Web Services and Widgets for Library Information Systems

Godmar Back and Annette Bailey

As more libraries integrate information from web services to enhance their online public displays, techniques that facilitate this integration are needed. This paper presents a technique for such integration that is based on HTML widgets. We discuss three example systems (Google Book Classes, Tictoclookup, and MAJAX) that implement this technique. These systems can be easily adapted without requiring programming experience or expensive hosting.

To improve the usefulness and quality of their online public access catalogs (OPACs), more and more librarians include information from additional sources into their public displays.[1] Examples of such sources include Web services that provide additional bibliographic information, social bookmarking and tagging information, book reviews, alternative sources for bibliographic items, table-of-contents previews, and excerpts. As new Web services emerge, librarians quickly integrate them to enhance the quality of their OPAC displays. Conversely, librarians are interested in opening the bibliographic, holdings, and circulation information contained in their OPACs for inclusion into other Web offerings they or others maintain. For example, by turning their OPAC into a Web service, subject librarians can include up-to-the-minute circulation information in subject or resource guides. Similarly, university instructors can use an OPAC's metadata records to display citation information ready for import into citation management software on their course pages. The ability to easily create such "mash-up" pages is crucial for increasing the visibility and reach of the digital resources libraries provide.

Although the technology to use Web services to create mash-ups is well known, several practical requirements must be met to facilitate its widespread use. First, any environment providing for such integration should be easy to use, even for librarians with limited programming background. This ease of use must extend to environments that include proprietary systems, such as vendor-provided OPACs. Second, integration must be seamless and customizable, allowing for local display preferences and flexible styling. Third, the setup, hosting, and maintenance of any necessary infrastructure must be low-cost and should maximize the use of already available or freely accessible resources. Fourth, performance must be acceptable, both in terms of latency and scalability.[2]

Godmar Back (gback@cs.vt.edu) is Assistant Professor, Department of Computer Science and **Annette Bailey** (afbailey@vt.edu) is Assistant Professor, University Libraries, Virginia Tech University, Blacksburg.

In this paper we discuss the design space of methods for integrating information from Web services into websites. We focus primarily on client-side mash-ups, in which code running in the user's browser contacts Web services directly without the assistance of an intermediary server or proxy. To create such mash-ups, we advocate the use of "widgets," which are easy-to-use, customizable HTML elements whose use does not require programming knowledge. Although the techniques we discuss apply to any Web-based information system, we specifically consider how an OPAC can become both the target of Web services integration and also a Web service that provides information to be integrated elsewhere. We describe three widget libraries we have developed, which provide access to four Web services. These libraries have been deployed by us and others.

Our contributions are twofold: We give practitioners an insight into the trade-offs surrounding the appropriate choice of mash-up model, and we present the specific designs and use examples of three concrete widget libraries librarians can directly use or adapt. All software described in this paper is available under the LGPL Open Source License.

Background

Web-based information systems use a client-server architecture in which the server sends HTML markup to the user's browser, which then renders this HTML and displays it to the user. Along with HTML markup, a server may send JavaScript code that executes in the user's browser. This JavaScript code can in turn contact the original server or additional servers and include information obtained from them into the rendered content while it is being displayed. This basic architecture allows for myriad possible design choices and combinations for mash-ups. Each design choice has implications to ease of use, customizability, programming requirements, hosting requirements, scalability, latency, and availability.

Server-side mash-ups

In a server-side mash-up design, shown in figure 1, the mash-up server contacts the base server and each source when it receives a request from a client. It combines the information received from the base server and the sources and sends the combined HTML to the client.

Server-side mash-up systems that combine base and mash-up servers are also referred to as data mash-up systems. Such data mash-up systems typically provide a Web-based configuration front-end that allows users to select data sources, specify the manner in which they are combined, and to create a layout for the entire mash-up.

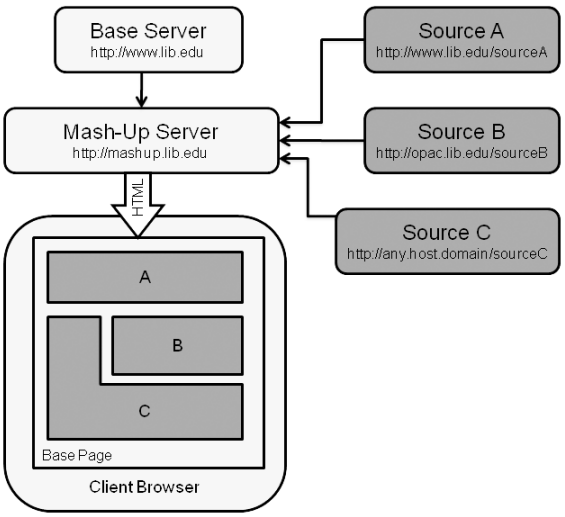


Figure 1. Server-side mash-up construction

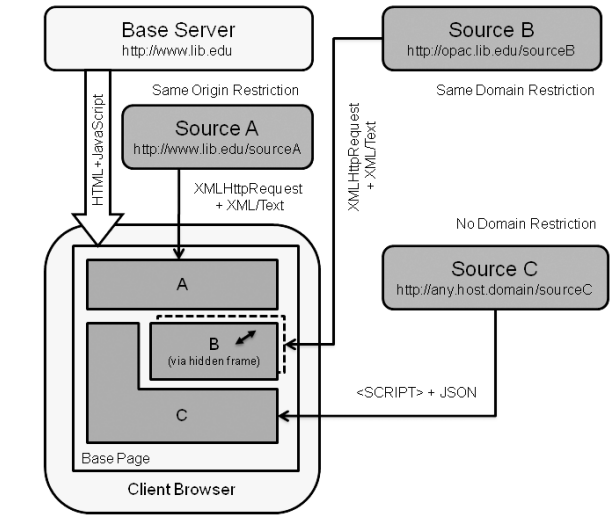


Figure 2. Client-side mash-up construction

Examples of such systems include Dapper and Yahoo! Pipes.[3] These systems require very little programming knowledge, but they limit mash-up creators to the functionality supported by a particular system and do not allow the user to leverage the layout and functionality of an existing base server, such as an existing OPAC.

Integrating server-side mash-up systems with proprietary OPACs as the base server is difficult because the mash-up server must parse the OPAC's output before integrating any additional information. Moreover, users must now visit—or be redirected to—the URL of the mash-up server. Although some emerging extensible OPAC designs provide the ability to include information from external sources directly and easily, most currently deployed systems do not.[4] In addition, those mash-up servers that do usually require server-side programming to retrieve and integrate the information coming from the mash-up sources into the page. The availability of software libraries and the use of special purpose markup languages may mitigate this requirement in the future.

From a performance scalability point of view, the mash-up server is a bottleneck in server-side mash-ups and therefore must be made large enough to handle the expected load of end-user requests. On the other hand, the caching of data retrieved from mash-up sources is simple to implement in this arrangement because only the mash-up server contacts these sources. Such caching reduces the frequency with which requests have to be sent to sources if their data is cacheable, that is, if real-time information is not required.

The latency in this design is the sum of the time required for the client to send a request to the mash-up server and receive a reply, plus the processing time required by the server, plus the time incurred by sending a request and receiving a reply from the last responding mash-up source. This model assumes that the mash-up server contacts all sources in parallel, or as soon as the server knows that information from a source should be included in a page.

The availability of the system depends on the availability of all mash-up sources. If a mash-up source does not respond, the end user must wait until such failure is apparent to the mash-up server via a timeout. Finally, because the mash-up server acts as a client to the base and source servers, no additional security considerations apply with respect to which sources may be contacted. There also are no restrictions on the data interchange format used by source servers as long as the mash-up server is able to parse the data returned.

Client-side mash-ups

In a client-side setup, shown in figure 2, the base server sends only a partial website to the client, along with JavaScript code that instructs the client which other sources of information to contact. When executed in the browser, this JavaScript code retrieves the information from the mash-up sources directly and completes the mash-up.

The primary appeal of client-side mashing is that no mash-up server is required, and thus the URL that users visit does not change. Consequently, the mash-up server is no longer a bottleneck. Equally important, no maintenance is required for this server, which is particularly relevant when libraries use turnkey solutions that restrict administrative access to the machine housing their OPAC. On the other hand, without a mash-up server, results from mash-up sources can no longer be centrally cached. Thus the mash-up sources themselves must be sufficiently

scalable to handle the expected number of requests. As a load-reducing strategy, mash-up sources can label their results with appropriate expiration times to influence the caching of results in the clients' browsers.

Availability is increased because the mash-up degrades gracefully if some of the mash-up sources fail, since the information from the remaining sources can still be displayed to the user. Assuming that requests are sent by the client in parallel or as soon as possible, and assuming that each mash-up source responds with similar latency to requests sent by the user's browser as to requests sent by a mash-up server, the latency for a client-side mash-up is similar to the server-side mash-up. However, unlike in the server-side approach, the page designer has the option to display partial results to the user while some requests are still in progress, or even to delay sending some requests until the user explicitly requests the data by clicking on a link or other element on the page.

Because client-side mash-ups rely on JavaScript code to contact Web services directly, they are subject to a number of restrictions that stem from the security model governing the execution of JavaScript code in current browsers. This security model is designed to protect the user from malicious websites that could exploit client-side code and abuse the user's credentials to retrieve HTML or XML data from other websites to which a user has access. Such malicious code could then relay this potentially sensitive data back to the malicious site. To prevent such attacks, the security model allows the retrieval of HTML text or XML data only from sites within the same domain as the origin site, a policy commonly known as same-origin policy. In figure 2, sources A and B come from the same domain as the page the user visits.

The restrictions of the same-origin policy can be avoided by using the JavaScript Object Notation (JSON) interchange format.[5] Because client-side code may retrieve and execute JavaScript code served from any domain, Web services that are not co-located with the origin site can make their results available using JSON. Doing so facilitates their inclusion into any page, independent of the domain from which it is served (see source C in figure 2). Many existing Web services already provide an option to return data in JSON format, perhaps along with other formats such as XML. For Web services that do not, a proxy server may be required to translate the data coming from the service into JSON. If the implementation of a proxy server is not feasible, the Web service is usable only on pages within the same domain as the website using it.

Client-side mash-ups lend themselves naturally to enhancing the functionality of existing, proprietary OPAC systems, particularly when a vendor provides only limited extensibility. Because they do not require server-side programming, the absence of a suitable vendor-provided server-side programming interface does not prevent their creation. Oftentimes, vendor-provided templates or variables can be suitably adapted to send the necessary HTML markup and JavaScript code to the client.

The amount of JavaScript code a librarian needs to write (or copy from a provided example) determines both the likelihood of adoption and the maintainability of a given mash-up creation. The less JavaScript code there is to write, the larger the group of librarians who feel comfortable trying and adopting a given implementation. The approach of using HTML widgets hides the use of JavaScript almost entirely from the mash-up creator. HTML widgets represent specially composed markup, which will be replaced with information coming from a mash-up source when the page is rendered. Because the necessary code is contained in a JavaScript library, adapters do not need to understand programming to use the information coming from the Web service. Finally, HTML widgets are also preferable for JavaScript-savvy users because they create a layer of abstraction over the complexity and browser dependencies inherent in JavaScript programming.

The Google Book Classes Widget Library

To illustrate our approach, we present a first example that allows the integration of data obtained from Google Book Search into any website, including OPAC pages. Google Book Search provides access to Google's database of book metadata and contents. Because of the company's book scanning activities as well as through agreements with publishers, Google hosts scanned images of many book jackets as well as partial or even full previews for some books. Many libraries are interested in either using the book jackets when displaying OPAC records or alerting their users if Google can provide a partial or full view of an item a user selected in their catalog, or both.[6] This service can help users decide whether to borrow the book from the library.

The Google Book Search Dynamic Link API

The Google Book Search Dynamic Link API is a JSON-based Web service through which Google provides certain metadata for items it has indexed. It can be queried using bibliographic identifiers such as ISBN, OCLC number, or Library of Congress Control Number (LCCN). It returns a small set of data that includes the URL of a book jacket thumbnail image, the URL of a page with bibliographic information, the URL of a preview page (if available), as well as information about the extent of any preview and whether the preview viewer can be embedded directly into other pages. Table 1 shows the JSON result returned for an example ISBN.

Table 1. Sample Request and Response for Google Book Search Dynamic Link API

Request:

http://books.google.com/books?bibkeys=ISBN:0596000278&jscmd=viewapi&callback=process

JSON Response:

```
process({
"ISBN:0596000278":
{ "bib_key": "ISBN:0596000278",
"info_url": "http://books.google.com/books?id=ezqe1hh91q4C\x26source=gbs_ViewAPI",
"preview_url": "http://books.google.com/books?id=ezqe1hh91q4C\x26printsec=frontcover\x26
source=gbs_ViewAPI",
"thumbnail_url": "http://bks4.books.google.com/books?id=ezqe1hh91q4C\x26printsec=frontcover\x26
img=1\x26zoom=5\x26sig=ACfU3U2d1UsnXw9BAQd94U2nc3quwhJn2A",
"preview": "partial",
"embeddable": true
}
});
```

Widgetization

To facilitate the easy integration of this service into websites without JavaScript programming, we developed a widget library. From the adapter's perspective, the use of these widgets is extremely simple. The adapter places HTML or tags into the page where they want data from Google Book Search to display. These tags contain an HTML <title> attribute that acts as an identifier to describe the bibliographic item for which information should be retrieved. It may contain its ISBN, OCLC number, or LCCN. In addition, the tags also contain one or more HTML <class> attributes to describe which processing should be done with the information retrieved from Google to integrate it into the page. These classes can be combined with a list of traditional CSS classes in the <class> attribute to apply further style and formatting control.

Examples

As an example, consider the following HTML an adapter may use in a page:

```
<span title="ISBN:0596000278" class="gbs
-thumbnail gbs-link-to-preview"></span>
```

When processed by the Google Book Classes widget library, the class "gbs-thumbnail" instructs the widget to embed a thumbnail image of the book jacket for ISBN 0596000278, and "gbs-link-to-preview" provides instructions to wrap the tag in a hyperlink pointing to Google's preview page. The result is as if the server had contacted Google's Web service and constructed the HTML shown in example 1 in table 2, but the mash-up creator does not need to be concerned with the mechanics of contacting Google's service and making the necessary manipulations to the document.

Example 2 in table 2 demonstrates a second possible use of the widget. In this example, the creator's intent is to display an image that links to Google's information page if and only if Google provides at least a partial preview for the book in question. This goal is accomplished by placing the image inside the span and using style="display:none" to make the span initially invisible. The span is made visible only if a preview is available at Google, displaying the hyperlinked image. The full list of features supported by the Google Book Classes widget library can be found in table 3.

Integration with legacy OPACs

The approach described thus far assumes that the mash-up creator has sufficient control over the HTML markup that is sent to the user. This assumption does not always hold if the HTML is produced by a vendor-provided system, since such systems automatically generate most of the HTML used to display OPAC search results or individual bibliographic records. If the OPAC provides an extension system, such as a facility to embed customized links to external resources, it may be used to generate the necessary HTML by utilizing variables (e.g., "@#ISBN@" for ISBN numbers) set by the OPAC software.

If no extension facility exists, accommodations by the widget library are needed to maintain the goal of not requiring any programming on the part of the adapter. We implemented such accommodations to facilitate the use of Google Book Classes within a III Millennium OPAC.[7] We used magic strings such as "ISBN:millennium.record" in a

Table 2. Example of client-side processing by the Google Book Classes widget library

Example 1: HTML Written by Adapter	**Browser Display**
Resultant HTML after Client-Side Processing	
  	
Example 2: HTML Written by Adapter	**Browser Display**
 class="gbs-link-to-info gbs-if-partial-or-full"> 	
Resultant HTML after Client-Side Processing	
  	

Table 3. Supported Google Book classes

Google Book Class	Meaning
gbs-thumbnail	Include an <img...> embedding the thumbnail image
gbs-link-to-preview	Wrap span/div in link to preview at Google Book Search (GBS)
gbs-link-to-info	Wrap span/div in link to info page at GBS
gbs-link-to-thumbnail	Wrap span/div in link to thumbnail at GBS
gbs-embed-viewer	Directly embed a viewer for book's content into the page, if possible
gbs-if-noview	Keep this span/div only if GBS reports that book's viewability is "noview"
gbs-if-partial-or-full	Keep this span/div only if GBS reports that book's viewability is at least "partial"
gbs-if-partial	Keep this span/div only if GBS reports that book's viewability is "partial"
gbs-if-full	Keep this span/div only if GBS reports that book's viewability is "full"
gbs-remove-on-failure	Remove this span/div if GBS doesn't return book information for this item

<title> attribute to instruct the widget library to harvest the ISBN from the current page via screen scraping. Figure 3 provides an example of how a Google Book Classes widget can be integrated into an OPAC search results page.

The Tictoclookup Widget Library

The ticTOCs Journal Table of Contents Service is a free online service that allows academic researchers and

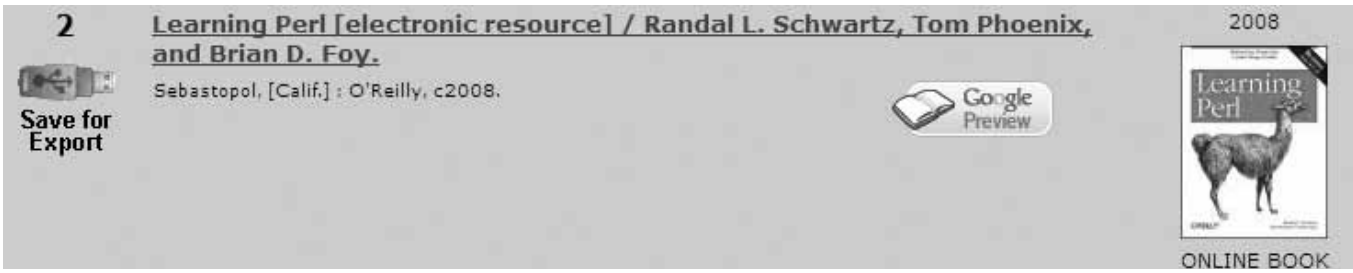


Figure 3. Sample use of Google Book Classes in an OPAC results page

other users to keep up with newly published research by giving them access to thousands of journal tables of contents from multiple publishers.[8] The ticTOCs consortium compiles and maintains a dataset that maps ISSNs and journal titles to RSS-feed URLs for the journals' tables of contents.

The Tictoclookup Web service

We used the ticTOCs dataset to create a simple JSON Web service called "Tictoclookup" that returns RSS-feed URLs when queried by ISSN and, optionally, by journal title. Table 4 shows an example query and response.

To accommodate different hosting scenarios, we created two implementations of this Tictoclookup: a standalone and a cloud-based implementation. The standalone version is implemented as a Python Web application conformant to the Web Services Gateway Interface (WSGI) specification. Hosting this version requires access to a Web server that supports a WSGI-compatible environment, such as Apache's mod_wsgi. The Python application reads the ticTOCs dataset and responds to lookup requests for specific ISSNs. A cron job downloads the most up-to-date version of the dataset periodically.

The cloud version of the Tictoclookup service is implemented as a Google App Engine (GAE) application. It uses the highly scalable and highly available GAE Datastore to store ticTOCs data records. GAE applications run on servers located in Google's regional data centers so that requests are handled by a data center geographically close to the requesting client. As of June 2009, Google hosting of GAE applications is free, which includes a free allotment of several computational resources. For each application, GAE allows quotas of up to 1.3 MB requests and the use of up to 10 GB of bandwidth per twenty-four-hour period. Although this capacity is sufficient for the purposes of many small- and medium-size institutions, additional capacity can be purchased at a small cost.

Widgetization

To facilitate the easy integration of this service into websites without JavaScript programming, we developed a widget library. Like Google Book Classes, this widget library is controlled via HTML attributes associated with HTML or tags that are placed into the page where the user decides to display data from the Tictoclookup service. The HTML <title> attribute identifies the journal by its ISSN or its ISSN and title. As with Google Book Classes,

Table 4. Sample request and response for ticTOCs lookup Web service

Request:

```
http://tictoclookup.appspot.com/0028-0836?title=Nature&jsoncallback=process
```

JSON Response:

```
process({
 "lastmod": "Wed Apr 29 05:42:36 2009",
 "records": [{
 "title": "Nature",
 "rssfeed": http://www.nature.com/nature/current_issue/rss
 }],
 "issn": "00280836"
});
```

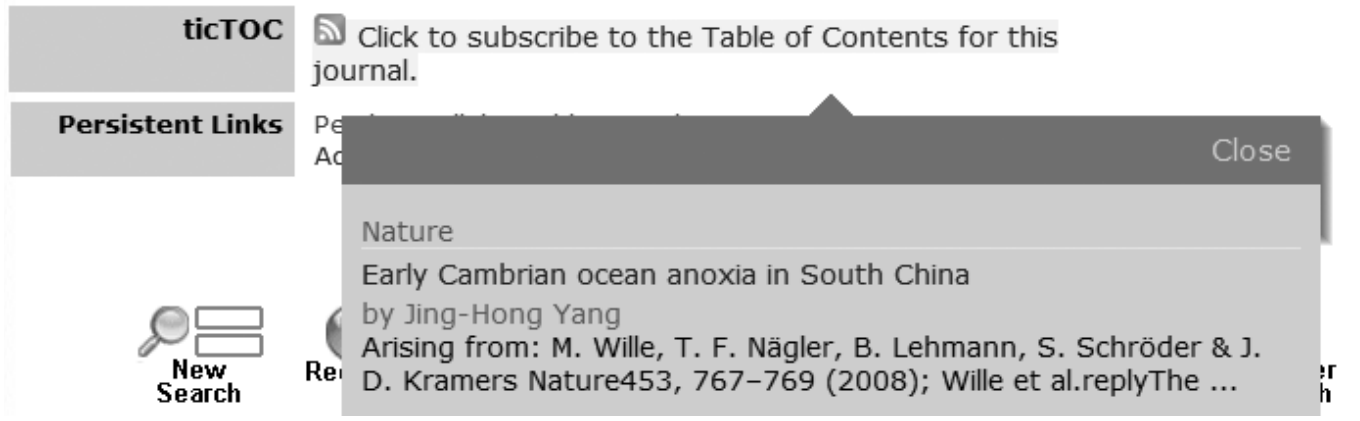


Figure 4. Sample use of tictoclookup classes

the HTML <class> attribute describes the desired processing, which may contain traditional CSS classes.

Example

Consider the following HTML an adapter may use in a page:

```
<span style="display:none"
    class="tictoc-link tictoc-preview tictoc-alternate-link"
    title="ISSN:00280836: Nature">
Click to subscribe to Table of Contents for this journal
</span>
```

When processed by the Tictoclookup widget library, the class "tictoc-link" instructs the widget to wrap the span in a link to the RSS feed at which the table of content is published, allowing users to subscribe to it. The class "tictoc-preview" associates a tooltip element with the span, which displays the first entries of the feed when the user hovers over the link. We use the Google Feeds API, another JSON-based Web service, to retrieve a cached copy of the feed. The "tictoc-alternate-link" class places an alternate link into the current document, which in some browsers triggers the display of the RSS feed icon in the status bar. The element, which is initially invisible, is made visible if and only if the Tictoclookup service returns information for the given pair of ISSN and title. Figure 4 provides a screenshot of the display if the user hovers over the link.

As with Google Book Classes, the mash-up creator does not need to be concerned with the mechanics of contacting the Tictoclookup Web service and making the necessary manipulations to the document. Table 5 provides a complete overview of the classes Tictoclookup supports.

Integration with legacy OPACs

Similar to the Google Book Classes widget library, we implemented provisions that allow the use of Tictoclookup classes on pages over which the mash-up creator has limited control. For instance, specifying a title attribute of "ISSN:millennium.issnandtitle" harvests the ISSN and journal title from the III Millennium's record display page.

Table 5. Supported Tictoclookup classes

Tictoclookup Class	Meaning
tictoc-link	Wrap span/div in link to table of contents
tictoc-preview	Display tooltip with preview of current entries
tictoc-embed-n	Embed preview of first n entries
tictoc-alternate-link	Insert <link rel="alternate"> into document
tictoc-append-title	Append the title of the journal to the span/div

MAJAX

Whereas the widget libraries discussed thus far integrate external Web services into an OPAC display, MAJAX is a widget library that integrates information coming from an OPAC into other pages, such as resource guides or course displays. MAJAX is designed for use with a III Millennium Integrated Library System (ILS) whose vendor does not provide a Web-services interface. The techniques we used, however, extend to other OPACs as well. Like many

legacy OPACs, Millennium does not only lack a Web-services interface, but lacks any programming interface to the records contained in the system and does not provide access to the database or file system of the machine housing the OPAC.

Providing OPAC data as a Web service

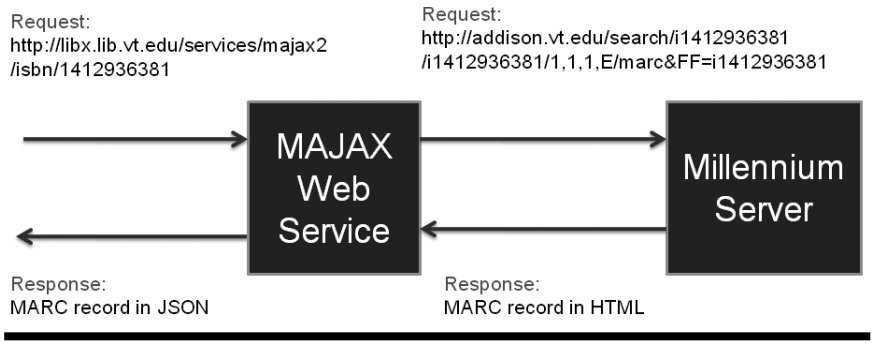


Figure 5. Architecture of the MAJAX 2 Web service

We implemented two methods to access records from the Millennium OPAC using bibliographic identifiers such as ISBN, OCLC number, bibliographic record number, and item title. Both methods provide access to complete MARC records and holdings information, along with locations and real-time availability for each held item. MAJAX extracts this information via screen-scraping from the MARC record display page. As with all screen-scraping approaches, the code performing the scraping must be updated if the output format provided by the OPAC changes. In our experience, such changes occur at a frequency of less than once per year.

The first method, MAJAX 1, implements screen scraping using JavaScript code that is contained in a document placed in a directory on the server (/screens), which is normally used for supplementary resources, such as images. This document is included in the target page as a hidden HTML <iframe> element (see frame B in figure 2). Consequently, the same-domain restriction applies to the code residing in it. MAJAX 1 can thus be used only on pages within the same domain—for instance, if the OPAC is housed at opac.library.university.edu, MAJAX 1 may be used on all pages within *.university.edu (not merely *.library.university.edu). The key advantage of MAJAX 1 is that no additional server is required.

The second method, MAJAX 2, uses an intermediary server that retrieves the data from the OPAC, translates it to JSON, and returns it to the client. This method, shown in figure 5, returns JSON data and therefore does not suffer from the same-domain restriction. However, it requires hosting the MAJAX 2 Web service. Like the Tictoclookup Web service, we implemented the MAJAX 2 Web service using Python conformant to WSGI. A single installation can support multiple OPACs.

Widgetization

The MAJAX widget library allows the integration of both MAJAX 1 and MAJAX 2 data into websites without JavaScript programming. The tags function as placeholders, and <title> and <class> attributes describe the desired processing. MAJAX provides a number of "MAJAX classes," multiple of which can be specified. These classes allow a mash-up creator to insert a large variety of bibliographic information, such as the values of MARC fields. Classes are also provided to insert fully formatted, ready-to-copy bibliographic references in Harvard style, live circulation information, links to the catalog record, links to online versions of the item (if applicable), a ready-to-import RIS description of the item, and even images of the book cover. A list of classes MAJAX supports is provided in table 6.

Examples

Figure 6 provides an example use of MAJAX widgets. Four tags expand into the book cover, a complete Harvard-style reference, the valid of a specific MARC field (020), and a display of the current availability of the item, wrapped in a link to the catalog record. Texts such as "copy is available" shown in figure 6 are localizable. Even though there are multiple MAJAX tags that refer to the same ISBN, the MAJAX widget library will contact the MAJAX 1 or MAJAX 2 Web service only once per identifier, independent of how often it is used in a page. To manage the load, the MAJAX client site library can be configured to not exceed a maximum number of requests per second, per client.

All software described in this paper is available under the LGPL Open Source License. The MAJAX libraries have been used by us and others for about two years. For instance, the "New Books" list in our library uses MAJAX 1 to provide circulation information. Faculty members at our institution are using MAJAX to enrich their course websites. A number of libraries have adopted MAJAX 1, which is particularly easy to host because no additional server is required.

Related work

Most ILSs in use today do not provide suitable Web-services interfaces to access either bibliographic information

or availability data.[9] This shortcoming is addressed by multiple initiatives. The ILS Discovery Interface task force (ILS-DI) created a set of recommendations that facilitate the integration of discovery interfaces with legacy ILSs, but does not define a concrete API.[10] Related, the ISO 20775 Holdings standard describes an XML schema to describe the availability of items across systems, but does not describe an API for accessing them.[11] Many ILSs provide a Z39.50 interface in addition to their HTML-based Web OPACs, but Z39.50 does not provide standardized holdings and availability.[12]

Nevertheless, there is hope within the community that ILS vendors will react to their customers' needs and provide Web-services interfaces that implement these recommendations. The Jangle project provides an API and an implementation of the ILS-DI recommendations through a Representations State Transfer (REST)–based interface that uses the Atom Publishing Protocol (APP).[13] Jangle can be linked to legacy ILSs via connectors. The use of the XML-based APP prevents direct access from client-side JavaScript code, however. In the future, adoption and widespread implementation of the W3C working draft on cross-origin resource sharing may relax the same-origin restriction in a controlled fashion, and thus allow access to APP feeds from JavaScript across domains.[14]

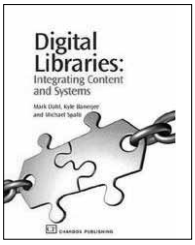
Table 6. Selected MAJAX classes

MAJAX Class	Replacement
majax-marc-FFF-s	MARC field FFF, subfields
majax-marc-FFF	concatenation of all subfields in field FFF
majax-syndetics-*	book cover image
majax-showholdings	current holdings and availability information
majax-showholdings-brief	…in brief format
majax-endnote	RIS version of record
majax-ebook	link to online version, if any
majax-linktocatalog	link to record in catalog
majax-harvard-reference	reference in Harvard style
majax-newline	newline
majax-space	space

HTML Written by Adapter

```
<table width="340"><tr><td>
  <span class="majax-syndetics-vtech" title="i1843341662"></span>
  </td><td>
  <span class="majax-harvard-reference" title="i1843341662"></span>
  <br /> ISBN:
  <span class="majax-marc-020" title="i1843341662"></span>
  <br />
  <span class="majax-linktocatalogmajax-showholdings"
     title="i1843341662"></span>
</td></tr></table>
```

Display in Browser after Processing



Dahl, Mark., Banerjee, Kyle., Spalti, Michael., 2006, *Digital libraries : integrating content and systems /*
Oxford, Chandos Publishing,
xviii, 203 p.
ISBN: 1843341662 (hbk.)
1 copy is available

Figure 6. Example use of MAJAX widgets

Screen-scraping is a common technique used to overcome the lack of Web-services interfaces. For instance, OCLC's WorldCat Local product obtains access to availability information from legacy ILSs in a similar fashion as our MAJAX 2 service.[15] Whereas the Web services used or created in our work exclusively use a REST-based model and return data in JSON format, interfaces based on SOAP (formerly Simple Object Access Protocol) whose semantics are described by a WSDL specification provide an alternative if access from within client-side JavaScript code is not required.[16]

OCLC Grid Services provides REST-based Web-services interfaces to several databases, including the WorldCat Search API and identifier services such as xISBN, xISSN, and xOCLCnum for FRBR-related metadata.[17] These services support XML and JSON and could benefit from widgetization for easier inclusion into client pages.

The use of HTML markup to encode processing instructions is common in JavaScript frameworks, such as YUI or Dojo, which use elements with custom-defined attributes (so-called expando attributes) for this purpose.[18] Google Gadgets uses a similar technique as well.[19] The widely used Context Objects in Spans (COinS) specification exploits tags to encode OpenURL

Table 7. Summary of features and requirements for the widget libraries presented in this paper

	Majax 1	Majax 2	Google Book Classes	Tictoclookup Classes
Web Service	Screen Scraping III Record Display	JSON Proxy for III Record Display	Google Book Search Dynamic Link API books.google.com	ticTOC Cloud Application tictoclookup .appspot.com
Hosted By	Existing Millennium Installation /screens	WSGI/Python Script on libx.lib.vt.edu	Google, Inc.	Google, Inc. via Google App Engine
Data Provenance	Your OPAC	Your OPAC	Google	JISC (www.tictocs .ac.uk)
Additional Cost	N/A	Can use libx.lib.vt.edu for testing, must run WSGI-enabled web server in production	Free, but subject to Google Terms of Service	Generous free quota, pay per use beyond that
Same Domain Restriction	Yes	No	No	No
Widgetization	majax.js: class-based: majax- classes		gbsclasses.js:class-based: gbs-	tictoc.js:class-based: tictoc-
Requires JavaScript programming	No	No	No	No
Requires Additional Server	No	Yes (Apache+mod_wsgi)	No	No (if using GAE), else need Apache+mod_wsgi
III Bibrecord Display	N/A	N/A	Yes	Yes
III WebBridge Integration	Yes	Yes	Yes	Yes

context objects in pages for processing by client-side extension.[20] LibraryThing uses client-side mash-up techniques to incorporate a social tagging service into OPAC pages.[21] Although their technique uses a element as a placeholder, it does not allow customization via classes—the changes to the content are encoded in custom-generated JavaScript code for each library that subscribes to the service.

The Juice Project shares our goal of simplifying the enrichment of OPAC pages with content from other sources.[22] It provides a set of reusable components that is directed at JavaScript programmers, not librarians. In the computer-science community, multiple emerging projects investigate how to simplify the creation of server-side data mash-ups by end user programmers.[23]

Conclusion

This paper explored the design space of mash-up techniques for the seamless inclusion of information from Web services into websites. We considered the cases where an OPAC is either the target of such integration or the source of the information being integrated. We focused on client-side techniques in which each user's browser contacts Web services directly because this approach lends itself to the creation of HTML widgets. These widgets allow the integration and customization of Web services without requiring programming. Therefore nonprogrammers can become mash-up creators.

We described in detail the functionality and use of several widget libraries and Web services we built. Table 7 provides a summary of the functionality and hosting requirements for each system discussed. Although the specific requirements for each system differ because of their respective nature, all systems are designed to be deployable with minimum effort and resource requirements. This low entry cost, combined with the provision of a high-level, nonprogramming interface, constitute two crucial preconditions for the broad adoption of mash-up techniques in libraries, which in turn has the potential to

vastly increase the reach and visibility of their electronic resources in the wider community.

References

1. Nicole Engard, ed., *Library Mashups—Exploring New Ways to Deliver Library Data* (Medford, N.J.: Information Today, 2009); Andrew Darby and Ron Gilmour, "Adding Delicious Data to Your Library Website," *Information Technology & Libraries* 28, no. 2 (2009): 100–103.

2. Monica Brown-Sica, "Playing Tag in the Dark: Diagnosing Slowness in Library Response Time," *Information Technologies & Libraries* 27, no. 4 (2008): 29–32.

3. Dapper, "Dapper Dynamic Ads," http://www.dapper.net/ (accessed June 19, 2009); Yahoo!, "Pipes," http://pipes.yahoo.com/pipes/ (accessed June 19, 2009).

4. Jennifer Bowen, "Metadata to Support Next-Generation Library Resource Discovery: Lessons from the Extensible Catalog, Phase 1," *Information Technology & Libraries* 27, no. 2 (2008): 6–19; John Blyberg, "ILS Customer Bill-of-Rights," online posting, Blyberg.net, Nov. 20, 2005, http://www.blyberg.net/2005/11/20/ils-customer-bill-of-rights/ (accessed June 18, 2009).

5. Douglas Crockford, "The Application/JSON Media Type for JavaScript Object Notation (JSON)," memo, The Internet Society, July 2006, http://www.ietf.org/rfc/rfc4627.txt (accessed Mar. 30, 2010).

6. Google, "Who's Using the Book Search APIs?" http://code.google.com/apis/books/casestudies/ (accessed June 16, 2009).

7. Innovative Interfaces, "Millennium ILS," http://www.iii.com/products/millennium_ils.shtml (accessed June 19, 2009).

8. Joint Information Systems Committee, "TicTOCs Journal Tables of Contents Service," http://www.tictocs.ac.uk/ (accessed June 18, 2009).

9. Mark Dahl, Kyle Banarjee, and Michael Spalti, *Digital Libraries: Integrating Content and Systems* (Oxford, United Kingdom: Chandos, 2006).

10. John Ockerbloom et al., "DLF ILS Discovery Interface Task Group (ILS-DI) Technical Recommendation," (Dec. 8, 2008), http://diglib.org/architectures/ilsdi/DLF_ILS_Discovery_1.1.pdf (accessed June 18, 2009).

11. International Organization for Standardization, "Information and Documentation—Schema for Holdings Information," http://www.iso.org/iso/catalogue_detail.htm?csnumber=39735 (accessed June 18, 2009)

12. National Information Standards Organization, "ANSI/NISO Z39.50—Information Retrieval: Application Service Definition and Protocol Specification," (Bethesda, Md.: NISO Pr., 2003), http://www.loc.gov/z3950/agency/Z39-50-2003.pdf (accessed May 31, 2010).

13. Ross Singer and James Farrugia, "Unveiling Jangle: Untangling Library Resources and Exposing Them through the Atom Publishing Protocol," *The Code4Lib Journal* no. 4 (Sept. 22, 2008), http://journal.code4lib.org/articles/109 (accessed Apr. 21, 2010); Roy Fielding, "Architectural Styles and the Design of Network-Based Software Architectures" (PhD diss., University of California, Irvine, 2000); J. C. Gregorio, ed., "The Atom Publishing Protocol," memo, The Internet Engineering Task Force, Oct. 2007, http://bitworking.org/projects/atom/rfc5023.html (accessed June 18, 2009).

14. World Wide Web Consortium, "Cross-Origin Resource Sharing: W3C Working Draft 17 March 2009," http://www.w3.org/TR/access-control/ (accessed June 18, 2009).

15. OCLC Online Computer Library Center, "Worldcat and Cataloging Documentation," http://www.oclc.org/support/documentation/worldcat/default.htm (accessed June 18, 2009).

16. F. Curbera et al., "Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI," *IEEE Internet Computing* 6, no. 2 (2002): 86–93.

17. OCLC Online Computer Library Center, "OCLC Web Services," http://www.worldcat.org/devnet/wiki/Services (accessed June 18, 2009); International Federation of Library Associations and Institutions Study Group on the Functional Requirements for Bibliographic Records, "Functional Requirements for Bibliographic Records : Final Report," http://www.ifla.org/files/cataloguing/frbr/frbr_2008.pdf (accessed Mar. 31, 2010).

18. Yahoo!, "The Yahoo! User Interface Library (YUI)," http://developer.yahoo.com/yui/ (accessed June 18, 2009); Dojo Foundation, "Dojo—The JavaScript Toolkit," http://www.dojotoolkit.org/ (accessed June 18, 2009).

19. Google, "Gadgets.* API Developer's Guide," http://code.google.com/apis/gadgets/docs/dev_guide.html (accessed June 18, 2009).

20. Daniel Chudnov, "COinS for the Link Trail," *Library Journal* 131 (2006): 8–10.

21. LibraryThing, "LibraryThing," http://www.librarything.com/widget.php (accessed June 19, 2009).

22. Robert Wallis, "Juice—JavaScript User Interface Componentised Extensions," http://code.google.com/p/juice-project/ (accessed June 18, 2009).

23. Jeffrey Wong and Jason Hong, "Making Mashups with Marmite: Towards End-User Programming for the Web" *Conference on Human Factors in Computing Systems, San Jose, California, April 28–May 3, 2007: Conference Proceedings, Volume 2* (New York: Association for Computing Machinery, 2007): 1435–44; Guiling Wang, Shaohua Yang, and Yanbo Han, "Mashroom: End-User Mashup Programming Using Nested Tables" (paper presented at the International World Wide Web Conference, Madrid, Spain, 2009): 861–70; Nan Zang, "Mashups for the Web-Active User" (paper presented at the IEEE Symposium on Visual Languages and Human-Centric Computing, Herrshing am Ammersee, Germany, 2008): 276–77.