags directly in a ZUML page, the XML name space must be used to distinguish them from XUL tags.

```
<window xmlns:h="http://www.w3.org/1999/xhtml">
      <h:table border="1">
      <h:tr>
            <h:td>
      <listbox>
            <listitem label="AA"/>
            <listitem label="BB"/>
      </listbox>
            </h:td>
      </h:tr>
      </h:table>
</window>
```

In additions, ZUML provides a simpler way if no need to mix HTML and XUL tags.

```
<html>
      <attribute name="content"><![CDATA[
Potix Rich Internet Solution
<ul>
<li>SIMPLE</li>
<li>RICH</li>
</ul>
      ]]></attribute>
</html>
```

Features and Benefits

Features	Benefits
Ajax-based Rich User Interfaces	Interactive and responsive. Users get the same level of use experience as using desktop applications, without any plugin at the browser.
Event Driven Model	Simple and intuitive. The cornerstone of desktop applications in 1990s is sound for its ease to learn and develop.
XUL-based Components	Rich and standard. Fastest way to build rich Web applications with over 60 off-the-shelf feature-rich components. No proprietary components secure your investment against vendor lock-in.
ZK User-interface Markup Language	Straight-forward and zero programming. Zero configuration. Neutral to components: XUL, XHTML or mixed is up to UI designers. Leverage the resources of XUL and XHTML community.
Server-Centric Processing	No replication of business logic at the clients. No asynchronous programming hassles. No RMI nor RPC. Use the same logic and data tiers you are used to.
Script in Java and EL Expressions	Quick prototyping and customization. No compilation. No JavaScript. No DOM. Just POJO (Plain Old Java cOde).
Modal Dialogs	Most intuitive way to interact with users for alternatives and decisions. Decomposing a sophisticated UI into several manageable dialogs.
Simple Thread Model	No thread knowledge and skills required, while the server remains the scalability to handle requests for different pages simultaneously. Yet flexible enough for suspending, resuming and multi-threading for handling sophisticated operations if necessary.
Live Data	Separating view and data, reduced development costs, loading large data effectively, and browsing large data easily.
Mixing Different Servlet Technologies	Evolved to rich Web applications at the speed you preferred. No need to replace or abandon what you have invested.
GPL	No vendor lock-in. Encouragement and beneficiary of global collaboration. Verification from large community and deployment.