Integration of Simultaneous Searching and Reference Linking across Bibliographic Resources on the Web

William H. Mischo, Thomas G. Habing, and Timothy W. Cole
Grainger Engineering Library Information Center
University of Illinois at Urbana-Champaign
1301 West Springfield Avenue
Urbana, IL 61820 USA
+1 217 333 7497, +1 217 244 4425, and +1 217 244 7837
w-mischo@uiuc.edu, thabling@uiuc.edu, and t-cole3@uiuc.edu

ABSTRACT
Libraries and information providers are actively developing customized portals and gateway software designed to integrate secondary information resources such as A & I services, online catalogs, and publishers full-text repositories. This paper reports on a project carried out at the Grainger Engineering Library at the University of Illinois at Urbana-Champaign to provide web-based asynchronous simultaneous searching of multiple secondary information resources and integrated reference linking between bibliographic resources.

The project has tested two different approaches to simultaneous broadcast searching. One approach utilizes custom distributed searchbots and shared blackboard databases. The other approach uses event-driven asynchronous HTTP queries within a single web script.

The reference linking implementation is built around the application of OpenURL and Digital Object Identifier (DOI) technologies and the CrossRef metadata database within a proxy server environment.

Categories and Subject Descriptors
H.3.3 [Information Storage and Retrieval]: Digital Libraries; H.3.7 [Information Storage and Retrieval]: Information Search and Retrieval – query formulation, retrieval models, search process; H.3.5 [Information Storage and Retrieval]: Online Information Services; H.2.4 [Database Management]: Systems – distributed databases; H.2.5 [Database Management]: Heterogeneous Databases.

General Terms
Design, Standardization, Experimentation, Performance

Keywords
Digital Library, Asynchronous Simultaneous Search, Reference Linking, OpenURL, Searchbots

1. INTRODUCTION
The continued development and enhancement of customized library portal and gateway sites designed to provide integrated user access to information resources has become an important part of the work done within academic libraries [6], [13]. This mission has been defined as a key role for libraries and librarians in the era of the user-centered, customized ‘digital library’ [11], [3], [4], [8].

Library portal services provide users with integrated paths to content of remote and local information providers and gateway services. Such portals attempt to bring together and integrate disparate e-resources such as full-text publisher repositories, secondary information resources and local finding tools/ aids in order to provide users with a systematic ‘view’ of the information landscape. This is particularly applicable within the full-text content that forms the cornerstone of digital libraries. The two key foci of library, publisher, and vendor portals are the twin initiatives of: (1) robust search/navigation, with (2) the ability to link everywhere from anywhere—including between discrete full-text articles and from A & I services and the online catalog to full-text [[11]].

Central to the implementation of these desirable integration services is the ability to simultaneously search multiple information resources and to efficiently and effectively link from retrieved resources to other, closely related information resources located elsewhere. There are a number of vendor and library systems in various stages of production or development that help address these needs to a limited degree. It is quite clear that vendors recognize reference linking and simultaneous search capabilities as important components in overarching digital library system architectures.

This paper looks at linking and simultaneous searching technologies that can be implemented and configured effectively at the local library level. Implementation by librarians at a local level allows for greater, and in many cases more economical, customization and integration of library portal services and information resources.

The search technologies discussed in this paper include lightweight scripting approaches relying on existing, off-the-shelf, Web-based services and utilities and more sophisticated distributed approaches requiring advanced programming and local search agent creation and implementation. The linking technologies discussed here provide ancillary services such as a reference link resolution service based on Digital Object Identifier (DOI) technologies. To demonstrate the potential of these technologies, we provide real-world illustrations of how these technologies are being used at the University of Illinois at Urbana-
Champaign (UIUC) Grainger Engineering Library Information Center.

The Grainger Library Search Aid project described here is designed to: (1) assist users in the selection of appropriate bibliographic databases by displaying search argument matches from candidate databases; (2) provide seamless cross-database searching by transforming user search arguments into their proper form for each database; (3) provide article level and e-journal Web site access to a broad range of publisher full-text repositories; and (4) utilize OpenURL and DOI standards for reference linking. The success of this proof-of-concept research suggests new opportunities for libraries and librarians in the management and integration of the burgeoning number of primary and secondary online information resources.

2. PREVIOUS & ONGOING WORK ELSEWHERE

One of the earliest multiple search vendor systems was the DialIndex system developed by the Dialog Corporation for their family of online databases, initially accessible over telnet connections. This system accepts user-entered search arguments expressed in Dialog command language and applies them to all the databases included in selected subject groups. DialIndex then returns the number of hits for each database. Any subsequent searching has to be done by re-entering the specific database. The primary value of this system lies in search strategy development. The Dialog Web version at http://www.dialogselect.com/main.html continues this technology of performing simultaneous searches over a set of databases.

Several OPAC vendors are committed to developing cross-content simultaneous search engines as part of their extended library system offerings. The MetaLib service from Ex Libris (see http://www.exlibris.co.il/metalib/) provides simultaneous search capability and the EnCompass (see http://encompass.endinfosys.com/) module of the Endeavor system will also provide this function. The Innovative Interfaces MetaFind (see http://www.iii.com/html/products/p_map.shtml) system is a universal search interface that allows access to multiple resources with a single search from a consistent interface.

In addition, the information providers are offering their versions of simultaneous search systems. Ovid is developing its Multifile and Deduping technologies, which operate over multiple A & I databases (see http://www.ovid.com/documentation/features/v410/multifile.cfm). The Institute for Scientific Information (ISI) has announced the ISI Web of Knowledge system, which integrates journal, patent, and proceedings literature with Web resources and provides cross-resource searching. Likewise, the Gale Corporation has released the InfoTrac Total Access system which features simultaneous search capabilities over specified electronic resources (see http://www.galegroup.com/servlet/ItemDetailServlet?region=9&imprint=000&titleCode=INFO26&type=4&id=172049).

The search integration company Webfeat, with their Knowledge prism product, is marketing a sophisticated simultaneous database search system to libraries and corporations (see http://www.paratext.com/webfeat/).

On the library side, the University of California at San Diego developed a cross-database search system called the Database Advisor [5]. This system performed simultaneous searches over 25 science databases. The software and search technology developed in this project have been incorporated in the California Digital Library’s SearchLight system (see http://searchlight.cdlib.org/cgi-bin/searchlight). SearchLight features simultaneous searching of databases in several subject areas under the general categories of science/engineering and social sciences/humanities. Also, the Los Alamos National Laboratory has developed the FlashPoint system as part of their Library Without Walls Project [9]. FlashPoint provides a Web-based interface that performs a broadcast, parallel search of nine bibliographic databases.

Clearly, there is a great deal of vendor interest in developing an overarching system architecture that will provide simultaneous search and discovery over multiple information resources. In particular, the integrated library system vendors are developing platforms that attempt seamless access services over a hybrid information environment, which includes publisher and vendor digital collections, specified Web resources, local collections of digital metadata, and secondary resources such as periodical index databases and the library online catalog.

In the following sections of this paper the Grainger Engineering Library Information Center’s implementation of searching multiple databases simultaneously and linking integration will be described. First, the integration aspects of the project will be discussed. Second, two separate implementations of the simultaneous search will be described.

3. GRAINGER LIBRARY SEARCH AID

As a working example of advanced search and linking portal technology, the Grainger Engineering Library at Illinois has developed a Web-based portal that provides simultaneous searching of the ILLINET statewide online catalog, the Compendex, INSPEC, Current Contents, and Applied Science and Technology databases, plus the Google search engine.

In a manner transparent to the user, the Grainger Search Aid utilizes simultaneous searching of periodical index databases, full-text availability displays determined from the UIUC Library e-resources database, and full-text resolution and display using a DOI obtained from the CrossRef metadata database.

Figure 1 shows the system’s top-level interface in which users select the desired database services to search and enter search
terms. This is a relatively simple interface that permits single term author, title, keyword, institution, journal name, and ISSN searches or Boolean AND or OR combination searches from multiple fields. In Figure 1, the user has entered the semiconductor material AlGaAs (Aluminum Gallium Arsenide) in the keyword field to be combined with the author last name Holonyak. When the form is submitted, the Search Aid software takes the user-entered search argument and normalizes it into the format expected by each of the database systems.

The searches are then performed asynchronously and simultaneously, and the numbers of search hits are displayed as they are returned. Figure 2 is a completed search results page indicating the number of search matches in each of the selected database resources.

Figure 2. Number of search hits from each database

The short entry results are displayed to the user, via a proxy server, from the actual vendor system that was searched. All subsequent links pass through the proxy server and are redirected to the vendor system with session connection information intact. The search results from the different database systems are not combined together in any fashion and duplicate citations from different services are not eliminated. Rather, the user must view results from the desired single system. Multiple search service results must be viewed consecutively.

Figure 3 shows a user-selected short-entry search results display from the ISI Current Contents service. This database is accessed through an Ovid server. While the search results are displayed in the standard Ovid format, note that the proxy server has added links to the ‘Journal Web Site’ and also to the ‘Article Full Text’ when those journals are available to UIUC patrons. The availability of a full-text version of a specific journal title is determined by an on-the-fly lookup in the UIUC custom e-resources registry database. When it is determined that the library has licensed full-text access to the specific title, selected metadata elements, including the first author last name, ISSN, journal title, volume, beginning page number, and year are extracted from the Ovid display. This information in OpenURL form is then programmatically added to the full-text link in the value-added Ovid display shown here in Figure 3.

Figure 3. Short citations from Current Contents embellished with added links to the ‘Journal Web Site’ and ‘Article Full Text’

When the user clicks on the ‘Article Full Text’ link, the metadata elements contained in the link are used to perform a search in the CrossRef metadata database. From the returned CrossRef metadata results, the article DOI is extracted. The DOI is then resolved against the Handles Server to redirect the user to the specific article-level and full-text information at the publisher site. The metadata of the first retrieved citation is shown in Figure 4.

Figure 4. Full-text article accessed from publisher’s site

Work is being done to add other additional periodical index databases and Web search engines into the Grainger simultaneous search system. It should be noted that the simultaneous search
component and full-text article resolution function within the Grainger Search Aid could be applied separately within other software implementations, such as a direct proxy module. It is expected that vendors and publishers will continue to incorporate and refine their searching and full-text linking functionality using many of the same techniques developed for the Grainger software.

4. IMPLEMENTATION OF ASYNCHRONOUS SIMULTANEOUS SEARCHING OF BIBLIOGRAPHIC DATABASES

The capabilities illustrated in the Grainger Library Search Aid rely on two underlying technologies: asynchronous simultaneous searching and OpenURL-based reference linking technologies. The objective of an asynchronous simultaneous search application is to perform searches of multiple bibliographic database resources concurrently, displaying the numbers of search hits found from each resource as they are returned from the multiple search systems, as shown in Figure 2.

We have implemented two different technical approaches to simultaneously searching bibliographic resources over the web. Each approach uses an identical web search form, and can perform the same types of searches against the same resources. However, each form is submitted to a different Microsoft Active Server Page (ASP) script for processing. Each ASP script handles the simultaneous searching quite differently. The two separate approaches are detailed below.

4.1 Shared Blackboard used by Independent Searchbots

The first approach implements the simultaneous search by using ‘shared blackboards.’ Each bibliographic resource to be searched has a blackboard. Both the ASP script to which the searches are submitted and multiple, independently running ‘searchbots’ can asynchronously access the blackboards. These searchbots are implemented as individual executables running independent of the Web server.

Currently, the blackboards are implemented as Microsoft Access relational databases. However, nearly any database system that supports the Microsoft ODBC standard could be used. For a production implementation, a more scaleable database system such as Microsoft SQL Server or Oracle would be used. The database schema used is simple, consisting of only one table with several columns.

<table>
<thead>
<tr>
<th>SearchArg</th>
<th>Result</th>
<th>UserID</th>
<th>Timeout</th>
<th>NewEntry</th>
<th>BotID</th>
</tr>
</thead>
</table>

The SearchArg column stores the search query using the native syntax expected by the specific bibliographic resource. The ASP script derives this search string from the submitted HTML form parameters, before inserting it into the table. For example, the search shown in Figure 1 for author Holonyak and keyword AlGaAs becomes algaas.mp. and holonyak-$au. when inserted into the blackboard database for the Ovid INSPEC resource.

The Result column stores the result of the search. This column remains empty until a search is picked up and completed by one of the searchbots. When a searchbot completes a search, it updates this column with the first page of the search results. Unless there is an error, this will always be the HTML returned by the web resource as a result of the search.

The UserID column stores the IP address of the client requesting the search. The ASP script inserts this when it adds a new search to the table. It is primarily used for logging purposes.

The Timeout column stores the date and time that the query was inserted by the ASP script. This column is used to determine possible timeout conditions and prevent a search from running indefinitely.

The NewEntry column stores the state of the query. A value of yes indicates that this is a new entry that can be picked up by any free searchbot. When a searchbot picks up the query, it updates the value to working on. This prevents other searchbots from picking up the query. When a searchbot completes a search, and updates the Result column, it also updates this value to changed. This indicates that the search is complete, so the ASP script can retrieve the result from the database for further processing and display to the user.

The BotID column is the unique identifier of the searchbot that is servicing the query. This column is updated by the searchbot when it picks up the query. When there are multiple searchbots servicing a single web resource, this column is used to distinguish which searchbot did which query.

Optimistic concurrency control is used by the ASP script and by the searchbots to prevent concurrency problems when updating the database.

After the ASP script has inserted new rows into the blackboard databases of each resource that is being searched, it enters a loop where it waits for the submitted queries to be completed by the searchbots. The ASP script repeatedly checks the rows it has inserted in all of the blackboard databases until the NewEntry column has been updated to changed by the searchbot. Once the NewEntry is changed, the ASP script retrieves the Result. The result is parsed, and the number of hits from that particular resource is displayed to the user. The ASP script will remain in this loop until a result is returned for each resource being searched. Since the process is asynchronous, the results are displayed in the order that they are returned by searchbots. This will vary depending on the search speed of individual bibliographic resources at any given time.

Timeout conditions are currently handled only by the searchbots. A searchbot will timeout if it does not retrieve any results from a resource in a given time. When it times-out it will update the Result column of the blackboard database with a fixed error message, and it will update the NewEntry column to changed. This will signal to the ASP script that a timeout has occurred, and the script can return an appropriate message to the user.

The searchbots have been implemented using Microsoft Visual Basic. Each searchbot is customized to search a specific bibliographic web resource. However, they all have a similar architecture. They have a timer that periodically queries the blackboard database looking for new entries that have not yet been processed. When it finds a new entry, it updates the NewEntry column to working on, and it retrieves the SearchArg.

It will use the search argument to construct a query that is sent to the URL of the resource either as an HTTP GET or POST, as required by the resource. The searchbots use the Microsoft WinHTTP object for HTTP access. The WinHTTP object has properties and methods for performing basic HTTP functions, and
may be set to operate asynchronously, if desired. After a web resource returns the search results, the searchbot will update the blackboard database, updating the Result column with the search results and updating the NewEntry column to changed. It will then start looking for new entries to process.

When multiple searchbots are used for querying a single Web resource, a second database indicating individual searchbot status (‘busy’ or ‘available’) is used to coordinate the work between the searchbots. Whenever a searchbot begins work on a new query it will update its status in this second database. The other non-busy searchbots monitor this database, and whenever a searchbot’s previous sibling becomes busy, the searchbot will become active, meaning it will begin checking the blackboard database for unassigned jobs. This logic will continue in a chain of sibling searchbots until all available searchbots are active. A searchbot will remain active until its previous sibling is no longer busy. It will then finish any pending queries and become inactive. This scheme allows the searchbots to become active as needed without contending for access to the blackboard database.

One feature of this searchbot implementation is the ability to initiate and ‘hold open’ database connections at the search prompt. This saves the time required to log on to the service, select the database, and put the system into the position of accepting user input. In this way, some searchbots are able to moderate the more complex interactions required by some web resources, such as navigating through several web pages to reach the search results. This is usually required in order to establish access privileges, but could also be required because of pop-up advertising, or other reasons. In some cases, searchbots will attempt to maintain previously established search session and session identifiers by periodically ‘pinging’ the web resource to prevent the resource from timing-out the session.

The shared blackboard approach has a number of benefits. Among them are flexibility and scalability. First, the ASP script, the blackboard databases, and the searchbots may all reside on different or the same machine as resources or performance dictate. Also, as search usage increases, it is fairly easy to add more searchbots for different resources, either running on the same machine or different machines. The flexibility also extends to the actual function of the searchbots. The current searchbots are all used to query web resources over HTTP, but searchbots could be programmed to use Z39.50, native relational databases, or even to use Telnet access.

Another benefit of this approach is that the blackboard databases provide a log of all search activity that can be used for various analyses.

There are a couple of drawbacks to this approach. Chief among them is complexity of implementation. System logic is spread across multiple disparate components, possibly running on different machines which makes debugging difficult. Concurrency issues also become difficult to analyze, especially as usage becomes high and multiple searchbots for the same database must be added and balanced.

Also, because the ASP script enters an active loop, continuously querying all of the blackboard databases, waiting for the results to be returned for each query, CPU usage on the web server where the ASP script is running can be very high, and may not be scaleable under heavy usage scenarios. Also, because of this, resource usage on the database server machine may also be very high, and not scaleable.

### 4.2 Event-Driven, Asynchronous HTTP Queries From within a Single Script

The second approach implements asynchronous HTTP queries from within a single ASP script using the ServerXMLHTTP object. The ServerXMLHTTP object is identical to the WinHTTP object, except that it has an onReadyStateChange property that allows callback functions to be registered for the object. The registered function is automatically called whenever the readyState of the ServerXMLHTTP object changes. A readyState of COMPLETED indicates that all of the data has been retrieved by the ServerXMLHTTP object and is ready for processing. Also, contrary to the object’s name, it supports any type of retrievable web resource, not just XML.

When the ASP script starts, a separate instance of the ServerXMLHTTP object is created for each web resource to be searched. A callback function is also registered for each resource. For example,

```vbscript
Set ccon_http = _
    Server.CreateObject("MSXML2.ServerXMLHTTP.4.0")
ccon_http.onReadyStateChange = _
    GetRef("CheckStateCCON")
The GetRef function returns a reference to a function with the given name. The CheckStateCCON function is defined elsewhere in the script. For example,

Function CheckStateCCON()
    If ccon_http.readyState=COMPLETED Then
        ReturnOvidHits ccon_http.responseText, "ccon"
        ccon_http.Abort
    End If
End Function
```

When the readyState is COMPLETED, a function, such as ReturnOvidHits, is called. The function takes the responseText, which is the HTML result returned by the web query, parses it and returns the number of hits to the client web browser. If the ASP Response.buffer property is set to False, the number of hits will be returned immediately to the client web browser, even before any other pending searches are completed. To signal that the query is complete, the Abort method is called. This sets the ServerXMLHTTP readyState back to zero.

After all of the ServerXMLHTTP object instances are created and their callback functions are registered, the ASP script parses the various input parameters from the HTML search form and turns them into the appropriate URLs and query parameters required by the various web resources that are being queried. The ServerXMLHTTP instances then either GET or POST those URLs, as required by the web resource. For example,

```vbscript
ccon_http.Open "GET", url, True
    ccon_http.Send
```

The url parameter contains the URL plus query string parameters for the web resource. True indicates an asynchronous request.

After all the queries have been asynchronously submitted, the code enters a loop where it waits for all of the queries to be completed by the ServerXMLHTTP instances or for the web browser client to disconnect. For example, if there are two web resources, ccon_http and asti_http, which are being queried, the code is as follows:
The proxy servers are written as ASP scripts. They use the standard (see http://library.caltech.edu/openurl/). It is under development as NISO standard for describing and transporting metadata about 'Article Full Text' and that also mediate all communication.

BASED PROXY SERVICES

5. IMPLEMENTATION OF OPENURL-BASED PROXY SERVICES

The proxy servers that add links to the ‘Journal Web Site” and ‘Article Full Text’ and that also mediate all communication between the user client and the various bibliographic web resources make use of OpenURLs. The OpenURL protocol [[11]] (see http://sfx1.exlibris-usa.com/openurl/openurl.html) is a standard for describing and transporting metadata about bibliographic resources. It is under development as NISO standard (see http://library.caltech.edu/openurl/).

The proxy servers are written as ASP scripts. They use the WinHTTP object for all HTTP communication functions. They also make use of a CiteParse object for parsing various citation formats and turning them into OpenURL queries.

The CiteParse object is a custom ActiveX object written in Microsoft Visual Basic. This object contains properties and methods for parsing various short citation syntaxes and turning them into OpenURL queries. The object will currently parse the short citations produced by Ovid for the Current Contents, Applied Science and Technology Index, Engineering Index, and INSPEC databases. For example, the following Ovid short citation HTML fragment:


is transformed into the following OpenURL query:

genre=article&amp;aulast=Reschikov&amp;auinit1=M&amp;atitle=Yellow +and+green+luminescence+in+a+freenstanding+GaN+template&amp;date=2001-05-14&amp;spage=3041&amp;epage=3043&amp;volume=78&amp;issue=20&amp;genre=article&amp;volume=78&amp;issue=20&amp;genre=article

The CiteParse object makes extensive use of regular expressions for parsing. First, a series of regular expressions are hierarchically executed to break the citation into its core sections, such as authors, article title, and source. Finally, successive regular expressions are applied from the most to least specific to these sections. The first regular expression to match defines the component parts of that section. Variations in the source (journal, volume, issue, page, date) are most difficult. Thus far, 26 regular expressions are required to accurately parse all the variations used by Ovid across the four databases we are searching. These not only include citations to journal articles, but also conference proceedings, book items, theses, and others.

Once the CiteParse object has derived the OpenURL query it can submit it to a service provider at given BaseURL and process the results. For example, the CiteParse object currently has methods and properties specifically tailored for retrieving DOIs from the CrossRef system. However, it can be used to retrieve results from nearly any service provider that supports OpenURLs. For example, we have implemented several ASP scripts that accept OpenURL queries and return very specific results. Our callno.asp script will return the shelf location and call number, if available, for a bound journal volume given an OpenURL. Our journal.asp script will return a URL to a publisher’s home page for a given journal.

The ‘Article Full Text’ links that are added by the proxy to the intermediate search results all point to our resolver.asp script. This script was modified from a script originally developed as part of a local resolver demonstration project for CrossRef and CNRI [[2]]. This script accepts an OpenURL query and performs a CrossRef lookup. If the CrossRef lookup is successful, it will redirect the user to the DOI and thus to the resource itself. If it is not successful, it will perform a call number lookup, using the calnno.asp script. If this is successful, the location and call number will be displayed to the user. If the call number lookup is unsuccessful, the user will be redirected to the publisher’s home page for that journal. In addition, we have experimented with the resolver.asp script to provide other value added features, such as finding other articles by the same author or with the same subject, or performing related queries with the library online catalog.
6. CONCLUSION
The Grainger Search Aid Project has successfully tested several technical approaches to integrating simultaneous search and reference linking. Clearly, the integration of simultaneous search over periodical databases with proxied, article-level reference linking provides greatly enhanced access to distributed digital full-text resources. Reaction from users to the Grainger Search Aid Services has been extremely positive, particularly in its ability to provide access to online full-text. The ability to navigate from references in a paper or extended citation to the referred-to article is considered extremely valuable to users [[6], [9]].

7. REFERENCES