Use the best open source client-side framework for cloud computing

Developing cloud-based Software as a Service applications

Dan Orlando  
August 18, 2009

Discover the pros and cons of using Adobe® Flex and OpenLaszlo — two leading client-side rich Internet application (RIA) development languages that are particularly useful for building Software as a Service applications for the cloud.

There is obvious value in learning how to build rich Software as a Service (SaaS) applications. But how do you choose a path to developing the skills necessary for building this type of software? In the past, SaaS development meant choosing a language responsible for rendering the necessary HTML in the browser when a respective URL was called, then choosing a database to store data as necessary. In contrast, modern SaaS application development requires that you make numerous additional decisions, as Figure 1 shows. One such decision is a client-side language or RIA-centric framework. Although there must be communication between the client-side language and the server-side platform, the client side runs independently of the server-side application.
The term *Software as a Service* is in large part synonymous with *rich Internet application*. After all, most current SaaS offerings are built with RIA technology. In fact, in modern-day computing, SaaS can only be successful if it is built using RIA technology because the RIA methodology centers on the concept of user experience design (UXD). Therefore, in the context of this article, I define successful SaaS as that which provides a rich user-friendly experience and, ultimately, allows the user to get more done in less time.

### Evaluation criteria

Each major client-side development framework available for RIA and SaaS has unique costs and benefits. Therefore, you must establish a set of guidelines and criteria by which you can evaluate each one. The important thing is to take an unbiased approach, and consider the facts and figures in your evaluation, rather than going on personal preference.

The first criterion is that the framework must be open source. There has been much debate over whether JavaFX is really open source, and the JavaFX FAQ does not seem to provide a clear
answer. One thing is clear, however: Sun Microsystems speaks about this technology in the future tense, as in "JavaFX will one day be fully open source." Considering its heavy support of open platforms, Sun will most likely make good on this promise. Although JavaFX does take contributions from the community, the technology probably won’t become fully open until 2010.

Join the cloud groups on My developerWorks

Discuss topics and share resources with other developers about reusable and customizable cloud computing architecture, tools, and killer applications in the My developerWorks Cloud computing and My developerWorks Cloud computing central groups.

Not a member of My developerWorks? Join now.

So, this criterion narrows the options to two main contenders: Adobe Flex and Laszlo Systems' OpenLaszlo. Interestingly, both compile down to SWF format, the major difference being that Flex is based on Adobe ActionScript, while OpenLaszlo has a JavaScript/Asynchronous JavaScript + XML (Ajax) foundation. It is important to note that although OpenLaszlo compiles to SWF format, it can also run in standard Dynamic HTML (DHTML)-enabled browsers without having to run from an SWF.

A high-level comparison of Flex and OpenLaszlo is shown in Figure 2. Your objective is to identify which of the four primary client-side RIA frameworks stand out among the rest for SaaS development.

Figure 2. Basic comparison chart of open source development frameworks

<table>
<thead>
<tr>
<th>Framework</th>
<th>Maturity</th>
<th>Developer</th>
<th>Requirements</th>
<th>Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adobe Flex</td>
<td>4.0</td>
<td>Adobe</td>
<td>Flash Player</td>
<td>ActionScript</td>
</tr>
<tr>
<td>OpenLaszlo</td>
<td>4.4</td>
<td>Laszlo Systems, inc</td>
<td>DHTML browser or Flash Player 7, 8, or 9</td>
<td>JavaScript</td>
</tr>
</tbody>
</table>

The additional criteria you use to evaluate these client-side development frameworks are:

- Maturity
- Web services support
- Development effort
- Community involvement
- Software requirements

**Maturity**

Maturity is measured by a combination of the current version of the framework and the length of time it spent in development.
Flex

Development of the Flex framework was initiated by Macromedia in 2002. Adobe inherited Flex when the company bought Macromedia in 2004. The Flex SDK was released to the open source community in 2006 with Flex V2. The beta version of the Flex V4 SDK was made available 1 Jun 2009 and, at the time of this writing, is expected to come out of beta by the end of 2009.

OpenLaszlo

OpenLaszlo started in 2001 with the development of the Laszlo Presentation Server (LPS), created by Laszlo Systems Inc. In October 2004, Laszlo Systems released the LPS source code under the GNU Public License (GPL) and started the OpenLaszlo project. With the release of V3 in 2005, the name of LPS was changed to OpenLaszlo. The current stable version of OpenLaszlo is V4.4.

The leader: OpenLaszlo

OpenLaszlo's development technically began before that of Flex, even though it was under a different name, and Laszlo Systems has consistently stayed ahead of Adobe with its releases. The Flex SDK remains in the beta of its fourth major iteration, while OpenLaszlo has been able to maintain a consistent release schedule — currently in V4.4 — giving OpenLaszlo a clean sweep over Flex, but not by a big margin.

Web services support

Web services support is an important category when comparing client-side development technologies because the supported protocols largely influence the perceived speed of the UI. For example, JavaScript Object Notation (JSON) and Action Message Format (AMF) messages can usually be sent over the wire more quickly than Representational State Transfer (REST)- and SOAP-based services because the data is compressed into binary format. This results in a perceived performance boost because data can be retrieved and processed faster. That said, it is important that a client-side development technology support a variety of communication protocols.

Another thing to keep in mind with regard to communications is that the client-side developer does not always have control over how he must send and receive data. All too often, that's something predetermined either by the server-side developers or, in most cases, by hooking into a pre-existing Web service that only supports a specific protocol. This is yet another reason for the ideal client-side development tool to support any and all existing communication protocols.

Flex

A heavy amount of consideration and development was paid to supporting nearly the entire gamut of Web services protocols with Flex. Support for Adobe's own AMF and Real Time Message Protocol (RTMP) is native to the ActionScript Virtual Machine (AVM), simplifying rapid binary data transfer between Flex applications and any server-side language that supports either protocol. JSON is also supported when you include the AS3CoreLib ActionScript library in the project. It is also worth noting that AMF and RTMP are open protocols, and most server-side languages have support for AMF through the use of plug-ins, libraries, and APIs. Flex also supports standard
HTTP services and XML over HTTP. It is somewhat limited in its internal support for truly RESTful services, but can generally integrate smoothly with just about any communication protocol.

**OpenLaszlo**

OpenLaszlo runs in two modes: **Proxied mode** and **SOLO**. In Proxied mode, the client side is directly coupled to a Java™ servlet and runs inside a servlet container or Java 2 Platform, Enterprise Edition (J2EE) application server such as Apache Tomcat, IBM® WebSphere®, or JBoss. In this mode, SOAP, XML-RPC, and Java Data Transfer Object (DTO) mapping are available. In the second mode — SOLO — the client application runs independently of the OpenLaszlo proxy server. Only data transfer by way of XML over HTTP is supported.

**The leader: Flex**

OpenLaszlo falls sadly short of Flex in the area of Web services support because of its required coupling with the OpenLaszlo Java servlet just to get support for SOAP and XML-RPC. Considering its reliance on Ajax, OpenLaszlo is built on the foundation of the XMLHTTPRequest object for Web services when running in SOLO mode. Unfortunately, the request-response time is extremely slow in comparison to Adobe's binary AMF, which has been adopted by nearly every server-side language. Therefore, I must declare Flex the obvious leader in this category.

**Development effort**

The Flex and OpenLaszlo frameworks implement their own declarative XML-based object declaration language. More specifically, Flex uses MXML to declare ActionScript objects and attributes, and OpenLaszlo uses LZX to declare JavaScript objects and attributes. MXML and LZX use the same base XML object notation — the only difference being the DTD structure and XML schemas used. The core purpose of both is exactly the same: to accelerate the development of client-side applications and UIs.

**Flex**

Flex uses MXML as its declarative markup language, which follows the conventions of basic XML (as do all declarative languages). Most of the time, a Flex application consists of both MXML and ActionScript classes. However, when the MXML files are compiled, the objects declared are converted to ActionScript before being compiled to a single binary SWF file. The code sample in Listing 1 demonstrates MXML markup for a Flex DataGrid component. Interestingly, you can place ActionScript code between curly braces ({}) included inside a set of quotation marks for an object attribute, as is seen with the dataProvider property.

**Listing 1. MXML sample code for the Flex DataGrid component**

```xml
<mx:DataGrid id="peopleGrid"
    dataProvider="{_controller.contactData}" editable="false"
    itemClick="dispatchItemClick(event);" >
    <mx:columns>
        <mx:DataGridColumn id="contact_id" headerText="ID" dataField="id" width="50" />
        <mx:DataGridColumn id="last_name" headerText="Last Name" dataField="last_name" />
        <mx:DataGridColumn id="first_name" headerText="First Name" dataField="first_name" />
        <mx:DataGridColumn id="grid_email" headerText="Email" dataField="grid_email" />
    </mx:columns>
</mx:DataGrid>
```

Use the best open source client-side framework for cloud computing
OpenLaszlo

Remember that OpenLaszlo uses LZX as its declarative markup language, which also follows the same basic principles as any well-formed XML. With LZX, you can place a JavaScript function inside a `<method />` tag and assign that method its own ID. The code in Listing 2 does almost the same thing as the Flex code in Listing 1, creating a grid to display table data.

Listing 2. LZX sample code for an OpenLaszlo grid component

```
<canvas width="800" bgcolor="#eaeaea">
  <bluestyle name="bluecolors"/>
  <dataset name="contactsData" request="true"
      type="http" src="http:contactsdata.xml"/>
  <grid datapath="contactsData:/*" sizetoheader="false">
    <gridtext editable="false" width="50"
        datapath="position()" resizable="false"
        sortable="false"> ID </gridtext>
    <gridtext datapath="@firstname"> First </gridtext>
    <gridtext datapath="@lastname"> Last </gridtext>
    <gridtext datapath="@displayname"> Fullname </gridtext>
  </grid>
</canvas>
```

The leader: Tie

As both technologies continue to mature, the basic principle of continued development for each framework remains the same: to do more with less. In other words, being able to do more in MXML means that less ActionScript code is required. Similarly, getting more done with LZX notation means that you don't need to write as much JavaScript code. This translates to less work for you. In most cases, applications written with OpenLaszlo currently require nearly an identical number of hours as an application written in Flex. This, combined with the myriad similarities between the two platforms, makes it nearly impossible to pick one over the other. In this respect, I declare this one a tie.

Community involvement

As with any open source technology, the level of community involvement is generally a key factor to its adoption, as well as its rate of evolution. A technology that has gained strong community support is not always measured by the number of people contributing to the code base. In fact, community involvement is more easily recognized by the number of "open journalists" out there publishing tutorials and specialized how-to documentation from their blogs for those who are new to the technology. Furthermore, a strong community often has a large group of quality-assurance analysts reporting bugs through the project's bug database.

Flex

Prior to V2, Flex had already gained a substantial amount of attention, largely because of strong marketing efforts after Adobe bought Macromedia. As a result, the Flex community seemed to scale exponentially as soon as the Flex SDK was released as open source in 2006. The size of
the community centered around the evolution of the Flex SDK continues to grow at this same rapid rate.

**OpenLaszlo**

Despite OpenLaszlo being successfully adopted by a number of prolific enterprises, including Pandora, YouTube, and Amazon, the size of the community around OpenLaszlo remains relatively small. Perhaps this has a lot to do with marketing, but there is no questions Adobe has an upper hand in terms of sheer size and marketing budget compared to Laszlo Systems.

**The leader: Flex**

Flex takes the crown in regard to community involvement hands-down. The Adobe news aggregator pulls in 2,370 feeds in the category of Flex alone. Additionally, a bevy of resources is made available to the open community from Flex.org and the online Flex Cookbook, which is like an encyclopedia of Flex "code recipes" that have all been published by the community.

**Software requirements**

This category encompasses any necessary plug-ins or software you must install before you can run an application built with the respective technology. This is generally a huge entry barrier because users are hesitant to trust anything that wants to install itself on their machines — especially when it is unexpected. Furthermore, some users become confused about what is going on when they are told something cannot run without a special piece of software; such users may go as far as leaving the Web site entirely when this happens.

**Flex**

Flex applications run from Adobe Flash® Player, which currently has the highest penetration rate of all software applications and plug-ins. In June 2009, worldwide penetration of the Flash Player had reached a record 98.8 percent, with 97.1-percent penetration in emerging markets. It is of particular interest to note that the latest version — Flash Player V10 — climbed from 55.9-percent penetration in December 2008 to 86.7 percent in June 2009.

**OpenLaszlo**

If there is one thing that makes OpenLaszlo especially interesting, it's that it compiles down to both SWF files that play from Flash Player and DHTML. The timeline of OpenLaszlo development shows that the intention of the framework is to eventually be capable of compiling to any format that runs in a Web browser, which includes Java applets (the same way JavaFX does). There is one area in which OpenLaszlo takes a huge hit, though: At the time of this writing, LZX applications cannot run in Flash Player V10, and it is unknown when this will be possible. OpenLaszlo applications are compatible with versions 7, 8, and 9 of Flash Player.

**The leader: OpenLaszlo**

The fact that OpenLaszlo can be compiled to support more than one file format is in itself quite an accomplishment and differentiates it from all other client-side RIA frameworks. In the case for maximum client compatibility, OpenLaszlo takes the crown.
Conclusion

With all the publicity that surrounds Flex in the RIA community, at first it might seem logical to assert that Flex would wipe the floor in a head-to-head comparison with OpenLaszlo. What I found, however, is that this is not the case by any means. The simple fact that the two RIA development technologies ended up in an even tie suggests that there is no "best" language or framework. Perhaps the moral of the story here is to focus on the needs of the business and allow the development technology to present itself based on those needs.

© Copyright IBM Corporation 2009
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)