Web services have become common place in today's connected world. Companies like Google, Amazon, eBay, Microsoft, Yahoo, and Twitter expose Web services to allow other developers to build on top of their infrastructure. In the consumer world these Web service APIs have allowed for a variety of fascinating "mash-ups" of data. In the business world, Web services have allowed for greater intra-company and inter-company (Business to Business or B2B) communication. What used to be done through TCP, FTP, and custom file formats can now be done with HTTP and XML.

In the early days of the Internet, a group backed by Microsoft started work on a standard called SOAP , a foundational layer for Web services. In the early days when people talked about building or consuming web services they were inherently talking about SOAP-based web services. Through the years SOAP has continued to evolve. The WS-* family of specifications provides greater support for reliable messaging, transaction control, and security. However, the increased functionality comes with a cost. A common criticism of SOAP is the complexity and overhead that come with it. Supporters have tried to counter these criticisms through better tooling and frameworks to abstract away the underlying complexity.

In the meantime, the Web has continued to evolve and a new age of technologies has introduced richer, more interactive behavior to browser-based apps. AJAX, Flash/Flex, and Silverlight (rich internet applications, or RIAs) have redefined our expectations of what browser-based apps can do. Each technology provides the ability to update the user interface without requiring the Web page to refresh. Web Services are key enablers of this capability, but not SOAP-based Web services. SOAP libraries for consuming SOAP-based Web services are either incomplete or altogether missing in JavaScript, Flash, and Silverlight.

As an alternative to the complexity and overhead of SOAP a new style for building web services, called REST , has been advocated. RESTful-based services leverage the capabilities of HTTP through the HTTP verbs of GET, POST, PUT, and DELETE. Data is often transmitted using POX (Plain Old XML) or JSON (JavaScript Object Notation) formats. These services interact similar to the way the browser works with a Web page; as such they are natural fits for use by RIAs.

Some architectural problems do arise in the RESTful world. The browser sandbox tends to disallow cross-domain calls (for example, calling an lds.org web service from another domain is generally disallowed). Most solutions for this issue require actions on the server-side to resolve it. For cross-domain services, the likelihood of a consumer influencing changes on the server-side is often unlikely. The lack of true SOAP support can also be a hindrance, including removing certain types of services as options. The need to support AJAX-enabled Web UIs could begin to dictate software architecture decisions throughout your organization (for example, like not using SOAP, what kind of authentication/authorization infrastructure you can use, etc.).

One of the emerging methods for enabling an RIA's use of services (regardless of their

underlying architecture) is the Proxy Pattern . The Proxy Pattern goes back to the original Gang of Four Patterns book
. It is applied by deploying a service on the RIA's Web host that abstracts the remote implementation (whether it is SOAP, REST, etc.) and exposes it to the client. In some cases, the proxy service might adapt the data format to one that is more palatable to the consuming client. In this instance the Proxy Pattern composts with the
Adapter Pattern
. This resolves cross-domain limitations because the remote service call is made on the server that doesn't have this restriction. Additionally, it enables the use of rich server-side libraries to simplify the communication with different services (like SOAP-based ones).

The current standard at the Church for Web services is SOAP. With the community development efforts that are underway, as well as the ongoing industry shift to richer Web applications, there is a growing need to support clients without SOAP support. The Proxy pattern is one approach to enabling interoperability with Church back-end systems without having the UI technology dictate our back-end architecture.

Bryan Hinton is a senior software engineer for the Church.

Add Comment