White Paper Report

Report ID: 104101

Application Number: HD5147211

Project Director: Geraldine Heng (heng@mail.utexas.edu)

Institution: University of Texas, Austin

Reporting Period: 9/1/2011-8/31/2012

Report Due: 11/30/2012

Date Submitted: 11/30/2012

White Paper: Bibliopedia, Drupal as a Web Application Framework, and the Semantic Web

Geraldine Heng, Michael Widner, and Jason Yandell
The University of Texas at Austin
November 30, 2012

Table of Contents

Introduction

Bibliopedia is a research platform designed to crawl scholarly resources including JSTOR, the Library of Congress, the Arts and Humanities Citation Index, and similar data sources, extract metadata about works cited, convert that data into a semantic web format, aggregate the different repositories, then display the results on a wiki-style website for the scholarly community to verify, add to, annotate, elaborate, and discuss. We envisage Bibliopedia as an open, research-enabling platform designed to unify the many disparate, closed silos of scholarly information available today, and that remain difficult and time-consuming to use. Our first goal was to extract and transform bibliographic data into a linked data format consistent with semantic web requirements, and to create large volumes of cross-references among texts, making digitized scholarly texts exponentially more useful to researchers and to machine analysis.

The primary innovations Bibliopedia achieves are: 1) the aggregation and cross-referencing of separate silos of scholarly data; 2) the transformation of that information into a format consistent with the semantic web; and 3) crowd-sourcing the verification and elaboration of that data. Mapping and cross-referencing large-scale, high-volume scholarship also means that unexpected connections can be found and brought to light, along with less-known original works that might otherwise remain unread. Moreover, formatting scholarly references for the semantic web will make this data available to a far broader community and enable unexpected innovations. Bibliopedia will generate custom bibliographies and visualizations based on search results, facilitating a wide variety of scholarly inquiry and discovery. Most importantly, Bibliopedia is designed for ease of use, so as to substantially broaden participation to attract the largest possible range of humanities scholars as its user base, in particular scholars who do not normally use digital tools.

The project team for this phase of development consisted of Geraldine Heng (The University of Texas at Austin), Principal Investigator; Michael Widner (formerly The University of Texas at Austin, now Stanford University), Technical Director and co-creator; Jason Yandell, co-creator and Lead Developer; and Ana Boa-Ventura (The University of Texas at Austin), Web Designer.

This paper describes the technologies used, our reasoning in choosing each, and our recommendations for future projects that could learn from our work. In particular, we focus on the importance of modular, test-driven design, the benefits of the semantic web, and the flexibility and power of the Drupal Content Management System (CMS).

System Architecture

The Bibliopedia project consists of four main components: 1) servers to host and run all the components of the system; 2) custom code to crawl data sources, retrieve article and book data, transfer the data to the 3) citation extraction engine, and then submit the results to the 4) Drupal-based web application. The Bibliopedia team successfully created a scalable, data-source agnostic crawling architecture, adapted the ParsCit citation extraction software, and developed a web-based application for publishing data in a linked open data format and for tracking the changes to the data made by the scholarly community. The code has been in stable release for some time and is available as open source software on Github: https://github.com/bibliopedia/bibliopedia/tree/master/Development

The Semantic Web and Scholarly Metadata

Coined by Sir Tim Berners-Lee, the semantic web, according to the W3C, has two main reasons for existence:

> It is about common formats for integration and combination of data drawn from diverse sources, where on the original Web mainly concentrated on the interchange of documents. It is also about language for recording how the data relates to real world objects. That allows a person, or a machine, to start off in one database, and then move through an unending set of databases which are connected not by wires but by being about the same thing. (http://www.w3.org/2001/sw/)

The semantic web also "provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries" ("What is the Semantic Web?", W3C).

The semantic web is transforming the Internet from a collection of pages and data readable only by humans to one that machines can understand and process. Semantic web technology promises the ability automatically to determine meaning and then infer connections among different elements, thereby vastly improving search capabilities, discovery of new information, and the overall usefulness of the Internet. Just as information accessible only to humans comprises the great majority of the general Internet, so too is data about scholarly literature locked away in text that computers cannot process without great difficulty. At best, search engines for repositories such as JSTOR permit researchers to query author name, journal titles, and keywords, but once a work is found, the search stops. No connections among works are found precisely because machines cannot currently read that data. Although Google Scholar attempts to show citations of articles, its usefulness is highly limited because it does not make clear the relationships among articles, present very limited metadata about each article (if any), fails to provide for community elaboration or correction, and includes only works that are publicly available. Yet despite its limitations, Google Scholar stands as a significant technological advance beyond keyword-based search engines such as those provided by JSTOR and Project Muse.

As we gain access to more data sources Bibliopedia will, by aggregating data from as many sources as possible, converting citations into semantic web format, and then cross-referencing an ever-growing database of scholarly works, be able not only to overcome many of the limitations of Google Scholar and become a powerful research tool in its own right, but also to make a valuable contribution to the growing semantic web. Introducing high quality metadata about humanities scholarship to the semantic web will enable others in the semantic web/linked data world to process that data in new, unexpected ways that will accrue further benefits to the scholarly community. For example, the standards underlying the semantic web make data visualization and automated inferences about relationships trivially easy rather than the complex problems such tasks currently present. Bibliopedia will, then, through the innovation of placing metadata about scholarly literature into a linked data format, open up a vast range of possible future innovations and analyses based on that data, which is currently locked away and readable only by select humans.

Another virtue of a linked data format is that it will help resolve many of the challenges inherent in metadata, some will inevitably remain. Rather than attempt to solve this incredibly complex problem through automation alone, then, Bibliopedia will, in the process of displaying its results for human consumption, also provide for human feedback in the form of correction and elaboration. A common disadvantage of fully automated text analysis and data extraction tools such as Google Books, Google Scholar, and other digital research tools is that their automatic parsers have errors in their metadata that they do not allow subject matter experts to repair. Bibliopedia will pursue the goal of unifying that information into an environment that not only displays the information efficiently, but actively encourages crowd-sourcing metadata on books, articles, and publications of all kinds. In thus opening data up to revision by the scholarly community, Bibliopedia can build on the strong work of mature data silos, improve overall data quality, and provide the academic community at large a continuously evolving research tool.

Thanks to the native support for RDFa, a lightweight semantic web data format, in Drupal 7, we are able to create and consume linked data in a very straight-forward manner. Drupal allows for the creation of mappings among its native content formats and the data structures described in different linked data ontologies. Moreover, Drupal provides a simple interface for importing other ontologies as needed. We settled upon some of the most widely used data formats: Dublin Core, Friend of a Friend (FoaF), and the Bibliographic Ontology (BIBO). Drupal allowed us to blend these ontologies and ensure that all records for journal articles, journals, authors, etc. are available as linked data.

Server Infrastructure

To ensure reliability and scalability while reducing server administration time, we settled upon cloud-based servers to host all databases, the website, and the crawler code. The most performant and cost-efficient providers we found were Linode.com and Amazon Web Services (AWS). Both Linode and AWS provide web-based dashboards that allow for server administration, backups, and monitoring. This choice allowed us to bypass the difficulties often encountered when trying to provision servers from a university's IT department. Further, the systems administration tasks (performed by Widner) were reduced to a bare minimum as the companies providing the cloud servers manage all backups and hardware maintenance.

The citation extraction and web application components of Bibliopedia exist on a Linode.com Linux server that provides an Apache web server, a MySQL database, PHP, Perl, and other common components of a LAMP (Linux-Apache-MySQL-PHP) stack. Widner installed and configured all necessary software. The crawler code runs on a Windows-based server hosted by Amazon's Elastic Cloud Compute (EC2) service.

Crawler Code

Design Principles

The crawler code discovers and parses data from various sources including metadata from JSTOR, citation links from WikiPedia, and MARC records from library sources such as the Library of Congress. Yandell designed the system to run on free and open source Linux platforms as well as on commercial platforms. It is modular, extensible and supported by automated tests. As we gain access to new data sources, the modular design allows them to be incorporated into the system with ease.

The technical design focused initially on performance, modularity, and interoperability. Yandell determined that these goals could best be met by developing an automated test suite at the same time as the project itself. The automated ("unit") testing framework provided a number of advantages, discussed below. Other critical design challenges gained priority when it was discovered that the structure and data JSTOR returned changed as the project progressed. This discovery increased the design focus on data provenance, the ability to continuously operate in the face of ever-changing data, and the ability to parse and unify the semantics of the data crawled by Bibliopedia.

Yandell designed the crawler to ingest data from numerous sources and to interoperate with a wide variety of technologies from the proprietary to the standards-compliant. The Bibliopedia crawler parses all data into an internally-standard format and operates on data in that format for later stages of the process. Yandell also decided that the internally-standard format to rely on should be one that was accessible by the widest array of tools available, rather than only by this one crawler. This design led to our choice of semantic web technologies as the basis for the project. The code retains extracted data (from any sources) in RDF XML files (http://en.wikipedia.org/wiki/RDF/XML), one of the central, standard technologies of the semantic web. This design choice means that the data in the system is in a common format and accessible with standard tools, thereby making it interoperable and comprehensible to a wide audience even in the earliest stages of the process.

Yandell established the performance of the crawler early in the project. It is capable of running in parallel on as many processors and network cards as the server running it possesses. Execution is entirely asynchronous, meaning that the crawler never stops processing due to input/output (IO) operations like retrieving data from the internet or writing it to disk. So long as there remains more data to process, the crawler will process it. This approach can reduce processing time by multiple orders of magnitude by, for example, reducing processing time from hours to seconds. The asynchronous approach to code, however, was historically quite challenging to write and even more challenging to maintain over time. Yandell chose to rely on functional programming techniques that have gained significant prominence in performance-intensive applications. This proved to be an effective choice as the stated performance characteristics were achieved with very few lines of code, and that module never had to be revisited in the face of other changes. These features provide Bibliopedia an architecture that is scalable and performant and thus can serve as a model for other large-scale infrastructure projects in the digital humanities. Scaling performance is often a problem for even projects with large teams and budgets. Bibliopedia thus provides a model for managing these challenges in the initial phases of design.

Another crucial aspect of the project was the retention of provenance data for every record, which requires tracking changes to each record back to its source. At a minimum, this requires tracking the date, time, and source URL for each piece of data. Initially some code was written to track these changes. However, locking curation into the code and thus outside of the realm of domain experts is at odds with the goals of the project. The final solution relies on the version management capabilities offered by the web application aspects of the project which Widner built using Drupal. This solution duplicates the revision tracking aspects of Wikipedia, thereby serving the dual goals of deploying standard technologies and methods and of tracking data as it is processed and transformed.

Results

Leveraging unit tests (and other forms of automated testing) proved useful in a number of ways. Since a test for the code could be established before the code itself was written, it was possible to evaluate the strategic value of technical tasks as they were being performed. It was also of vital importance in identifying correct functioning of the code as well as continued functioning of the code. For example, automated tests made it possible for our team to discover immediately the changes to the JSTOR data format and the disappearance of the JSTOR API.

The ability to operate in the face of ever changing data was necessitated by JSTOR's internal (and not publicly communicated) technology choices. Soon after the first crawler was developed, it was discovered that JSTOR stopped adhering to the broadly used Dublin Core metadata standards. The fact that the data were constantly in flux necessitated numerous refinements to the JSTOR parser module of the Bibliopedia code. What is more, JSTOR seems to have abandoned any adherence to Dublin Core, OAI-*, or any other public ontologies as of our latest data retrieval. By October 2012, the nature and depth of the data changes thus demanded a rewrite of the JSTOR parsing module.

Yandell's adherence to a modular programming approach nevertheless minimized the impact to the project as a whole and to the crawler specifically. This experience suggests that modularity of code that relies on data from outside sources is not just a nice feature: it is critical to the completion and continued utility of any project. Such a design philosophy allows us to anticipate and adapt to dramatic fluctuations in data sources beyond the control of the project.

One solution to issues arising from data instability would be to repeatedly query and catalog data (i.e., data scraping: http://en.wikipedia.org/wiki/Data_scraping). While this approach effectively addresses issues related data instability, it may not be legally advisable, particularly in the case of JSTOR. Many data sources have Terms of Service that explicitly prohibit non-API access from programs. Moreover, such methods are less reliable than API queries. It is thus critical that all levels of project leadership pay close attention to data retention policies of all data sources.

Citation Extraction

To extract citation data from plain text resources (such as JSTOR's Data for Research service) requires a machine-learning engine trained to recognize bibliographic data in a variety of formats. Rather than train our own engine (a time-consuming an error-prone process), we used the open source Perl software ParsCit (http://aye.comp.nus.edu.sg/parsCit/), which powers CiteSeerX, to identity and extract citations. The source code for ParsCit is available on Github here: https://github.com/knmnyn/ParsCit. This software provides a web service to which one submits plain text data and receives back XML files that identify authors, titles, journal names, date, books, and other relevant bibliographic data. Moreover, its authors trained it on a wide array of citation formats, including those common to the humanities, and multiple (mostly Western European) languages. Widner installed and configured ParsCit on the Linode server running the web application and databases for the project. This aspect of the project demonstrates once again the benefits of using existing technologies wherever possible. The development of a citation extraction engine is, itself, a lengthy and difficult process that would have been well beyond the scope of this project.

Web Applications in Drupal

The web application is responsible for receiving data from the crawler via a web services API, transforming them for consumption by the semantic web, managing user accounts and access levels, displaying data, and enabling the scholarly community to edit the machine-generated data. Widner built the web application using Drupal 7, an open source Content Management System (CMS), and a large number of third-party modules that extend Drupal's functionality (see "Appendix A: Drupal Modules" for a complete list). The final capabilities of the web application significantly exceed the initial design specifications and include the follow features:

Features

- RESTful API (http://en.wikipedia.org/wiki/Representational_state_transfer) to receive and expose data; receives data in JSON format
- Transforms data into linked data using Dublin Core (http://dublincore.org/), BIBO (http://bibliontology.com/), and FoaF (http://www.foaf-project.org/) ontologies
- Provides a SPARQL (http://en.wikipedia.org/wiki/SPARQL) endpoint for linked data queries
- Consumes linked data from other resources to enrich Bibliopedia-generated data
- Wiki-style editing to allow users to curate metadata; all changes create new versions to enable a review of the revision history and ensure integrity of data
- Provides social platform for discussion/contextualization of citations
- User account authentication and management
- Access control to content
- Semantic similarity scores among content
- Advanced search capabilities using Apache Solr
- Traffic and activity logging (Drupal logs and Google Analytics)
- Support for saving pages to Zotero libraries

Along with the theming and overall configuration, Widner also designed the content types to store journal, article, book, and author data, configured the system to transform that data into linked data (via the RDFx module; http://drupal.org/project/rdfx), and designed the way different content types refer to others. A concrete example clarifies this process:

The crawler code, after discovering a journal article and extracting citation data, submits the data to the web application RESTful API in JSON format. The web application maps the different data elements (author name, article title, etc.) to the appropriate linked data ontologies and publishes the entry. A journal article entry links to a pages for its author(s) and the journal in which it was published.

Web application development also demonstrated the difficulty in using Drupal 7 to build a complicated linked data web service. Although the modules exist, many of them are still under heavy development and required the use of development rather than production code to meet all of this project's needs. These challenges aside, the web application Widner designed exceeds the goals laid forth in the original project proposal. The full application is available on Github: https://github.com/mwidner/bibliopedia/tree/rebuild0.2/bibliopedia.org

Moreover, despite the difficulties involved, the functionality Widner designed with Drupal shows yet again the benefits of using stable, mature open source software. Any one of the features present in the web application would require years of development were they undertaken from scratch with custom code. By using Drupal, Widner was thus able to design a complex, feature-rich application in a relatively short time span. Drupal's ease of use and widespread adoption among academic communities, moreover, means that others can implement the platform Widner designed for use in their own projects.

Concluding Recommendations

The design philosophy and technologies behind Bibliopedia can serve as a model for how digital humanities and library science tools and infrastructure can make use of linked data, why they should do so, and how we can take advantage of coding practices (such as test-driven development and modular architecture) common among software professionals.

Drupal has proven to be a robust, flexible, and feature-rich platform for developing web applications that are interoperable with other technologies, for publishing and consuming semantic web data, and for providing a community portal for engagement with that data. Its ability to work with standard protocols and formats (e.g., XML, linked data, RESTful APIs) also makes Drupal an excellent choice for a large array of web publishing applications. Further, it saves a great deal of time and allows projects to develop complex workflows and websites that far exceed the possibilities of what could be accomplished with custom code in the same timeframe.

Our experience with cloud-based solutions has been entirely positive. Not only are such servers cost effective and easy to maintain, but they are also easily scaled to handle any amount of traffic. Moreover, they eliminate the need for the procurement, housing, and administration of server hardware, which leaves project members free to focus on software development.

Finally, the adoption of coding practices such as modular design and test-driven development allowed us to be confident that the different parts of a complex system worked as intended. It also allowed us to recognize problems with our data sources as soon as they appeared. Perhaps most importantly, the modular design allows us to extend the project to encompass other data sources without requiring any major changes to the code or the overall system, thereby making it adaptable to unforeseen opportunities.

Related to our use of test-driven development is the issue of dealing with unexpected problems. A significant roadblock appeared near the end of this phase of the project because JSTOR disabled its Data for Research API near the end of our project phase. The loss of the JSTOR web service API means that we were unable to crawl their data as planned and will need to turn to other data sources. While the Bibliopedia system is designed to make such changes possible and even expected, it will nevertheless require further work to make this data usable. On a brighter note, JSTOR reports that they are working on a new API, which would allow us to surmount this challenge. Despite this challenge, the design of the Bibliopedia system means that we able to discover this change quickly and that we will be able to move to other data sources and other formats without problems. Indeed, our focus from an early stage on the principles of our design and the need for flexibility was proven effective in the face of this challenge. Had we, for instance, written code focused only on the JSTOR API, we would have been left with an unusable system that required a complete rewrite. Instead, our system remains functional and adaptable to such challenges, thereby proving the importance of a logical and clear design philosophy to guide development of large digital humanities projects.

Appendices

Appendix A: Drupal Modules

Below is a list of the non-core Drupal modules used in the Bibliopedia web application. The prevalence of alpha, beta, release candidates, and developer versions (14 total) indicates the cutting-edge nature of the application. For all of these instances, stable production releases are either unavailable or possess bugs or other limitations when used in the ways Bibliopedia requires.

Package	**Name**	**Type**	**Version**
Chaos tool	Chaos tools (ctools) suite	Module	7.x-1.2
Features	Features (features)	Module	7.x-1.0
Feeds	Feeds (feeds)	Module	7.x-2.0-alpha5+56-dev
Feeds	Feeds Admin UI (feeds_ui)	Module	7.x-2.0-alpha5+56-dev
Fields	Entity Reference (entityreference)	Module	7.x-1.0-rc3
Fields	Link (link)	Module	7.x-1.0
Other	Backup and Migrate (backup_migrate)	Module	7.x-2.4
Other	Entity API (entity)	Module	7.x-1.0-rc3
Other	Job Scheduler (job_scheduler)	Module	7.x-2.0-alpha3
Other	Libraries (libraries)	Module	7.x-2.0
Other	Pathauto (pathauto)	Module	7.x-1.2
Other	Raphaël (raphael)	Module	7.x-1.0
Other	Semantic Similarity (semantic_similarity)	Module	7.x-1.0-alpha1
Other	Token (token)	Module	7.x-1.3
RDF	External RDF Vocabulary Importer (evoc)	Module	7.x-2.0-alpha4
RDF	RDF UI (rdfui)	Module	7.x-2.0-alpha4
RDF	RDFx (rdfx)	Module	7.x-2.0-alpha4
RDF	SPARQL API (sparql)	Module	7.x-2.0-alpha4
RDF	SPARQL Endpoints Registry (sparql_registry)	Module	7.x-2.0-alpha4
RDF	SPARQL Views (sparql_views)	Module	7.x-2.0-beta1+11-dev
Services	Services (services)	Module	7.x-3.1+72-dev
Services - servers	REST Server (rest_server)	Module	7.x-3.1+72-dev
User interface	Wysiwyg (wysiwyg)	Module	7.x-2.1
Views	Views (views)	Module	7.x-3.5
Views	Views UI (views_ui)	Module	7.x-3.5
Other	Bibliopedia (bibliopedia)	Theme	1.x

Appendix B: Automated Tests

Section 1: Unit Tests

```
CitationSemanticExtraction
  CombineJStorAndOpenCalais
```

```
    ExploreComposition
---------------------------------------------
Data.Tests
  SampleDomainFixture
    CanSaveSampleDomainItem
---------------------------------------------
Jstor
  JstorTests
    CanGetProperlyFormattedXmlFromJstor
    CanPersistJstorTypes
    CanScourMassiveSubject
      CanScourMassiveSubject(10,1005)
    dc_deserializes_into_multiple_attributes
    EscapeQueryWorksProperly
    ExtensionMethodReturnsCitations
    GetCitationsXml
    JstorServiceIsUp
    SearchReturnsResults
  ResourceTests
    ArgumentContainsMultipleResults
    ArgumentContainsSingleResult
    CanLoadResource
    CannedDocumentHasCitations
---------------------------------------------
JstorCrawl
  ControllerFixture
    SearchBySubjectByAuthorByWork
---------------------------------------------
MongoConnect
  MongoDB: Connecting
    Can connect
    Can inspect database
---------------------------------------------
OpenCalais
  BasicConnectionFixture
    CanExtractComplexText
    CanExtractSimpleText
---------------------------------------------
ParCit.Test
  ParsCitServiceIntegrationTests
    is_result_deserializable_as_json
    is_service_running
---------------------------------------------
PersistJstorAsTriples
  DublinCoreConverterTests
    CanConvertCannedDcsToRdf
---------------------------------------------
PublishedWorks.Tests
  DatabaseFunctionality
    Can_CrossReference_Two_Works
    Can_Save_Work
  MappingsAreWorking
    Can_Correctly_Map_Article
    Can_Correctly_Map_Author
    Can_Correctly_Map_BinaryData
    Can_Correctly_Map_Book
    Can_Correctly_Map_DataMinedWork
    Can_Correctly_Map_Journal
    Can_Correctly_Map_JsonData
    Can_Correctly_Map_LibraryIdentifier
    Can_Correctly_Map_MinedData
    Can_Correctly_Map_Publisher
    Can_Correctly_Map_Range
    Can_Correctly_Map_Subject
    Can_Correctly_Map_TextData
    Can_Correctly_Map_Work
    Can_Correctly_Map_WorkDetails
    Can_Correctly_Map_XmlData
```

```
WikiInteraction
  EntityQueueTests
    CanCreateEntityQueue
    CanLocatePersistentQueueByType
    CanPopItem
    CanPushItem
  InteractionTests
    CanConnectToWiki
    RenderTestData
```

Section 2: Behavioral

```
Feature: Drupal data
        In order to work with data in Drupal
        As a bot
        I want to be able to perform basic operations

Background:
        Given credentials
        And a connection

Scenario: Create
        When I create a new test node
        Then I can determine that it succeeded
        And I get some data back
```

```
Feature: Drupal connection
        In connect to Drupal
        As a bot
        I want to be able to connect
        And I want to be able to perform basic GET on a Node

Scenario: Connect
        Given a connection
        And credentials
        When I can perform a basic get
        Then I get some data back

Scenario: Create a test item
        Given sample data
        And a connection
```

Then I can create a new node