OpenLaszlo

An Open Architecture Framework for Advanced Ajax Applications

November 2006

Table of Contents

1.0 On the Cusp of a Revolution in Web Applications

We live with one of the accidents of the Internet boom: The web, a platform intended for the delivery of online documents, now serves as the primary vehicle for the delivery of networked applications. Generally speaking, these applications leave a lot to be desired:

- They deliver an inferior user experience;
- They burden data centers with capital and operating costs by consuming excessive data bandwidth and server hardware;
- They subject developers to inconsistent execution environments and sub-optimal development methodologies.

Thus for providers, consumers and developers of web applications, the experience is less than satisfactory.

Yet the web application model persists because the advantages of web-based application deployment outweigh the flaws.

- web delivery of applications is beneficial for users, because it frees them from the burden of configuring software and administering version updates, and allows them access to programs and data from more than one machine.
- For application providers, web-based application delivery means server-based, centrally managed deployment, which is extremely cost-effective.
- And developers, of course, follow the dictates of the market.

All three of these parties—consumers, providers, and developers—put up with the current state of affairs, but they all want something better. This "something better" is sometimes called the rich Internet application, or RIA.

Java applets were the original attempt to deliver on the promise of rich Internet applications. Applets provide rich, sophisticated client functionality, but client-side Java application start-up and execution were notoriously slow, and inconsistent client run-time execution across operating systems has compromised the original vision of "write once, run anywhere".

Then, after Netscape and Microsoft enabled their browsers to be extended with plug-ins or ActiveX controls, hundreds of downloadable plug-ins were developed. But users have demonstrated a reluctance to install them, and today, outside of the Flash Player and the Java Runtime Environment (JRE), none remain in wide distribution. The lesson here is that any new web application software needs to work in browsers without requiring installation of new or unfamiliar plugins.

Over the last few years, and at an increasing rate, a new approach rich Internet applications has begun to appear. Known variously by terms like "Ajax" and "web 2.0," these solutions, taken together, demonstrate a clear trend towards a radically more usable web than the one we've known.

The prevailing outmoded technology of the web is rapidly giving way to applications that communicate asynchronously with servers that provide data and services in standardized formats – typically XML. Because of its inclusion in all modern browsers, JavaScript has become the default language for procedural programming. Applications built using this suite of technologies are sometimes called "Ajax"—for asynchronous JavaScript + XML.

The term "Ajax" has gained wide currency since the publication, in February 2005, of the paper "Ajax: A New Approach to web Applications" by Jesse James Garrett, of Adaptive Path. Because this paper has become the canonical reference for defining Ajax, we'll assume you've read it. It can be found at: http://www.adaptivepath.com/publications/essays/archives/000385.php .

Ajax is thus an evolutionary development of Dynamic HTML or DHTML, which, according to the Wikipedia "is a method of creating interactive web sites by using a combination of static markup language (such as HTML), a client-side scripting language (such as JavaScript), the presentation definition language (e.g. Cascading Style Sheets or CSS), and the Document Object Model (or DOM)."

Although much exciting progress has been made with DHTML/Ajax, large obstacles remain in the path of would-be web application developers. Perhaps chief among these obstacles is the problem of browser incompatibilities—programs and websites that work fine in some browsers fail spectacularly in others.

As an indication of just how problematic these browser inconsistencies are, consider that in the index to the book Ajax in Action by Crane and Pascarello, the heading "browser differences" has 63 entries, including such things as "attaching callbacks" "IE asynchronous communication" and "security of local filesystem".

The absence of a mature tools ecosystem is a further obstacle to Ajax development. The scaffolding, or infrastructure used in creating client-based web applications is still primitive when compared to that used by developers of server software.

In contrast to the techniques described above, OpenLaszlo, a free, open source platform, was built from the ground up for application development — not "page" development, not "movie" development — and is centered on standard development approaches. OpenLaszlo is a proven technology, sustained by a rapidly growing community that has tens of thousands of developers. Applications based on OpenLaszlo, such as the Pandora music discovery service, and major websites like IBM's, are in use by millions of users.

OpenLaszlo applications are written in LZX, an XML language that includes embedded JavaScript. Unlike many Ajax applications, however, OpenLaszlo applications are portable across browsers. The OpenLaszlo compiler technology takes care of runtime particulars (such as the 63 browser incompatibilities), leaving the developer free to concentrate on the application's behavior and appearance. An OpenLaszlo application can be as short as a single source file, or factored into multiple files that define reusable classes and libraries.

OpenLaszlo is the only Ajax technique that compiles to targets as distinct as Flash and DHTML. The OpenLaszlo platform architecture is designed to accommodate multiple run-time rendering environments.

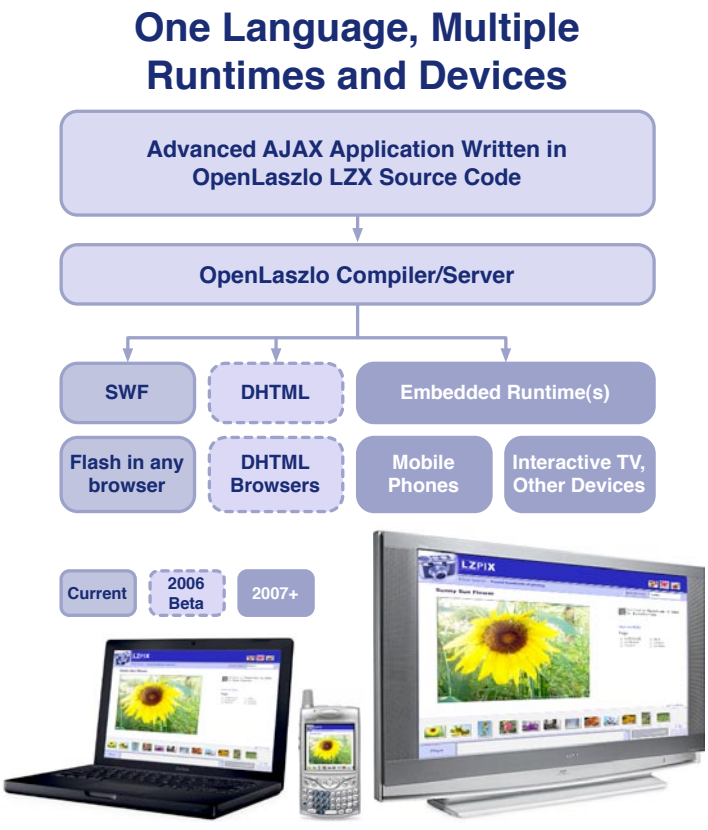

Figure A.

In this white paper we'll explain how OpenLaszlo, an open source Ajax platform, fits into the architecture of the newly emerging web. We will show why OpenLaszlo is an excellent strategic choice for building full-featured, best-of-breed web applications. In order to do that we'll have to explain what a full-featured, best of its kind web application is, and why OpenLaszlo is a superior platform for building them.

So, before proposing OpenLaszlo as a solution, let's consider what a good web application framework would look like—to the consumer, provider, and developer.

2.0 Aspects of the Emerging Web

Above, we talked about web applications from the points of view of three constituencies:

- users – people who interact with the application
- developers – usually teams that include designers and software engineers
- providers – companies responsible for hosting and deploying the application

Here's a closer look at what's important to each of these groups. We'll use these criteria to evaluate OpenLaszlo and various alternative technologies.

2.1 What Consumers Want: a Seamless, Pleasant Experience

People who use web applications want, at the very least, an experience that is at least as good as they get with applications that are native to the desktop. And they are coming to expect an experience that is much better than using desktop applications—because web applications, after all, are backed by the seemingly infinite resources of the Internet. In particular, users want:

- Fast response
- Graphical interfaces that change state with visual continuity rather than through disorienting page refreshes
- Seamless integration of data, media, animation
- Support for real-time interaction and data retrieval

The clichéd term surfing the web is actually not a bad metaphor for how users relate to the best web applications. Surfing is fluid, continuous. There is an element of serendipity to it. The waves guide you, but you decide how you want to ride them. It s fun. And people are increasingly expecting their experience on the web to be like that.

2.2 What Providers Want: Cost-effective, Standards-based Deployment

Software application providers are moving to Ajax technology for three main reasons: it's cheaper to provide, easier to upgrade, and better for their customers than the alternatives of traditional server-based "thin client" applications and "shrinkwrap" software to install locally. Ajax applications are cheaper than traditional web-applications because they shift processing tasks from overburdened servers to underutilized clients; they're easier than shrinkwrap because they eliminate complicated distribution and upgrade procedures. And they're better for their customers because they free customers from software maintenance chores and don't tie them to a single machine, while at the same time providing a more engaging, fluid experience.

There are lots of ways to create and deploy web applications. For maximum benefit to the provider, web applications must:

- Deploy into existing standard server infrastructure
- Run without requiring users to download additional client (plugin) software
- Leverage industry-standard developer skills and methodologies that enable rapid, economical and maintainable development
- Use proven Internet security mechanisms

The Ajax approach meets all these criteria, which accounts for its meteoric rise in popularity.

2.3 What Developers Want: Power and Elegance

In the early days of the web, developers were willing to take a giant step backwards, in terms of programming tools, from modern object oriented languages to HTML and Flash and similar technologies, in order to get the benefit of working on the web. It was a time of innovation and experimentation and any tool at all was better than nothing.

Today, those tools are no longer good enough. Developers want to use modern languages and programming techniques. They want to be able to refactor and reuse code, to use debugging tools, to easily manage versions and releases, to integrate the work of many contributors on the same team. They want even more than that. Rich web applications are a new kind of thing that open up entirely new vistas of collaboration and interactivity, and developers want tools and technology that will that enable them to build the products that they conceive. They want tools that will help them to envision new things entirely.

3.0 Ajax Under the Covers

It's important to understand that Ajax is merely a general approach to solving a problem. It is not a language, or a tool set, or a product. That means that a variety of concepts and platforms and architectures fall under this umbrella term. Let's take a look at some of them.

3.1 Ajax = Asynchronous JavaScript and XML

The XMLHttpRequestObject() API , which was introduced in Internet Explorer version 5, Mozilla version 1 and Safari version 1.2, is the defining capability of Ajax. This call allows XML data to be retrieved by a web page without requiring a page refresh.

Let's start by looking at the meaning of the individual words of the acronym.

Asychronous – As explained in the Adaptive Path paper, "asynchronous" in the Ajax context means that client applications can continue doing useful work even as they await responses from servers. To users, the benefit of asynchronicity is that they spend less time spent waiting for their computers to respond, and there are no disruptive page refreshes when they do. We can call this the asynchronous execution model—in other words, the term "asynchronous" describes the behavior of the application, not the word "JavaScript". We'll see why this distinction is important in a little while.

JavaScript – The term "JavaScript" is similarly imprecise. It refers to any of the various derivatives of the loosely-typed, prototype based language originally devised by Brendan Eich as part of the Netscape browser. Unlike Ajax, JavaScript (or one flavor of it, anyway) is defined by a specification that is maintained by a standards body – the European Computer Manufacturer's Association, or ECMA, whose language ECMAscript defines versions of JavaScript. For example, JavaScript 1.4 corresponds to ECMAscript 262 and so forth. This mapping between ECMAscript versions and JavaScript versions can get quite baroque and confusing and is beyond the scope of this paper. ECMA-262, which is used in LZX, is a widespread standard-compliant variant of the language commonly called JavaScript. In addition, Microsoft's JScript is often lumped in with JavaScript. If you know how to program in one of these languages, learning any of the others won't pose any real obstacles.

XML – XML is a format originally developed for data exchange. XML, which stands for Extensible Markup Language, defines a syntax, which means that it can be used to implement programming languages, of which there are many. One reason that XML is important to Ajax is that web services publish and consume data in XML format (including formats that are built on top of XML, such as SOAP). This standardization makes possible program architectures that separate back-end data processing from the user interface that runs in the client. To a certain extent, it is this concept of consuming and posting data in standardized formats that characterizes Ajax, not the XML itself. With XML you do get some things for "free," such as the XPath syntax for manipulating data.

Some web services publish data in JSON ("JavaScript Object Notation") format, which can be more efficient than XML. A JavaScript client program that consumes JSON data instead of XML is logically Ajax, even if that name is a bit of a misnomer.

Another way in which the concept of XML bears upon Ajax is that some Ajax frameworks, such as LZX, Microsoft's XAML and Mozilla's XUL, use XML to represent the application, not just that data consumed by the application. We'll take a look at them a little later on.

3.2 JavaScript: Toolkits and Frameworks

Years before the Ajax paradigm took root, JavaScript originated as a language for making web pages interactive. Because web pages are implemented in HTML and Cascading Style Sheets (CSS), JavaScript was not designed as a complete user interface language; rather, it was designed to manipulate HTML and CSS entities. JavaScript was used for things like popping up alerts and forms, for validating input. But the language itself had no built-in user interface componentry.

A lack of powerful libraries and reusable components meant that coding a DHTML application was an exercise in error-prone tedium. Furthermore, execution of DHTML was so fraught with inconsistencies between browsers and operating systems that developers were often forced to create multiple versions of production code.

Over time, certain patterns evolved for doing common things. Useful bits of functionality were gathered into libraries, or toolkits. As the term "toolkit" implies, their purpose is to supply miscellaneous ad hoc solutions to common problems. Ajax toolkits, (such as, for example, the Yahoo UI library) add discrete elements of functionality (events, animation, effects, etc) without delivering a cohesive framework for developing highly interactive applications. As such they represent a step towards modern programming methodology, but not a very big step.

As Ajax emerged and developers began doing more and more ambitious things on web sites—that is, as web sites began to evolve into web applications—creators and users of toolkits began to create a more useful programming environment. Meta-toolkits, or frameworks, emerged. Among such frameworks are JackBe, Nexaweb, and TIBCO General Interface.

Such frameworks include much more than miscellaneous classes to decorate traditional web pages. Rather, they provide the scaffolding for developing robust, enterprise-scale applications. For example, the Dojo software stack includes a "package" rubric for managing large libraries, language constructs, an event system, user-interface utilities, and a system for building widgets.

Although such frameworks represent another great step forward, they still each have significant drawbacks. Most such toolkits are immature products, lacking many essential APIs, and even basic documentation.

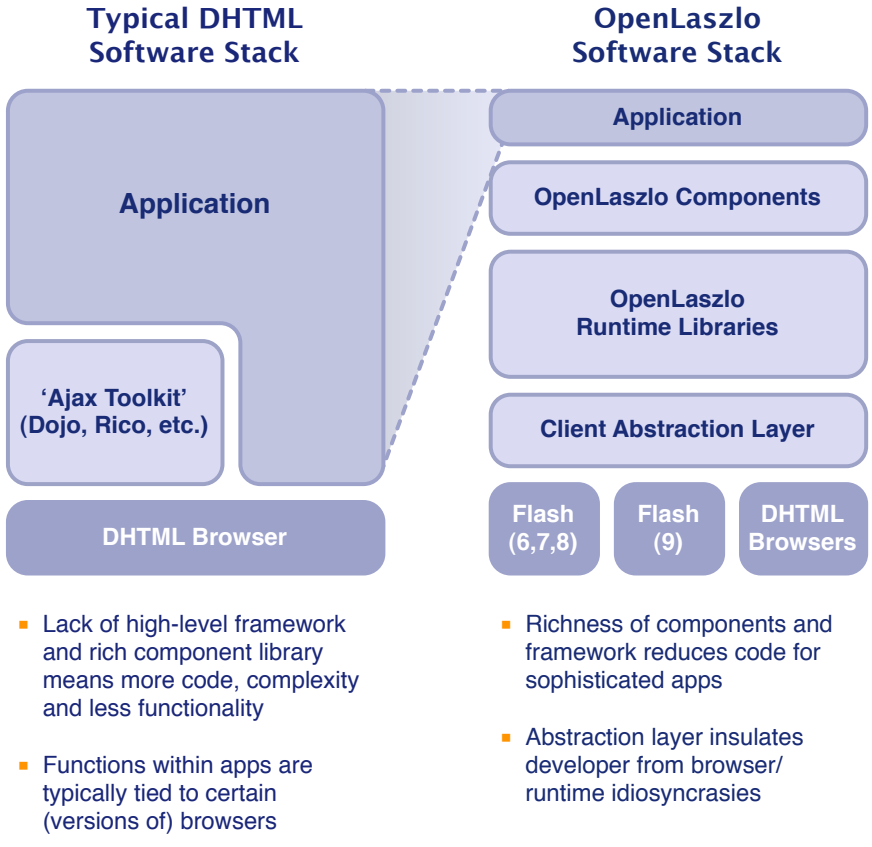

Fig B.

Finally, all of these frameworks are, at their core, JavaScript. They do not help developers to think in terms of the user experience, nor do they shield the developer from chores like direct manipulation of the DOM. JavaScript is a fine language, but it is inherently procedural, whereas declarative languages (like LZX) are simply better for building user-facing applications.

3.3 Don't Mistake the Language for the Runtime

We have said that AJAX stands for Asynchronous JavaScript plus XML. But does that refer to the source code or the target runtime? In order to understand the full implications of the Ajax movement, we need to get a little more precise.

In the model described by Jesse James Garrett in his original Ajax paper, when a user browses to a website, JavaScript code is downloaded from the server to the client and executed in the browser, so that Ajax code is both what is written by the developer and what is executed on the client. Notice that this model conflates the programming language of the source code with the execution environment of the target runtime. There are other possible scenarios.

For example, the Google web Toolkit compiles Java to JavaScript. This allows Java programmers to create Ajax programs without writing any JavaScript themselves.

A similar model is provided by OpenLaszlo, which can compile source programs in LZX to the .swf format (bytecode) to be executed by the Flash Player plugin. In this case the programmer uses JavaScript in the source, but the browser's JavaScript runtime engine does not execute it.

A variant on this model is provided by OpenLaszlo's forthcoming compiler option to generate DHTML output instead of .swf. In this case the developer writes LZX (using JavaScript inside <method>, <handler> and <script> tags) and the program is executed by the browser's JavaScript runtime engine. But the JavaScript code executed is not exactly what the developer wrote. Rather, the OpenLaszlo compiler generates the JavaScript that the browser executes—converting tags to JavaScript, merging that with JavaScript written by the developer, and optimizing the resulting JavaScript output.

In order to properly discuss the benefits and drawbacks of Ajax we need to be clear about whether we're talking about the source language or the execution environment.

4.0 The New Web Software Stack

Let's summarize what we've said so far about the emerging web 2.0 software paradigm and where OpenLaszlo fits into it. When discussing software contexts it's sometimes useful to think in terms of a software stack, in which types of software are placed in terms of their level of abstraction from the hardware. For web applications, the stack looks something like this:

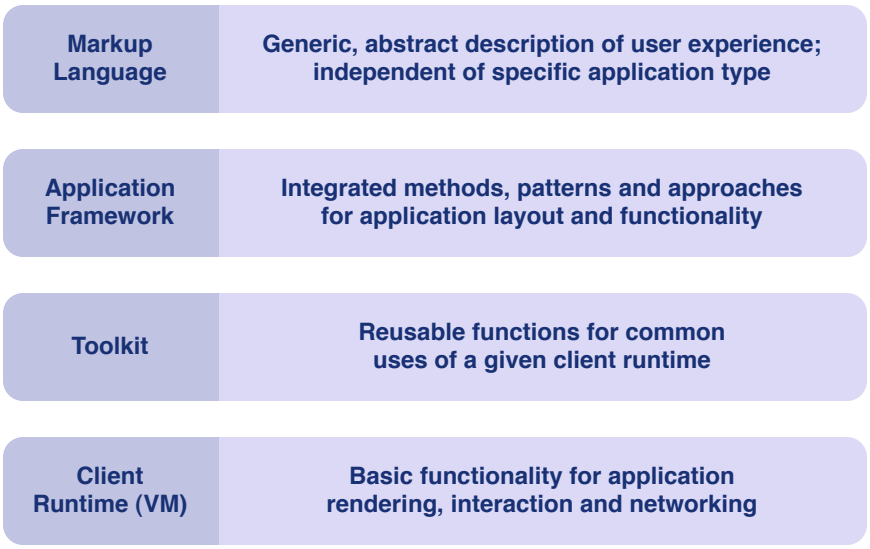

Fig C.

Laszlo Systems, Inc, was founded by experienced software engineers with a clear vision of how to implement this logical stack, and this vision was made real in 2002, with the first release of the Laszlo Presentation Server. At that time the mainstream way to build user applications was on Microsoft Windows. Note that such applications were not web-aware, and there was no declarative markup. Two years later Laszlo Systems took the further step of opening the source to their platform, and OpenLaszlo was born.

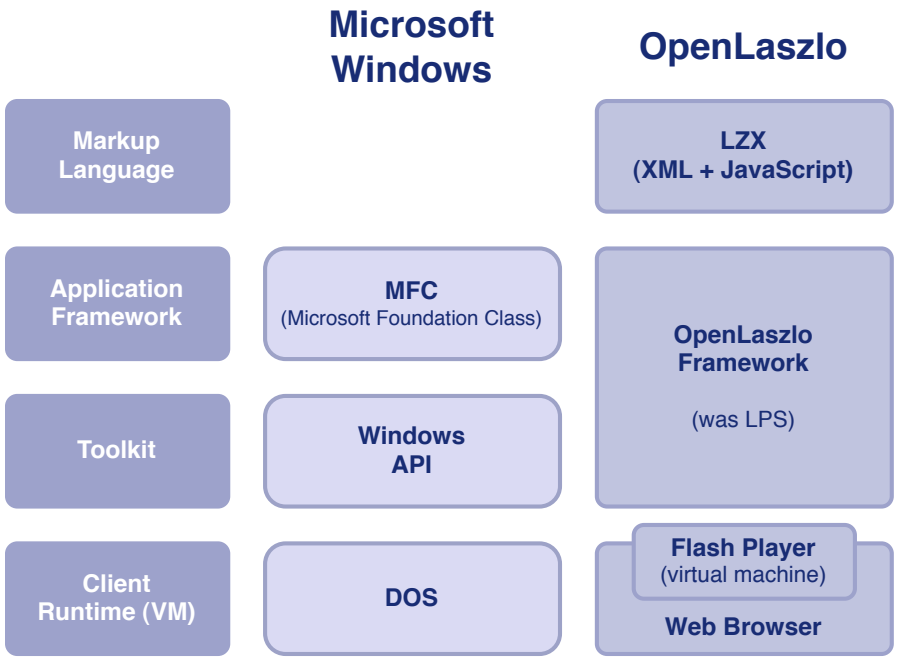

Fig D.

4.1 Runtimes

The runtime is the execution environment that provides the basic functionality for application rendering, interaction, networking. In the context of traditional web applications running on a computer, the runtime is typically a browser such as Internet Explorer or Mozilla Firefox, or a browser plugin, such as Macromedia's Flash Player or a Java Virtual Machine. Many mobile phones and other non-PC devices have similar software stacks. As the web evolves to include more devices of various kinds, presumably new runtime execution environments will appear as well.

4.2 Toolkits

As we've described above, toolkits provide reusable functions for common uses for a given client runtime. There are dozens, perhaps hundreds of JavaScript toolkits available to provide anything from simple checkboxes to complex drag-and-drop behavior.

4.3 Frameworks

Frameworks include integrated methods, patterns and approaches for application layout and functionality. Frameworks implicitly embody some notion of extensibility, such that you can organically create new parts that seamlessly integrate with the whole.

4.4 Markup languages

Markup languages are declarative programming tools that enable generic abstract description of the user's experience independent of application type. We introduce declarative markup languages and explain why they are suited to web application development below.

4.5 Components

Components are interlocking building blocks of large-scale functionality, such as grids, tree controls, windows, and the like. Components generally have a visual representation, although some components may not. For example, a persistent connection manager or a media format transcoder might be thought of as a component despite having no visual aspect.

Trends in web application development confirm the vision that was built into LZX from its inception, but none of the other entrants match OpenLaszlo. Ajax toolkits lack a uniform, full-featured markup language, and MXML and XAML are each limited to a single target runtime.

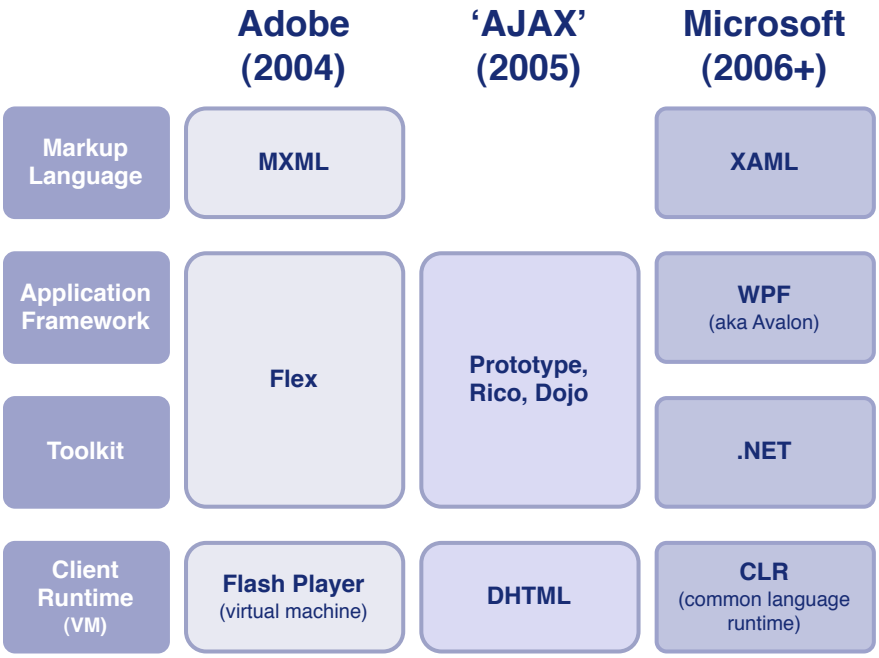

Fig. E

5.0 Declarative Markup and User Interfaces

LZX is an XML language that is a hybrid of a declarative markup language and a procedural object-oriented language, JavaScript.

A declarative language, as opposed to procedural languages, defines what things are, not the procedures (algorithms) that are used to create them. Markup languages, as the name implies, are analogous to the shorthand used by editors to "mark up" text to indicate how pages are to be laid out –where the paragraphs go, how illustrations are to be positioned, and so forth. The mostly widely known markup language is HTML – the hypertext markup language.

Markup languages are well suited to describing user interfaces because they are concise and intuitive. A single tag, for example, <button> or <window>, can be used to program a complex object that has complex behavior, and the hierarchical structure of the code hints at the visual structure of the application.

So, for example, the code snippet

```
<window>
  <button text="hello"/>
</window>
```

programs a button, labeled "hello", inside a window.

5.1 LZX: An Object-Oriented Markup Language

LZX is an object-oriented markup language for describing rich client applications that interact with backend data and services. It is designed and optimized for describing client applications regardless of the target runtime in which they will be executed.

LZX includes primitive operations to implement animation, databinding, constraint-based layout, and other typical behaviors. It's easy to do things like drag-n-drop and translucencies; in fact as the LZX developer community grows people are discovering new innovative ways to structure user interaction.

LZX's syntax and semantics are immediately familiar to developers who have experience with traditional web application development. LZX uses XML tags to describe the structure of an application and JavaScript for client-side logic. LZX APIs provide animation, layout, data binding, event handling, and server communication. There is a large set of pre-built components and widgets that implement most elements of traditional user interfaces. An OpenLaszlo application can be as short as a single source file, or factored into multiple files that define reusable classes and libraries.

Here's the code for a short application that uses the <grid> component:

```
<canvas height="250">

  <dataset name="weatherdata" request="true"

  src="http://www.laszlosystems.com/cgi-pub/weather.
cgi?zip=10022"/>

  <grid datapath="weatherdata:/weather"`contentdatapath="forecast/
day"/>

</canvas>
```

and here's the result:

label	imageurl	desc	temp
TODAY	http://www.srh.noaa.gov/	Mostly Sun	Hi 65°F
Tonight	http://www.srh.noaa.gov/	Chance Rai	Lo 50°F
Wednesday	http://www.srh.noaa.gov/	Breezy	Hi 60°F
Wednesday Night	http://www.srh.noaa.gov/	Mostly Clea	Lo 38°F
Thursday	http://www.srh.noaa.gov/	Breezy	Hi 58°F
Thursday Night	http://www.srh.noaa.gov/	Mostly Clea	Lo 47°F
Friday	http://www.srh.noaa.gov/	Breezy	Hi 63°F
Friday Night	http://www.srh.noaa.gov/	Breezy	Lo 50°F
Saturday	http://www.laszlosystems	Partly Clou	Hi 63°F

Fig. F

LZX affords fine-grained control for power users and complex constructs for ease of development. As an object-oriented, XML language with hundreds of APIs, scores of extensible components, and powerful libraries, LZX provides an ideal foundation for teams of serious developers.

LZX "All The Way Down"

All OpenLaszlo components are written in LZX, and of course the source is freely available. This means that you can extend or modify any OpenLaszlo class without learning a new language or programming paradigm. Also, these UI components are portable between runtimes.

OpenLaszlo's Object System

LZX is a robust object-oriented, prototype-based language that provides inheritance, polymorphism and encapsulation. LZX makes it easy to do "instance first" development in which you create a class from an object by the mere use of a <class> tag. Because LZX is based on prototypes, you can modify instances of classes by adding or overriding methods and attributes. As a practical matter, this means that LZX is ideal for rapid prototyping. LZX supports runtime and compile time inclusion of libraries, which facilitates code modularization and makes it practical for large teams to collaborate on source code.

5.2 LZX Compared to other Declarative Markup Languages

In addition to LZX, three notable, similar XML languages are currently in use: The Mozilla Foundation's XUL, Microsoft's XAML, and Macromedia's MXML. At a first glance, they each resemble LZX. Closer inspection reveals significant differences:

XUL – XUL is a declarative markup language that is executed in either the Mozilla Firefox browser, or else in its standalone runtime environment called XULRunner. Like OpenLaszlo, Mozilla Foundation software is free and open source. Like LZX, XUL incorporates JavaScript methods that allow to you include procedural sections in the declarative code.

Unlike LZX, XUL is not a "no download/run everywhere" solution, since it only runs in the Mozilla Firefox browser. (To run XUL applications outside of Firefox you must download a separate runtime.) XUL is not an object oriented language (in the sense that you cannot define and extend classes) and thus does not offer any of the well-proved benefits of object-oriented methodology.

XAML – XAML is a markup language developed by Microsoft for its Windows Presentation Framework. Like LZX, XAML is a fully object-oriented language (in the sense that XAML tags map to .NET classes); XAML programs are usually compiled to .NET classes rather than interpreted by the browser. As a Microsoft language, XAML is supported by a rich set of development tools.

However, XAML is closed-source and runs only on one platform.

MXML – MXML from Adobe (formerly Macromedia) is an object-oriented markup language similar in feel to LZX. It is supported by the Flex Builder developer tool. Unlike LZX, which is "LZX all the way down", MXML classes are implemented in ActionScript, which means that developers must master two programming models in order to extend MXML components. Unlike LZX, which has been architected to allow its porting to other target runtimes, MXML is fundamentally tied to the Flash Player.

Flex 2 is an incompatible upgrade from earlier versions of Flex and requires Flash Player 9.

6.0 OpenLaszlo Architecture

The OpenLaszlo platform consists of a compiler and optional presentation servlet on the server side and set of services and classes on the client side that are available to every running LZX application. This client-side core is called the LFC, for Laszlo Foundation Classes.

The OpenLaszlo Client

The LFC includes things like a view system for visually rendering the application, an event system, a data manager, a timer, a focus manager, and so forth. Every OpenLaszlo application operates in the context of the LFC, regardless of the actual runtime execution environment, whether that be Flash Player 6, 7 or 8 (now) or, (soon) DHTML or Flash Player 9.

When you compile your application, you specify which runtime it will execute in. The compiler

then generates the appropriate code for the target (for example, Flash bytecode or DHTML), along with a "kernel" for that runtime that provides an abstraction layer to insulate you from target-specific details.

The Event System recognizes and handles application events such as user mouse clicks or data returned from the server, thus improving the responsiveness of OpenLaszlo applications relative to conventional web implementations that process such things on the server. OpenLaszlo reduces the processing load placed on the host server by enabling tasks such as client-side sorting, processing, validation, and dynamic display across all application states.

The Data Loader/Binder serves as a traffic director, accepting data streams across the network from the OpenLaszlo Server and binding data to appropriate visual display elements such as text fields, forms, and menu items.

The Layout and Animation System provides OpenLaszlo applications with constraint-based screen layout of interface elements and algorithm-driven animation of interface state changes. This enables developers to easily build dynamic application interfaces with minimal programming. It allows developers to position a variable number of interface elements using relative positioning or absolute pixel positioning. Dynamic layout mechanisms enable simple modifications to such properties as an application's overall size to be intelligently applied by the platform. This simplifies adapting an application to work on screens and devices of different size.

All screen visualizations in OpenLaszlo applications use time-based rather than frame-based animation, and thus transparently accommodate the processor speed differences of various device types.

The Services component provides a number of supporting capabilities, including timers, sounds and modal dialogs.

Currently the compiler is bundled with the OpenLaszlo Server. This means that in order to develop applications you must have Java and a servlet container or web applications server installed on your developmert machines (though not necessarily on your machines used to deploy the applications). The OL Server and compiler are logically separate, however, and they might be decoupled in the future.

The OpenLaszlo Server

When present, the optional OpenLaszlo Server provides additional capabilities, such as transcoding media on the fly from one format to another, proxying requests to back end data servers, allowing access to SOAP and XML-RPC services, and mapping OpenLaszlo objects to Java objects. The OpenLaszlo Server (formerly LPS, Laszlo Presentation Server) is implemented as a Java servlet and runs inside a servlet container or standard J2EE application server running JRE 1.4 or higher, such as Apache Tomcat, websphere or JBoss. OpenLaszlo thus inherits the proven scalability of these application server environments and can run on any OS supported by them. OpenLaszlo supports Windows, Solaris, Linux and Mac OS X server environments.

The OpenLaszlo Security Model

The OpenLaszlo application platform supports the proven SSL security model. Data transmissions across the Internet can be encrypted using SSL encryption over HTTPS. OpenLaszlo applications in swf format execute on the client computer within the secure "sandbox" environment of the Flash Player, and thus cannot write to the local file system or access the client's native environment.

web services and databases used by an OpenLaszlo application can also be secured using a per-user authentication model. This mechanism is used to protect against using the OpenLaszlo Server as a proxy or gateway into secure services or data.

One Source, Many Targets

LZX is designed to be a run-time independent language. Its APIs do not rely on particular features of any particular runtime environment, and its internal data structures (for example, its class system, view system, etc) translate well to different platforms. You can compile your applications for any of several Flash Player formats, and soon for DTHML. Further down the road, according to market demand and community interest, other target environments will be supported. Although Laszlo Systems has not announced plans to port to, for example, Java or .NET framework, there is no reason that that could not be done. Sun Microsystems recently announced its project Orbit, to make possilble compilation of OpenLaszlo applications for the Java(tm) Platform Micro Edition (Java ME). See below for a closer examination of the benefits of the various targets.

Deployment Options: Proxied or Stand-Alone

There two ways that OpenLaszlo applications can be deployed: either proxied by the OpenLaszlo Server, or as free-standing applications (called "SOLO" as explained below).

Most OpenLaszlo applications can be run without being proxied by the OpenLaszlo Server. These non-proxied applications are called Standalone OpenLaszlo Output, or SOLO, applications. They are simply .swf files that have been compiled by the OpenLaszlo compiler. On the deployment machine, all you need is a web server, such as Apache or IIS. In fact, you can even send SOLO OpenLaszlo applications as email attachments. Once downloaded into the client browser, SOLO applications operate without need to stay connected to the server that deployed them.

Solo Applications can transact with back end data sources so long as those sources are based on XML over HTTP, and if media are involved, it does not require transcoding.

Proxied and unproxied OpenLaszlo applications are essentially the same, but have two significant differences:

1) Proxied applications are deployed in source format (.lzx). When you browse to a proxied .lzx application, the OpenLaszlo Server determines if the source has changed since the last time you downloaded it. If it has, the application is recompiled. This is ideal for iterative development: you modify the source and hit the "refresh" button. SOLO applications are deployed as (static) .swf files.

2) Proxied applications are connected to the rest of the Internet through the agency of the OpenLaszlo server. Any connections to other resources on the Internet, such as, for example, a database or art resources, are proxied though the OpenLaszlo server, which handles security and authorization. SOLO applications connect directly to their respective back ends, and may need to be modified to take security protocols into account.

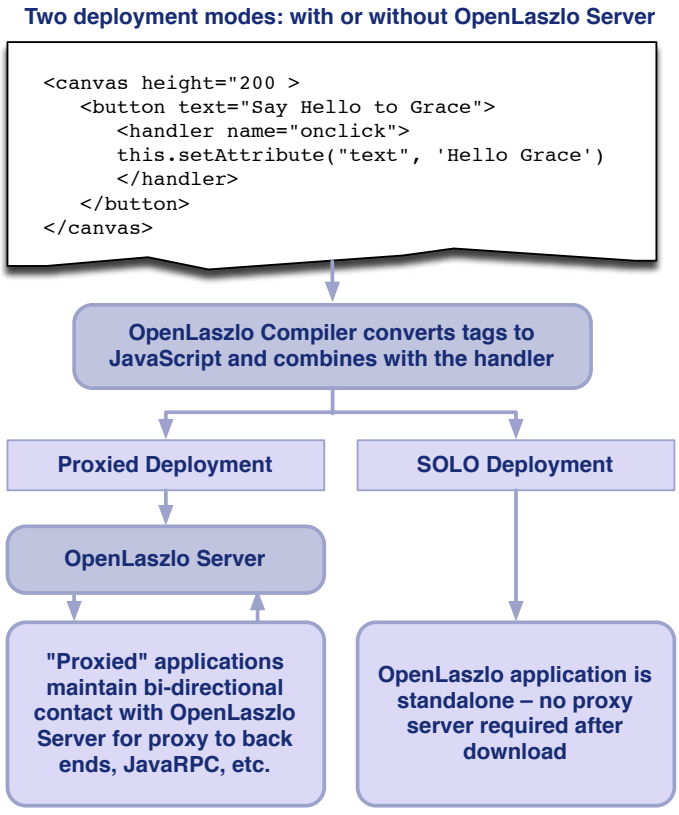

Fig. G

7.0 Flash and DHTML Runtimes: The Choice is Yours

With the 4.0 release of OpenLaszlo, you will have the option of compiling to any of several variants of the FlashPlayer (.swf) format, or to DHTML (Ajax). Here are some considerations to keep in mind when deciding which to target.

Benefits of Flash

Adobe's Flash Player (formerly Macromedia) is an extremely successfully software product, installed on 97% of user desktops. The Flash plug-in is small, highly reliable and consistent across operating systems, browsers and device types. With its built-in support for audio and video formats, it is an excellent platform for media-centric applications (such as, for example, Pandora).

In choosing which version of Flash to compile to (that is, swf6, sw7 or swf8), the main consideration is the tradeoff between market penetration of the player and benefits from the more recent versions. The newer versions of the player will have better performance and new features, but fewer users will have the newer version installed.

Benefits of DHTML ("Ajax")

DHTML is an open standard that requires no plugin. Some organizations stipulate that open source software must be used where available; OpenLaszlo applications compiled to DHTML provide the only such solution for web applications. (Although the Flash Player is free on personal computers, it is not open source.)

Document-centric applications, such as those that include a lot of text and HTML, often perform better in DHTML than in Flash because many browsers have better support for text rendering than Flash does.

8.0 OpenLaszlo Application Development

The OpenLaszlo architecture accommodates the natural separation of the presentation layer from business logic.

Because LZX files are XML, multi-developer teams can use familiar text-format editors, search tools, version differencing and source control systems to build and maintain complex systems.

Developing an OpenLaszlo application is as simple as Edit, Save, Refresh. Use the Eclipse-based IDE4Laszlo or any text editor. The IDE or server automatically compiles the application into a Flash object file, and the browser displays it.

The overall work flow is directly analogous to HTML web page development. To develop an OpenLaszlo application, a developer creates one or more OpenLaszlo application description files, identified by the .lzx file extension. These plain text LZX files reside in an assigned directory for the OpenLaszlo Server, are compiled on demand, and executed when the consumer directs a web browser to the appropriate URL. To test updates to an application description file, developers update the .lzx files, and click the Refresh button of the web browser.

Whenever the URL that points to the source file is browsed, changes in the LZX file are automatically re-checked for valid syntax, re-compiled, re-cached and made immediately viewable in the browser.

The vocabulary and usage rules for the LZX are defined in the OpenLaszlo XML DTD and XML schema. The Interface Compiler references these documents to validate the OpenLaszlo application description (.lzx file) and report any exceptions to the OpenLaszlo debugger.

Unit Testing

In order to foster robust engineering, OpenLaszlo includes a complete unit-testing API based on the W3C unit testing specification.

Powerful Debugger

The OpenLaszlo debugger, which can be compiled into any OpenLaszlo application and also used remotely, offers an "eval" capability that can be used to query and modify the properties of a running application. For example, you can easily check variable values or invoke object methods. The debugger has trace and monitor options, and includes extensive formatting capabilities.

Size and Speed Profiling

The OpenLaszlo platform includes a size and speed profiler which can help you identify performance bottlenecks and areas for potential optimization.

9.0 The OpenLaszlo Project

The Open Ajax Initiative: In February, 2006, Laszlo Systems joined with prominent computer industry vendors and Internet-based businesses including BEA, Borland, the Dojo Foundation, Eclipse Foundation, Google, IBM, Laszlo Systems, Mozilla Corporation, Novell, Openwave Systems, Oracle, Red Hat, Yahoo, Zend and Zimbra in forming the Open Ajax Initiative. The purpose of this effort easier for an open-source community to form and popularize Ajax. They intend to promote Ajax's promise of universal compatibility with any computer device, application, desktop or operating system, and easy incorporation into new and existing software programs.

OpenLaszlo is a mature, robust open source Ajax technology. Currently at release 3.3, OpenLaszlo has been used by tens of thousands of developers, and the LZX API is mature and extensive. There is a wide assortment of data-backed components, and OpenLaszlo applications are in use by millions of users.

OpenLaszlo is free:

- Free to develop
- Free to deploy
- Free to bundle

OpenLaszlo is developed in the open.

- Open Sources, including nightly builds, are publicly available for download.
- An OpenLaszlo contributor process provides legal protections for contributors
- OpenLaszlo plans and roadmap are regularly updated.
- Open JIRA bug report/feature-request/task tracking system is open to all who register for it.
- OpenLaszlo's extensive documentation is freely available, as are tools for users to build their own docs.
- An active forum fosters self-learning among the community, and there are active mailing lists.

The OpenLaszlo user community is large (hundreds of thousands of downloads), active, worldwide, and rapidly growing. Its forums, mail lists, and user groups foster community learning and problem solving, and the community is actively involved in testing and contributing code back to the platform itself.

Licensing

The OpenLaszlo platform is released under the OSI-certified Common Public License. OpenLaszlo uses the CPL without modification. IBM authored this license for general-purpose use by other companies wishing to release their products under an open source license.

The Common Public License is a 'reciprocal' license that provides maximum flexibility for commercial reuse of source code, aims to protect users against broad liability claims, and requires that modifications to the source be made available under terms compatible with this license. It is not a "viral" license like GPL, and applications developed on top of OpenLaszlo do not automaticallly themselves become open source.

Laszlo Systems chose the CPL because it is a simple, clear and well-written license that allows customers great flexibility in building on the OpenLaszlo platform, such as permitting proprietary applications written using OpenLaszlo to remain proprietary, but ensures that any contributions to the OpenLaszlo platform itself remain open source and are "given back" to the open source community. OpenLaszlo is developed by contributors worldwide.

Professional Open Source

OpenLaszlo is fully backed by Laszlo Systems, Inc. The platform's original creators lead the project. Laszlo Systems employs a full-time OpenLaszlo development team, including documentation and quality engineering. The company offers a wide range of training and support options.

10.0 Finding the Sweet Spot: Designing and Delivering Optimal User Experience

> *"The biggest challenges in creating Ajax applications are not technical. The core Ajax technologies are mature, stable, and well understood. Instead, the challenges are for the designers of these applications: to forget what we think we know about the limitations of the web, and begin to imagine a wider, richer range of possibilities."*
>
> *– Jesse James Garrett*

User-centered, web-based applications are a new kind of thing under the sun. Leveraging both the limitless power of the Internet and the endlessly increasing computational horsepower of personal machines, such applications must and will evolve new modes of interaction, new metaphors for computing. People ("customers", "clients" "users") are coming to expect not simply disparate software "applications" that help them to do chores, but an integrated "digital life experience" that seamlessly blend work and play and enhance their lives. Developing these new software experiences will require developers to acquire a new skill set; a new way of thinking. And the language they use either constrains or liberates their ability to think in new ways. OpenLaszlo, with its powerful abstractions like databinding and animation built into a simple declarative language, frees developers from tedious housekeeping opens the door to new kind of thinking.

The spectrum of client technologies: Depending on what you're trying to accomplish, there is an Ajax technology that is best suited to your needs. Toolkits provide incredible value-add for incremental improvements to page-based applications. Ajax frameworks can be used to create simple Ajax applications that include, for example, some interaction with a backend database. For complex Ajax applications that are truly rich and powerful, OpenLaszlo is simply the best option.

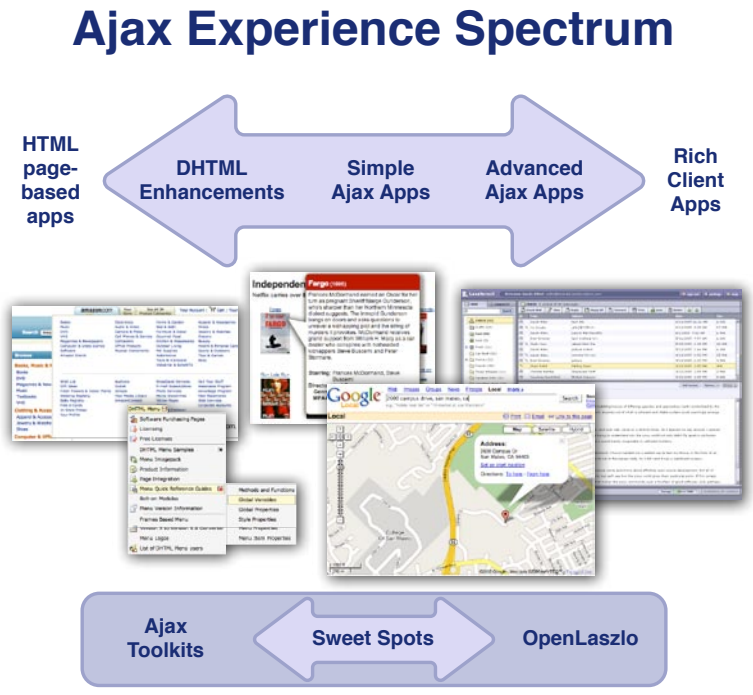

Fig. I

11.0 Explorations in Application Space

OpenLaszlo's compelling technology has unleashed the creativity that had been until recently kept locked inside developers, and applications that would have been hard to even imagine a few years ago have begun to spring up all over the globe. Here is a sampling.

Pandora (http://www.pandora.com) is a program for discovering new music. You type in the name of a song or artist that you like, and Pandora then provides an endless stream of music based on your initial seed and your subsequent tuning. You can create different "stations" with different styles and share stations with others on the internet.

Fig. J

When Barclays Global Investor, one of the world's largest administrators of institutional assets, introduced a new division called iShares, they wanted a tool for their investors that would allow them to compare the return results of multiple index funds, and update the information with real-time stock data.

Laszlo Studios responded with a Rich Internet Application written in OpenLaszlo, that runs in users browsers without special plug-ins, and communicates over standard web ports, allowing the application to be used behind firewalls. The Index Returns Chart application is far more interactive in presenting the information than static HTML pages, and has become one of the most popular features of the Barclays iShares website, helping to generate awareness of Barclays products, and drive traffic to their site.

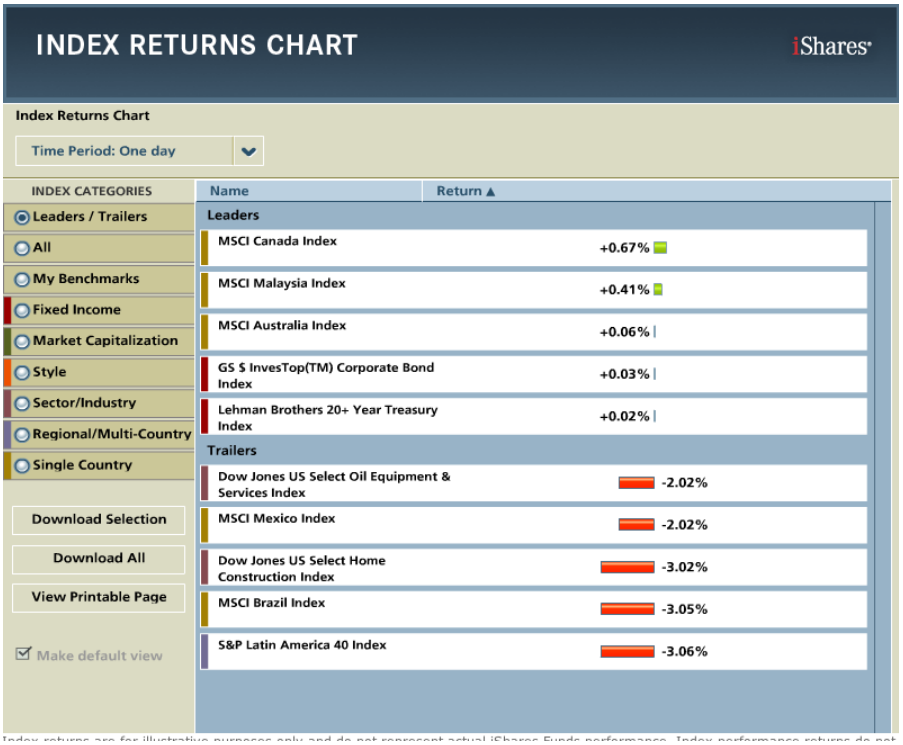

Fig. K

12.0 Conclusion: OpenLaszlo for Maximum Ajax

web applications have arrived. They are no longer "cutting edge"; rather, they are a necessity for any business that uses the Internet to do business. The question faced by businesses is not whether to develop web applications, but which technology to use to develop them.

OpenLaszlo maximizes the prospects for success of any rich Internet application by judiciously leveraging new and existing technologies. It fits into existing hardware and software environments and is easily learned, yet it gives developers the power to create astonishing interfaces. The LZX language meets the needs of professional web/IT development organizations, which include software developers, information architects and graphic designers. It's an ideal tool for rapid prototyping. This means that companies can rapidly build Internet applications that combine the usability of desktop software with the administrative advantages of server-based deployment.

OpenLaszlo is offered under the OSI-certified Common Public License, and is free for development and deployment. It allows developers to create applications with a cinematic user experience, combining the features and responsiveness of client software with the instantaneous no-download web deployment.

In order to minimize the costs and risks of deploying from different code bases for different target browsers, web applications should be developed on a standard platform, and be flexible enough to adapt to changing realities of the web. OpenLaszlo currently compiles to the ubiquitous Flash player, but its architecture is runtime-neutral and will soon support compilation to DHTML. This means that applications developed with OpenLaszlo are universally deployable today, and able to be retargeted in the future.

In order to minimize the risk of a technology's becoming an orphan, web applications should be developed on a framework guaranteed to be around for the long haul. With a robust open source project structure and strong corporate backing, OpenLaszlo is here to stay.

To learn more visit the OpenLaszlo.org website, where you will find complete documentation, self-guided tutorials, user forums, mailing lists, contributor agreements, and much more.

Laszlo Systems, Inc., provides a wide range of training, support, and custom development services.

About Laszlo Systems

Laszlo Systems is the original developer of OpenLaszlo, the leading open source platform for building and deploying web 2.0 applications. OpenLaszlo technology has been widely adopted by application and service providers in the consumer, enterprise, education and government markets. Laszlo Systems provides updates, training and support for OpenLaszlo and offers rich-experience web-based digital life applications such as Laszlo Mail, built on OpenLaszlo.

For more information about San Mateo, Calif.-based Laszlo Systems, visit www.laszlosystems.com.

For additional information, please contact:

Laszlo Systems, Inc.
2600 Campus Drive, Ste. 200
San Mateo, CA 94403

www.laszlosystems.com