Annual Report to the NSF:

Integrating and Navigating Eprint Archives through Citation Linking

Project Number IIS-9907892

June 30, 2001

1 Introduction

NSF funded the Cornell Digital Library Research Group (part of the Computer Science Department at Cornell) to work with researchers at Southampton University to investigate the linking of electronic archives through Citation Linking (known more widely in the United States as Reference Linking). A good example of an electronic archive is the E-Print archive at Los Alamos National Laboratory, a third partner in this project.

The overall project has been named Open Citations, or OpCit for short [1, 7].

While researchers at Southampton undertook the linking of the LANL archives using tools they developed for the Open Journal[6] project, we at Cornell took a more general view of the reference linking problem as it applied to online scholarly material, in order to determine a good framework for linking the online literature and to come up with tools that would encourage the linking of electronic publications.

It should be noted that a large group of journal publishers have their own project underway, called CrossRef[10]; this is a very worthwhile project, and when successful will be of great assistance to scientists and other technical workers. However, it relies on Digital Object Identifiers (DOIs), which might not be available to all online publications. At Cornell we are focusing on more informal materials, including born-digital materials such as D-Lib, grey literature such as articles published online by individuals, and repositories of technical reports, such as NCSTRL.

2 Accomplishments in Second Year

In the first year, we considered the question, "what would be the ideal behavior of a digital object that supported reference linking (both incoming and outgoing)"? Answering this question led to an API that included four principle methods:

1. getMyData() - the digital object should emit standard metadata describing that object, i.e. title, authors, year of publication, etc.

2. getReferenceList() - the digital object should say what its list of references is (this is the fixed number of references contained in the online document).

3. getCitationList() - the object can say what other works the object knows have cited it. (This list grows as more and more items are analyzed.)

4. getLinkedText() - returns the original content of the digital object but with link information added to it so that each reference can be used to go directly to an online copy of the referenced work, if an online copy is available.

In the second year, we finished implementing 1 and 2, and implemented 4. Implementation of 3 is underway as a student Masters of Engineering project. The digital object was implemented as a Java class, called Surrogate. To instantiate a Surrogate for an online document, one simply calls the Surrogate constructor with the URL of the online document. We decided to concentrate initially on HTML documents; work on PDF documents is underway at U. of Southampton, and has just begun at Cornell on the ACM online library.

The first problem we encountered was in parsing HTML documents. We started off using Java's Swing HTML parser, but found it to be undocumented and not very functional. At that point, we switched to using the JTidy HTML to XML converter, from W3C. That works very well, and there are several robust XML parsers around, such as xml4j, jaxp, xalan, and xerces.

Implementation of the `getLinkedText()` method raised many interesting problems and issues. We were required to locate all the references within the text based on simple patterns, such as "(name, year)" and "{ACRONYM}" and "[1-3]". This is a challenging problem, but we can locate most references using collection of about six different grammars. After locating the first occurrence of a reference, we use the matching grammar for the rest of the paper. Most of the variation in reference format is between papers, not within a paper.

The second problem, once the reference in the text has been found, is to match it to a literal reference of the end of the paper. For example, if the first reference is "1. —", then the reference"[1-3]" refers to the work listed in reference 1. Matching a reference like "Bragg (1998) said..." with its literla reference "(Bragg, 1998)..." requires some simple heuristics.

Finally, we save the context in which the reference occurred, aiming to extract just the sentence containing the reference anchor.

The main difference between the original document and the analyzed, XML version is that the XML contains our own <reflink> tags around each reference in the text. These tags enclose bibliographic data about the referenced work. Later, XSLT can be used to translate the <reflink> elements into <openURLs>, <Xlinks>, etc.

Having implemented the getLinkedText method, we were next interested in using this with an archive search engine, such as Dienst or Eprints. This required building a repository of Surrogates to correspond to items in the repository. Hence we added a fifth method, "`save()`", which causes a Surrogate object to save its instance data as short XML files in the appropriate directory. Correspondingly, a special constructor was implemented that would re-instantiate a Surrogate object using the saved XML documents.

Thus with the current implementation, one can analyze an entire repository, save the surrogates, and then have them resurrected on by one, “ on demand”. In other words, the user wishing to retrieve a document from the archive could (if he or she chooses) be shown the linked version by resurrecting the appropriate Surrogate and invoking the get linked text method on it.

A demonstration of the API being used by a browser application can be viewed at http://cs-tr.cs.cornell.edu/RefLinkingDemo. This demonstration fulfills our goal of using the link data obtained by the analysis of an online document in a presentation/rendering application. We used XSLT to transform the link data into JavaScript code segments suitable for retrieving linked full text.

The Java implementation of the reference linking API runs on Linux, Solaris, Windows2K, and NT. Thus we can claim it to be quite portable (which you would expect for Java codes). The API has been distributed to our colleagues at the University of Southampton, and is available for other academic users. In fact, a key part of the implementation code is based on software originally written at the University of Southampton, and subsequently enhanced at Cornell. All other packages used (e.g. Tidy, Xalan, etc.) are open source or GPL’ed.

The following design decisions were taken and have been found to be good ones:

1. Do everything in XML. Convert the input document to XML (i.e. XHTML). Make the output of every API method an XML document. Store the Surrogate’s instance data as XML documents.

2. Use Dublin Core for bibliographic data. This makes our work consistent with the Open Archives Initiative.

3. Because works have to be recognized if encountered again, we use a URN (Uniform Resource Name) based on the work’s bibliographic data. The URN format is a single string which contains: first author’s last name, or ”*” if not known; the 4 digit year of publication, or ”*” if not known; and the first 20 characters of the title, or ”*” if not known. This has proved to be quite effective in looking up documents, to see if we have seen them before. Each work is represented by its URN (a string of several dozen characters), and index number, and the DOI of a Surrogate. (The DOI of a Surrogate can be the official DOI of an online journal, such as 10.1045 for D-Lib magazine; it could be an OAI Repository name, like arXiv; or any unique names collection name.) The index number is 0 if this URN is that of the Surrogate; otherwise its value is i where this work is the Surrogate’s i^{th} reference.

4. We decided to use <`reflink`> elements to contain a reference’s bibliographic data, the reference literal, and URLs of its online locations if known. This proved to be a very good decision, as the <`reflink`> can

then be translated to any of a variety of actual links, such as plain URLs, XLinks, OpenURLs, and so on. Thus in the output of `getLinkedText()`, each reference in the document is enclosed in a <`reflink`> element.

5. Send all debug messages to *syserr*. This allows application programs to send the XML documents from Surrogate public methods to *stdout*, which could be a file or a document for a browser. In this way, an application can simply display the XML output, without intervening error messages. (See the demo for an example.)

During the second year of this work, we ran a number of experiments.

1. Intralink D-Lib. At the end of year one we had `getMyData()` and `getReferenceList()` working for a single (randomly selected) D-Lib paper. In year two, we processed hundreds of D-Lib papers.

2. We experimented with various presentation methods and came up with the demo given above.

3. Last year we did a broken link analysis of D-Lib [3]. This year we looked at D-Lib's well-formedness. Of 280 journal items, one crashed JTidy, 60 were suffiently ambiguous or ill-formed that they could not be converted into legal XML with any precision. Thus about 75% of D-Lib papers can be converted into XML and analyzed.

4. For the papers that can be analyzed, we extract reference linking information at better than 80% accuracy [5].

5. Analyzing a D-Lib paper takes about 5 seconds on average, on a Sparc 10.

It is problematical to parse a document based on presentation inputs (e.g. HTML tags), but it is sometimes necessary. Input tags are important clues as to where new subsections start, where the title is, and so on. Furthermore, automatic extraction techniques must include heuristics to handle words such as `et al.`, `et.  al.`, `et.  al`, and `see also`. (Note that many authors do use the misspelled versions.) The project continues to pose interesting challenges.

3 Plans for Year 3

Our work in reference linking made good progress during the first two years, 1999-2001. We need to finish off and polish the implementation. We have already made a good start on inserting JavaDoc comments, which in turn can be used to produce documentation of the API.

One goal initially was to fold the reference linking API into the widely used Dienst[8] architecture and protocol. However, it is now most likely that OpCit's Eprints architecture and implementation is going to replace Dienst in most cases. We plan to install Eprints at Cornell, and if reasonable, add a reference linking service to it, built on our API. This would simplify the task of interlinking various archives.

The work on the reference linking API, as well as its accuracy in automatic extraction of bibliographic data (better than 80%) will be presented at the European Conference on Digital Libraries in September 2001 in Darmstadt, Germany. Some problems remain, and we would like to find solutions, if possible:

1. There is no one good way to determine content type of a URL. It is still the case that `text/html` is returned for most text-based online documents, even if they are `.xml` or `.xhtml` documents. The MIMETYPE `application/xml` has been slow to come into widespread use.

2. How to find additional URLs to include in the <`reflink`> elements is an open problem, probably a basis for another research project. Should one use online databases? Google? services such as ResearchIndex?

3. Getting the date of publication is often problematical. Various sources are the: URL of the item, <`meta`> elements at the start of the paper, and references from other items to this publication. The date is most often *not* included in the content of the paper itself.

4. Handle servers are not reliably available yet. They would be useful to map bibliographic data to a DOI, and from the DOI to full text. A handle server should be used to find Surrogates, for example.

5. We would like to repackage the implementation from its current three packages into one, perhaps called `edu.cornell.edlrg.reflink`. Another advantage of this is that method status (public, protected, private) would make more sense, because only one package is involved.

We have also produced a number of technical reports, some of which will be turned into formal publications ([2, 4, 9]).

References

[1] The Open Citation Project. <http:opcit.eprints.org>.

[2] D. Bergmark. Automatic extraction of reference linking information from online documents. Technical Report TR 2000-1821, Cornell Computer Science Department, Nov. 2000.

[3] D. Bergmark. Link accessibility in electronic journal articles. Technical Report TR 2000-1793, Cornell Computer Science Department, Mar. 2000. <http://www.cs.cornell.edu/bergmark/LinkAnalysis.ps>.

[4] D. Bergmark, W. Arms, and C. Lagoze. An architecture for reference linking. Technical report, Cornell University, Digital Library Research Group, Oct. 2000. TR 2000-1820.

[5] D. Bergmark and C. Lagoze. An architecture for automatic reference linking. In *ECDL, September 2001*, Sept. 2001. <http://www.cs.cornell.edu/bergmark/www10.pdf>.

[6] S. Hitchcock, L. Carr, W. Hall, S. Harris, S. Probets, D. Evans, and D. Brailsford. Linking electronic journals: Lessons from the Open Journal project. *D-Lib Magazine: The Magazine of Digital Library Research*, Dec. 1998.

[7] S. Hitchcock, L. Carr, Z. Jiao, D. Bergmark, W. Hall, C. Lagoze, and S. Harnad. Developing services for open eprint archives: Globalisation, integration and the impact of links. In *5th ACM Conference on Digital Libraries, San Antonio, Texas, June 2 - June 7, 2000, also titled ACM Proceedings of Digital Libraries, 2000 (DL2000)*, San Antonio, Texas, 2000.

[8] C. Lagoze and J. Davis. Dienst: An architecture for distributed document libraries. *Communications of the ACM*, 38(4):47, Apr. 1995.

[9] S. Payette and C. Lagoze. Value-added surrogates for distributed content. *D-Lib Magazine: The Magazine of Digital Library Research*, 6(6), June 2000. <http://www.dlib.org/dlib/june00/payette/06payette.html>.

[10] I. Pila. CrossRef: The central source for reference linking. <http://www.crossref.org/>.