Applying SOAP to OAI-PMH

Sergio Congia, Michael Gaylord, Bhavik Merchant, and Hussein Suleman

Department of Computer Science, University of Cape Town
Private Bag, Rondebosch, 7701, South Africa
{scongia, mgaylord, bmerchan, hussein}@cs.uct.ac.za

Abstract. The Web Services paradigm for distributed computing promises to provide a breakthrough in interoperability by defining standardised mechanisms for inter-process communication. The SOAP standard, in particular, is widely discussed but not as widely adopted by standards bodies. The OAI is one such organisation that has been criticised for not adopting SOAP. Since the OAI-PMH is driven by semantics and SOAP describes syntax, a merger of the two technologies seems natural and inevitable. This paper discusses an attempt to remodel and repackage the OAI-PMH as a layer over SOAP and implement an end-to-end solution based on this experimental protocol. The project highlighted important concerns, such as the relative efficiency of layering in structured textual data and the problem of moving standards. The results show that few compromises are needed for a move to SOAP provided that protocol design is appropriately abstracted, and this has far reaching implications for the adoption of SOAP and Web Services within the DL community and OAI in particular.

1 Preamble

It must be noted at the very outset that the OAI protocol (OAI-PMH v2.0 [12]) is a fixed standard that is implemented in a consistent manner in a growing community of users. This work was simply an experiment, in consultation with members of the OAI, rather than an attempt to suggest a new alternative standard. Based on the results of this experimental work, it is hoped that the OAI and other organisations will more seriously consider an informed adoption of Web Services standards as an underlying layer for future interoperability efforts.

2 Background

2.1 OAI-PMH

The Open Archives Initiative (OAI) was formed to drive the process of developing low-barrier solutions to the problem of interoperability among digital library systems [11]. The primary product of experimentation and standards development was the Protocol for Metadata Harvesting (PMH) [12], a high level application layer network protocol that defines how to synchronise a source of

metadata with a remote copy, or user of the data. From the early stages of development in 1999, it was agreed that the protocol would adopt current standards such as XML Schema [4] and Dublin Core so that it would additionally serve as an implicit testbed and reinforcement for those standardisation activities. The OAI-PMH has since established itself as an important standard for information exchange among a large and varied group of digital archives [3].

Technically, OAI-PMH is a client-server protocol layered over HTTP, with requests specified using URL-encoded parameters and responses delivered in strictly validifiable XML. Specifically, there are 6 request/response pairs: Identify, ListMetadataFormats and ListSets return administrative information about the archive; ListIdentifiers, GetRecord and ListRecords facilitate the transfer of metadata from a source archive on demand. Some of these requests have optional and/or mandatory parameters to restrict the list of results to a subset of the full set, for example, the response for ListIdentifiers can be restricted by specifying a particular set of the archive to list identifiers for, as opposed to the full contents of the archive.

A typical request and response is shown in Fig 1.

Request

```
http://abc.org/OAI?verb=Identify
```

Response

```
<OAI-PMH>
   <responseDate>2004-02-02T12:00:00Z</responseDate>
   <request verb="Identify"/>http://abc.org/OAI</request>
   <Identify>
      <repositoryName>Somewhere</repositoryName>
      <baseURL>http://abc.org/OAI</baseURL>
      <protocolVersion>2.0</protocolVersion>
      <earliestDatestamp>2001-01-01T01:00:00Z</earliestDatestamp>
      <deletedRecord>no</deletedRecord>
      <granularity>YYYY-MM-DD</granularity>
   </Identify>
</OAI-PMH>
```

Fig. 1. Typical OAI-PMH v2.0 request and response (namespace and schema attributes are not depicted)

2.2 SOAP

SOAP [8] is a standard for encoding messages in a distributed computing environment. It originated from industrial efforts to leverage XML technology for

standardised messaging and has tracked best practices as they emerged. SOAP is part of a larger framework to specify external interfaces to network-accessible services: SOAP specifies the syntactic encoding of messages, the Web Services Description Language (WSDL) [2] is a formal specification of a network-accessible API to a service and the Universal Description, Discovery and Integration of Web Services (UDDI) registries list public interfaces. Together these specifications are at the core of the Web Services paradigm of computing, where applications are sequences and aggregations of independent service-oriented components [17].

The current status of Web Services can be confusing to adopters as there is much talk about how it will revolutionise computing, but there are few actual case studies illustrating its use in large-scale interoperable scenarios. This is partly because of the divergence of standards - Microsoft adopted v1.1 of the SOAP specification [1] for its Web Services work, while the W3C only recognises v1.2 [8] as a "recommendation". Between the release of these two versions, the XML Schema standard was formalised - thus, while the earlier SOAP specification allowed the use of XML Schema but supported its own type system, the newer v1.2 SOAP specification does not define its own type system to avoid overlapping with XML Schema functionality. In spite of these issues, some well-known services, such as Google [7], have been publicly exposed using a version of the SOAP protocol.

A typical request and response is shown in Fig 2.

Request

```
<SOAP:Envelope>
   <SOAP:Body>
      <DigitOfPIRequest>1023</DigitOfPIRequest>
   </SOAP:Body>
</SOAP:Envelope>
```

Response

```
<SOAP:Envelope>
   <SOAP:Body>
      <DigitOfPIResponse>6</DigitOfPIResponse>
   </SOAP:Body>
</SOAP:Envelope>
```

Fig. 2. Typical SOAP v1.2 request and response (namespace and schema attributes are not depicted)

3 Context for OAI-PMH + SOAP

The earliest version of the OAI-PMH (v1.0) was disseminated through many avenues, including a paper presented at JCDL 2001 [12]. One of the first questions

posed to the presenter of that paper asked why SOAP was not used. This has led to much discussion in the OAI community, but it was felt that as long as SOAP was not a formal standard, it should be avoided. This was largely motivated by prior experiences with the change in status of the XML Schema specification, an event that necessitated the release of OAI-PMH v1.1.

In mid-2003, SOAP v1.2 was finally released by the W3C as a "recommendation", their equivalent of a standard. At around this point in time, it was decided to try to use the SOAP specification as the basis of an updated experimental OAI protocol, to investigate its applicability and tease out any implementation issues that could inform future standards efforts.

4 Experimental Systems

This investigation involved 4 parts:

- Analysis and design of a new protocol using SOAP as an underlying layer instead of HTTP, hereafter referred to as SOAP-PMH
- Implementation of a SOAP-PMH-based data provider
- Implementation of a SOAP-PMH-based service provider
- Implementation of a SOAP-PMH-based testing and validation tool

Each of the latter 3 parts was implemented to support both the former experimental protocol as well as the original OAI-PMH, to determine the feasibility of supporting multiple transports or bindings in a single implementation.

4.1 Protocol Specification

Since SOAP defines only a syntactic framework, the encoding of OAI-PMH requests and responses needed to be modified, keeping the core semantics unchanged. Unlike standards such as the IMS Metadata Set [10], the protocol specification for OAI-PMH does not define an abstract information model and concrete bindings. As a result, it was not possible to define a new binding as a layer over the abstract semantics. To overcome this, in the context of this experimental work, a parallel specification was defined by editing a copy of the original specification.

Requests were encoded in a manner similar to the SOAP Request/Response use case [9]. Schema for these requests were defined to match the existing OAI-PMH responses. During the development of OAI-PMH v2.0, XML Schema data types were explicitly defined for each PMH parameter in preparation for a possible migration to SOAP. Thus, in SOAP-PMH these data types are imported directly from the original OAI-PMH schema [16] to maintain a high degree of data type consistency.

In order to maintain independence from the underlying transport, the responses were embedded within SOAP envelopes. To some degree, this design choice goes against SOAP recommendations for error handling, which state that semantic errors are handled by special cases defined in the SOAP protocol. It

was felt that this yet again mixed syntax and semantics at different layers of a structured model. Instead, OAI-PMH errors were retained at the OAI-PMH level, while only SOAP errors were handled at the SOAP protocol level. A typical example of a SOAP error is a message with non-SOAP tags at the top level of its body. HTTP errors were yet another case to consider and had to be handled at the layer below SOAP. In the existing OAI-PMH, errors are specified at the semantic, encoding and transport levels, thus HTTP errors that should already have been handled at a SOAP layer still exist at the upper semantic layer. This was not dealt with in this study, but is an important consideration for future protocol design activities.

Fig 3 depicts the current onion-peel design of OAI-PMH, an onion-peel version of SOAP-PMH and an ideal conceptual view of the layers. In this model, OAI-PMH is built around HTTP, implying that they are not separable. The SOAP-PMH version is built over SOAP, layered over HTTP – implying that neither the SOAP nor HTTP encodings are essentially separable from the core protocol, but that SOAP and HTTP are not necessarily bound together. Conceptually, in the ideal case, the semantics of the protocol should be specified independently of the encoding mechanism and transport layer. Thus, if the protocol was written with this in mind, it would have been simpler to retarget it from the combination of URL encoding and XML responses to SOAP messages. Similarly, since SOAP does not intrinsically depend on HTTP, upper level protocols ought not to do so – so that if HTTP was replaced by, say, SMTP, no changes in the core semantics would be necessary.

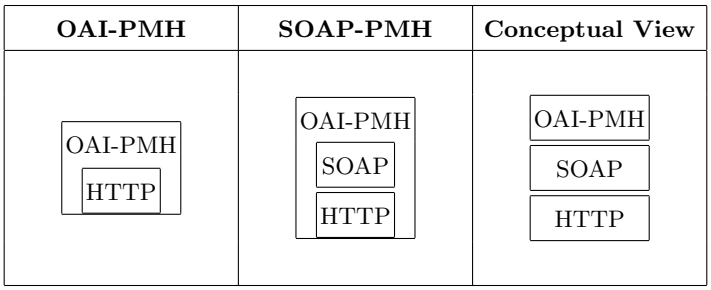

Fig. 3. Different conceptual views of OAI-PMH as a transport-independent protocol

A typical request and response pair from this experimental SOAP-PMH is shown in Fig 4. The regular OAI-PMH response is cleanly encapsulated within the Body of a SOAP Envelope. The SOAP Header is not used since its definition suggests it is intended for intermediate or non-payload processing and the OAI-PMH essentially requires end-to-end payload delivery.

Request

```
(Sent to: http://abc.org/SOAPOAI)

<SOAP:Envelope>
   <SOAP:Body>
      <OAI-PMH-Req>
         <verb>Identify</verb>
      </OAI-PMH-Req>
   </SOAP:Body>
</SOAP:Envelope>
```

Response

```
<SOAP:Envelope>
   <SOAP:Body>
      <OAI-PMH>
         <responseDate>2004-02-02T12:00:00Z</responseDate>
         <request verb="Identify"/>http://abc.org/SOAPOAI</request>
         <Identify>
            <repositoryName>Somewhere</repositoryName>
            <baseURL>http://abc.org/SOAPOAI</baseURL>
            <protocolVersion>SOAP2.0</protocolVersion>
            <earliestDatestamp>2001-01-01T01:00:00Z</earliestDatestamp>
            <deletedRecord>no</deletedRecord>
            <granularity>YYYY-MM-DD</granularity>
         </Identify>
      </OAI-PMH>
   <SOAP:Body>
</SOAP:Envelope>
```

Fig. 4. Typical SOAP-PMH request and response (namespaces and schema attributes are not depicted)

4.2 Data Provider

A database-driven data provider was built to conform to the SOAP-PMH. For a reasonable level of completeness, this module supported the following optional features of the OAI-PMH:

- multiple metadata formats
- flow control using resumption tokens
- sets for selective harvesting

The implementation was done in Java, using JDBC for database connectivity to a MySQL source of data. The Web interfacing was accomplished using Java servlets. Processing of requests and responses was performed in layers to avoid duplication of code in supporting different bindings of the core OAI-PMH semantics.

4.3 Testing Tool

An intemediate-level testing tool was developed to test data provider implementations of both OAI-PMH and SOAP-PMH. This tool built on typical tests carried out by the Repository Explorer [15] and OAI Data Provider Registry [14] by including the following capabilities:

- Users may choose which tests to conduct in a batch, as a subset of the full suite.
- The testing tool was implemented as a portable Java application, thus enabling efficient testing of local data providers, even those behind firewalls or those that are domain-restricted.
- Exploiting its nature as an application, the human interface was designed for greater usability e.g., tabbed dialogs where used where this seemed most natural and progress indicators were used during batch testing.

4.4 Service Provider

From a service provider perspective, a simple search engine was built, based on the Lucene [6] open source package. A Web-based interface was developed using Java servlets, both for searching through harvested data and for management of the harvesting operations. Once again, the harvester was developed such that it would work with either the regular OAI-PMH or the experimental SOAP-PMH.

5 Evaluation

Evaluation of the software was conducted on multiple levels, to confirm that the testbed was realistic and then to deduce emergent properties that were a result of using SOAP.

In the first instance, usability testing was conducted on the testing tool and the service provider. These tests confirmed that non-expert users were able to successfully harvest data, conduct searches and test data providers.

All three components were then tested for OAI protocol compliance, by interconnection and by connection with external components. For example, the testing tool was used to validate existing OAI data providers and the data provider was validated by the Repository Explorer.

Finally, performance testing was conducted on the data provider to determine the effect of the intermediate layer on network traffic and necessary processing. For this purpose, the data provider was primed with a copy of approximately 77000 ETD records, themselves obtained from an OAI data provider. Measurements for a typical **ListRecords** request formulated to disseminate a subset of records specified by *set*, *metadataPrefix*, *from* and *until* parameters are shown in Table 1. The first column confirms that there is a large increase in the size of the request, and this is due to the switch from URL encoding to XML. The response, on the other hand, has a size differential that is due only to the additional layering and therefore does not have a sizable impact on bytes transferred. The processing time, similarly, is only minimally affected by SOAP layering.

Protocol	Request(bytes)	Response (bytes)	Processing Time (ms)
SOAP	645	167037	1609.22
HTTP	140	166616	1594.87
Diff	505	421	14.4

Table 1. Performance measurements for SOAP-PMH vs. OAI-PMH for a typical request/response pair

6 Analysis and Conclusions

The process of developing a protocol and development and testing of initial implementations conforming to it have revealed a number of issues that could assist with future protocol design efforts.

Firstly, and most importantly, the migration to SOAP was deemed to be relatively straight-forward by the developers. The implication of this is that there need be no additional complexity in software development because of the use of SOAP. Further, it was possible in all instances to create tools that could understand multiple bindings of the core protocol.

In most cases, the use of SOAP also resulted in only marginal increases in bytes transferred and processing time required. During the course of performing these tests, it was discovered that XML indentation for human-readability

caused a major increase in bytes tranferred (typically 6 bytes per line of XML embedded in a SOAP Body). This reinforces the notion that XML should be optimised for data transfer by removing indentation and linefeeds, and introducing compression where possible. Such techniques will further reduce the effects of adding SOAP layers.

While SOAP supports the use of alternative lower layers, the tight integration of OAI-PMH and HTTP prevents a clean separation at the SOAP layer. In contrast, the Blox project successfully used SMTP as a SOAP transport because of less overhead than HTTP [13]. This is not possible in SOAP-PMH because of, for example, the requirement for a *baseURL* in the *request* header tags of all responses. There is understandably tension between the usefulness of abstractions and the utilitarian benefit of complete specifications. However, unless a new complete specification is to be released for every underlying transport, abstracting the semantics of OAI-PMH should be considered.

The SOAP standard is itself not cleanly separated from the semantics it encapsulates. By specifying that semantic errors are to returned as SOAP Faults, it makes tight integration a requirement and this once again introduces complications, albeit minor, into the protocol design/layering process.

All these issues notwithstanding, the benefits of Web Services as a standard framework, with integrated support in many modern development tools, coupled with the verified ease of a migration of OAI-PMH to SOAP, makes it imperative that this is pursued by the OAI.

7 Future Work

It is hoped that the results of this experimental work will lead to the formation of a new working group to investigate the migration of OAI-PMH towards a SOAP-based standard.

This study was initially intended to include investigations into WSDL specifications for the OAI protocol. This was abandoned largely because of the state of flux of the WSDL protocol, with differences in encoding between the existing WS-I version and the draft W3C standard (as at mid-2003). By the completion of the project, WSDL was not yet standardised by W3C - when this happens, a formal description of the existing and SOAP protocols can be created.

New formalisms such as REST (REpresentational State Transfer) [5] also need to be taken into account. Where SOAP's request/response message pattern suggests that requests be encoded as XML messages, REST recommends that if no state is changing, requests should be encoded as HTTP GETs. These sometimes conflicting notions must be reconciled so that Web Service interfaces are indeed standardised. XML messages have the advantage of standardised data encoding while services such as ERRoLs have the advantage that they can be completely referred to by a simple URL. More work is needed to investigate how to bridge the gap from RESTful URLs to possibly RESTful SOAP communications.

References

1. Box, Don, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte and Dave Winer (2000), Simple Object Access Protocol (SOAP) v1.1, W3C, 8 May 2000. Available http://www.w3.org/TR/SOAP/
2. Christensen, E., F. Curbera, G. Meredith and S. Weerawarana (2001), Web Services Description Language (WSDL) 1.1, W3C. Available http://www.w3.org/TR/wsdl
3. Dobratz, Susanne, and Birgit Matthaei (2003), "Open Archives Activities and Experiences in Europe: An Overview by the Open Archives Forum", in D-Lib Magazine, Vol. 9, No. 1, January 2003. Available http://www.dlib.org/dlib/january03/dobratz/01dobratz.html
4. Fallside, David C. (editor) (2001), XML Schema Part 1: Structures and Part 2: Datatypes, W3C, 2 May 2001. Available http://www.w3.org/TR/xmlschema-1/ and http://www.w3.org/TR/xmlschema-1/
5. Fielding, Roy, T. and Richard N. Taylor (2002), "Principled design of the modern Web architecture", in Transactions on Internet Technology, Vol. 2, No. 2, ACM Press, pp. 115-150.
6. Goetz, Brian (2000), The Lucene search engine: Powerful, flexible and free, in JavaWorld. Available http://www.javaworld.com/javaworld/jw-09-2000/jw-0915-lucene.html
7. Google (2004), Google Web APIs. Website http://www.google.com/apis/
8. Gudgin, M., M. Hadley, N. Mendelsohn, J. Moreau and H. F. Nielson (2003), SOAP Version 1.2 Part 1: Messaging Framework and Part 2: Adjuncts, W3C, 24 June 2003. Available http://www.w3.org/TR/2003/REC-soap12-part1-2003-0624/ and http://www.w3.org/TR/2003/REC-soap12-part2-2003-0624/
9. Ibbotson, J. (2002), SOAP Version 1.2 Usage Scenarios, W3C, 26 June 2003. Available http://www.w3.org/TR/2002/WD-xmlp-scenarios-20020626/
10. IMS Global Learning Consortium, Inc. (2001), IMS Learning Resource Meta-data Information Model, IMS, 28 September 2001. Available http://www.imsglobal.org/metadata/imsmdv1p2p1/imsmd_infov1p2p1.html
11. Lagoze, Carl and Herbert Van de Sompel (2001), "The Open Archives Initiative: Building a low-barrier interoperability framework", in Proceedings of JCDL 2001, Roanoke, VA, USA, June 2001, ACM Press, pp. 54-62.
12. Lagoze, Carl, Herbert Van de Sompel, Michael Nelson, and Simeon Warner (2002), The Open Archives Initiative Protocol for Metadata Harvesting Version 2.0, Open Archives Initiative, June 2002. Available http://www.openarchives.org/OAI/2.0/openarchivesprotocol.htm
13. Moore, David, Stephen Emslie and Hussein Suleman (2003). BLOX: Visual Digital Library Building, Technical Report CS03-20-00, Department of Computer Science, University of Cape Town. Available http://pubs.cs.uct.ac.za/
14. Open Archives Initiative (2004), Open Archives Initiative Data Provider Registry. Website http://www.openarchives.org/data/registerasprovider.html
15. Suleman, Hussein (2001), "Enforcing Interoperability with the Open Archives Initiative Repository Explorer", in Proceedings of the ACM-IEEE Joint Conference on Digital Libraries, Roanoke, VA, USA, 24-28 June 2001, pp. 63-64.
16. Van de Sompel, H. and S. Warner (2004), XML Schema which can be used to validate replies to all OAI-PMH v2.0 requests, 29 March 2004. Available http://www.openarchives.org/OAI/2.0/OAI-PMH.xsd
17. Yang, J. (2003), "Web Service Componentization", in Communications of the ACM, Vol. 46, No. 10, October 2003, ACM Press, pp. 35-40.