Reference Linking the Web's Scholarly Papers

Donna Bergmark* and Carl Lagoze†
Cornell Digital Library Research Group

November 7, 2000

Abstract

Along with the explosive growth of the Web has come a great increase in online scholarly literature. This literature comes in many forms. Informal online archives are repositories for papers and technical reports. Proceedings are more and more commonly published on the web. The collection of online journals is growing. Thus the web is becoming an efficient resource for up-to-date information for the scientific researcher, and more and more researchers are turning to their computers to keep current on results in their field. Not only is Web retrieval usually faster than a walk to the library, but the information obtained from the Web is potentially more current than what appears in printed publications.

The increasing proportion of online scholarly literature makes it possible to implement functionality desirable to all researchers – the ability to access cited documents immediately from the citing paper. Implementing this direct access is called "reference linking".

This paper describes the object oriented approach the Digital Library Research Group at Cornell has taken to help solve the reference linking problem. This approach employs *value-added surrogates* to enhance Web documents with reference-linking behavior. Given the URL of an online paper, a Surrogate object is constructed for that paper. The Surrogate fetches the content of the document and parses it to automatically extract reference linking data. User applications can then use the surrogate to access this reference, encoded in XML, via a well-defined API.

We use this API to reference link the D-Lib magazine, an online journal of technical papers relating to digital library research. Currently we are (automatically) extracting reference linking information from the papers in this journal with a rate of near 80% accuracy.

1 Background and Motivation

Reference Linking is actually an old idea. Classical reference linking arose from a desire to study citation patterns among scholarly articles. The Science Citation Index, founded by Eugene Garfield in the

*DARPA/CNRI Grant #2057/57-02
†NSF Grant # IIS-9907892

70's, was invented to do just that, and was a spectacular success. It was, however, based on human labor. For every paper examined, the staff captured that paper's metadata, and then went to the reference section and did the same for each reference there, or at least for those references to journals covered by the SCI.

As a result, one could look up links using the Science Citation Index and build a graph as shown in Figure 1. From this graph we can observe that Paper C has 4 references, that Papers C, D, and G have been analyzed, that Paper A has two citations, and that Papers C and G are bibliographically coupled (i.e. they have a reference in common). The links in the graph are *explicitly* contained in the Science Citation Index.

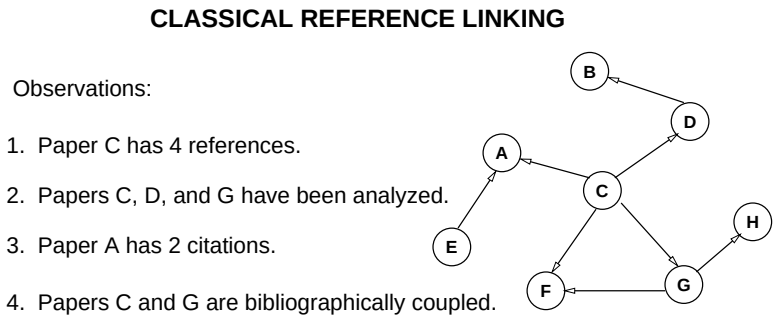

Figure 1: Classical Reference Linking

We then fast-forward some 25 years to the current time, where there is a growing amount of scholarly literature online. Much of this has HTML links to other works on the web. As in classical reference linking, the references are inserted by authors. Some references are accompanied by URLs, but not all.

Unlike SCI and classical reference linking, citations cannot be directly discovered from the Web. It is a daunting task to analyze items on the web to find out who might have cited a paper of interest.

Figure 2 shows how interlinked papers on the Web might exist. The graph is implicit, defined by links between papers. It is likely to be quite large. From the fragment shown here, we can deduce the HTML page C has four links in it to other HTML pages; page A has at present two links to it; and papers C and G are linked to a common page. But, discovering this fragment from traveling the web is nearly impossible. The graph exists *implicitly* on the Web.

In our reference linking project we are aiming somewhere between the classical view and what exists today on the web. We wish to make the graph in Figure 2 explicit, as well as supply additional links where possible. We hope to augment the work currently being done by CrossRef (which grew out of the DOIX project described by Atkins [1]) to link together the online copies of a group of scientific and technical journals. Our work is directed at online work not covered by the DOI initiative, and is in partnership with ECS at Southampton University which is reference linking arXhiv, the technical report repository at Los Alamos.

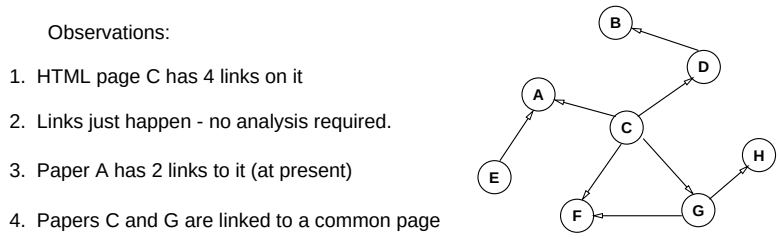

Figure 2: Linking on the Web

By making the links between online papers explicit, new applications are possible. Figure 3 is just one example of a reference linking application.

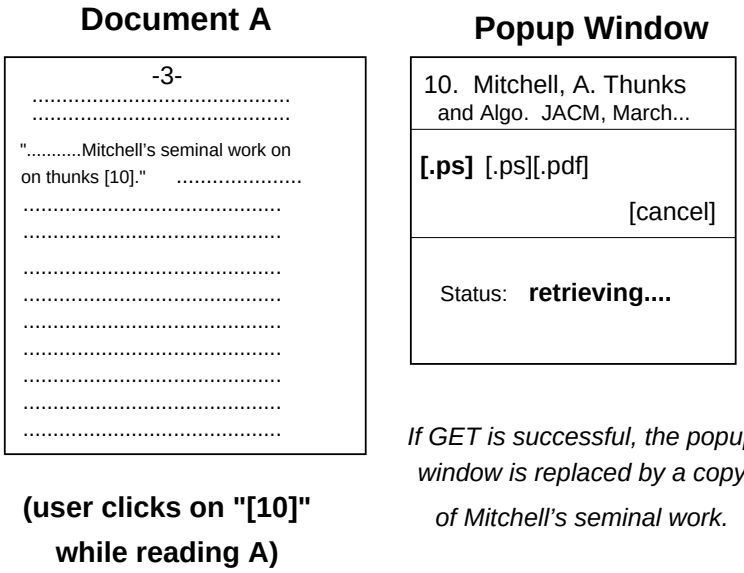

Figure 3: A Reference Linking Application

Imagine sitting in front of a computer screen, reading Document A (or hearing it on your speakers, etc.) and you come across an intriguing reference: "...Mitchel's seminal work on thunks[10]." If there is a copy of this work somewhere online, the "[10]" would be turned into a clickable live link, so that the user could start fetching that copy while continuing to read the original paper. One interface that would support this goal might be a JavaScript popup window that looks something like the one on the right side of Figure 3; the complete reference string is shown along with some choices of format (PostScript, PDF) in which the document might be retrieved; the user can retrieve one of these or cancel.

Implementing the functionality shown in Figure 3 requires solving at least two problems: 1) Figuring out that "[10]" is a reference and that it matches the reference string, [*10*] *Mitchell, A. Thunks and Algol...*; then parsing the reference string to decide what work it is and whether it is linkable (this is a *tough* problem!) and whether it is something we've seen before so we can credit Mitchell with a citation.

2) Turning the "[10]" into a live link. In HTML and PDF you can

turn this into an anchor that can be clicked. For other formats some kind of auxiliary display is needed.

In any case the first problem is one of *analysis* and the second is a *presentation* problem. Our work has been concentrating on the analysis problem, which is the extraction of reference linking data from online literature. An API, to be described in this paper, is responsible for supplying this data to client applications.

2 Definitions

The previous section was a quick introduction to reference linking. In this section we present some basic terms and definitions, so that we can explore the problem in more detail.

2.1 Items and Works

There are two different documents contained in Figure 3: there is Document A which the user is reading, and there is thing B, a work by Mitchell, which is referred to by A. There is a subtle, but important, difference between A and B. A is an *Item*, something that has a format type, is online, and can be analyzed by a computer program. B is a *Work*, or an abstraction of a paper. This work will also exist in the form of zero or more items. Some of them may even be online. We say that B is one of A's references. It probably even shows up in a section of A titled `References`.

The words we just defined - *Work* for the abstract paper and *Item* for a concrete instance of that paper – is taken from the IFLA model [8, 10], shown in an abbreviated form in Figure 4. Since in dealing with online literature one only cares about Works and Items, we ignore the middle two levels. In our experience with scholarly and scientific publications, the reference is usually to the Work (rather than to, say, a specific instance of that work held in a particular collection, in a particular format or manifestation). See Svenonius [11] for a good philosophical discussion of what a work is.

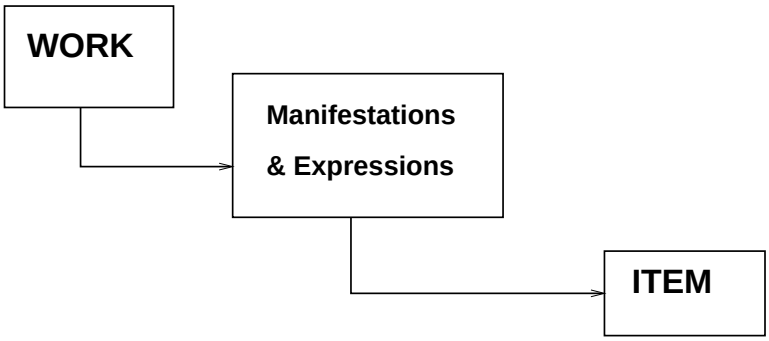

Figure 4: IFLA Model

In the rest of this report, we drop the capitalization of work and item, but continue to distinguish between the two terms.

2.2 References and Citations

Going back to Figure 3, item A references work B, an abstraction of a paper by Mitchell. If in fact, a copy of Mitchell's work can be found online, then it is a *linkable reference*. A *citation* is the inverse of reference. Here, the abstract paper of which A is an instance is a citation of B. Tracking citations is not immediately needed for reference linking, but is a valuable addition to any reference linking service. There is, of course, a subtle difference in tractability of making, for an item, its list of references and its list of citations. The labor involved in finding the citations is what made the SCI a huge success. The main thing to note here is that like references, citations are works.

In the remainder of this paper, the term *reference* will refer to the reference in the text of the paper being analyzed, e.g. `[10]` in Figure 3, the *context* will be the sentence containing that reference, and the *reference string* will be complete description of the work, e.g. `10. Mitchell. A. Thunks ....`

2.3 Repository

A repository, also known sometimes as an archive, is any collection of online items, e.g. an online journal, a department's technical reports, or a person's online bibliography. A repository has some sort of special identifier. This project does not build repositories; it only analyzes them for a variety of reference linking information.

2.4 Intralink and Interlink

If we analyzed every item in a repository, then we can *intralink* that repository, since if one item references a work that is itself an item in the repository, we have a linkable reference.

If we analyze several repositories, then we can *interlink* items in these repositories. The extreme limit of this work, as more and more items are analyzsed, is to interlink the Web, at least the scholarly side of it.

3 The Reference Linking API

The reference linking architecture developed at Cornell is unique in several respects. First, because we are aiming to link online literature which has not been indexed by hand or accompanied by author-supplied metadata, we are taking an automatic approach to parsing document source in order to extract the item's metadata as well as the item's references' metadata.

Secondly, we have a different approach to collecting and storing reference linking data. Most other reference linking projects (e.g. Open Journals[5] and ResearchIndex[6]) use databases to store information about works and do a lot of "database crunching". For example, there would be one database of all the titles, and perhaps another database of all the authors, and a third with references. Instead of

using databases to store this information, we use item surrogates. A *surrogate* is a digital object that encapsulates reference linking information ??and behaviors?? relating to one item ??in a repository.?? Reference linking data is thus distributed across the collection of surrogate objects, and all the data relating to one item is grouped together within a single surrogate.

This use of surrogates for reference linking is consistent with our overall architectural approach in digital library research at Cornell. We make use of "value-added surrogates" [9] as a vehicle for endowing digital objects with a wide variety of extensible behaviors (e.g. preservation, access management).

A third unusual aspect of reference linking at Cornell is that we do not put together one monolithic reference linking service. Rather, we provide an API on top of which such services can be built. Such an approach has been quite successful in other, unrelated endeavors (see, for example, [2]).

Having an API specifies the operational semantics of reference linking; it also allows us to cleanly separate the analysis phase of reference linking from the presentation phase. The advantage of creating an API is that no decision is made in advance of what the data should be used for.

3.1 How the API Works

The combination of surrogates and an API essentially allows us to walk up to a paper and ask it "what are your references" and "what is your metadata?" One surrogate is instantiated for each repository item, and can answer questions about that item. In fact, the API includes a set of methods, where each *method* is simply one of the questions or requests that can be directed at the surrogate. Each surrogate answers the same set of questions (to the best of its ability).

A typical use of the API would be to analyze all the papers in some collection or repository. This architecture is depicted in Figure 5. The central column represents some repository of network-accessible documents. The items listed in this column are linkable (they are online) and therefore analyzable (we have their bits).

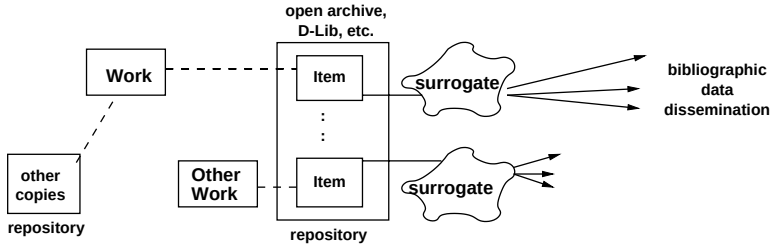

Figure 5: An Architecture for Reference Analysis

On the left are drawn the works that the items represent. Any work might have several copies spread across several archives. All of these copies are "items" corresponding to that work. If more than one copy of a work is encountered, we could pool the information collected

so that both surrogates have consistent data, but this requires either that the surrogates be able to find and communicate with each other, or that there be a central database. Arranging for the surrogates to communicate among themselves is an interesting research problem; for now we keep a small database of works seen so far which at least allows sketchy information to be updated.

To the right of the archive items are the surrogates, shown as "blobs". They know how to disseminate bibliographic data about the item, and indirectly, about the work. As stated above, client applications ask the questions and then display or otherwise use the results. The API supplies the data.

4 Two-Phase Architecture

Up to this point, we have discussed the API: its methods and outputs. However, recall that the application shown in Figure 3 requires solving two very difficult problems: analysis, to find the references; and presentation of the live references. Figure 6 shows this two-phased architecture with structured data – XML – at the interface. Suppose in this case that the XML is in response to "`getLinkedText`".

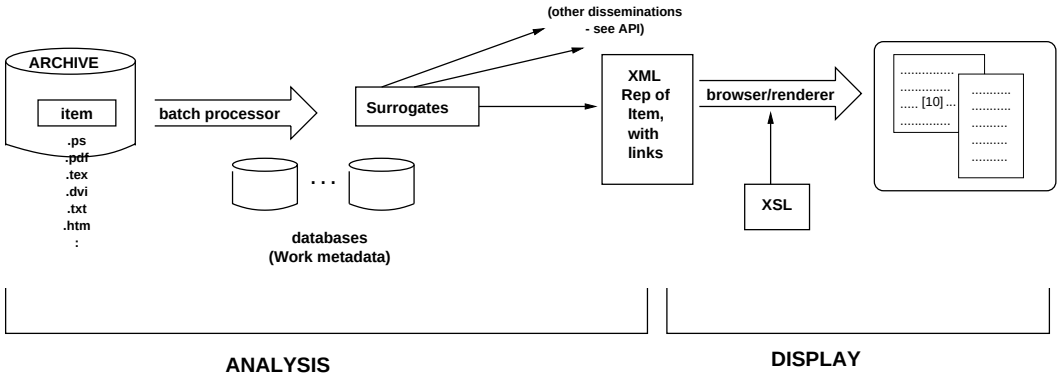

Figure 6: Overall Reference Linking Architecture

.

The "links" contained in the XML output of `getLinkedText()` are elements of our own devising. They can be translated (by a XSLT processor) into "actionable links", such as HREF's, XLinks, or OpenURLS. Our link elements, `<reflink>`, are sufficiently rich to point to various on-line copies of the reference, to retrieve the reference string itself, and so on. Figure 7 shows an example `<reflink>` element, along with how it might look after being converted into an XLink of type simple.

4.1 Components of the API

Surrogates could answer a number of different questions, but the four primary questions in the Cornell reference linking API are:

Original Text:

```
... it was said [5] that ...
```

Linked Text with custom tag:

```
<reflink ord="5" author="last-name-of-first-author"
title="title of this work"
year="1999"
url="http://www.some.org/filename">[5]</reflink>
```

Linked Text with XLink:

```
<ref-xl xmlns:xlink="http://www.w3.org/1999/xlink"
xlink:type="simple"
xlink:href="http://www.some.org/filename">[5]</ref-xl>
that...
```

Figure 7: A linked XML Item, first with reflink then with XLink.

- `getLinkedText` – contents of the paper (as data) augmented with reference linking information. This method would be invoked by browsers that wanted to display the document with some of its references turned into anchors of live links, as in Figure 3.
- `getReferenceList` – this interface would be used by applications that wish to know what references are contained in this paper. For example, if one were building the SCI, this would be the question to ask, along with the next one.
- `getMyData` - this returns that paper's own metadata. This is not directly related to reference linking, but is required for building up citation relationships. It could have other uses; for example, one client might have a button labeled "get BibTeX"; when the button is pushed, the client invokes `getMyData` on the surrogate, and reformats the results into something suitable for cutting and pasting into a LaTeX bibliography.
- `getCurrentCitationList` – the list of works citing this paper to the best of the surrogate's knowledge. As stated before, this function is not strictly required for reference linking, but would be very useful to client applications that want to know what other documents cite this one, as they might be related or provide more current information. If online, we have a *linkable citation*.

In addition to these API methods, surrogates can be told to save themselves and they can be resurrected from stored data.

4.2 Output from the API

Figure 5 showed the surrogates disseminating bibliographic information about their items, in response to a particular method in the API being invoked. Each method returns a byte-stream of structured data coded in XML, to permit further processing. For example, one method in the API is `getReferenceList`, which returns harvested metadata for each reference contained in a repository item, such as its title, publication, context in which it was cited, year and authors. This data, encoded in XML, is suitable for further processing by other applications.

Figure 8 shows what part of the XML information disseminated by `getReferenceList` might look like. This component is the second referenced (`ord="2"`) of this surrogate's item.

First comes bibliographic data related to the reference work. We are using Dublin Core for convenience, so for example, dates must be in `CCYY-MM-DD` format.

Next comes item-related information, such as the reference string exactly as it appeared in the item (enclosed in a `<literal>` element and entified), and all the contexts in which the work was cited. The context is usually one complete sentence, as shown near the bottom of Figure 8. Note the "[2]" in the context. Since the Maly paper does have a URL, this may become the anchor of a live link in any text returned by a call to this surrogate's `getLinkedText` method.

```
<api:reference_list length="17"
      xmlns:api="http://www.cs.cornell.edu/cdlrg/..."
      xmlns:dc="http://purl.org/DC">
  <api:reference ord="1">
          :
          :
  <api:reference ord="2">
    <dc:title>
       Smart Objects, Dump Archives: A User-Centric,
       Layered Digital Library Framework
    </dc:title>
    <dc:date>1999-03-01</dc:date>
    <dc:identifier>10.1045/march99-maly</dc:identifier>
    <dc:creator>K Maly</dc:creator>
    <api:displayID>
       http://www.dlib.org/dlib/march99-maly/03maly.html
    </api:displayID>
    <api:literal tag="2.">
       Maly K, "Smart Objects, Dumb Archives: A User-Centric,
       Layered Digital Library Framework" in D-Lib Magazine,
       March 1999,
       &lt;http://www.dlib.org/dlib/march99-maly/03maly.html&gt;.
    </api:literal>
    <api:context list>
      <api:context>
         The need for standards to support the interoperation of
         digital library systems has been reported on before in
         D-Lib[1],[2] as have efforts to discover common ground
         in related standard processes(Dublin Core and INDECS[3]).
      </api:context>
    </api:context list>
  </api:reference>
          :
          :
</api:reference_list>
```

Figure 8: XML for a Reference Object

5 A Java Implementation of the API

The API can be easily implemented in Java, Perl, or even as part of a larger protocol. Our Java implementation will briefly be discussed in this section. Readers uninterested in implementation details may skip directly to Section 4.

The API is implemented as three packages, only one of which (`Linkable.API`) is needed by client applications. The other packages include one for parsing of source documents (`Linkable.Analysis`) and another for helper routines (`Linkable.Utility`).

Only one parameter is required for constructing a surrogate object, the URL of the item to be parsed. The surrogate invokes one or another analyzer depending on the item's format. Typically the item is translated to XML before further analysis. Formatting hints are retained in the XML version, to enable decomposition of the item into header, body, and reference sections.

When the surrogate is returned to the client application, the item has been parsed and preliminary reference data has been stored into data fields within the surrogate. Invoking one of the four methods on the surrogate, e.g. `getLinkedText()`, causes further analysis of the reference data and culminates in an XML byte array.

The surrogates can be constructed and used on the fly and then discarded, or they may be stored in a repository for further use. This allows for a wide range of applications, from constructing a database of citation information to providing a completely dynamic reference linking service.

The Java implementation consists of less than 6000 lines of code and uses both DOM and SAX parsing of XML data.

6 Applications

The reference linking API can be used for a large variety of applications. This section briefly sketches two of them.

6.1 A Simple Display Application

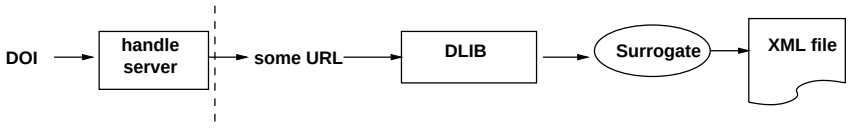

Figure 9: A Simple Reference Linking Application

In Figure 9, the client application is given the URL of some online item. (Alternatively, the application might instead be handed a DOI, and then use a handle server[4, 7] to get a URL.) In the case of Java, the application instantiates a surrogate object, passing it that URL. Instantiation of a surrogate is what causes an item to be analyzed. All further interactions with the reference linking API are via this surrogate.

The right-hand side of Figure 9 shows the client application invoking various methods on the surrogate. Here is a sample snippet of Java code that might appear in such an application:

```
Surrogate s = new Surrogate ( url );
clientDisplay ( s.getLinkedText() );
```

This application uses the API to obtain the linked text for the item located at the specified url; the result of this request is a XML byte array, which is then passed to a routine, `clientDisplay()`, which will display the linked text to a user. For a display similar to that shown in Figure 3, the steps in the presentation would be as follows:

1. Run XSLT or a similar translator to convert the API's `<reflink>` elements into an "actionable link", such as a URL, an XLink, an OpenURL, or JavaScript code.
2. Display the translated XML object to the user.
3. When the user clicks on a reference that has a live link, bring up the *retrieving...* dialogue, showing the complete reference string, and show what formats exist for this work.
4. If the user clicks on the cancel button, quit. Otherwise retrieve the format selected by the user and display it in a separate window.

This example has shown how the reference linking API would be used on the fly to display to a user an online item with live links to linkable references.

6.2 Reference Linking the D-Lib Magazine

We are currently using the Java implementation of the reference linking API in batch mode to analyze D-Lib articles. D-Lib is an online journal that has been appearing eleven times a year since July 1995; it makes an excellent test bed for automatic extraction software because there is little editorial imposition on the format of the papers submitted to the journal, and therefore provides a wide selection of paper layouts. All D-Lib articles are written in HTML.

Figure 10 illustrates the major steps in analyzing a D-Lib paper. The application, running from the command line, (1) inputs a file of D-Lib URLs. (The file was automatically generated from D-Lib table of contents pages.) For each URL, the application (2) constructs a surrogate object, which proceeds to extract reference linking information, and (3) returns itself to the application; the application simply (4) stores the surrogate. The Java code to perform this processing is as follows:

```
Surrogate s = new Surrogate ( url );
s.save();
```

(The reference linking API contains `save()` and `restore()` surrogate methods). At some future date, that surrogate can be used to respond

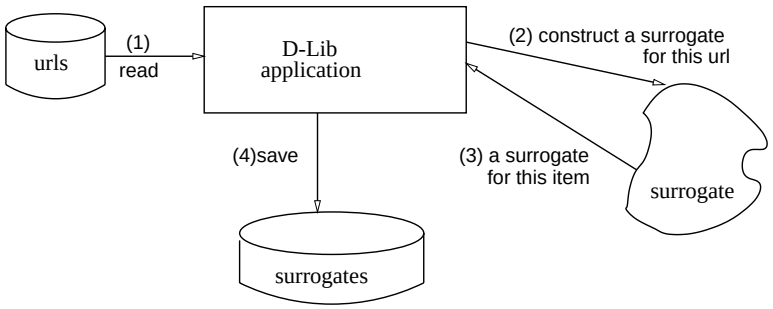

Figure 10: The application to intralink D-Lib.

to Open Archive requests, to provide data for a reference linking application, to build a citation database, etc.

The next section evaluates the accuracy of parsing D-Lib papers. If more accuracy is needed, it is certainly possible to run an offline "upgrade" procedure which allows a human to edit private surrogate data. When an edited surrogate is resurrected, it will have the upgraded information. Currently, however, we see no need to do this for D-Lib.

7 Results

Because our approach extracts all reference linking and bibliographic data *automatically*, it cannot be expected that the data will be 100% accurate. Fortunately (unlike for library services) a reference linking service for online documents does not have to be completely accurate. Rather, one aims for the "sweet spot" where at least one copy of the reference can be retrieved (so *recall* is not that important), and where there are not too many false links (*precision* has to be good enough). We believe that an 80% accuracy level is sufficient for most applications.

Our current results show that we are very near to achieving this desired level. The accuracy of our parsing has improved considerably as more and more papers have been parsed.

There are two categories of parsing errors: incorrectly extracting bibliographic data about the item being analyzed; and incorrectly parsing the reference strings contained in the analyzed items. We therefore devised a performance metric based on both of these inputs.

For each item analyzed, the *item accuracy* is the number of elements parsed correctly, divided by the total number of elements in the item. Specifically, some of the elements used are: the item's title, the item's authors (each author counts as one element), the item's year of publication, the reference contexts (each contex counts as one element) and the average reference accuracy times the number of references.

The *reference accuracy* for one reference string is the per centage of its elements that are correctly parsed. These elements include: title, each author, year, contexts, and URL (if present). To give a concrete feeling for how the metric is calculated, Figure 11 shows a hypothetical item, the parsing results, and calcluation of the item accuracy.

Reference Accuracy (16 items)

Ordinal	Number Elements	Number correct	Reference Accuracy	Ordinal	Number Elements	Number correct	Reference Accuracy
1	7	4	57	9	4	3	75
2	5	1	20	10	5	5	100
3	5	5	100	11	5	5	100
4	5	5	100	12	8	6	75
5	7	7	100	13	5	2	40
6	5	5	100	14	6	6	100
7	4	1	25	15	5	1	20
8	7	6	86	16	4	1	25

Total Reference Accuracy = 1123; Average = 1123/16 = 70.19

Item Accuracy

What	How Many	How Many Correct	%
title	1	1	
authors	2	0	
year	1	1	
contexts	8	8	
references	16	11	
Totals	28	21	75%

Figure 11: Example of Item Accuracy for hypothetical item with 2 authors, 16 references and 8 reference contexts. First calculate the average Reference Accuracy (top figure, 70%). Then in the bottom table, use 70% of 16 (11) references as the average accuracy of reference parsing. The Item Accuracy metric is then 21 divided by 28, or 75%.

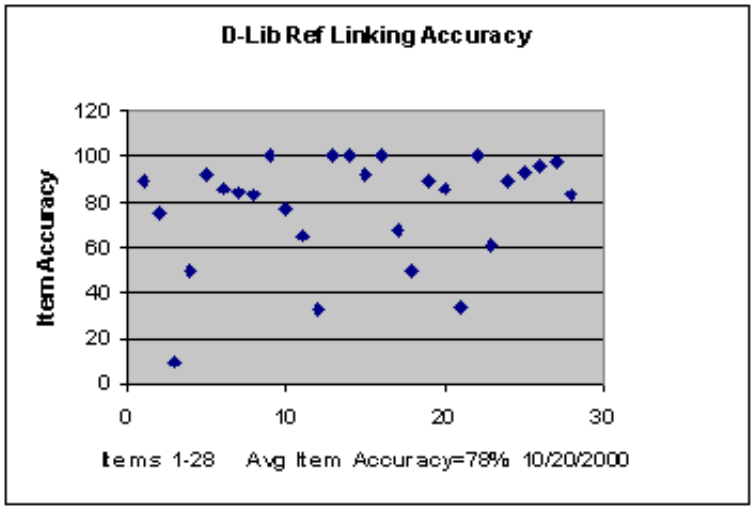

Figure 12: Item Accuracies for a set of 28 D-Lib papers

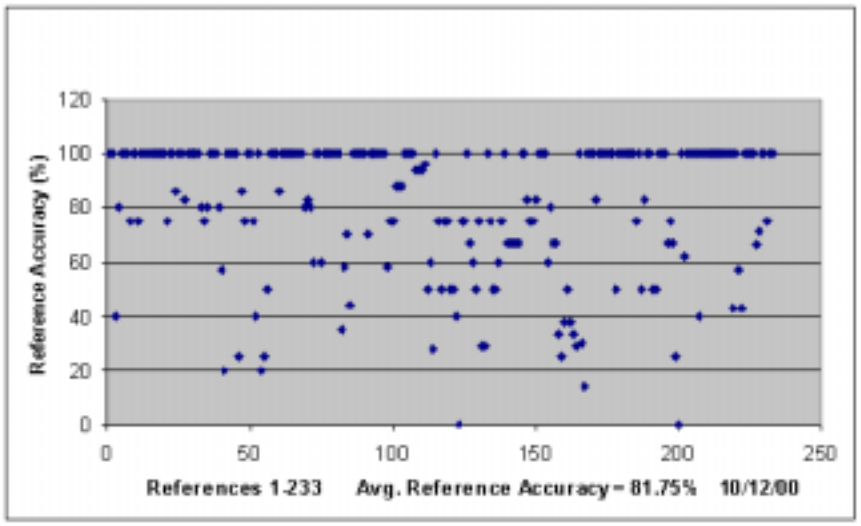

Figure 13: Reference Accuracies for a set of 26 D-Lib papers

We have processed a set of 29 D-Lib papers. Of this number, 3 were not able to be converted to XML (i.e. XHTML) and so were discarded. **NOTE: RESULTS HERE ARE NOT YET COMPLETE. Plots 12 and 13 will be extended as more D-Lib papers are analyzed.** For the remainder, item accuracies were determined by human inspection of the data contained by stored surrogates; the item accuracy is plotted in Figure 12. As can be seen, most of the items lie above our desired 80% level of accuracy.

The collection of references is much larger and varied than the items in a single repository. Figure 13 shows the accuracy of parsing the references in the same set of D-Lib papers. Again the majority of the references parse to the desired degree of accuracy, with a surprising number parsed perfectly. In fact the overall level of accuracy is above 80%.

While the overall averages are acceptable, it is harder to get accu-

D-Lib 1995 to August 2000: Metadata Extraction			Subsample	
Description	Number	% of Total	Number	% of Total
Number of D-Lib papers:	280	100	29	100
Converted to XHTML:	220	79	26	90
Extraction is Perfect			5	19
Good (70% or more)			13	50
Poor (below 70%)			8	31

Table 1: Number of D-Lib items whose bibliographic data was correctly extracted. The rightmost 2 columns are the subset of D-Lib papers processed to date. The bottom 3 rows are a per centage of row 2, that is, of the items that could be turned into XHTML.

D-Lib 1995 though August 2000			Subsample	
Description	Number	% of Total	Number	% of Total
Number of References:		100	266	100
Parsing is Perfect			149	56
Good (70% or more)			55	21
Poor (below 70%)			62	23

Table 2: Number of correctly parsed references in D-Lib Papers. This table is incomplete. The two right-most columns show the results for 28 D-Lib papers that contain 266 references

racy concentrated into one place – that is, all the item's metadata and each of the item's references gets parsed correctly. We therefore look at how often it was possible to perfectly extract a paper's metadata, which would correspond to the number of times the user would get a perfect answer in response to the `getMyData()` method. We also looked at how often references in a paper are perfectly parsed, which corresponds to the quality of the response to the `getReferenceList` request. The results are contained in Tables 1 and 2. **NOTE: RESULTS HERE ARE NOT YET COMPLETE. Tables 1 and 2 will be filled in as more D-Lib papers are analyzed.**

8 Conclusions

This project shows that automatic extraction of reference linking information is very difficult to get right. The extraction of reference linking data is difficult mainly because parsing text produced by many different authors in many different formats with many different conventions is problematical. However, we have found that there are a relatively limited set of variations in format, and have successfully developed grammars to handle most of them. A separate paper [3] discusses this problem in more detail, and presents some algorithms for extracting reference linking information.

At this point we are analyzing papers, examining the errors, patch-

ing up the Java API, and then analyzing new papers. As each additional paper gets processed, the implementation improves a little. If we look at the *proportion* of elements that can be correctly extracted from an item or from a reference, we have 79% item accuracy and more than 80% reference accuracy.

Of course, using any available metadata would improve this accuracy. But because such metadata has only recently begun to be available, we extract this information ourselves. It should be noted that the Open Archives initiative asks authors to submit metadata along with their papers. A tool like that described in this paper would be helpful in providing an initial data set which could then be refined by the author during the submission procedure.

ResearchIndex also automatically extracts data from items discovered online, and does a remarkably good job. Its main strength lies in applying clustering methods and other artificial intelligence techniques to the analyzed material. Our software does not incorporate AI methods, but does almost as well.

The work done so far indicates that the architecture and design for the reference linking API are sound. The object-oriented API makes it exceptionally easy to build new reference linking applications.

References

[1] H. Atkins, C. Lyons, H. Ratner, C. Risher, C. Shillum, D. Sidman, and A. Stevens. Refererence linking with DOIs: A case study. *D-Lib Magazine*, 6(2), February 2000. <http://www.dlib.org/dlib/february00/02risher.html>

[2] D. Bergmark and S. Keshav. Building blocks for IP telephony. *IEEE Communications Magazine*, 38(4):88–94, April 2000.

[3] Donna Bergmark. Automatic extraction of reference linking information from online documents. Technical report, Cornell Computer Science Department, October 2000. in preparation.

[4] Priscilla Caplan and William Arms. Reference linking for journal articles. *D-Lib Magazine*, 5(7/8), July/August 1999. <http://www.dlib.org/dlib/july99/caplan/07caplan.html>

[5] Steve Hitchcock, Les Carr, Wendy Hall, Stephen Harris, S. Probets, D. Evans, and D. Brailsford. Linking electronic journals: Lessons from the Open Journal project. *D-Lib Magazine*, December 1998.

[6] Steve Lawrence, C. Lee Giles, and Kurt Bollacker. Digital libraries and autonomous citation indexing. *IEEE Computer*, 32(6):67–71, 1999. <http://www.researchindex.com>

[7] Norman Paskin. DOIs and reference linking, February 1999. A presentation to the NISO/NFAIS/SSP Linking Workshop, available online at http://www.niso.org/paskin.html

[8] Norman Paskin. E-citations: actionable identifiers and scholarly referencing, 1999. <http://www.doi.org/citations.pdf>

[9] Sandra Payette and Carl Lagoze. Value-added surrogates for distributed content. *D-Lib Magazine*, 6(6), June 2000.

[10] K. G. Saur. Functional requirements for bibliographic records, 1998. UBCIM Publications - New Series Vol. 19.

[11] Elaine Svenonius. *The Intellectual Foundation of Information Organization*. M.I.T. Press, 2000.