Improving Software Sustainability: Lessons Learned from *Profiles in Science*

Marie E. Gallagher; Lister Hill National Center for Biomedical Communications, U.S. National Library of Medicine; Bethesda, Maryland, USA

Abstract

The Profiles in Science® digital library features digitized surrogates of historical items selected from the archival collections of the U.S. National Library of Medicine as well as collaborating institutions. In addition, it contains a database of descriptive, technical and administrative metadata. It also contains various software components that allow creation of the metadata, management of the digital items, and access to the items and metadata through the Profiles in Science Web site [1]. The choices made building the digital library were designed to maximize the sustainability and long-term survival of all of the components of the digital library [2]. For example, selecting standard and open digital file formats rather than proprietary formats increases the sustainability of the digital files [3]. Correspondingly, using non-proprietary software may improve the sustainability of the software--either through in-house expertise or through the open source community.

Limiting our digital library software exclusively to open source software or to software developed in-house has not been feasible. For example, we have used proprietary operating systems, scanning software, a search engine, and office productivity software. We did this when either lack of essential capabilities or the cost-benefit trade-off favored using proprietary software. We also did so knowing that in the future we would need to replace or upgrade some of our proprietary software, analogous to migrating from an obsolete digital file format to a new format as the technological landscape changes. Since our digital library's start in 1998, all of its software has been upgraded or replaced, but the digitized items have not yet required migration to other formats.

Technological changes that compelled us to replace proprietary software included the cost of product licensing, product support, incompatibility with other software, prohibited use due to evolving security policies, and product abandonment. Sometimes these changes happen on short notice, so we continually monitor our library's software for signs of endangerment. We have attempted to replace proprietary software with suitable in-house or open source software. When the replacement involves a standalone piece of software with a nearly equivalent version, such as replacing a commercial HTTP server with an open source HTTP server, the replacement is straightforward. Recently we replaced software that functioned not only as our search engine but also as the backbone of the architecture of our Web site. In this paper, we describe the lessons learned and the pros and cons of replacing this software with open source software.

Introduction

When making choices about the file formats we would use in our digital libraries, we chose standard and open file formats over proprietary formats. Our goal was to increase the chances that the contents of the files would survive because they could be sustained over time. Keeping files in open formats that are readable by multiple, widely available software packages owned or maintained by different sources, rather than locked inside of a proprietary package, seemed compatible with that goal. So far this strategy has worked. In the twenty years since we started digitizing items for our digital libraries, we have not yet had to migrate our master files to other formats.

Because of the attention understandably focused on the digital objects, the software that is essential to build, house, protect and disseminate the contents of our digital libraries is easy to overlook. After software is up and running properly, it is tempting to consider that part of the library finished with the exception of adding new features to the library. But such complacency would be a mistake. Similar to the objects in the digital library, we have found that the software requires ongoing monitoring. Unlike our digital items, our software has required continual modifications, upgrades and replacements. This has been our experience even during periods when new features were frozen, and it held true whether the software was proprietary, commercial, public domain, open source, or written in-house. Various factors, often external to our project, have driven our need for upgrades, modifications and replacements. These factors included our inability to sustain the rising costs of ongoing product licensing and product support, software upgrades that lacked backward compatibility with previous versions, discontinued support for needed functions, the end of support for an entire product, existing software's inability to meet new policy requirements, incompatibility with new hardware or new operating systems, discovery of security flaws, and discovery of software bugs.

When making choices about software improvements to overcome these problems, we tried to choose solutions that could be sustained for the long-term. Sometimes we did not have multiple choices for solutions; sometimes we had no readily available choices at all. Sometimes a well-documented, well-supported proprietary solution that met our needs existed, so we used it. When our data (and the custom business rules we developed) did not become locked inside the proprietary software, proprietary software solutions have worked for us. When open source software solutions existed that met our needs without requiring extensive customization, they have worked. When no software solution to our needs was available, we wrote our own. Each solution has had advantages and pitfalls.

Background

In the early 1990s, our experiments in digitization and metadata involved a digital library containing scanned papers, a library that was shared among a group within the same building. At that time, only a few Web servers existed, no graphical Web browsers were available, and "metadata" was not a widely known term. We used a client-server system of integrated proprietary commercial off-the-shelf (COTS) hardware and software that let us perform black & white scanning, data entry and editing, search and retrieval, black & white printing, and backup of the software, database, and images. The client used for data entry and editing ran on multiple operating systems. One of the strengths of the system was that we were able to import and export metadata and scanned images into and out of the system. However, the digital library was accessible only through the proprietary client software, and there was a license fee for each copy of the client. We were also not able to make basic software changes, such as modifying the search function to be case-insensitive.

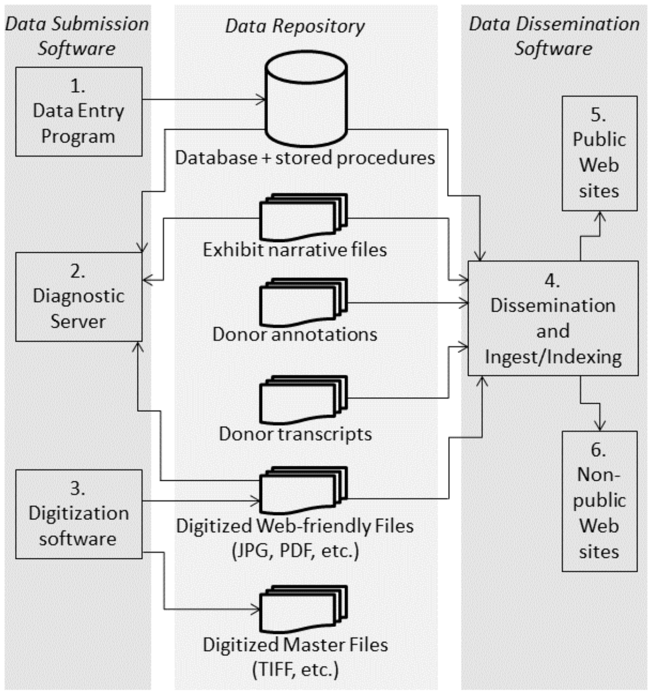

Figure 1. Software underlying the Profiles in Science digital library support either data submission (data creation and editing) or data dissemination (browse and search through the Web).

Although this system of proprietary commercial hardware and software worked well and did not require specialized expertise to use, we needed to be able to change and add functionality, and we wanted to make the digital library's contents freely and widely available. As Web browsers and servers came along, we added our own Web interface to the digital library and served the content using the public domain National Center for Supercomputing Applications (NCSA) HTTPd [4] software. Our system grew into a hybrid of proprietary and open source software. We could create data and images and import them, or use the proprietary client to enter and edit data. The content was made available through the proprietary client as well as to anyone with a Web browser and Internet connection. Our file formats expanded, too. Because TIFF was not natively supported in Web browsers, we made our images available in GIF format and eventually PDF format. The TIFF images, some of which are almost twenty years old, are still our digital masters and have not been modified. We no longer serve the GIF images, but we still make available the PDF files.

When we started building the *Profiles in Science* digital library in 1998, the database and digital items remained at the heart of the system, and the purpose of most of the software components was to either create (submit) content or distribute (disseminate) content. Our overall architecture, as illustrated in Figure 1, has been stable over time. We found no commercial data entry program (Item 1 in Figure 1) that met our needs, so we contracted with a software developer to build one using Microsoft (MS) Access Visual Basic for Applications (VBA) [5] in part because MS Access was available on all our project staff PCs. Although we remained dependent on the proprietary MS Access software, writing the data entry program in-house allowed us to modify and customize it in ways not possible with the proprietary client software. However, we did not need software development services when using the proprietary client software, while we did need such ongoing expertise for writing and maintaining our own custom software. Occasionally Microsoft discontinued support for functions that our data entry program employed, so upgrading to new versions of MS Access required ongoing modifications.

Our software developer also wrote our Diagnostic Server (Item 2 in Figure 1) which provides a Web-based read-only interface to the database, dozens of reports and views of the contents of the database, and previews of how the data would look after being made available to the public. The Diagnostic Server software was written in-house and depends on Adobe ColdFusion Enterprise Edition [6] software. So far, ColdFusion upgrades have not resulted in emergency software modifications due to discontinued vendor support for functions. Software development services are employed for bug fixes, security fixes, and adding new features.

We use proprietary commercial software for digitization (Item 3 in Figure 1) including the vendor-supplied scanner drivers and software, as well as Adobe Photoshop [7] and Adobe Capture software. New scanner drivers and Photoshop upgrades cause temporary disruptions while operators learn how to use the upgraded interfaces. The Adobe Capture software is no longer supported, so we expect it will eventually stop functioning due to future hardware or software incompatibilities. Using the proprietary digitization software has required ongoing development of protocols and training, but no software development services.

Our dissemination and ingest/indexing software (Item 4 in Figure 1) were also written in-house. Dissemination software that extracted the relevant fields from the database was written in MS VBA. This software was less complicated than the data entry program, so it did not make use of functions that were discontinued and it required little maintenance.

Other dissemination software extracted and copied other types of data such as contributed files and digitized files from the data repository (Figure 1), transformed the database into HTML pages, and created a search index. This software was written in the open source Perl programming language [8]. It was complicated and required software development expertise to modify and maintain. We used the proprietary commercial Verity Search '97 Information Server as the search engine on our Web servers (Items 5 and 6 in Figure 1). While Verity's search results were highly configurable, we could not modify it to add or change functionality such as changing the way it handled synonyms. We did not encounter software or hardware compatibility problems with the Verity search software in spite of several hardware and operating system upgrades. However, support for the product ended, and Verity was acquired by Autonomy.

We used the open source Apache HTTP Server [9] on our Web servers. Upgrades to the Apache HTTP server sometimes required changes to the accompanying configuration files as new configuration directives were added.

Replacing the Search Engine

Four factors drove the effort to replace the proprietary Verity Search '97 Information Server with the in-house search software Essie [10] originally developed for ClinicalTrials.gov [11]. The first factor was the desire to find a replacement for the unsupported Verity Search '97 Information Server software. The second was the desire to experiment with the Essie search software and its synonymy capabilities while expanding Essie's capabilities to accommodate *Profiles in Science* needs such as highlighting search terms within PDF files. The third was to improve response time of search requests. The fourth factor was our desire to move away from the Solaris hardware platform due to rising hardware support costs.

Verity Search '97 Information Server was a standalone search engine that could be replaced by another search engine without requiring modification to the other data dissemination software. Adopting Essie resulted in adopting the full architecture in which it was used by ClinicalTrials.gov: Essie was used not only during searches, but also when generating all Web pages sent to the user's browser. This replacement affected all of the *Profiles in Science* data dissemination software.

A software developer wrote a new program in the freely available Java programming language [12] to extract the relevant fields from the database as well as to extract and copy the contributed files and digitized files from the data repository. The Java program created master XML files, suitable for input to Essie. Essie used the master XML files in addition to its synonymy and stop word lists to create its own search indexes used during searches. The developer also wrote Java programs to convert user queries into requests to Essie to either return appropriate data needed to generate each Web page at run time, or to return data needed to generate a list of search results. The open source SAXON XSLT processor [13] used the in-house XSL style sheets and output from Essie packaged as XML to create HTML that is returned to the user's Web browser. The left column in Figure 2 illustrates the path taken by a user request to see a browsable page as well as to initiate a search and obtain a list of results. In order to improve performance, the software developer built a cache of the generated HTML pages, eliminating the need to make repeated calls to Essie. The right column in Figure 2 illustrates the request process for viewing a digitized file or a cached file. The switch from the Verity Search '97 Information Server to Essie resulted in speedier replies, an ability to search for synonyms of terms, and the end of our dependence on software that could not be maintained.

While the effort to implement the Essie search engine on *Profiles in Science* was underway, the open source search platform Apache Solr [14] was under development. As Apache Solr gained popularity, we focused on replacing the in-house developed Essie software with the open source Apache Solr. Apache Solr is flexible and highly configurable, and it required effort for us to understand it well enough to use it. In order to make the replacement as quickly as possible, the software developer's plan for the Solr implementation was to reuse as much of the existing data dissemination code as possible. This resulted in minimal changes to the Java code that extracted the relevant fields from the database, and to the code that extracted and copied the contributed files and digitized files from the data repository. The software developer configured Solr to reuse the same XML files produced for ingest into Essie. New code had to be written to appropriately format the output from Solr into the previously established XML format. New code was also needed to handle Solr's different search syntax. Minimal changes to the XSL style sheets were needed. The most complicated code that had to be rewritten converted the many possible user requests for Web pages into the appropriate format for Solr instead of for Essie.

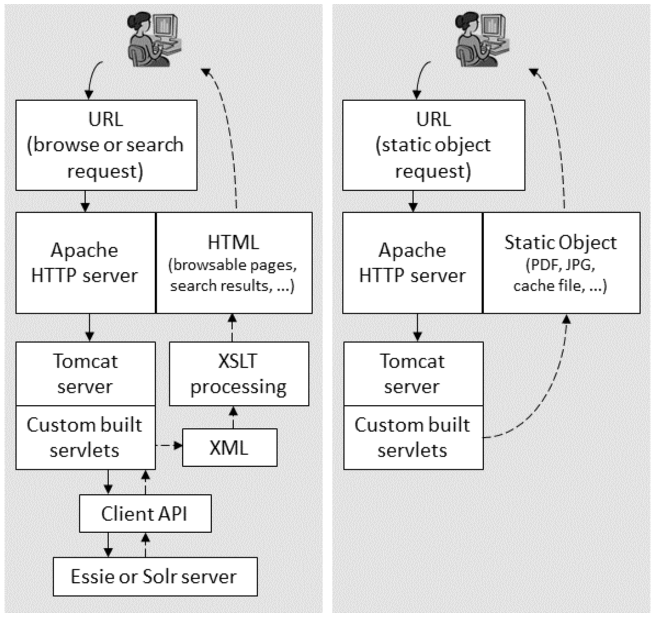

Figure 2. Path of a user request to view search results or a browsable page.

The first step was getting Solr to index the XML to create its search indexes. Essie handles XML natively, while Solr required configuration changes to handle the XML files as input. Problems were sometimes difficult to debug, usually because of unexpected behavior or a lack of warning messages. Although there are books about Solr as well as an active user community, documentation was sometimes lacking or confusing. When reference books, consulting the user community, and consulting co-workers did not explain unexpected behavior, trial and error was employed. The software developer used Solr's admin interface to confirm that Solr successfully indexed the XML and could return expected search results.

The next part of the replacement process required rewriting, in Java, all of code that transformed user requests for browsable pages into Solr queries so Solr could return the appropriate data to form the correct browsable page. As shown in Figure 2, when a user requests a page on *Profiles in Science*, the Apache HTTP server passes the request to the open source Apache Tomcat [15] server. Java servlet code, running inside Tomcat, passes the request to a client API, which transforms and sends a request to Solr. Solr responds to the client API, which passes the response to the Java servlet code, which produces the XML content needed to form the page. The XML and appropriate XSL style sheet are then processed to generate the HTML page that is returned to the user (and stored in the cache for quick retrieval by the next user).

More Java code had to be written to highlight search terms within PDF files. Essie had already been enhanced to return the character offsets and word lengths of search terms within PDF files. Page numbers, character offsets and word lengths were needed to produce the Adobe Highlight File Format XML to generate the highlights within a PDF inside of a Web browser. Solr returned marked up text with tags identifying the terms to be highlighted. This required new code to parse for the Solr tags and then to calculate page numbers, character offsets, and word lengths of search terms.

One of Essie's strengths is its ability to search for synonyms. Initially, Solr could not load Essie's large (545 MB) file of multi-word synonyms. During the course of our Solr implementation, Solr developers improved the Solr synonym handling to require less memory. Simultaneously, our software developer omitted redundant and obscure synonyms to reduce the size of the synonyms file until Solr would load it. This will be an ongoing effort because synonyms are added over time, so experiments with Solr synonymy will continue.

Testing was a significant part of the replacement effort. Because Essie and Solr played a part in generating every Web page on *Profiles in Science*, over 100,000 pages needed to be compared. We employed an in-house developed tool, htmldiff, written in open source Perl, to compare and report differences between pages. Differences pinpointed the areas to debug. At the end of the replacement, the pages generated through Essie and Solr were verified to be 100% identical. Testing Essie and Solr search results was less straightforward. We expected that Essie and Solr would rank results differently. This was at least in part due to Essie and Solr's different handling of synonymy and the likelihood that searching unclean optical character recognized (OCR) text was causing confusion. More experimentation with Solr's weighting and ranking may improve results in the future.

The replacement required hundreds of hours of software developer time over the course of a year. During the replacement period, at least five releases of Solr became available, fixing some problems and introducing others. Implementing a new installation rather than a replacement might have been less complicated.

Table 1 contains a brief summary of our experiences and observations comparing the in-house developed software Essie with the open source software Solr.

Table 1. Essie and Solr comparison

Cost	We obtained both Essie and Solr at no cost. However, both require ongoing software developer expertise to install, configure, customize, document, enhance, maintain, and upgrade.
Customization	Anyone can report bugs, request changes, and contribute code to the open source Solr. If we want changes to the in-house developed Essie, we ask the software developer.
Documentation	Reference books, commercial training classes, online documentation, tutorials, and user communities provide help for Solr. The in-house software developer answers questions about Essie.
Features	Solr possesses more features than our application currently uses. As needs arise, we will use Solr's additional capabilities such as facets. The user community influences the types of features added to Solr. Our software developer wrote workarounds for features we need that Solr did not have. Essie possesses the features we need, and more are added as needs arise.
Maintenance	For both Essie and Solr, some upgrades are optional, such as those adding new features that we do not use. Other upgrades are required to fix bugs, compatibility problems, or security issues. Maintenance requires software developer expertise. If backward compatibility is lacking, development of new code or workarounds is required.
Performance	At index time, Solr indexes faster than Essie ingests. At search time, we detect no difference in performance between Essie and Solr.
Platform	Both Essie and Solr are written in freely available Java.
Synonymy	Being able to search for synonyms is useful for our digital library. Sometimes familiar terms such as "heart attack" are used, and other times medical terms such as "myocardial infarction" are used. We want to be able to find both regardless of which term the user specifies. We are still experimenting with Solr synonymy and expect its development to continue. Essie, developed specifically with synonym handling in mind, meets our application's synonymy needs.

Lessons Learned

We strive to select, write, build or configure our software to minimize the need for maintenance and maximize its sustainability. Continual threats to the stability of our software (see Figure 3) have made ongoing monitoring, fixing, upgrading and replacement necessary, independent of adding new functionality. Over the life of our project, these threats to stability have increased rather than decreased. Using proprietary software, open source software or in-house developed software each has advantages and risks.

The proprietary software we have used was well tested, well documented, and stable. Although it performed as advertised, it was sometimes not possible to extend it to suit our needs or to meet new security or policy requirements. We avoided using proprietary software that did not have import and export capabilities. The absence of this characteristic could be an impediment to the sustainability of both our software and data. Without export capabilities, it could be labor intensive or impossible to migrate to alternative software in the future. We remained aware of the possibility of discontinued development of proprietary software we chose, and tried to have alternative plans in case support ended. Some proprietary software had minimal support or upgrade costs, but some had significant costs. Future license, support and upgrade costs became an important consideration.

Widely used, mature open source software, such as the seventeen year-old Apache HTTP server, has allowed us to leverage the expertise of software developers external to our organization. Also, every user of that software potentially served as a collaborator who could report bugs, contribute code, and lead to improvements. Open source software that was standalone, well documented, straightforward to understand, and easy to configure required less intensive maintenance efforts. Using open source software in a way that called for extensive configuration, modification and customization required a considerable investment of resources. Attempts to highly customize or extend open source software can require significant software development expertise and many hours of ongoing effort, all of which could impede sustainability. Our implementation of Solr required a thorough understanding of our application and of Solr's capabilities. Open source software that is young and actively under development sometimes produces frequent releases, and we occasionally experienced unexpected changes in behavior between releases. The quality of open source software documentation has varied and is dependent on the developer community donating its uncompensated time.

In-house developed software has been customized to meet our project's specific needs. When we could not fill a need with existing software, either proprietary or open source, we have had software developers at the National Library of Medicine, either with our project or another project, create new software. When we wanted in-house software changed, our developers have had the in-depth understanding to change it. When a bug or security flaw was detected, they could fix it. We could specify new features to add, and we could give priority to the addition of each feature. When we had a question, we could ask in-house developers directly. When we wanted to perform an experiment using the software, we could ask in-house developers to do it. However, hiring and keeping in-house software development expertise requires significant ongoing resources. All of our in-house software depends upon proprietary software, public domain software and open source software. Our in-house developed software has been subject to the same threats posed by rapidly changing technology as proprietary and open source software. So experienced software developers, intimately familiar with their software, have been needed to successfully monitor and maintain in-house developed software. Documentation varies depending on the application and the individual software developer. Sometimes only the creator of the in-house developed software possesses an in-depth understanding of it, which may put its sustainability at risk.

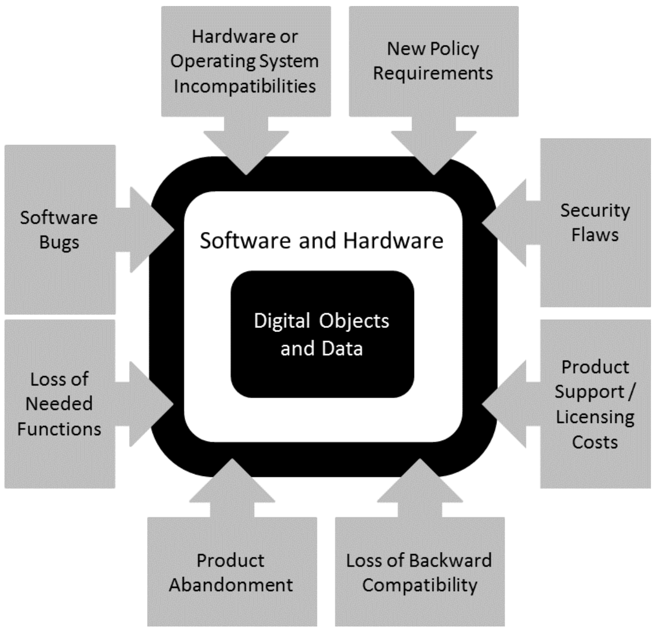

Figure 3. Some of the forces that threaten the stability of our software.

Acknowledgments

The author thanks Mr. Karl Wolf, the software developer who writes and maintains the data dissemination software underlying the *Profiles in Science* Web site. Mr. Wolf replaced the in-house developed Essie software with the open source Solr/Lucene software and provided helpful consultations during the preparation of this paper. Mr. Ajay Kanduru and Ms. Xiaohui Ma shared their experiences with implementing the Solr/Lucene software on the NLM Gateway [16]. Dr. Alexa McCray lead our digital library research and development efforts from 1992 through 2004. Mr. Michael Flannick wrote the original data entry program and Diagnostic Server software. Mr. Loi Hang and Ms. Xiaohui Ma maintain the data entry program and Diagnostic Server, and are actively migrating the data entry program. Mr. Michael Gruen wrote the first Essie-based version of the data dissemination software. Dr. Russell Loane wrote the Essie search software and enhanced it for use by *Profiles in Science*. Mr. Erik Dorfman wrote the htmldiff tool. Essential encouragement and guidance has been provided by project leaders Drs. Olivier Bodenreider, Clem McDonald, and Aaron Navarro, who spearheaded the

experiment to replace Essie with Solr/Lucene on *Profiles in Science*.

This work is supported by the Intramural Research Program of the NIH, National Library of Medicine. The Library, the world's largest library of the health sciences, is a component of the National Institutes of Health, U.S. Department of Health and Human Services.

References

[1] *Profiles in Science* (U.S. National Library of Medicine, Bethesda, MD) http://profiles.nlm.nih.gov/. Accessed Feb 4, 2013.

[2] A.T. McCray and M.E. Gallagher, "Principles for Digital Library Development." Communications of the ACM 44, 5 (May 2001).

[3] Sustainability of Digital Formats: Planning for Library of Congress Collections (Library of Congress, Washington, DC) http://www.digitalpreservation.gov/formats/. Accessed Feb 4, 2013.

[4] NCSA software and technologies. (Board of Trustees of the University of Illinois) http://illinois.edu/lb/imageList/2943. Accessed Feb 4, 2013.

[5] Visual Basic for Applications Language Reference for Office 2010 (Microsoft Corporation) http://msdn.microsoft.com/en-us/library/gg278919%28v=office.14%29.aspx. Accessed Feb 4, 2013.

[6] Adobe ColdFusion 10 family (Adobe Corporation) http://www.adobe.com/products/coldfusion-family.html. Accessed Feb 4, 2013.

[7] Adobe Photoshop family (Adobe Corporation) http://www.adobe.com/products/photoshop.html. Accessed Feb 4, 2013.

[8] The Perl Programming Language (Perl.org) http://www.perl.org/. Accessed Feb 4, 2013.

[9] Apache HTTP Server Project (The Apache Software Foundation) http://httpd.apache.org/. Accessed Feb 4, 2013.

[10] N.C. Ide, R.F. Loane and D. Demner-Fushman. "Essie: A concept-based search engine for structured biomedical text," J Am Med Inform Assoc. 14, 3 (2007) http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2244877/. Accessed Feb 4, 2013.

[11] ClinicalTrials.gov (U.S. National Library of Medicine, Bethesda, MD) http://clinicaltrials.gov/. Accessed Feb 4, 2013.

[12] Java (Oracle Corporation) http://www.java.com/. Accessed Feb 4, 2013.

[13] SAXON: The XSLT and XQuery Processor (Michael H. Kay, Saxonica Limited) http://saxon.sourceforge.net/. Accessed Feb 4, 2013.

[14] Apache Solr (The Apache Software Foundation) http://lucene.apache.org/solr/. Accessed Feb 4, 2013.

[15] Apache Tomcat (The Apache Software Foundation) http://tomcat.apache.org/. Accessed Feb 4, 2013.

[16] NLM Gateway (U.S. National Library of Medicine, Bethesda, MD) http://gateway.nlm.nih.gov/. Accessed Feb 4, 2013 .

Author Biography

Marie E. Gallagher, a computer scientist in the U.S. National Library of Medicine's Lister Hill National Center for Biomedical Communications since 1990, is the project leader of the Digital Library Research and Development team. The team investigates systems and develops the software underlying Profiles in Science. Ms. Gallagher earned her B.S. degree in Computer Science and Mathematics from the College of William and Mary in Virginia.