SRW/U with OAI

Expected and Unexpected Synergies
Robert Sanderson
University of Liverpool
<azaroth@liv.ac.uk>

Jeffrey Young
OCLC Research
<jyoung@oclc.org>

Ralph LeVan
OCLC Research
<levan@oclc.org>

Abstract

SRW/U (the Search/Retreive Webservice) and OAI (Open Archives Initiative) are both modern information retrieval protocols developed by distinct groups from different backgrounds at around the same time. This article sets out to briefly contrast the two protocols' aims and approaches, and then to look at some novel ways in which they have been or may be usefully co-implemented. While using SRW as a search service to an OAI repository or aggregated data set is an obvious synergy, there are also many other useful architectures that can be constructed without bending the protocols' semantics.

Introduction

Since the first meeting of the Open Archives Initiative in October 1999 [1], the OAI Protocol for Metadata Harvesting (henceforth just OAI) has flourished in terms of support, technical developments and usage all around the globe. Its beauty lies in its simplicity and focus; it aims to do one thing easily and well: facilitate the sharing of data via a harvesting model. Recently adventurous architectures have been developed and discussed such as the use of complex digital objects in the Los Alamos National Laboratory Digital Library [2]. Others have looked back to OAI's roots in the Dienst protocol and resurrected some of the other functions to include search features [3]. At the end of the day, OAI's primary function and raison d'être is to facilitate the aggregation and distribution of data, and it has been extremely successful in that field.

SRW (the Search/Retreive Webservice), on the other hand, has grown out of the search oriented Z39.50 protocol in an attempt to bring it into line with current expectations and technology. The idea of the useful conjoining of the web and Z39.50 has existed since the web's inception, and about the same time as OAI was created, the same process was starting for the transition of Z39.50 to SRW.

In February of 1996, Eliot Christian posed and provided a well thought out answer to the question "What is the proposal for fitting Z39.50 into the Web?" [4]. Later that year
Sebastian Hammer wrote in a *D-Lib* article:

"...we believe that there is a strong potential for a profitable and synergetic relationship between the WWW and Z39.50. We see the two worlds merging together, with each one growing stronger by using the best elements of the other..." [5]

In November 1998, Eliot Christian brought up the use of XML to encode the Z39.50 operations [6] which was later taken up by Matthew Dovey [7]. In March of 1999, Ray Denenberg proposed a profile for Z39.50 over HTTP [8] and eventually the first official SRW/U [9] (then Z3950, Next Generation) meeting was held in 2001. After a successful experimental release, a stable version of the protocol, version 1.1, was released in February 2004.

These two protocols look set to form a strong, interoperable backbone for the distribution and remote discovery of data for many years to come, so it is worth looking at how they can be used together advantageously. While some combinations are immediately obvious, there are other architectures that have not been discussed and are worthy of notice. It is also interesting to look, briefly, at how the two protocols handle the same fundamental requirements.

### Some Differences and Similarities

#### Too Much Information

An OAI [10] server may decide to split a large response into several more manageable sized chunks. A resumptionToken is then used to signify that there is further data left to retrieve and the client can simply send that token back in order to continue fetching the records. In this scenario, the server makes the decision as to when the data should be split, and must either locally maintain the token such that it may be used again in the future, or somehow encode everything about the segmented operation into the token.

This has both differences and similarities to the current SRW approach to the same problem. Here the client decides how many records it wants to retrieve in a response, up to a maximum limit imposed by the server. Paging through chunks of records is then achieved by incrementing the start position within the same ordered set of records. There are two options in SRW for how to accomplish this. Either the server issues a token to identify the result set (as opposed to a token that continues a segmented operation) or the client must re-issue the query.

The approaches of both OAI and SRW have merit within their own contexts. The OAI approach of segmenting the operation fits the goals of the protocol in that it ensures that every matching record is harvested once, but the approach does not permit skipping chunks of records or moving backwards. SRW's startPosition and maximumRecords parameters, combined with either repeated queries or persistent result sets, allow the client to identify and retrieve exactly the records it requires. The utility of the two different approaches in SRW becomes apparent with large and popular data sets. If Google, for example, were to put up an SRW interface to their data, maintaining the thousands of new result sets created every second with a unique identifier for each might be an impossible burden to bear, making re-issuing the query cost effective. On the other hand, if the context was scientific research, re-issuing a query to a database that has possibly changed since the last time it was issued is not suitable and persistent result sets should be used.

#### Versioning

In SRW version 1.1 a mandatory 'version' parameter was added to the operations and the version number was removed from the namespace for the SOAP [11] binding. In comparison, OAI does not carry the protocol version in the request, even though there are two quite different major versions (1.X and 2.0). Why this difference, and does it matter?
The developers of OAI can be very proud of the fact that the protocol can be implemented on top of a database in relatively little time by a competent programmer. If there were to be a version 3.0 of the OAI-PMH that was not backwards compatible with the current version, and not significantly more complex, it would likely only take programmers a day or two to update from the previous version, at both the client and server end points.

In contrast, the additional features exposed by SRW mean that it has a somewhat higher barrier to entry (although nowhere near as high as its predecessor protocol Z39.50). When a new version of SRW is released it may take quite a while for services and clients to update, if they do at all. In the mean time, if the version were carried in the XML namespace, it would make version 1.1 and 1.2 clients and servers totally unable to interoperate, even if a 1.2 server supported all of the 1.1 parameters, as the SOAP toolkit would reject the request before it got to the application to process. Equally, a 1.1 client would reject a 1.2 response as invalid due to this namespace difference, even if only 1.1 compliant elements were present. SRW's solution to this is to have a namespace that will not change for the lifetime of the protocol, and to put the version information as a mandatory parameter in both the request and the response.

Both protocols have faced the same questions and arrived at different answers as appropriate for their function. OAI segments operations whilst SRW allows the client to negotiate the records at its leisure. OAI only records version in the Identify response, whereas the more complicated SRW includes it in every request and response to permit the interoperation of clients and servers with different versions. Most importantly, even though the two protocols have come to different conclusions, they still work effectively side by side, which we shall now turn to.

**SRW Interfaces to OAI Aggregated Data**

There are many web interfaces to search services being run over OAI harvested data [12], each with their own query syntax and conventions. And while this isn't necessarily a bad thing, as different communities and audiences have different requirements, it is a distinct disadvantage when viewed in the metasearch context [13] or even in the context of a single remote search client rather than a user with a web browser.

The first and most obvious strategy is to allow the data harvested via OAI to be searched via SRW. This model has been described several times and is in use at the Resource Discovery Network [14], The European Library [15] and Ockham [16], amongst many others.

This model is the basis for several large-scale architectures, such as the JISC Information Environment described by Andy Powell [17], and it is recognised by the NISO Metasearch Initiative. In effect, it creates an inverse pyramid where the top layers are the easiest to implement and most prolific. Descending the pyramid towards the point, fewer implementations are required, but they become more complex. If there are 100 OAI providers, these could be divided amongst 10 SRW search interfaces and, at the bottom, one metasearch service that targets the 10 SRW services. Providing data via OAI is straightforward; providing a search interface via SRW is more complicated but still relatively easy compared to the troubles of de-duplication and combined relevance ranking of a metasearch engine.

Herbert van de Sompel and colleagues at Los Alamos have described their multi-layered internal architecture for dissemination of complex digital objects [18] here in past issues. If this architecture were to be duplicated elsewhere, the top level of the pyramid could instead be either handled by fewer surrogate OAI providers, or have access to significantly more resources for the same amount of work by the harvester.

**OAI Interfaces to SRW Provided Data**

We can also turn the hourglass over and run the sand back the other way, building OAI on top of SRW. One of the problems with programming an OAI provider implementation is that
data can be stored in a variety of databases engines, in a variety of storage formats, and with a variety of metadata schemas. Data providers are forced to either find an implementation that happens to match their existing database configuration or adopt a new configuration wholesale. Much too often, the chosen solution is to write a new OAI implementation to cope with local conditions. Despite claims that OAI is easy to implement, though, the world is full of problematic OAI repositories.

OCLC's OAICat implementation [19] addressed this concern by abstracting the database engine, record storage format, and crosswalk mechanism. The problem of developing an application that can access diverse databases isn't peculiar to OAI, however. The ideal solution would be to have a standard search API that can work with any database configuration. For relational databases, SQL is that standard. For practically everything else, now, we have SRW.

Technically speaking, it would be wonderful if the base SRW protocol alone were enough to support OAI. If this were true, someone could set up a gateway service that accepted OAI requests and instantly provide access to the universe of SRW services. But, alas, OAI has some requirements that SRW services in turn are not required to support. These requirements can, however, be satisfied by a minimal profile. To conform to the profile, an SRW service must provide the following features:

1. an oai.identifier index containing a unique identifier for each record in the database
2. an oai.datestamp index containing the date/time the record was added or changed in the database
3. an optional oai.set index, browsable via the scan operation, to support selective harvesting of records
4. an extraRecordData element included in an SRW record container with an oai:header fragment. For example:

```xml
<record xmlns="http://www.loc.gov/zing/srw/">
  <recordPacking>...</recordPacking>
  <recordSchema>...</recordSchema>
  <recordData>...</recordData>
  <extraRecordData>
    <oai:header xmlns:oai="http://www.openarchives.org/OAI/2.0/OAI-PMH.xsd">
      <oai:identifier>...</oai:identifier>
      <oai:datestamp>...</oai:datestamp>
      <oai:setSpec>...</oai:setSpec>
    </oai:header>
  </extraRecordData>
</record>
```

For SRW implementations that don't support the inclusion of an extraRecordData element, an alternative is to provide a separate recordSchema to contain the oai:header result instead. This approach is less efficient since the OAI code must perform two SRW queries instead of one to collect all the information it needs, but it is better than the approach not working at all. The first query would retrieve the header information, and the second query would retrieve the record itself.

Using this minimal profile, any OAI request can be satisfied by transforming an SRW response into an OAI response using a stylesheet, such as implemented by XSLT [20].

- Identify: generated from selected parts of an SRW Explain response.
- ListMetadataFormats: generated from the schemaInfo section of the Explain response.
- ListSets: generated from an SRW Scan of the oai.sets index.
- ListRecords: generated from an SRW Search/Retrieve against the oai.datestamp and oai.set indexes.
- ListIdentifiers: the same as for ListRecords
- GetRecord: generated from an SRW Search/Retrieve against the oai.identifier index.
The only piece missing is the OAI requirement to support an oai_dc metadata format (or as SRW would call it, an oai_dc recordSchema). One solution would be to add this to the profile as well. Since SRW already defines an analogous recordSchema an oai_dc variant might be easy to add. More on this below, though.

Once an SRW service conforms to this profile, there are at least two options for achieving OAI access.

- Local server: Download an open-source OAI implementation [22] that understands the profile and configure it with the URL of the conforming SRW service.
- Gateway server: Register the URL of the conforming SRW service in the SRW Registry [23] that will recognize the existence of the profile and provide OAI access as a value-added service. See below for more information about this and registries in general.

A good example of the gateway option is the OCLC Office of Research Publications Catalog. The SRW/U service URL for this collection is <http://alcme.oclc.org/srw/search/ORPublications>. This SRW/U URL was then added to the SRW/U Registry at <http://errol.oclc.org/srwRegistry.oclc.org/orpubs.alcme.oclc.org.html>.

As mentioned above, one of the features of this registry is that it provides OAI access to profile-compliant SRW/U services. In this case, the OAI baseURL is <http://errol.oclc.org/srwRegistry.oclc.org/orpubs.alcme.oclc.org.oaigateway?verb=Identify>. If you look deep inside this Identify response, you will see a gateway element in the description that reflects the gateway's operation.

**OAI Retrieval of SRW Discovered Data**

In OAI, sets are defined as "an optional construct for grouping items for the purpose of selective harvesting" and the definition continues by saying that these sets are predefined but are otherwise left to design and describe. The protocol does not state how the contents of those sets are defined nor by whom, though typically this is done by the maintainer of the repository. SRW also has the notion of sets, but SRW does define how they are created and by whom. Result sets are created by a client performing a search and are subsequently named and exposed by the server.

If a server had both SRW and OAI interfaces to the same collection, a search could be performed in SRW thereby creating a set. Using SRW's extension mechanism, metadata about the search could also be sent at the same time, such as a suggested human readable name and description. Once the result set has been created, it could be automatically exposed in the OAI interface for retrieval. This might be especially appropriate for large records or very large databases where it may be easier to harvest them via the simpler protocol than to harvest them by paging through the set with SRW.

Using common authentication mechanisms between the two protocols would additionally allow users to create sets of records that held particular interest to the users without overburdening the global list of sets with hundreds or thousands of entries. The server could then also store the query and re-apply it at a later date rather than simply referencing the original result set, thereby allowing users to stay up to date with the types of records they are interested in harvesting without the maintainer of the repository having to lift a finger to define a set just for those users.

At the very least, an administrative interface could be provided such that sets can be configured in a standard way across different OAI implementations. All that would be needed is a CQL [24] query handler with which to evaluate the search used to create the set; several free and open source CQL parsers exist in a variety of languages that could be used to help accomplish this.
Registries with SRW and OAI

Unlike the web, services such as Z39.50, SRW and OAI do not have a built-in means to advertise their existence, and no reason to link to other services. A ‘friends and neighbors’ approach was tried in the Z39.50 world where servers would return a list of links to other known servers, but there was no impetus to implement this functionality. Instead of hyperlinks, information retrieval protocols have come to put more faith in registries that maintain a list of known services. Recently, OAI has been used to good effect to maintain and allow the distribution of such registry information.

Given a registry of OAI or SRW services, both SRW and OAI have a part to play. OAI allows the registry to be harvested such that other duplicate or aggregate registries may be kept up to date without manual intervention. SRW allows an agent, either human or software, to discover appropriate services for the agent's current information need.

A perfect example of this is the registry of OAI repositories [25] that Thomas Habing at UIUC has created. Repositories are described using both a Dublin Core schema and the ZeeRex service description schema [26]. The registry can be searched via SRU, harvested via OAI, monitored via an RSS based alerting service, and interactively browsed or searched via an HTML web interface. With regards to the OAI to SRW profile discussed above, Thomas Habing is also the author of a similar study for Z39.50 [27].

A similar registry of SRW services is also available [28] with the same features as the OAI repository registry. The SRW registry also has various value-added services, including an enrichment gateway that can enable support for record transformations. So, for example, if an SRW repository only supports a MARC recordSchema, accessing it through the registry's SRU enrichment gateway will provide a host of other output formats made possible from a catalog of available XSLT metadata crosswalks. So, getting back to the oai_dc requirement discussion above, the reason an oai_dc recordSchema isn't required by the profile is because the SRW Registry's OAI gateway service always uses the Registry's SRU format enrichment gateway as its source rather than directly accessing the registered SRW URL. If oai_dc can be generated from one of the recordSchemas that are available, the registered SRW service will not be obligated to produce it directly.

In a metasearch environment, the existence and interoperability such registries is critical to the success of the venture. Existing metasearch engines require complicated customisation; whereas, if standards based registries were available that contained all the relevant information to describe a service, the target services could be discovered and selected easily, or could even be automatically handled, based on a user's query. In the NISO Metasearch Initiative, Taskgroup 2 is actively discussing this, and earlier an experimental service called the Information Environment Service Registry [29] was commissioned by the JISC.

Registries are not limited to listing services, however. Both OAI and SRW use global, unique identifiers which could be stored in a registry in order to facilitate their use. Both protocols can retrieve records in different XML schemas, and SRW also has identifiers for context sets (a disambiguation method for CQL), diagnostics, extensions and profiles, all of which would be more easily discovered and maintained if they were stored in a registry. Similar registries already exist at several levels. For example, the info URI registry [30] maintains a list of registered namespaces within the URI scheme, and OCLC hosts a registry of components for OpenURL [31]. These registries are powered by OAI, and SRW support would not be difficult to add. SRW and OAI themselves could benefit from similar such registries, rather than just being the protocols used to power them, especially in the realm of SRW’s context sets where duplication of indexes across different context sets may lead to a decrease in the ease of interoperability in the future.

Conclusions

SRW and OAI clearly complement each other. Although the two protocols have chosen different answers to certain questions, this does not prevent them from being stacked up like
building blocks into very different and interesting configurations. OAI's lower barrier to entry and specific goal make it easy to recommend for anyone to implement, whereas SRW is somewhat more complicated but aims to reproduce the essential functions of Z39.50 in facilitating distributed searching rather than harvesting.

Apart from the typical inverted pyramid metasearch model, there are also great benefits to be had from implementing OAI as a gateway interface to an SRW server. This progresses to having both protocols available and interlinked in the same server, such that records selected with a search can then be harvested at leisure.

Not only can regular databases of records have value added to them by these protocols, the protocols can also be used to maintain registries. Service and collection description documents are important to have available such that appropriate routes to information can be taken, but also important are the internal identifiers within the protocols which could be usefully maintained in registries.

**Notes and References**


Copyright © 2005 Robert Sanderson and the OCLC Online Computer Library Center, Inc.