Outside-In: Application Interoperability Using an OSID-Based Framework

Adam Franco - Middlebury College

Abstract

This paper describes an interoperability demonstration to be given at OpeniWorld - Europe 2008.

Segue and Concerto are two curricular applications built upon Harmoni, an Open Service Interface Definition-based (OSID) service-oriented application framework. This demonstration will show how website content created in Segue is stored as OSID Assets in Harmoni's OSID Repository. Similarly, the demonstration will show how multimedia assets created in Concerto can be stored in the same repository. Interoperability will be demonstrated as each application is used to view and make real-time modifications to the OSID Assets created using the other application, while at the same time respecting the authorizations given to those assets. Additionally, an OSID Repository to OAI-PMH gateway will be shown providing the LibraryFind meta-search tool with access to the metadata for content created in Segue, Concerto, and a lightweight, read-only OSID Repository.

Background: Application frameworks

When building web applications, software developers have a wide variety of tools at their disposal to help them in their work. Application frameworks give developers a core set of libraries and structures that provide the common functionality needed by many or all applications. This allows developers to focus on the unique utility of their application rather than needing first to build all of the supporting code. As with all technology, individual application frameworks are products of the environment and goals envisioned at the time of their design. These pressures lead to design decisions that open up some doors, while closing others. At the time of this writing, there are more than 25 well-known PHP application frameworks and many more lesser-known efforts. Similar numbers are available for other languages and environments.

During my surveys of available PHP application frameworks, I found that most fit into two general categories. The first of these I think of as the 'nuts and bolts' category. These frameworks provide a wide range of light-weight helper systems for handling things such as input validation, file management, database access, session management, and templating. Moreover, they often include support for model-view-controller (MVC) or action-based program-flow organization. Members of this category include CakePHP, CodeIgniter, and symfony among others.

The second category are what I think of as the 'cms/platform' frameworks. Examples in this category include Drupal and Joomla! While these frameworks may include many of the features of the 'nuts and bolts' category, they generally are designed around building a certain kind of application composed of various plug-ins or widgets.

Building an OSID-based framework: Harmoni

When we sat down to design our new-generation curricular systems in 2003, we wanted to keep the benefits of using a shared framework for all of our applications. While we wanted something richer than what the 'nuts and bolts' frameworks provided, we also wanted to avoiding the limited focus of the 'cms/platform' frameworks. What we wanted was a cohesive set of APIs that would form a portable service and object model capturing the core needs of many systems. The set of services on which we developed Harmoni are those defined in the Open Knowledge Initiative (O.K.I.) Open Service Interface Definitions (OSIDs).

While evaluating the OSIDs, we also looked at several other published service APIs, but found them all to be too heavily focused in one area or another. The Fedora API, for instance, is heavily focused on the repository space with little to say about course-information or other curricular concepts. Other service APIs were strong in the authentication and user-information spaces, but lacked repository support. An attempt at merging multiple APIs would have resulted in conflicts where the the divergent APIs overlapped (such as authentication).

The use of the OSIDs as our service API provided us with a set of subsystems that operate in a unified and consistent way, while also providing clear separation of responsibilities. By their nature as a published API, the OSIDs provide a service model that is compatible with a range of other systems. To date, Harmoni contains OSID version-2 provider implementations of the Agent, Authentication, Authorization, Hierarchy, Id, Logging, and Repository services. In addition to the OSID services, Harmoni includes a number of the 'nuts and bolts' sort of utilities found in other application frameworks which are available for optional use by applications.

An additional benefit of the OSIDs is that they are designed as a very high-level abstraction and not dependent on any particular underlying technologies. This technology independence allows the OSIDs to be implemented in any programming language and to use any technology for data transport. For instance, Harmoni implements the OSIDs as PHP objects; Harmoni clients make PHP-language method calls and receive PHP objects or primitives in response. Another OSID provider may implement the OSIDs as a SOAP endpoint on a web-server; clients would make a SOAP request for each OSID method call and receive a SOAP XML document in response.

The service-oriented architecture of the OSIDs is not to be confused with 'web services'. 'Web services' are a combination of technologies such as SOAP, WSDL, UDDI, and others that provide a transport system for cross-network/cross-language messaging. They define *how* messages are passed, not *what* those messages are. Conversely, the OSIDs define what the object names, method names, method parameters, and method return values are, not what technology is used to implement them.

OSID client applications on Harmoni: Segue and Concerto

Concerto

http://concerto.sourceforge.net/

Concerto is a repository viewer and digital asset management system in the style of ContentDM, DSpace, and Fedora. It allows users to build 'collections' (repositories) of images, audio, video, and other asset types, add and edit metadata, and browse and search the assets in the system. Concerto is used at Middlebury College for managing a number of faculty-created collections of images and multimedia.

The majority of functions that Concerto performs map directly to the objects and methods defined in the Repository OSID. Concerto is simply a web-based front-end that makes calls to Repository providers for all data storage and retrieval. Similarly, authentication and authorization needs are handled by the corresponding OSID providers with Concerto only providing the web-forms for user interaction.

Building Concerto on Harmoni and the OSIDs provided us with a number benefits. Because Concerto is only a user-interface layer and an OSID client, the Authorization, Hierarchy, and Repository providers have all been completely overhauled several times throughout development without necessitating any changes to Concerto. The rebuilt implementations still have the same OSID objects and methods as seen by Concerto, they just return results more quickly.

The second major benefit is that Concerto as a user-interface layer is not tied to any particular OSID provider or providers. With several configuration lines, an additional Repository provider can be added to those available by default. We have used this ability to enable Concerto to browse and search several data-sets stored in various MySQL databases. This was accomplished by configuring Concerto to add a simple read-only Repository provider that translates the rows of database tables into Asset objects.

Segue

A description from the Segue project page: https://segue.middlebury.edu/sites/segue

> Segue is a curricular content management system designed for teaching, learning and research. It is essentially a synthesis of wikis, blogs and traditional content management systems. Segue borrows from the blogosphere the notion of multiple entries or posts on a page, each with their own URL and attached comments. Segue then provides a version history of each post and allows users to link between posts using 'wiki' markup. Finally, like many content management systems, Segue allows users to organize their sites hierarchically and control access by specifying who can discuss, edit or delete any existing content, as well as who can add new content.
>
> Segue is designed for collaboration. Its very granular access control enables Segue to manage individual and group blogs, wikis, courses, e-portfolios, peer review, portals, research collaboration, and personal repositories. Central to Segue is the notion of 'versioned microcontent,' many individual content blocks on a page, each with their own URL, RSS feed, and version history. These content blocks can be sorted in various ways, including reverse chronological or completely custom ordering. Moreover, these content blocks can be organized hierarchically (i.e. top-down) through sets of navigation links and associatively (i.e. bottom-up) by means of tags and tag aggregation.

From the end user's perspective, Segue is a very different sort of system than Concerto, and it is not one whose abilities translate as directly to the OSID Repository service. Rather than the user-interface layer directly making method calls to the Repository provider, Segue introduces an additional abstraction layer broken into three parts.

The first part of this abstraction layer is the 'site engine', a system that translates the site-objects seen by the user-interface layer into the OSID Asset objects that provide the data storage. A Segue website is made up of a hierarchy of nodes: the root 'site' node itself, 'sections', 'pages', and 'content-blocks,' all of which are interspersed with various 'layout-container', 'menu-container', and 'content-container' objects that provide positioning and ordering of the nodes within them. The site engine translates these site nodes into Asset objects, formatting data and passing off modification requests to appropriate methods. The container objects in the site do not map to OSID Assets, but are rather represented as XML data stored in the Asset corresponding to the site-node above them in the hierarchy.

The second part of the Segue abstraction layer is a content plugin system that enables a variety of data types to be displayed in a site. Plugin instances access and store data via their corresponding Asset and provide the user-interface layer with a consistent means of interacting with a range of content such as HTML text, files-for-download, inline RSS feeds, audio, video, calendars, etc.

The third part of the Segue abstraction layer is the commenting system. Comments are stored as OSID Assets attached hierarchically beneath a content-block's corresponding Asset. The commenting system provides comment sorting and display utilities to the user-interface layer.

Building Segue on top of the the Repository OSID, as described above, does add a measure of complexity to the architecture over and above what could be achieved via a custom-storage model, but there are a number of benefits that would be lost without the OSIDs. As with Concerto, the most important benefit so far has been a stable API below which the provider implementations can be improved without requiring changes to the Segue application code. Similarly, the complexity of the hierarchical authorization system, the multiple-source authentication/agent/group management system, and other services is hidden away, greatly simplifying the application code. Lastly, as will be seen, Segue content may be repurposed via direct access to Segue's repository with authorization restrictions still intact.

Segue-Concerto interoperability via OSIDs

As Segue and Concerto both use the OSID services as their primary data access API, basic interoperability between the systems essentially comes along for free. Segue is a specialty OSID client that interacts with certain asset types in a particular way, while Concerto is a more general-purpose OSID client that can be used to view and modify arbitrary types of Assets. With appropriate configuration, each can access data created by the other through the OSIDs. The current goals for using this interoperability are the following:

- Enable placing of assets archived in Concerto into Segue sites.
- Enable searching and retrieval of Segue content through Concerto.
- Enable metadata addition to Segue content through Concerto.

Concerto Assets in Segue
The first interoperability demonstration is the usage of Concerto assets in a Segue site. Because all images and media files in Segue are assets, specifying the id of an asset from one of Concerto's repositories is enough to use it within a Segue site. Segue will look for Dublin Core metadata in media assets and, if the Concerto-created asset has Dublin Core metadata, then that will be displayed in Segue.

While Segue will make use of Concerto assets, this interoperability is not without room for improvement. First, the current version of Segue expects assets it uses to be represented in its local authorization hierarchy. Future versions will remove this assumption and allow access to repositories elsewhere across the network. A second improvement would be the addition of a browse and search panel to allow end users to easily find assets in repositories outside of Segue's local one.

Viewing Segue Assets in Concerto
The second interoperability demonstration is the browsing, searching, and viewing of Segue data in Concerto. As discussed above, the Segue site nodes are Assets and, as such, can be viewed in Concerto. As the logic for displaying formatted rich content is in Segue, Concerto will show any HTML or XML data without special formatting. Images and other files appear as they do when created directly in Concerto.

Modifying Segue Assets in Concerto
The third interoperability demonstration is the modification of Segue metadata in Concerto. Concerto can be used to safely add Dublin Core metadata to Segue site assets, as the structure of the Dublin Core schema is defined in the repository and the data types are restricted. This metadata can then be viewed and searched in Concerto. Segue will display the Dublin Core metadata when it is applied to media assets, but will currently ignore it for navigation and content-block assets. Unlike the Dublin Core metadata, the repository does have built-in rules for Segue-specific data stored in each asset's unstructured 'content' field and will allow modification of the raw Segue data in ways that may be incompatible with Segue.

Improving Segue's repository implementation to disable modification of Segue-specific data by other clients-- while allowing modification of other metadata-- would enable wider editing access to Segue nodes without risking data integrity of the Segue-specific parts.

LibraryFind and the OSID to OAI-PMH gateway
While the OSIDs are the primary interoperability API for objects stored in these systems, they are not the only way to access data. For the past year we have been testing versions of the LibraryFind meta-search tool (http://www.libraryfind.org) developed by the Oregon State University Library. In addition to performing Z39.50 searches of library catalogs, LibraryFind will harvest and index metadata records from the many systems that support the Open Archives Initiative's Protocol for Metadata Harvesting (OAI-PMH).

To enable Concerto assets to be searched and located in LibraryFind alongside records from the library catalog, a DSpace repository, and a ContentDM repository, I wrote an OSID Repository to OAI-PMH gateway. This gateway consists of two parts: OSID client and an OAI-PMH provider. The OSID client periodically iterates through the assets in a repository provider, recording their metadata in a temporary database that is kept until the next refresh. The OAI-PMH provider in turn reads the records out of the temporary database and formats them into XML response documents.

As this gateway reads from the Repository OSID, it can also be used to harvest data from Segue or other OSID-based systems, though this is yet to be tested.