



Kuali Student Service System
User Interface Technology Selection

Prepared for:
Kuali Student Service System Board

June 2008

Document Information and Revision History

Document Revisions			
Version	Date	Author(s)	Description of Change
0.1	June 1 2008	Wil Johnson	Draft outline for document
0.2	June 26, 2008	Wil Johnson	2 nd revision

TABLE OF CONTENTS

1 ASSUMPTIONS.....	4
2 USER INTERFACE DEVELOPMENT FRAMEWORK.....	5
2.1 SELECTION PROCESS SUMMARY	5
2.2 COMPARISON MATRIX.....	6
2.3 KEY REASONS FOR SELECTING GWT	8
2.3.1 <i>Rich Interface</i>	8
2.3.2 <i>Cross browser</i>	8
2.3.3 <i>Internationalization support</i>	8
2.3.4 <i>Accessibility</i>	8
2.3.5 <i>Scalability</i>	9
2.3.6 <i>Development</i>	9
2.3.7 <i>Deployment</i>	10
2.3.8 <i>Ease of use</i>	10
2.4 LICENSING CONCERNS	11
3 MODEL-VIEW-CONTROLLER ARCHITECTURE WITH GWT	12
4 PORTAL INTEGRATION	15
4.1 AJAX/GWT CONSIDERATIONS FOR JSR168/286 PORTLETS	15
4.1.1 <i>Session management</i>	15
4.1.2 <i>RPC proxying</i>	15
4.2 CONCERNS ABOUT APPLICATION SIZE AND COMPLEXITY	15
4.3 CONCERNS ABOUT TIGHT COUPLING BETWEEN KUALI STUDENT UI AND JSR168/286 PORTLET SPECIFICATION	16
4.4 RECOMMENDED PORTAL STRATEGY	16
5 CONFIGURATION AND CUSTOMIZATION.....	17
5.1 STYLING	17
5.2 VALIDATION	17
5.3 INTERNATIONALIZATION AND LOCALIZATION	17
5.4 COMPLEX MODIFICATIONS.....	17
6 APPENDIX A	18

1 Assumptions

The Program Charter of Kuali Student commits the project to supporting a rich user experience. Much of the Kuali Student user interface will be student facing. As such it will need to meet the expectations of the millennial generation.

To quote the Charter (Appendix C section 1.1): “The understanding of user experience and mechanisms for enriching user interactions will evolve over the next few years...the standard repertoire of well-understood actions will expand, as will users’ expectations for personalized options, meta information, and general richness of the interfaces they engage. It is our intention that the user interface of Kuali Student will stay current with these developments”

In addition to this general conceptual framework, the Project Charter also commits the project to supporting accessibility and internationalization.

These basic assumptions focused the user interface team on finding the base toolkit for developing rich Ajax user interfaces (that supported both accessibility and internationalization).

The decision to implement a rich user interface using Ajax also forced the team to re-think the way in which Kuali Student integrates with a Portal.

2 User Interface Development Framework

2.1 Selection Process Summary

With limited time to select a UI toolkit, an initial pass was made to select a "short list" of libraries for an in-depth evaluation (see Appendix A for a quick survey of the entire Ajax toolkit space). The primary focus was on:

- Ease of development including:
 - Coding
 - Debugging
- Unit testing
- Cross-browser compatibility
- Internationalization (i18n)
- Accessibility
- Performance
- Scalability
- Extendibility
- Sustainability
- Over-all design

The Google Web Toolkit (GWT) feature set that led to the selection is listed in the table below.

2.2 GWT feature selection matrix

Feature	GWT
Ease of use	Easy
Cross browser	Yes
Internationalization	Yes, excellent
Accessible	Yes, WAI roles are automatically added. Accessibility is a major initiative for GWT.
Requires server-side components	No
Provides server-side components	Yes
Server-side scales well	Yes
JSR168 portlet compatible	Yes, using custom portlet wrapper
Java knowledge required	Moderate to high
HTML knowledge required	Minimal
Javascript knowledge required	Minimal to moderate
CSS knowledge required	Moderate
Compatible with other JS/Ajax frameworks	Yes
Eclipse support	Yes, excellent
Security *	Good
Widget library	Large GWT provided library, many 3rd party libraries available
Build/deploy tools	Ant deploy tasks and Maven plugins
Documentation	Excellent
Post-developer configurable	Possible, but not built in
Extendibility	Excellent
Sustainability	Excellent, backed by Google and community support as well
Community size	Moderate, and growing
Debugging support	Excellent, hosted mode browser allows full debugging via Eclipse
Performance	Excellent
Future proofing**	Excellent

* AJAX applications are insecure by default, but GWT provides assistance in reducing vulnerabilities. See: <http://groups.google.com/group/Google-Web-Toolkit/web/security-for-gwt-applications>

** Because all GWT client code is translated from Java source, future enhancements to GWT result in improvements to GWT applications when they are recompiled.

2.3 Key reasons for selecting GWT

2.3.1 Rich Interface

GWT is aimed at developing complete “web applications,” rather than “web sites.” It provides a robust way to create rich applications that behave in the manner of an actual desktop application, but running completely in the browser as Javascript. This will allow us to create a Kuali Student system meeting any possible user interface requirements, with few of the design limitations of a traditional web application.

2.3.2 Cross browser

GWT is as cross browser as any other Javascript library. It is fully supported under Internet Explorer, Firefox, and Safari, and is mostly supported under Opera. This can be said of most other Javascript toolkits, but with GWT the cross-browser issue is also “future proofed” in that applications can simply be recompiled if a newer, incompatible browser is released, automatically

2.3.3 Internationalization support

Support for internationalization is built into GWT. Some additional functionality was required on the Kuali Student project, but it was simple to implement and was completed in the early phases of the evaluation process. The only remaining internationalization issues (those of operational data vs. configuration data) are actually service layer related, and require no additional work on the part of the user interface.

2.3.4 Accessibility

- Google has a strong commitment to ensuring that GWT supports accessibility. The upcoming 1.5 release includes the most robust accessibility support to date in a web development framework. Kuali Student developers have been using pre-release versions of GWT 1.5 as the basis of the toolkit evaluation.
- GWT is “screen reader friendly,” and provides complete Accessible Rich Internet Application (ARIA) metadata support automatically at compile time, requiring no extra work on the part of the developer.
- Nearly all GWT components are keyboard-accessible by default. The update of the remaining ones is underway, as GWT 1.5 nears release. Robust support for custom keyboard accessibility is provided via a “FocusWidget” mechanism that allows developers to easily implement any keyboard accessibility behaviors required.

2.3.5 Scalability

There are two sides to the scalability issue. The one that generally comes to mind is scalability in deployment, e.g. how many transactions per second, concurrent users, etc can it support? Of equal concern, though, is development scalability. Large projects such as Kuali Student can tend to “fold under their own weight” when a myriad of tools and technologies result in an exponential increase in project complexity. Increased complexity can also lead to a decreased adoption rate for Kuali Student, as institutions may be reluctant to use Kuali Student if their development staff has trouble understanding how it works, or has difficulty modifying or maintaining it. Our goal is to “lower the bar” by reducing the number of required skills, as well as reducing the required degree of skill.

2.3.6 Development

- Nearly all code is written in Java, with a minimal amount of HTML, CSS, and the rare piece of Javascript code. Few developers are expert in all areas of Java, HTML, CSS, and Javascript, so this creates a wider audience capable of developing, enhancing, and maintaining Kuali Student by reducing the number of skills required.
- Code is reusable on both the client presentation layer and the server. This is important due to the inherent lack of security in a rich internet application. The client can never be trusted, so any checks such as validation or business logic performed on the client must be repeated on the server before any transaction is completed. In a traditional AJAX/DHTML rich internet application, these checks must be written twice, both in Javascript for the client and then again in Java on the server. GWT reduces complexity by allowing reuse of code both on the client and server.
- GWT is fully unit-testable, and unit tests are written in the same manner as any other Java unit test (such as JUnit). No additional Javascript test frameworks are required, but they are supported. Complexity is reduced by eliminating additional testing frameworks, and quality can be increased by the use of a cohesive and comprehensive set of automated unit tests.
- As all code is written in Java, code quality can be ensured using standard static analysis tools, style checking, etc. Complexity is reduced by eliminating additional analysis tools and reducing the number of post-compile reports that must be evaluated.
- Compile-time optimization allows developers to focus on writing code that “solves the problem at hand” rather than focusing on writing fast code, or worrying about cross-browser issues.
- GWT focuses on component oriented development. Rather than implementing code reuse via include files, developers focus on writing reusable “widgets” that can be leveraged via aggregation and composition to build complex user interfaces. Custom widgets can be defined at any level of detail, and can be extended via inheritance. An example would be an abstract search widget that provides the majority of the boilerplate for the implementation of a search screen, which could then be extended for use as a Person search, as well as a Course search, with

minimal additional work. Our goal on Kuali Student is to build a comprehensive suite of widgets to facilitate the rapid creation of new user interfaces with minimal work.

- Debugging AJAX applications can be difficult, as it requires stepping through both client Javascript as well as server-side Java. While Javascript debuggers are becoming more robust, even the best are still far behind the capabilities of the Java debugger provided by the Eclipse framework. GWT provides a “hosted mode” debugger that allows a developer to debug their client code in the Eclipse debugger, yielding much higher productivity.

2.3.7 Deployment

- Student information systems tend to experience peak periods of high load, such as registration and add/drop periods. Reducing load on the application server is critical. Traditional web applications can require multiple calls to the server “per click,” resulting in high load on the server. GWT applications run entirely in the browser, with calls to the server only when new data is needed or when performing write operations.
- The GWT compiler produces Javascript and HTML that are highly optimized and compressed, greatly reducing network bandwidth usage. These pages are also statically compiled before deployment, resulting in minimal resource utilization on the server per request when a client loads the application.
- GWT remote procedure calls (RPC) from the client to the server are low overhead asynchronous calls, improving scalability by reducing server utilization.
- GWT reduces overall “round trip” traffic by transparently bundling requests into batch form when appropriate, such as for fetching images. In a traditional web application that has multiple images, a separate request to the server is required for each image on the page. GWT consolidates these into a single request for an image canvas containing all of the required images. No additional server resources are required to create this image canvas, as it is created at compile time.
- GWT implements a robust caching framework that is transparent to the developer. Once an application has been cached by the browser, there is no need for the browser to reload it unless there is an actual change to the application (such as a bug fix or enhancement). When a browser makes a request for the application, a small “stub” is returned which will only reload the application if a change is detected.

2.3.8 Ease of use

- The development model for GWT is relatively simple, when compared to traditional rich internet application frameworks. With a few exceptions (such as inline anonymous classes due to Java's lack of closures), most code implemented in GWT can be easily understood by even an entry level Java developer. Once a base framework/strategy for using GWT is in place, the majority of the development effort focuses on implementing business requirements, rather than technical design issues.

This allows high productivity with a reduced number of expert or senior level developers. A team composed of one or two senior developers and multiple entry level developers would be highly productive, whereas in a traditional rich internet application a higher degree of skill would be required of all developers due to the number of “moving parts.”

2.4 Licensing concerns

Concerns have been voiced regarding Google's licensing of GWT. Recent history shows that Google has a strong commitment to open source, and is continually improving their open source licensing, ensuring that their software remains open. Google has a desire to see GWT become the platform of choice for the development of rich internet applications, and as such is unlikely to close the source or move it to a more restrictive license.

Admittedly there is also a minor issues connected with 3rd party components that are bundled with the standard GWT distribution. The only potentially questionable component of this nature is “Browser Detect”, a small “snippet” of Javascript (approximately 100 lines of code) that is sent to the client to detect the browser name and browser version, which is then used to load the appropriate version of the compiled GWT page. It is licensed under the Creative Commons. However, no method of attribution is specified and so Kuali Student is not affected.

Concerns about licensing apply equally to all Ajax frameworks. Many of the popular Javascript frameworks are primarily written by individuals or small groups of primary developers, with additions made by others. Sometimes, such as in the recent case of ExtJS, the developers may decide to move to a more restrictive license in order to promote their commercially licensed version. ExtJS recently moved to a GPL license, resulting in the need for users of ExtJS to either license their own software as GPL, purchase a commercial license, or to continue to only use older versions under the previous license.

In addition, GWT is a compiler-based framework. Even if GWT itself were moved to a more restrictive license, the code written using GWT, as well as the compiled Javascript form, would not be affected by the license change.

In the highly unlikely worst case scenario that a licensing change is made such that we can no longer use GWT, all is not lost. Our design focuses on a SOA approach that separates the services from the user interface, so only the interface layer is affected. Because the interface layer is written in Java and then compiled to Javascript, it would be possible to migrate to a different compiler/renderer framework such as Echo3. While this would be difficult, the scenario is highly unlikely, and much less likely in the case of GWT than in the case of many other Javascript frameworks.

3 Model-View-Controller architecture with GWT

3.1 Summary

Google Web Toolkit (GWT) is a departure from standard web design frameworks, and as such it requires a different approach to the separation of concerns provided by standard MVC frameworks. Web applications are typically designed around a request/response framework, with a clear demarcation between page states. For example, when a user clicks a button labeled "Edit" next to a record, the request is sent to the server, and an "edit" page is displayed. Traditional web MVC frameworks such as Spring MVC are built around this relatively static, document centered behavior. GWT allows us to develop user-event-driven interfaces that are conceptually closer to the Swing MVC model.

GWT applications run entirely in-browser, there is no transition from one page to the next to manage. Instead, the action of clicking the "Edit" button might simply display an edit widget. In addition, an application might be a composite made up of multiple sub-applications each with their own state management.

An additional challenge is the hierarchical nature of the widgets used to build a GWT application. Widgets are self-contained, and by default only have access to other widgets defined within the same scope. Directly referencing widgets within another container leads to tight coupling of components, preventing reuse.

After surveying the emerging MVC frameworks for GWT, it was found that none existed that provided the separation of concerns required for Kuali Student. Consequently, our decision was to design a new MVC framework.

3.2 Design Considerations

3.2.1 Synchronous vs. Asynchronous

The GWT mechanism for remote procedure calls (RPC) requires that all RPCs be asynchronous. This requirement cascades to any code potentially requiring use of a remote service. The solution is to build an MVC framework that is entirely based on message passing, via events. A widget invokes an operation by firing an event to request the operation, and then asynchronously receives a response event containing the operation results.

3.2.2 Component Oriented Design

Extensibility is a key requirement for the Kuali Student project. We have focused on designing reusable components that represent logical portions of the user interface. This enables both reuse and rapid development, as well as extensibility via component replacement. For example, rather than writing the code to display an address in each place it is required, we would build a DisplayAddress component that would be reused globally. This component may contain some logic to conditionally display portions based on certain criteria. An implementor may choose to either modify or entirely replace this DisplayAddress component with one that meets their needs, and the change would take effect globally.

3.2.3 Controller Location and Hierarchy

In traditional web applications, the controller is a central point through which all requests are handled. In GWT, nested components may not have direct access to their controller. This is solved by providing a utility function that allows a widget to find its parent controller dynamically at runtime.

Because GWT applications can be comprised of multiple composite widgets, which in turn can be comprised of further composite widgets, it is necessary to support a hierarchy of controllers, which can conditionally pass messages between tiers.

3.2.4 Event Type Extension

Given that Kuali Student is intended to be a platform on which to build customized implementations, event types must be extensible to allow the definition of new event types and subtypes without modifications to the core of the MVC framework. GWT lacks support for Java runtime reflection on the client, thus it was necessary to build a "generator hook" for GWT which embeds the event type hierarchy information in the event classes at compile time.

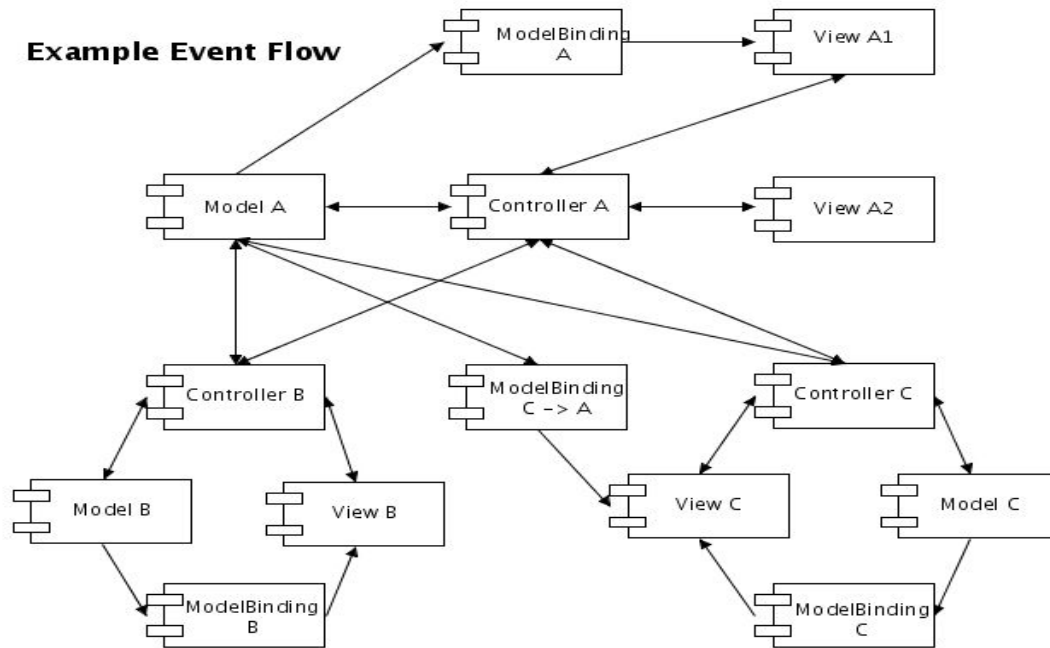
3.2.5 Shared Models

Because nested controllers may require access to the same model information, the MVC framework must support the sharing of models between different controllers. Models are local to the controller by default, but the developer may specify to use a shared model as if it were local.

3.2.6 Model Events and Model Binding

A shared model may be updated by any of its controllers. Rather than require all controllers to listen for all events that may result in an update operation, the decision was made to also dispatch "model events" directly from the model. This way, views may be updated directly from model events originating from outside their own controller. Rather than allow views to directly bind to the model, though, they must implement an interface that marks them as being "model aware." Then, the controller may create a linkage between the view to receive model events via a "model binding."

3.2.7 Example event flow representing 3 controllers



4 Portal Integration

4.1 *AJAX/GWT considerations for JSR168/286 portlets*

There are a few technical issues to be considered when developing AJAX enabled portlets:

4.1.1 **Session management**

The JSR specification(s) dictate that application state be managed on the server, while GWT based AJAX applications manage state entirely in the browser.

- GWT applications must detect any navigation event away from the currently running application, and take appropriate action based on the current client-side application state, either:
 - Prompting the user with a "Are you sure you wish to discard unsaved changes?"
 - Automatically saving the state to the server via a remote procedure call and reloading that state (which may be a partially complete process such as filling out a form) on the next render.

4.1.2 **RPC proxying**

The JSR for portlets does not define whether a portlet implementation must be hosted in the same container as the portal.

- When the portlet is hosted in the same container, or at least exposed via the same host and port, then remote procedure calls work correctly.
- When the portlet is exposed via a different host or port (still visible to the client), and proxied by the portal, remote procedure calls will generally fail due to the "same origin" policy used by browsers to prevent cross-site scripting attacks.
- When the portlet is entirely proxied by the portal, and not exposed publicly at all, then some remote procedure calls may work if the call itself is correctly proxied by the portal, but some difficulties have been encountered dealing with proxy support for synchronous vs. asynchronous HTTP requests.
- The portlet specification defines a method for encoding URLs that should allow for better proxying of RPC requests.

4.2 *Concerns about application size and complexity*

Given the size and level of complexity of the Kuali Student system, many parts of the system may not be well suited for implementation as a portlet.

- Portlet design limitations
 - Portlets are best suited to narrowly purposed applications, such as "Display My Grades," vs. an "Enrollment Management" portlet.

- Portlets can be aggregated by a portal, but composition is limited to a relatively flat structure.
- Lack of flexibility in creating logical application flow, as there is no "next page" concept with portlets.
- Complex operations can be difficult to perform given limited the "screen real estate" of a small portlet window.

4.3 Concerns about tight coupling between Kuali Student UI and JSR168/286 portlet specification

With the rapid growth of both AJAX and non-AJAX alternatives to JSR168/286, such as the OpenSocial API, we have decided not to tie the Kuali Student UI directly to JSR168/286 portlets.

Instead, anything that needs to be deployed in a "portlet style" will be developed with that in mind, but will use an extra wrapper. The wrapper will most likely be a SimplePanel widget that contains the application, and provides environmental access it needs, such as user preferences, etc. This way an application could be exposed both via an OpenSocial application on FaceBook, as well as a JSR168/286 portlet in uPortal, with only a different wrapper used as a connector.

4.4 Recommended portal strategy

The user interface technical team recommends that the Kuali Student system be developed primarily as a suite of reusable user interface components ("widgets") which can be used via composition to form a stand-alone reference implementation. These widgets could then be reused to create small "dashboard" applications that are exposed via a portal, or any other aggregation mechanism such as an existing CMS, or directly embedded within other institutional web pages.

Applications exposed via a portlet or gadget type interface would need to have a limited scope and strictly defined transaction management policy. For example, an address portlet may display an address, allow the editing of an address, and save the address. When a more complex operation is required, portlet or gadget type interfaces should provide a "dashboard view" of information, but link into the Kuali Student system itself for the operation. For example, a portlet may display a student's current class schedule, but to modify their schedule the portlet would take the student to the enrollment screen of the Kuali Student system.

Case-by-case exceptions may be made for complex operations directly within a portal, but these should be the exception rather than the rule, due to the complexity of state management and effective use of screen real estate.

5 Configuration and Customization

5.1 Styling

Styling is done via standard CSS. Widgets will be given a hierarchy of style names to allow customization at multiple levels. For example, a TextBox widget used to display a person's age would be associated with the styles "gwt-TextBox" (the default GWT stylename for TextBox), "KS-TextBox" (a global Kuali Student style for textboxes), and "KS-TextBox-Age."

In this manner, the majority of styling can be easily configured by a web designer, with the normal caveats regarding cross browser compatibility, etc.

An additional feature that will be added later is support for CSS variables and macros. Upcoming releases of GWT will support server-side compilation of CSS based on variables and macros, allowing for easier configuration.

5.2 Validation

Validation will be performed using a metadata driven validator framework. Most validations will be customized by simply changing the validator parameters in the configuration application. Examples include setting the minimum and maximum length for text fields, or setting the begin and end dates for a date field with a specified range.

Complex validation can be added or changed via Javascript. The validator implementation will either evaluate the Javascript natively on the client, or via the Rhino JS engine on the server. This has been mostly implemented, and an expansion of the validator library is underway.

5.3 Internationalization and localization

Internationalization is implemented using a custom "messages" framework that allows for end user configuration of text values. The messages are keyed on locale and a message identifier. When displaying a message, the developer simply fetches the message from the message provider, and it will automatically retrieve the message based on the locale and identifier. The messages can be automatically interpolated against runtime variables, reducing the need for duplicate messages. For example, instead of needing separate messages for "First Name is required" and "Last Name is required," the developer need only define "{\$valueName} is required," and the message will be returned correctly formatted.

Localization of number, currency, and date formatting is done via the standard Java mechanisms such as NumberFormat, and GWT automatically compiles the appropriate Javascript for the client.

5.4 Complex modifications

In some cases the implementor will require customization that goes beyond the scope of the methods above. In this case, the implementor can simply create their own version of a Kuali Student widget, and rebuild the project. While the details have not been finalized, the plan is to modify the GWT build process (via the Generators framework provided by GWT) to detect widgets created by the implementor, probably via a configuration file, and use them in place of the provided widgets to which they correspond.

6 Appendix A

Name	License	Kuali Compatible?
Adobe AIR		
Appcelerator	GNU Public License v2	
BackBase		
Base2	MIT License	
Bindows		
Curl	http://www.curl.com/products_licensing.php	
Dojo	modified BSD license or the Academic Free License version 2.1.	
DWR	Apache Software License v2.	Y
Echo	Mozilla Public License.	Y
ExtJS	http://extjs.com/license	Y
Flex	Mozilla Public License (MPL) http://labs.adobe.com/wiki/index.php/Flex:Open_Source	Y
GWT	http://code.google.com/webtoolkit/terms.html \Apache License, v. 2.0, Creative Commons Attribution 1.0, Eclipse Public License v. 1.0 GNU Lesser General Public License v. 2.1, Mozilla Public License v. 1.1	Y
JavaFX	JavaFX Script will be available via the GPL license	
jQuery	MIT, GPL	
Kabuki	Yahoo Public License (YPL)	
MochiKit	MIT license or Academic Free License, v2.1	
mootools	MIT license	
Nexaweb	http://dev.nexaweb.com/home/us.dev/index.html@cid=1752.html	
Novulo	http://novulo.com/products-pricing.php	
OpenAjax		
OpenLaszlo	Common Public License (Version 1.0) The third party software components that are distributed with OpenLaszlo (and LPS) are released under their own license terms and conditions	Y
Prototype	MIT (source code)	
Qooxdoo	LGPL/EPL dual license	Need to check LGPL with License WG
Rico	Apache 2.0 License	Y

Scriptaculous	MIT License	
Seam		
Silverlight		
Spry (Adobe)	BSD http://labs.adobe.com/technologies/spry/License.html	
ThinWire	GNU General Public License (GPL), GNU Library or Lesser General Public License (LGPL)	Need to check LGPL with License WG
Tibco GI	Open Source BSD License or TIBCO License and Support Agreements	
ULC		
XAP (Apache)	Apache	
Yahoo UI	BSD license	
Zend	New BSD License	
Zk	GPL or commercial http://www.zkoss.org/license/	