Protocol Considerations for Web Linkbase Access

Erik Wilde
Computer Engineering and Networks Laboratory
Swiss Federal Institute of Technology, Zürich

TIK Report 143
July 2002

Abstract

We propose the *Open Web*, which aims at transforming the Web into an Open Hypermedia System. Based on the *Extensible Linking Language (XLink)*, we investigate the possibilities for implementing linkbase access methods. Linkbases are collections of so-called *third-party* links, which are links which live outside the resources that they are linking, and thus must be found and retrieved somehow when presenting the resources that they are linking. We focus on the protocol issues of accessing linkbases, concentrating on how such a new protocol could and should be designed. In addition to our design goal of specifying a protocol for accessing the linkbase Web service, be believe that our protocol considerations can serve as a blueprint for other areas where Web access to services is required.

1 Introduction

The *Hypertext Transfer Protocol (HTTP)* [8] is the most widely used protocol on the Internet. It is mainly used as a client/server-protocol between a client (in most cases, a Web browser) requesting a Web resource, and a server responding to that request. While currently most browser interactions with servers are simple Web resource retrievals for immediate display (in most cases HTML pages or images), future developments will lead to an increasing exchange of non-display data, in particular metadata as the core matter of the *Semantic Web* [2].

In this paper, we investigate access to a special kind of metadata, which is link information. The *Extensible Markup Language (XML)* [4] is rapidly becoming the universal foundation for structured data on the Web, and the *Extensible Linking Language (XLink)* [7] is an additional specification defining how to embed links in XML documents [17]. One of the most remarkable features of XLink is its support for third-party links, which can be aggregated in so-called *linkbases*. XLink's features make it possible to realize the vision of the Web as an *Open Hypermedia System*, which we call the *Open Web*, and one of the core issues of the Open Web is the question of how browsers communicate with linkbases.

2 XLink and Linkbases

XLink treats links as first-class objects, which means that links may be regarded as separate entities outside of the resources that they are linking (they are called third-party links because they may be provided separately from the resources). XLink supports a sophisticated linking model that can support advanced hypermedia presentations [16]. However, how links are generated, maintained, transferred, and presented it not within XLink's scope. In XLink, linkbases are simply XML documents which are collections of third-party links. From XLink's point of view, this is sufficient, but it is not sufficient from the perspective of transforming the Web into the Open Web. For this vision to become a reality, an open protocol must exist which can be used by clients to access linkbases and request links from them.

Linkbases potentially contain huge numbers of links, and it is therefore not practical for clients to retrieve the entire linkbase in order to find the links that are relevant for a given document. What therefore is necessary is some kind of query mechanism that can be used to retrieve the links from a linkbase that are relevant for a given Web resource. And because client and linkbase must communicate over the Internet, a protocol must be used which supports this query mechanism. It is the purpose of this paper to describe the potential solutions to this problem, which also serve as an interesting lesson in Web engineering.

3 Linkbase Queries

XLink defines links in terms of a set of global attributes. This is the XML syntax of XLinks, and even though the underlying link model is informally described in the XLink specification, so far there is no formal specification of XLink's link model. This situation is similar to the history of XML itself, where the document syntax predated the specification of the data model, which was later specified in the *XML Information Set (XML Infoset)* [6]. The XML Infoset was specified because it was discovered that a number of applications needed to be based on the data model rather than the syntax (examples are APIs, query languages, and presentation mechanisms). We assume that similar reasoning will lead to an *XLink Infoset*, but so far this is purely hypothetical. However, a first step has been made in a W3C note [15] dealing with the question of XLinks and their presentation by styling mechanisms. This note describes a (limited) draft of a formal XLink Infoset, and it could become the starting point of a specification of the XLink Infoset.

While using *XML Query (XQuery)* [3] (or a subset of it) for linkbase queries might be interesting at first sight, there are problems of availability (XQuery is still under construction, but this will change in the near future), and the more serious problem of the data model. While linkbase queries need to operate on the XLink Infoset, XQuery operates on the XML Infoset, which is inappropriate for linkbases. We therefore decided against basing linkbase access on XQuery.

Furthermore, we prefer a *service-oriented* rather than a *query-oriented* perspective,

which means that we do not want to force linkbase providers to use XLink internally. We only want to base the linkbase queries on the XLink Infoset (and extensions to it which providers might choose to use), and XLink syntax is only required as the exchange format between the linkbase and the requesting client. Specifically, our query model handles the support of different linking models as follows:

- *XLink link model*

  As the most important aspect of the query model, we support querying the XLink information set. This means that queries can be used which specifically address XLink's link model.

- *XLink supported extensions*

  XLink contains mechanisms for specifying link semantics (the so-called *semantic attributes*), but these mechanisms do not prescribe a way of how these semantics have to be represented. We provide basic support for this type of application-specific link semantics.

- *Additional XML attributes*

  If XLinks are represented in XML, then they can be easily extended with additional XML attributes. While this way of extending XLink can become very complex and therefore hard to support generically, we also provide basic support for this type of application-specific link semantics.

- *Generic links*

  XLink does not explicitly support generic links, but generic linking can have very powerful applications[1]. Generic links make it necessary for the linkbase server to have access to the resource, and therefore generates additional load on the communication link as well as on the linkbase server. We therefore intend to support generic linking in the next version.

This rather short presentation of the linkbase access model should be sufficient to serve as a foundation for the following protocol considerations. Basically, we designed linkbase access as a service and then were looking for a suitable protocol binding for this service.

4 Protocol Considerations

In this section, we address the design issues that were important for deciding which protocol binding to choose for the linkbase access service. It should always be kept in mind that for our model of the Open Web, linkbase access should become widely used and should be supported by many clients and servers.

[1] It can also have highly questionable implications when under control of one company, as has been shown by the controversial debate about Microsoft's *Smart Tags*.

- *Proxies*

  Proxies are a common components of today's network topology, and in order to fit into this topology, linkbase access must be able to operate over one or even chained proxies. Seeing that many proxies also implement firewall strategies, it becomes almost mandatory to use one of the well-known port numbers, such as HTTP's port 80.

- *Caching*

  Caching in the case of linkbase access is difficult, because effectively linkbase accesses are query operations which may be affected by many factors, not only the query itself. For example, the server might also use other knowledge to better serve the request, such as the client's properties and preferences. We therefore believe that caching will not play an important role for linkbase access, but we also believe that there are opportunities for more research in this area.

- *Redirection*

  The protocol must support redirection in the sense that a client can be *temporarily* or *permanently* redirected to another server, if the linkbase server is not capable of processing the request. This redirection must be transparent to the user in order to avoid disruptive messages. However, a client may choose to inform users about permanent redirections, so that the access information for the redirected linkbases may be updated.

- *Authentication*

  Linkbase services may be operated commercially. Thus, clients must have a way to authenticate against the server, so that the server can charge registered users for its services. Additionally, it should be possible for clients to authenticate the server, so that they can be sure that they are communicating with the right server.

- *Privacy*

  Links can be considered normal Web resources, and as with all Web resources, there are scenarios where users are willing to accept cleartext transmission, while other scenarios might require a high level of privacy, provided by cryptographic methods. Consequently, linkbase access must be able to switch between cleartext and encrypted operation.

- *Integrity*

  There must be ways to guarantee the integrity of linkbase accesses, if required by the client and/or server. This means that it should not be possible be a third party to change the information being exchanged between client and server. Like privacy, integrity is probably not required in all scenarios, but it should be supported as an optional property of the protocol.

This list of design consideration lists all the requirements that a protocol binding for the linkbase access service should satisfy. In the following section, we describe some possible implementations of the protocol binding, and in Section 6 we revisit the protocol considerations and look at how well the different protocol bindings work.

5 Protocol Designs

In this section, we describe a number of possible implementations of the protocol binding for linkbase access. In the descriptions it is assumed that the size of the accessed linkbase makes it impractical to return it completely to the client, so that some form of filtering is required.

5.1 Implicit Information

In the simplest case, implicit information could be used to convey the information that is used by the server to select links from the linkbase. For accessing a linkbase, a client would simply retrieve it (probably using HTTP's `GET` method), and the linkbase server would respond with the links that are appropriate. The request's `Referer` header field (and possibly others) could be used by the server to decide which links are returned to the client. However, this method of accessing linkbases becomes highly problematic if the client wants to exert control on the linkbase service's operation.

5.2 URI Query Strings

In this case, some form of query string would be appended to the linkbase's URI which would be interpreted by the server. This query string could contain information about which links to select, such as certain semantic attributes of XLink. In this case, the client can control the server's behavior using the query string. The query string could be regarded as being specific for linkbase resources, so that would only be applicable to this type of resource. In effect, this model would be comparable to the processing of HTML forms (when using the `GET` method), where some sort of program on the server is executed to process the form values submitted by the client.

One limitation of this method is that the linkbase query would have to be coded using URI syntax, which requires a lot of escaping and becomes cumbersome for complex structures.

5.3 HTTP Extensions

HTTP can be extended by using new methods and/or header fields. This has already been done by *WebDAV* [11], and the *HTTP Extension Framework* [10] defines a formal framework for how this could be done[2]. Using this approach, HTTP could be extended to

[2]WebDAV predates the HTTP Extension Framework and does not conform to it.

include methods and headers that support linkbase access. The HTTP Extension Framework is carefully designed to work across unextended proxies, and it also makes it easy for peers to detect whether the partner does support a particular extension.

5.4 HTTP Entities (I)

Instead of coding the linkbase access information within URI query strings or specific header fields, it could also be embedded inside the HTTP message body. If only the the access information would be contained in the message body (with the URI addressing the linkbase service), this method would be comparable to the processing of HTML forms (when using the `POST` method). It would be necessary to define the format of the message body, but this is unconstrained by HTTP.

5.5 HTTP Entities (II)

Another way to embed the access information into the HTTP message body would be to not only include the access information, but also the information about which service is being accessed on the server. This would effectively use HTTP as a generic transport protocol for remote services, and there are several established technologies for doing this. One of the earliest approaches is *XML-RPC*, which is very basic.

A more elaborate approach is the *Simple Object Access Protocol (SOAP)* [13], which can be used in different ways, one of them being RPC-like services. Using SOAP, it is be possible to describe a service using the *Web Services Description Language (WSDL)* [5], and then define protocol bindings, such as HTTP. This layer of abstraction can be very useful for services which should be made accessible through different protocol bindings.

If required, additional descriptions could be used to define the interaction between a linkbase access client and a server. HP's *Web Services Conversation Language (WSCL)* [1] or IBM's *Web Services Flow Language (WSFL)* [12] currently are the most prominent languages for higher-level descriptions of Web Services, in particular their usage (in terms of activity sequences) and their composition.

5.6 XQuery

Even though we decided against using XQuery as the foundation for our linkbase access, it would still be possible to use XQuery's protocol binding and maybe extended it with XLink-specific functionality. However, at the time of writing XQuery itself is still under construction, and there is no publicly available draft of the planned protocol binding, so we concluded that even though XQuery might provide a valuable alternative in the future, today it is not suitable as a foundation.

5.7 Dedicated Protocol

Instead of using any of the available Web infrastructure, it would of course also be possible to design a completely new protocol, either being based on very low-level transport mechanisms such as TCP, or using supporting frameworks for building distributed applications, such as CORBA. Both approaches have different consequences:

- *Built on transport-level*

  Building on top of the transport-level means that everything in the protocol could be designed as required, but it would also mean that the complete functionality would have to be implemented. In the light of the fact that most clients as well as browsers already have a number of built-in communication layers (such as SSL/TLS and HTTP), this strategy would not re-use any components.

- *Using supporting framework*

  In this case, the framework could be selected to match the needs of linkbase access as closely as possible. However, using a framework would result in making this framework a necessity for all clients and servers supporting linkbase access, and this is a decision which should be very carefully considered.

The approach to define a completely custom protocol, based on low-level transport mechanisms or supporting frameworks, would in both cases result in less-than-ideal implementations, which either do not re-use existing components, or would introduce some rather heavy framework into both clients and servers. We therefore decided against this approach.

6 Design Choice

In the previous section we listed the considerations for the protocol binding of our linkbase access service. After considering the design criteria and the possible solutions, we decided to follow the path described in Section 5.5, ie modelling linkbase access as a Web service and using existing infrastructures for making the Web service accessible. Based on this decision, we now shortly revisit the design considerations presented in Section 4 (assuming that our Web service is specified using WSDL and protocol bindings are defined for HTTP and HTTPS):

- *Proxies*

  Since SOAP uses HTTP and HTTPS as transport mechanism, we can safely build on the Web's standard protocol's ability to be used through proxies and firewalls. Some firewalls may still reject SOAP messages (based on the HTTP header fields), but if a firewall is configured this strictly, there is not much we can do.

- *Caching*

  In principle, SOAP can use all of HTTP's caching mechanisms, but since linkbase access often will result in unique requests, we assume that caching of linkbase accesses will have a very low hit rate (in fact, it may make sense to configure caches so that they do not cache any SOAP or at least linkbase access messages).

- *Redirection*

  Linkbase servers can easily use HTTP's mechanisms to redirect linkbase accesses. HTTP's `301` and `302` status codes (as well as maybe `410` for linkbases that have disappeared) can be used by the server to inform the client about redirections.

- *Privacy*

  HTTPS is the Web's most widely used mechanism for encrypted exchange of messages, and it can simply be used to exchange SOAP messages.

- *Authentication*

  HTTP's authentication mechanisms [9] can be used to implement authentication. They do not provide a high level of security, in case of security-critical applications it is therefore highly advisable to combine them with HTTPS.

- *Integrity*

  HTTP's `Content-MD5` header or *HTTP Instance Digests* [14] can be used to ensure the integrity of the messages being exchanged. Both methods should be combined with HTTPS for security-critical applications.

As can be seen, the design criteria have been met by our choice of protocol binding, and it hopefully will prove useful to build on a standard platform for Web services, for example when linkbase access should be combined with other Web services and the overall service could be described using WSCL/WSFL.

7 Conclusions

In this paper we have described our linkbase access service and its protocol binding. We have completed a prototype implementation of the service (using simple XSLT programs to perform the linkbase queries), and will be testing the service and the protocol for completeness and openness, and then submit it to the W3C. However, there still needs a lot of work to be done (in particular, the XLink Infoset and a presentation model) until the Open Web can be implemented in an interoperable way.

References

[1] Arindam Banerji, Claudio Bartolini, Dorothea Beringer, Venkatesh Chopella, Kannan Govindarajan, Alan Karp, Harumi Kuno, Mike Lemon, Gregory Pogossiants, Shamik Sharma, and Scott Williams. Web Services Conversation Language (WSCL) 1.0. World Wide Web Consortium, Note NOTE-wscl10-20020314, March 2002.

[2] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, May 2001.

[3] Scott Boag, Don Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie, Jérôme Siméon, and Mugur Stefanescu. XQuery 1.0: An XML Query Language. World Wide Web Consortium, Working Draft WD-xquery-20011220, December 2001.

[4] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler. Extensible Markup Language (XML) 1.0 (Second Edition). World Wide Web Consortium, Recommendation REC-xml-20001006, October 2000.

[5] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web Services Description Language (WSDL) 1.1. World Wide Web Consortium, Note NOTE-wsdl-20010315, March 2001.

[6] John Cowan and Richard Tobin. XML Information Set. World Wide Web Consortium, Recommendation REC-xml-infoset-20011024, October 2001.

[7] Steven J. DeRose, Eve Maler, and David Orchard. XML Linking Language (XLink) Version 1.0. World Wide Web Consortium, Recommendation REC-xlink-20010627, June 2001.

[8] Roy T. Fielding, Jim Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, Larry Masinter, Paul J. Leach, and Tim Berners-Lee. Hypertext Transfer Protocol — HTTP/1.1. Internet proposed standard RFC 2616, June 1999.

[9] John Franks, Phillip M. Hallam-Baker, Jeffery L. Hostetler, Scott D. Lawrence, Paul J. Leach, Ari Luotonen, and Lawrence C. Stewart. HTTP Authentication: Basic and Digest Access Authentication. Internet proposed standard RFC 2617, June 1999.

[10] Henrik Frystyk Nielsen, Paul J. Leach, and Scott D. Lawrence. An HTTP Extension Framework. Internet experimental RFC 2774, February 2000.

[11] Y. Goland, E. James Whitehead, A. Faizi, S. Carter, and D. Jensen. HTTP Extensions for Distributed Authoring — WebDAV. Internet proposed standard RFC 2518, February 1999.

[12] Frank Leymann. Web Services Flow Language (WSFL 1.0). Technical report, IBM, May 2001.

[13] Nilo Mitra. SOAP Version 1.2 Part 0: Primer. World Wide Web Consortium, Working Draft WD-soap12-part0-20020626, June 2002.

[14] Jeffrey C. Mogul and Arthur van Hoff. Instance Digests in HTTP. Internet proposed standard RFC 3230, January 2002.

[15] Norman Walsh. XML Linking and Style. World Wide Web Consortium, Note NOTE-xml-link-style-20010605, June 2001.

[16] Harald Weinreich, Hartmut Obendorf, and Winfried Lamersdorf. The Look of the Link — Concepts for the User Interface of Extended Hyperlinks. In *Proceedings of the 12th ACM Conference on Hypertext and Hypermedia*, pages 19–28, Århus, Denmark, August 2001. ACM Press.

[17] Erik Wilde and David Lowe. *XPath, XLink, XPointer, and XML: A Practical Guide to Web Hyperlinking and Transclusion*. Addison Wesley, Reading, Massachusetts, July 2002.