MyLibrary: A Digital Library Framework and Toolkit

Eric Lease Morgan

This article describes a digital library framework and toolkit called MyLibrary. At its heart, MyLibrary is designed to create relationships between information resources and people. To this end, MyLibrary is made up of essentially four parts: (1) information resources, (2) patrons, (3) librarians, and (4) a set of locally defined, institution-specific facet/term combinations interconnecting the first three. On another level, MyLibrary is a set of object-oriented Perl modules intended to read and write to a specifically shaped relational database. Used in conjunction with other computer applications and tools, MyLibrary provides a way to create and support digital library collections and services. Librarians and developers can use MyLibrary to create any number of digital library applications: full-text indexes to journal literature, a traditional library catalog complete with circulation, a database-driven website, an institutional repository, an image database, etc. The article describes each of these points in greater detail.

Background and history

The term "MyLibrary" was coined by Keith Morgan, Doris Sigl, and myself in 1997 when we worked in the Department of Digital Library Initiatives at the North Carolina State University Libraries. At that time it denoted a personalizable/customizable user interface to sets of library collections and services. It was a reaction to the then-popular portal applications called My Netscape, My Yahoo!, and My Dejanews.[1]

In that form, MyLibrary was a monolithic turnkey application. Librarians were expected to use the administrative interface to organize information resources into three distinct groups: databases, electronic texts, and library links (services). Each item in each group was expected to be associated with one or more discipline terms. Patrons were expected to come to the system, register, select a discipline, and use the databases, texts, and library links to do library research. Patrons had three additional functions at their disposal. The first was the ability to add "personal" links—bookmarks to their favorite websites. Second, they had the ability to select multiple disciplines and thus refine the number of resources associated with "their" page. Finally, and to a small degree, patrons had the ability to change the graphic design of the page. Because of these customizable features and its implementation at NCSU Libraries, the system was officially called MyLibrary@NCState.

MyLibrary@NCState was packaged and distributed as open-source software, a newly coined term at that time. It was subsequently downloaded and installed in roughly two dozen libraries across the world. Some of these libraries used it in exactly the manner it was designed, and some of them are still accessible today.[2] Other libraries used parts and pieces of the system to build their own applications. For example, the OpenUniversity used only the underlying database structure.[3] On the other hand, Los Alamos National Laboratory used to MyLibrary@NCState concept and completely re-wrote the Perl modules.[4]

More importantly, the concept of MyLibrary—a user-driven, customizable interface to sets of library collections and services—became very popular. MyLibrary-like applications sprang up all over the library landscape. These implementations did not use the Perl modules and scripts written under the MyLibrary@NCState rubric, but they did organize content in an underlying database and allowed patrons to mix and match the content for their specific purposes.[5]

As a turnkey application, MyLibrary@NCState functioned correctly. It did not crash and it did not output invalid data. At the same time, MyLibrary@NCState did not fare very well when it came to usability tests. For example, Gibbons describes how the usability of MyLibrary was improved to meet the needs of course offerings.[6] In another article, Brantley describes how users had difficulty "understanding the discipline-specific nature" of MyLibrary@NCState.[7]

Its installation process was nonstandard and therefore difficult to implement. As written, MyLibrary@NCState was difficult to extend and enhance, and thus it did not truly benefit from its open-source nature. Data entry was tedious and for this reason its content was difficult to initialize and maintain. The idea of actively customizing a user interface was foreign to many users. People do not take an active role in customizing their user interfaces. They accept the defaults or unconsciously expect the user interface to adapt to their needs.[8] For all these reasons, MyLibrary@NCState's popularity lasted about five years, but for many of the reasons outlined previously, the concept of MyLibrary still seems viable.

The balance of this article describes two things: (1) how the current implementation of MyLibrary has evolved beyond the turnkey nature of MyLibrary@NCState, and (2) how the "new and improved" MyLibrary has been and can be used to create a number of digital library applications.

Eric Lease Morgan is Head of the Digital Access and Information Architecture Department, Hesburgh Libraries, University of Notre Dame, Indiana.

MyLibrary, relationships, and facet/term combinations

More than anything else, MyLibrary is intended to provide a framework for creating relationships between information resources and people. Most of the time these information resources are the traditional things of libraries such as books, journals, indexes, catalogs, manuscripts, and photographs. The people of MyLibrary are patrons and librarians. Relationships can be drawn between information resources and people through the use of facet/term combinations—a locally defined and institution-specific controlled vocabulary.

Information resources and people can be described in similar fashions. Resources, for example, are described with subjects. They are described according to their physical format and function. Patrons and librarians focus much of their energies in specific subjects: "I am majoring in philosophy." Sometimes people focus their attention on specific formats: "I need a journal article on . . ." Sometimes people are interested in particular functions: "I need a definition for . . ." People can belong to particular audiences and they might want to use audience-specific resources: "These resources are particularly useful for students in GEOG 203."

In our increasingly networked environment, it is just as important to create relationships between people as it is to create relationships between information resources and patrons. Librarians are not seen as the only authority on data and information. The opinions of one's peers play an important role too. Users want to read reviews, rank items according to various weights, and make decisions based on the thoughts of people like them. Through facet/term combinations applied to users, this is possible. Moreover, since users do not visit libraries as often as they used to, librarians need to figure out ways of staying in touch with their populations. By applying facet/term combinations to librarians as well as users, the librarians can know who their users are and users can easily identify subject experts.

Intended for use as the framework for a controlled vocabulary, the facet/term combinations of MyLibrary give the librarian and developer an opportunity to describe and relate the primary components of libraries—information resources and people. Through these facet/term combinations, conceptual links can be created between information resources and users, between users and librarians, and between librarians and resources. After creating a set of facet/term combinations, the librarian and developer can address increasingly popular desires such as but not limited to:

- As a librarian, this is the set of resources I curate . . .
- Because you are in this class, you might want to use . . .
- Here is a list of all the encyclopedias on the topic of . . .
- Here is a list of patrons who use the resources I curate . . .
- Here is a list of the full-text article indexes . . .
- Here is a list of articles on . . .
- The library owns the following special collections . . .
- These special collections can be used for this class . . .
- Other people in this class have also used . . .
- Other people like you have used . . .
- Recommended resources for this subject are . . .
- Resources for this subject are . . .
- The subject-specific librarian is . . .

To be able to address these issues, the librarian and the developer first create sets of facet/term combinations and then assign one or more of them to information resources, patrons, and/or librarians. After the assignments have been made, lists of relevant MyLibrary objects (information resources or people) can be generated by specifying—"joining" in relational database parlance—facet/term combinations held in common between the objects. For example, if many information resources, patrons, and librarians were classified using a Subjects/Astronomy facet/term combination, then the librarian and developer can create a list of astronomy-related resources for patrons, a list of astronomy-interested patrons for librarians, and list of astronomy-responsible librarians for patrons.

MyLibrary facets and terms

MyLibrary facets are intended to be the headings for very broad categories. MyLibrary terms are expected to denote examples of the facets. Facet/term combinations are expected but not required to be defined for every MyLibrary implementation. Every librarian and developer who uses MyLibrary is expected to define his or her own set of facet/term combinations. In the form of a simplified entity-relationship diagram, figure 1 illustrates how the relationships between information resources and people are modeled in MyLibrary.

An easy-to-understand facet might be Formats denoting the physical manifestation of an information resource. Terms associated with a Formats facet might include Books, Manuscripts, Journals, Microforms, Articles, Maps, Pictures, Movies, or Datasets. Given just about any information resource, a Formats facet/term combination can be assigned to it. For example, a library that owns the *Encyclopaedia Britannica* might "catalog" it with the

Formats/Books facet/term combination:

Title—Encyclopaedia Britannica
Facet/Term—Formats/Books

Another easy-to-understand facet might be called Research Tools, denoting things used to find data and information. Example terms might include Dictionaries, Thesauri, Manuals, Journal indexes, Library catalogs, Internet indexes, Encyclopedias, Atlases, or Almanacs. Continuing with the example above, *Encyclopaedia Britannica* might have an additional facet/term combination assigned to it:

Title—Encyclopaedia Britannica
Facet/Term—Formats/Books
Facet/Term—Research tools/Encyclopedias

An Audience facet might be created to denote classes of users. In an academic library, possible Terms might include Freshman, Sophomores, Juniors, Seniors, Graduate students, Instructors, Faculty, and Staff. Using a different information resource—say, *Dissertation Abstracts*—we might come up with a different set of facet/term combinations:

Title—Dissertation Abstracts
Facet/term—Research tools/Bibliographic indexes
Facet/term—Audiences/Graduate students

Using MyLibrary's facet/term combinations, it is almost trivial to create an authorities list. An Authors facet can be created to denote the creators of works. Specific names can be used as terms. Similarly, there might be a need or desire to include genre headings. Consequently, *The Adventures of Huckleberry Finn* might be described like this:

Title—The Adventures of Huckleberry Finn
Facet/term—Audiences/Adolescents
Facet/term—Authors/Mark Twain
Facet/term—Formats/Books
Facet/term—Genre/Coming of age stories
Facet/term—Genre/Novels

MyLibrary objects

Facet/term combinations are used to describe and create relationships between MyLibrary objects. These objects include information resources and people, and the people consist of users and librarians. The idea of facet/term combinations has been described above. This section describes the MyLibrary objects—information resources and people—in greater detail.

Information resources

Information resources are the traditional information-carrying "things" of a library. Typically they include books, journals, articles, manuscripts, indexes, catalogs, finding aids, etc. In order to organize and increase access to these materials, libraries systematically describe collections using rigorous cataloging procedures. With the advent of ubiquitous computing and the Internet, at least two things have happened regarding the "things of a library." First, they are increasingly less bibliographic in nature. While the number of books, journals, and articles is certainly not decreasing, the number of conference presentations, simulations, images, sounds, movies, and data sets is multiplying at an astounding rate. Second, because of this additional content, the traditional rigorous cataloging procedures of librarianship do not scale to the amount of work that needs to be done. Dublin Core metadata elements were created to address these problems. Facet/term combinations form the foundation for creating simple but local controlled vocabularies. Facet/term combinations plus Dublin Core metadata elements plus a number of other attributes brought along from MyLibrary@NCState for backwards compatibility are used to describe information resource objects in MyLibrary.

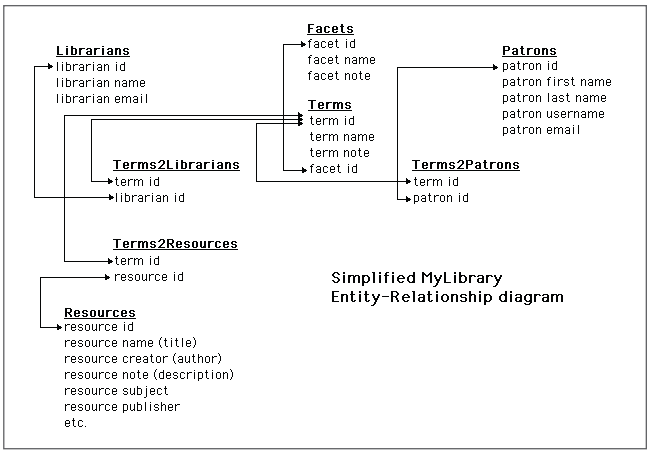

Figure 1. Simplified MyLibrary entity-relationship diagram. Facets have a one-to-many relationship with terms. Terms have a many-to-many relationship with resources, patrons, and librarians. After defining sets of facet/term combinations, the MyLibrary API allows librarians and developers to build interconnections between resources, patrons, and librarians.

Attributes

A few things ought to be noted about some of the MyLibrary attributes. First, many of the Dublin Core elements can be duplicated with facet/term combinations. The prime candidates are elements that can be expressed as database many-to-many relationships. The Dublin Core element called creator is an excellent example. Any

single information resource may have many creators, and any creator may be associated with many resources. Librarians and developers who use MyLibrary are able to place creator information in an attribute of a MyLibrary resource object and/or in a facet/term combination. The former usage is similar to traditional library cataloging technique and consequently requires additional overhead for editing records. The application of facet/term combinations makes it much easier to maintain database integrity as well as create browsable lists.

Just like creators, subjects might be better implemented as facet/term combinations, and the MyLibrary subject attribute might be used as a placeholder for keywords or non-controlled vocabulary terms. Each MyLibrary resource object might have multiple subjects. Using the facet/term approach, this is no problem to implement. Using the Dublin Core subject field approach, this is challenging, since the field is not repeatable. To circumvent this, librarians and developers are encouraged to delimit subject term values with predefined characters (such as "|"). Upon indexing or display, the subject attribute can be parsed into multiple values.

Identifiers

MyLibrary resource objects possess three distinct types of identifiers, and each has it own explicit use. The first is the MyLibrary resource identifier, which is a relational database key. It is non-assignable and non-editable by librarians or developers. It is an internal value used to maintain relational database integrity.

The second type of MyLibrary resource identifier is called the fkey, and it is used to denote a foreign key. This attribute is primarily intended to contain the value of an identifier from a remote information system like the 001 field of a MARC record. A better example includes the harvesting of records from OAI data repositories. Each record in each OAI repository has an Internet-wide unique identifier. This value is not a URL, but usually a combination of characters and numbers analogous to the 001 field of a MARC record. Each repository may also implement a concept called "sets," and each record might belong to multiple sets. When harvesting from a repository, the librarian and developer can save the OAI identifier as an fkey value, and when the same record from an alternative set is discovered, the associated resource object can be updated instead of duplicated.

The third type of identifier are Resource::Location objects. They are primarily intended for but not limited to URLs. Unlike all of the other resource attributes, Resource::Location objects are intended to have many values because information resources have many locations. For example, a library might have a printed version of *The Adventures of Huckleberry Finn*, and its location is denoted by a call number. A library might also have an electronic version, and its location is a URL. An online bibliographic database might be located at a particular URL, but its locally developed help text might be located at a different URL. Each Resource::Location object has three qualities: (1) a key, (2) a type, and (3) a value. The key is an internal relational database identifier. The type is an institution-defined value denoting the kind of location. Examples might include primary URL, help text URL, call number, local file name, and ISBN or ISSN. The value is an example of the type, and, in the case of Dublin Core elements, might very well be the identifier. Using MyLibrary Resource::Location objects, single information resources can be displayed and multiple locations can be associated with them.

Library services

Think creatively regarding the definition of resource objects. Think library services as well as books, journals, and databases.

Libraries are about more than collections. They are also about services applied against those collections. Libraries want to promote their services just as much as they want to promote access to bibliographic indexes, special collections, and the wealth of monographs. These services include bibliographic and information literacy sessions, circulation services (such as interlibrary loan, item recalls, renewals, or document delivery), library tours, one-on-one reference consultations, and online chats.

Each of these services has a title, a description, and probably a URL where details can be read online. MyLibrary resource objects provide a means to embody this information in a concise package. All that is missing are facet/term combinations to relate them to other information resources or people. Consider an Audience facet. Putting things on reserve is something of interest to instructors. Consider an Audience term called Instructors. Assign an Audience/Instructors facet/term combination to instructions for putting things on reserve. Things put on reserve are intended for use by students. Again, consider assigning something like an Audience/Students facet/term combination to instructions for using the reserve book room.

People—patrons and librarians

MyLibrary includes two types of objects representing people: patrons and librarians. Like information resource objects, librarian and patron objects are characterized using a number of attributes plus facet/term combinations. On one level, the patron attributes are simple and rudimentary only including things like first name, last name, username, password, e-mail address, URL, and image. This type of information was explicitly designed to map to the FOAF (Friend of a Friend) architecture in the hopes of future compatibility. Patron objects also

include attributes for things like last date visited and total number of visits. This information forms the basis for potential What's New? functionality. The patron object also includes functionality to record personal links for bookmarking features. The MyLibrary librarian object is even simpler than the patron object since it only includes attributes for name, e-mail address, and URL.

Just like the MyLibrary information resource objects, both the patron objects and the librarian objects can be mapped to facet/term combinations. Just as MLA Bibliography might be "cataloged" using a Subjects/English literature facet/term combination, a patron or librarian object can be "cataloged" in the same way. Once these sorts of relationships are established, recommendations can begin to take shape. Once patrons start bookmarking and associating particular resources and services to their identity, the system can take the next step and address things such as "People like you also used . . . " or "Popular resources in this area are . . ." Moreover, once facet/term combinations are associated with people, then relationships between people can be created and the system can answer statements such as "Other people interested in this topic include . . ." or "The patrons who are interested in this subject are . . ."

Establishing facet/term combinations for people is not as difficult as it may seem at first. In an academic library, much of this information can be gleaned from human resources data or the institution's registrar office. Libraries probably already get this information in one shape or another to populate their integrated library system circulation module. At the very least, this information includes a first name, a last name, and a unique institution identifier (possibly a username). Given this information, the librarian and developer could query the institution's directory services to discover institutional department and/or major field of study. Just as this information is loaded into the integrated library system to support borrowing, it can be loaded into a MyLibrary instance. Each department or major can then be mapped to facet/term combinations.

Privacy is a real issue with the inclusion of patron information in a MyLibrary instance. It should be taken very seriously. The use of MyLibrary does not assume the inclusion of patron information; it is more than possible to use MyLibrary and not have it contain any information about people. On the other hand, without this information a library prevents itself from providing the sort of services increasingly expected by its patrons. A discussion of the professional ethics of providing personalized services to library users in a computer-networked environment is beyond the scope of this article. Each library must weigh for itself the strengths, weaknesses, advantages, and threats of using information about patrons to provide individualized services.

Combining MyLibrary with other "toolboxes"

As a framework or toolbox, MyLibrary is intended to support only certain aspects of a digital library, namely, the collection of content, information about people, and a means of making relationships between them. MyLibrary is not intended to be an "integrated library system." It has no acquisitions module. It has no circulation module. It includes the only the most basic functionality for searching. Instead, librarians and developers are expected to combine MyLibrary with other tools to fulfill these functions.

For example, acquisitions functionality can be implemented by harvesting OAI content. By combining MyLibrary with another set of Perl modules called Net::OAI::Harvester, librarians and developers can import OAI-based content into a MyLibrary instance.[9] Feed Net::OAI::Harvester an OAI root URL, and it will systematically harvest remote metadata in any number of metadata formats. Since Dublin Core metadata is required of all OAI data repositories, and since MyLibrary supports a one-to-one mapping to Dublin Core elements, it is trivial to create MyLibrary resource objects based on each of the harvested records. Appendix A illustrates a simple yet complete OAI acquisitions application. It harvests journal article metadata from the Directory of Open Access Journals.

Just about any bibliographic metadata format can be mapped to Dublin Core. Examples include MARC, MARCXML, MODS, EAD, and TEI. To get content in these forms into a MyLibrary instance, the librarian and developer need to write a program reading bibliographic data, parsing out the desired information, and saving it to MyLibrary. Considering MARC data, the venerable Perl module called MARC::Record could be used to read and parse the data.[10] The other data formats are XML-based, and a Perl-based application supporting XSTL or XPath could be used to read and parse the data. In all of these cases the content of the MyLibrary instance should be considered brief and the fkey value might point to the original file on the local file system. Such MyLibrary resource objects are useful for syndication, search result displays, or browsable lists. If more detail is required, then the brief records can point to the full metadata through the fkey value.

MyLibrary is not intended to support search. That is because search is best supported not by a database but by an indexer.[11] There are myriad indexers available. Some of them include Swish-e, KinoSearch, Zebra, and Lucene.[12] To search the content of a MyLibrary instance, librarians and developers are expected to write reports against the instance and use them as the content for indexing. Appendix B illustrates a rudimentary but complete pro-

gram creating a KinoSearch index against a MyLibrary instance. Once the index is created, librarians and developers are expected to write interfaces to search the index. Appendix C illustrates one searching technique: get a query as input, search the index, return a record's ID value, lookup the record in MyLibrary, display.

In summary, MyLibrary first defines a number fundamental library objects (information resources, people, and a controlled vocabulary). It then supports a Perl-based application programmer interface (API) for doing input/output against these objects. The input can be garnered from any number of streams—manual data entry, tab-delimited text files, MARC or XML files, OAI, etc. The output can be XML files, RSS or Atom feeds, OAI, HTML subject pages, e-mail messages, or PDF files.

Production and demonstration applications

A number of diverse applications have been created with MyLibrary. Some of them are production services. Some of them are not fully developed and only exist to demonstrate the possibilities. This section briefly describes a number of them.

Alex Catalogue of Electronic Text

The Alex Catalogue of Electronic Text is a collection of just less than 14,000 public-domain documents from American literature, English literature, and Western philosophy. Much of the content comes from Project Gutenberg, but it also includes content from the defunct Eris Project of Virginia Tech and the Internet Wiretap Archive. Each MyLibrary resource object includes as much Dublin Core data as possible. The description attribute of each MyLibrary resources includes not an abstract of the electronic text, but an RDF/XML version of the original text. A report was written against the MyLibrary instance that saves the RDF/XML to the local file system. These files were then indexed with an open-source indexer called Zebra, and access to the index was provided through a Web Services–based protocol called SRU (Search/Retrieve via URL). Consequently, the catalogue is full-text searchable as well as searchable via title, creator, and subject. The contents of the subject fields were computed by analyzing each document and extracting statistically significant words. The searchable interface supports a Did You Mean? service by comparing search terms to alternative spellings and a WordNet thesaurus. The Catalogue's title and creator browsable lists are static HTML files built by a script written against the underlying MyLibrary instance. Finally, links to all of the documents and their subjects have been uploaded to Del.icio.us. To facilitate this, a script was written against the database extracting all the titles, their creators, and subjects ("tags"). These things were then sent to Del.icio.us via a Perl module implementing the Del.icio.us API.

Article Index

The Directory of Open Access Journals includes an OAI interface to its journal titles as well as some of its articles. The Article Index system harvested the article metadata and saved it to a MyLibrary instance. Along the way, journal titles and publishers were saved to underlying facet/term combinations and linked to each article. This enabled the creation of browsable lists via publisher and source. The content of the database was indexed using KinoSearch and made accessible via a Perl module written to implement SRU. Search results are displayed in a brief format. Details are available via a simple Asynchronous JavaScript and XML (AJAX-y) link. Appendixes A, B, and C illustrate the core of this application.

Catholic Research Resources Alliance

The Catholic Research Resources Alliance (CRRA) is a "portal" intended to highlight rare and unique materials of interest to Catholic scholars. Much of this content exists in archives. Archives use an XML format called EAD to describe their holdings. The CRRA provides a mechanism for ingesting these EAD files, parsing out controlled vocabularies, populating facet/term combinations accordingly, full-text indexing the EAD, and supporting a searchable/browsable interface to the entire content via SRU. The CRRA also supports ingesting MARC records as well as getting its input from online data-entry forms. Reports are written against the underlying MyLibrary instance allowing the CRRA's content to be accessible via OAI.

Facebook

A Facebook application has been written against the MyLibrary data of the Hesburgh Libraries University of Notre Dame's database-driven website. After Facebook users load the application into their profile, they are presented with a set of default recommended resources. The user then has the option to select a different set of resources based on subject terms presented in a pop-up menu. The resulting list of resources is then saved to the user's profile pane, giving easy access to the pertinent databases and indexes of his or her selected subject.

Library catalog

MyLibrary has been used to create a demonstration library catalog. About 300,000 MARC records were downloaded

from the Library of Congress. A program was written that reads each MARC record, crosswalks it to Dublin Core, and creates MyLibrary resource objects accordingly. Each MARC record is saved as an individual file on the file system. The whole collection is indexed with KinoSearch, and an SRU interface provides access to the index. As search results are returned, the existence of ISBN numbers is checked. If found, cover art and user reviews are retrieved and displayed from Amazon. Each record is displayed in a brief format, but links to a fully tagged format is available as well as MARCXML and MODS formats. Each record is also associated with a "Get it for me" link. Once clicked, the item is essentially checked out to the user. Each user then has a "bookshelf" link displaying the items they have borrowed.

Hesburgh Libraries, University of Notre Dame's database-driven website

The Hesburgh Libraries' database-driven website is probably the most extensive MyLibrary application in existence, and its primary purpose is to support the majority of the libraries' website. The system begins with the integrated library system where much (but not all) of the library's website content has been cataloged using traditional methods. Each item in the catalog destined for the website has been flagged with a local note denoting such. Each item's description has also been enhanced with facet/term combinations. On a nightly basis, all of the items destined for the website are exported from the catalog as MARC records. On a nightly basis, another script reads these records and updates a MyLibrary instance. Reports are written against the instance creating subject pages, format pages, tool pages, etc., complete with descriptions, recommendations, and links to associated librarians. Some information resources on the website are not deemed worthy of a record in the catalog. For these items, a manual data-entry form was created allowing bibliographers and subject specialist librarians to supplement the website's content. These resources are seamlessly integrated into the website along with the resources from the catalog. To facilitate search, reports are written against the MyLibrary instance and fed to Swish-e. The resulting index is then supplemented with the content of static Web pages to support Search This Site functionality. Using this database-driven and MyLibrary-based system, the content of the libraries' website has many fewer broken links because the links are all centrally maintained. The site also sports a common look and feel, making it easy for users to know where they are located in the system. This process also eliminates the need for selectors and subject specialist librarians to know any HTML. They can focus on content and the system can focus on presentation.

Future directions and conclusion

The MyLibrary modules work in the manner in which they were intended, and they continue to be distributed and supported as open-source software, but software is never complete.

MyLibrary is available from CPAN (Comprehensive Perl Archive Network). It is supported by a website complete with voluminous documentation, sample applications, access to a CVS repository, blog commentaries, and a mailing list with about 150 subscribers.[13] Yet despite the support, use of MyLibrary outside the University of Notre Dame has been underwhelming. I assume this is true because the number of Perl programmers in libraries is shrinking as the number other programming languages (PHP, Python, Ruby, Java, etc.) grows. The modularity of the system may also be a factor since most of the library profession can not write a computer program and therefore will have a difficult time understanding how to put MyLibrary into practical use. The idea of facet/term combinations used to describe information resources as well as people may be off-putting. Finally, because MyLibrary requires an underlying database to operate, the normal Perl installation process (perl Makefile.PL; make; make test; make install) can only be done after a bit of pre-installation processing. This is possibly another impediment to adoption—the installation process is a bit unusual.

Despite these issues, MyLibrary works very well for the University of Notre Dame, and a number of improvements are planned. First, the underlying database contains a table for user reviews, and a Perl module needs to be written allowing input/output against these tables. Similarly, MyLibrary presently includes tables for keeping track of how often a particular resource is used and by whom, but there is no module to update the table. Future work will enhance this statistics table and implement the statistics module. Finally—and most importantly—work will be done to make it easy to do input/output against a MyLibrary instance through a REST-ful (Representational State Transfer) interface. As defined by REST, this interface will exploit the four transfer methods of HTTP (GET, POST, PUT, and DELETE) to retrieve, create, edit, and remove MyLibrary objects from the underlying database. By exploiting REST-ful computing techniques, at least two things will be enabled. First, application programmers will be able to use their favorite computer language to maintain a MyLibrary instance. There will be no need to know Perl; REST is computer-language independent. Second, through the use of REST-ful computing MyLibrary content will be more easily syndicated. For example, the output of a REST-ful MyLibrary interface could be manifested in many flavors of XML. Atom comes to mind, but an RDF/XML representation may be

more expressive. The output of a REST-ful interface to MyLibrary could also be manifested as a JSON (Javascript Object Notation) data structure, making it easier to integrate MyLibrary content in AJAX-y interfaces.

As more and more library collections and services are manifested in a computer-networked environment, the need to provide these collections and services in new and different ways increases. MyLibrary is an attempt to address this issue, and it has met with qualified success.

Acknowledgments

An enormous debt of gratitude goes to Rob Fox of the Hesburgh Libraries, University of Notre Dame for writing the bulk of the MyLibrary Perl modules. Rob and I sat down together for a couple days in 2003 to learn about object-oriented Perl programing techniques from Ed Summers (now working at the Library of Congress). We then coupled that experience with the needs and desires of the libraries to articulate and design MyLibrary as it is today. While I wrote bits and pieces of the modules and used them to write many applications, Rob was the person who really got his hands dirty.

References and Notes

1. Keith Morgan and Tripp Reade, "Pioneering Portals: MyLibrary@NCState," *Information Technology and Libraries* 19, no. 4 (Dec. 2000): 191–98.

2. The author has identified at least four MyLibrary@NCState implementations still up and running from across the world, including The Wellington City Libraries in New Zealand, www.wcl.govt.nz/mylibrary (accessed Feb. 19, 2008); the Buswell Library Electronic Access Center of Wheaton College, http://libweb.wheaton.edu/mylibrary (accessed Feb. 19, 2008); the Biblioteca Mario Rostoni at the Universita Carlo Cattaneo, http://mylibrary.liuc.it/mylibrary (accessed Feb. 19, 2008); and Auburn University, http://mylibrary.auburn.edu (accessed Feb. 19, 2008).

3. Anne Ramsden, James McNulty, Fiona Durham, Helen Clough, and Nicola Dowson created MyOpenLibrary for the OpenUniversity in the United Kingdom. "MyOpenLibrary is an online personalised library system developed for Open University students and staff. Every individual user can have a virtual library 'shelf' or space which is tailored to meet their particular needs. The system is based on the MyLibrary software originally developed at North Carolina State University and now supported at Notre Dame University. The software has a simple basic interface, groups resources under clear headings, and provides a tick box facility for selecting and removing resources. Users sign in because it is a personalised service, but then they can customise the colour and settings of their page according to need, and if they are familiar with the Internet, they add their own personal favourite links. There is a quick search facility for searching individual databases and Internet search engines. The system is currently being used by 20 Open University courses and this is expected to increase year on year. For more information see http://myopenlibrary.open.ac.uk/." MyOpenLibrary includes 80,768 patrons (79% of the total student population of OpenUniversity), 111 disciplines, 12,731 e-books, 500 databases, and 38,708 journals. From personal correspondence between the author and James McNulty (Feb. 19, 2008).

4. "The LANL implementation of MyLibrary @ LANL is an object oriented redesign of the Mylibrary source code created by Eric Lease Morgan of North Carolina State University. The code was designed by two summer students Andres Monroy-Hernandez and Cesar Ruiz-Meraz from Monterrey, Mexico. The code is currently maintained by Mariella di Giacomo and Ming Yu." From http://library.lanl.gov/lww/mylibweb.htm (accessed Feb. 19, 2008).

5. A search against Google for "mylibrary" returns myriad results, many of which are MyLibrary-like applications and services. Representative samples include MyLib of Malaysia's National Digital Library, www.mylib.com.my (accessed Feb. 19, 2008); My Library of Hennepin County Library, www.hclib.org/pub/ipac/MyLibrary.cfm (accessed Feb. 19, 2008); and MyLibrary of Coastal Carolina University, www.coastal.edu/library/mylibrary.html (accessed Feb. 19, 2008).

6. Susan Gibbons, "Building Upon the MyLibrary Concept to Better Meet the Information Needs of College Students," *D-Lib Magazine* 9, no. 3 (Mar. 2003), www.dlib.org/dlib/march03/gibbons/03gibbons.html (accessed Feb. 19, 2008).

7. Steve Brantley, Annie Armstrong, and Krystal M. Lewis, "Usability Testing of a Customizable Library Web Portal," *College and Research Libraries* 67, no. 2 (Mar. 2006): 146–63, www.ala.org/ala/acrl/acrlpubs/crljournal/backissues2006a/marcha/Brantley06.pdf (accessed Feb. 19, 2008).

8. Udi Manber, Ash Patel, and John Robison, "Experience with personalization on Yahoo!" *Communications of the ACM* 43, no. 8 (Aug. 2000): 35–39.

9. Net::OAI::Harvester, http://search.cpan.org/dist/OAI-Harvester (accessed Feb. 19, 2008).

10. MARC::Record, http://search.cpan.org/dist/MARC-Record (accessed Feb. 19, 2008).

11. Search is a function best supported by an indexer, not a relational database. Relational databases are tools for organizing and maintaining data. Through the process of normalization, relational databases store data unambiguously and efficiently. Because relational databases store their information in tables, records, and fields, it is necessary to specify the tables, records, and fields when querying a database. This requires the user to know the structure of the database. Moreover, standard relational databases do not support full-text searching nor relevance-ranked output. Indexers excel at search. Given a stream of documents, indexers parse tokens (words) and associate them with document identifiers. Searches against indexes return document identifiers and provide the means to retrieve the documents without the necessary knowledge of the index's structure. Indexers are weak at data maintenance. In a well-designed database, authority terms can be updated in a single location and reflected throughout the database. Indexers do not support such functionality. Databases and indexers are two sides of the same information retrieval coin. Together they form the technological core of library automation.

12. There are a growing number of open-source indexers available on the Web, including Swish-e, http://swish-e.org (accessed Feb. 19, 2008); KinoSearch, www.kinosearch.com/kinosearch (accessed Feb. 2008); Zebra, http://indexdata.com/zebra (accessed Feb. 19, 2008); and Lucene, http://lucene.apache.org (accessed Feb. 19, 2008).

13. The canonical home page for MyLibrary version 3.x is http://mylibrary.library.nd.edu (accessed Feb. 19, 2008).

APPENDIX A

```
# harvest DOAJ articles into a MyLibrary instance

# require

use MyLibrary::Core;

use Net::OAI::Harvester;

# define

use constant DOAJ => 'http://www.doaj.org/oai.article';  # the OAI repository

MyLibrary::Config->instance( 'articles' );  # the MyLibrary instance

# create a facet called Formats

$facet = MyLibrary::Facet->new;

$facet->facet_name( 'Formats' );

$facet->facet_note( 'Types of physical items embodying information.' );

$facet->commit;

$formatID = $facet->facet_id;

# create an associated term called Articles

$term = MyLibrary::Term->new;

$term->term_name( 'Articles' );

$term->term_note( 'Short, scholarly essays.' );

$term->facet_id( $formatID );

$term->commit;

$articleID = $term->term_id;

# create a location type called URL

$location_type = MyLibrary::Resource::Location::Type->new;

$location_type->name( 'URL' );

$location_type->description( 'The location of an Internet resource.' );

$location_type->commit;

$location_type_id = $location_type->location_type_id;

# create a harvester and loop through each OAI set
```

```
$harvester = Net::OAI::Harvester->new( 'baseURL' => DOAJ );
$sets      = $harvester->listSets;
foreach ( $sets->setSpecs ) {
 # get each record in this set and process it
 $records = $harvester->listAllRecords( metadataPrefix => 'oai_dc', set => $_ );
 while ( $record = $records->next ) {
  # map the OAI metadata to MyLibrary attributes
  $FKey      = $record->header->identifier;
  $metadata  = $record->metadata;
  $name      = $metadata->title;
  @creators  = $metadata->creator;
  $note      = $metadata->description;
  $publisher = $metadata->publisher; next if ( ! $publisher );
  $location  = $metadata->identifier; next if ( ! $location );
  $date      = $metadata->date;
  $source    = $metadata->source;
  @subjects  = $metadata->subject;
  # create and commit a MyLibrary resource
  $resource = MyLibrary::Resource->new;
  $resource->fkey( $FKey );
  $resource->name( $name );
  $creator = ''; foreach ( @creators ) { $creator .= "$_|" }
  $resource->creator( $creator );
  $resource->note( $note );
  $resource->publisher( $publisher );
  $resource->source( $source );
  $resource->date( $date );
  $subject = ''; foreach ( @subjects ) { $subject .= "$_|" }
  $resource->subject( $subject );
  $resource->related_terms( new => [ $articleID ]);
  $resource->add_location( location => $location, location_type => $location_type_id );
  $resource->commit;
 }
}
```

```
# done
exit;
```

APPENDIX B

```
# index MyLibrary data with KinoSearch
# require
use KinoSearch::InvIndexer;
use KinoSearch::Analysis::PolyAnalyzer;
use MyLibrary::Core;
# define
use constant INDEX => '../etc/index';        # location of the index
MyLibrary::Config->instance( 'articles' );   # MyLibrary instance to use
# initialize the index
$analyzer   = KinoSearch::Analysis::PolyAnalyzer->new( language => 'en' );
$invindexer = KinoSearch::InvIndexer->new(
  invindex  => INDEX,
  create    => 1,
  analyzer  => $analyzer
);
# define the index's fields
$invindexer->spec_field( name => 'id' );
$invindexer->spec_field( name => 'title' );
$invindexer->spec_field( name => 'description' );
$invindexer->spec_field( name => 'source' );
$invindexer->spec_field( name => 'publisher' );
$invindexer->spec_field( name => 'subject' );
$invindexer->spec_field( name => 'creator' );
# get and process each resource
foreach ( MyLibrary::Resource->get_ids ) {
  # create, fill, and commit a document with content
  my $resource = MyLibrary::Resource->new( id => $_ );
  my $doc      = $invindexer->new_doc;
  $doc->set_value ( id          => $resource->id );
```

```
  $doc->set_value ( title       => $resource->name )      unless ( ! $resource->name );
  $doc->set_value ( source      => $resource->source )    unless ( ! $resource->source );
  $doc->set_value ( publisher   => $resource->publisher ) unless ( ! $resource->publisher );
  $doc->set_value ( subject     => $resource->subject )   unless ( ! $resource->subject );
  $doc->set_value ( creator     => $resource->creator )   unless ( ! $resource->creator );
  $doc->set_value ( description  => $resource->note )     unless ( ! $resource->note );
  $invindexer->add_doc( $doc );
}
# optimize and done
$invindexer->finish( optimize => 1 );
exit;
```

APPENDIX C

```
# search a KinoSearch index and display content from MyLibrary
# require
use KinoSearch::Searcher;
use KinoSearch::Analysis::PolyAnalyzer;
use MyLibrary::Core;
# define
use constant INDEX => '../etc/index';       # location of the index
MyLibrary::Config->instance( 'articles' );  # MyLibrary instance to use
# get the query
my $query = shift;
if ( ! $query ) { print "Enter a query. "; chop ( $query = <STDIN> )}
# open the index
$analyzer = KinoSearch::Analysis::PolyAnalyzer->new( language => 'en' );
$searcher = KinoSearch::Searcher->new(
  invindex => INDEX,
  analyzer => $analyzer
);
# search
$hits = $searcher->search( qq( $query ));
# get the number of hits and display
$total_hits = $hits->total_hits;
```

```
print "Your query ($query) found $total_hits record(s).\n\n";
# process each search result
while ( $hit = $hits->fetch_hit_hashref ) {
  # get the MyLibrary resource
  $resource = MyLibrary::Resource->new( id => $hit->{ 'id' });
  # extract dublin core elements and display
  print "            id = " . $resource->id    . "\n";
  print "          name = " . $resource->name . "\n";
  print "          date = " . $resource->date . "\n";
  print "          note = " . $resource->note . "\n";
  print "     creators  = ";
  foreach ( split /\|/, $resource->creator ) { print "$_; " }
  print "\n";
  # get related terms and display
  @resource_terms = $resource->related_terms();
  print "       term(s) = ";
  foreach (@resource_terms) {
    $term = MyLibrary::Term->new(id => $_);
    print $term->term_name, " ($_)", '; ';
  }
  print "\n";
  # get locations (URLs) and display
  @locations = $resource->resource_locations();
  print "  location(s) = ";
  foreach (@locations) { print $_->location, "; " }
  print "\n\n";
}
# done
exit;
```