Open Journal Systems

Research Report by Salman Bakht, Pehr Hovey, Aaron McLeran

Summary

Open Journal Systems (OJS) is an online management and publishing system for peer-reviewed journals developed by the Public Knowledge Project, a partnership among the University of British Columbia, the Simon Fraser University Library, the School of Education at Stanford University, and the Canadian Centre for Studies in Publishing at Simon Fraser University. Open Journal Systems's open source software is designed with the purpose of "making open access publishing a viable option for more journals, as open access can increase a journal's readership as well as its contribution to the public good on a global scale"[1] and provides tools to aid with every stage of the publishing process. Additionally, OJS offers a set of reading tools which can optionally be added to the journal. As of January 2009, OJS was currently used by over 2000 journals.[2]

Description

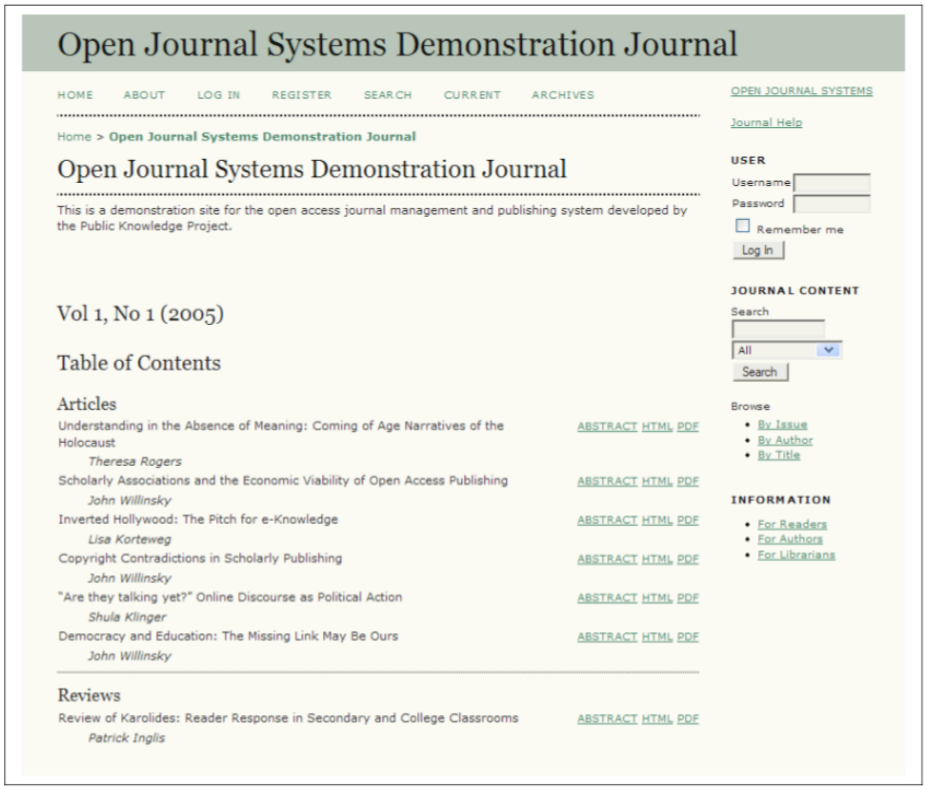
The Public Knowledge Project, founded by John Willinsky in 1998, investigates "knowledge management strategies to improve both the scholarly quality and public accessibility and coherence of this body of knowledge in a sustainable and globally accessible form."[3] Open Journal Systems offers one solution for improving both the quality and accessibility of online journals. Several features of OJS are listed in the documentation:

1. OJS is installed locally and locally controlled.
2. Editors configure requirements, sections, review process, etc.
3. Online submission and management of all content.
4. Subscription module with delayed open access options.
5. Comprehensive indexing of content part of global system.
6. Reading Tools for content, based on field and editors' choice.
7. Email notification and commenting ability for readers.
8. Complete context-sensitive online Help support.
9. Payments module for accepting journal fees.[4]

OJS allows for a large amount of content and layout customization as is shown in screenshots of various journal located at http://pkp.sfu.ca/ojs-customization.[5] However, the screenshots of the

[1] "Open Journal Systems," Public Knowledge Project, 2009, 6 Feb. 2009 <http://pkp.sfu.ca/?q=ojs>.
[2] "A Sample of Journals Using Open Journal Systems," 2009, 6 Feb. 2009 <http://pkp.sfu.ca/ojs-journals>.
[3] "What Is the Public Knowledge Project?," 2009, 6 Feb. 2009 <http://pkp.sfu.ca/about>.
[4] "OJS in an Hour: An Introduction to Open Journal Systems, Version 2.2.1.0," Public Knowledge Project, 2008, 6 Feb. 2009 <http://pkp.sfu.ca/files/OJSinanHour.pdf>.

OJS demonstration journal below show the basic layout and functionality presented to the reader. (See Figure 1 and Figure 2.[6])


Open Journal Systems Demonstration Journal

HOME ABOUT LOG IN REGISTER SEARCH CURRENT ARCHIVES

Home > Open Journal Systems Demonstration Journal

Open Journal Systems Demonstration Journal

This is a demonstration site for the open access journal management and publishing system developed by the Public Knowledge Project.

Vol 1, No 1 (2005)

Table of Contents

Articles

Understanding in the Absence of Meaning: Coming of Age Narratives of the Holocaust — ABSTRACT HTML PDF
Theresa Rogers

Scholarly Associations and the Economic Viability of Open Access Publishing — ABSTRACT HTML PDF
John Willinsky

Inverted Hollywood: The Pitch for e-Knowledge — ABSTRACT HTML PDF
Lisa Korteweg

Copyright Contradictions in Scholarly Publishing — ABSTRACT HTML PDF
John Willinsky

"Are they talking yet?" Online Discourse as Political Action — ABSTRACT HTML PDF
Shula Klinger

Democracy and Education: The Missing Link May Be Ours — ABSTRACT HTML PDF
John Willinsky

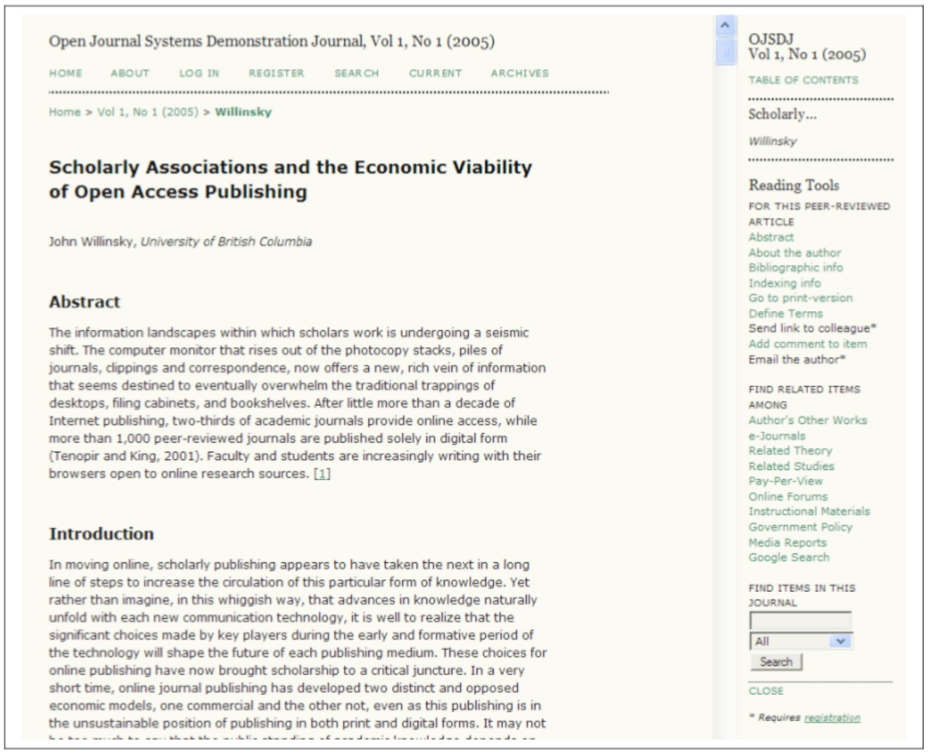
Reviews

Review of Karolides: Reader Response in Secondary and College Classrooms — ABSTRACT HTML PDF
Patrick Inglis

Figure 1 OJS Table of Contents

Figure 1 shows a table of contents listing articles within the selected volume, with navigation and text search shown in the header and right sidebar. Figure 2 shows an HTML article shown after choosing a link on the table of contents. (Articles can also be made available in PDF format.) In this screen, the right sidebar shows a set of reader tools, as chosen by the journal editors. These reading tools link to information about the article and author, articles by the same author or related by subject, search tools, a dictionary, online forums, a reader comment box, etc. Additional HTML content, such as text about the journal and submission procedures, ads, embedded audio and video, can be added to the pages. In addition, OJS provides a set of journal statistics including number of articles, acceptance rate, and registered readers. A means for payment and subscription-based access control are also provided, although there are also options for allowing open access to archived articles a number of months after publication.

[5] "OJS Customization," 2009, 6 Feb. 2009 <http://pkp.sfu.ca/ojs-customization>.
[6] "OJS in an Hour: An Introduction to Open Journal Systems, Version 2.2.1.0," Public Knowledge Project, 2008, 6 Feb. 2009 <http://pkp.sfu.ca/files/OJSinanHour.pdf>.



Scholarly Associations and the Economic Viability of Open Access Publishing

John Willinsky, *University of British Columbia*

Abstract

The information landscapes within which scholars work is undergoing a seismic shift. The computer monitor that rises out of the photocopy stacks, piles of journals, clippings and correspondence, now offers a new, rich vein of information that seems destined to eventually overwhelm the traditional trappings of desktops, filing cabinets, and bookshelves. After little more than a decade of Internet publishing, two-thirds of academic journals provide online access, while more than 1,000 peer-reviewed journals are published solely in digital form (Tenopir and King, 2001). Faculty and students are increasingly writing with their browsers open to online research sources. [1]

Introduction

In moving online, scholarly publishing appears to have taken the next in a long line of steps to increase the circulation of this particular form of knowledge. Yet rather than imagine, in this whiggish way, that advances in knowledge naturally unfold with each new communication technology, it is well to realize that the significant choices made by key players during the early and formative period of the technology will shape the future of each publishing medium. These choices for online publishing have now brought scholarship to a critical juncture. In a very short time, online journal publishing has developed two distinct and opposed economic models, one commercial and the other not, even as this publishing is in the unsustainable position of publishing in both print and digital forms. It may not

To aid in project management, the journal editorial process is divided into a set of steps, and tasks are divided into several "editorial roles." An in-depth analysis of the details of setting up an online journal with OJS are described in the "Technical Analysis, End-User" section below. Additionally, as the software is open source, customization and modification at the software developer level is also encouraged, allowing for the addition of functionality or the integration of OJS with another software framework. A detailed guide for development is given in the the "Technical Analysis, Software Developer" section.

Research Context

In the research proposal, "Supporting the Reading of Research in Online Settings," John Willinsky states the research objectives associated with OJS: "This study takes up the question of what happens once readers find what they imagine they have been looking for when the subject of their search has been the research literature. It will investigate how journal websites can be designed to better support the reading of research in online settings for a wider range of readers than has traditionally been the case with research."[7] Primarily, OJS attempts to improve online reading by introducing a set of reading tools.

The document, "A Study of Professional Reading Tools for Computing Humanists" (http://etcl-dev.uvic.ca/public/pkp_report/), reports on a study performed to examine whether this set of reading tools can assist academics in humanities computing in understanding and using online literature in their field. Thirteen academics were presented with the set of reading tools and interviewed regarding their experience. Although mostly positive, the feedback identified several issues with the tools.

[7] J. Willingsky, "Supporting the Reading of Research in Online Settings" <http://pkp.sfu.ca/files/sshrc04.pdf>.

Continued research in the field could take several forms, ranging from the introduction of new tools to the integration of existing tools into contexts other than the online journal. Further details on the potential areas of expansion, specifically with respect to the Transliteracies Project and the Bluesky Group, are described in the section "Evaluation of Opportunities/Limitations for the Transliteracies Topic (and the Bluesky Group)" below.

Technical Analysis, End-User

The goal of this section is to assess the level of technical ability required to use OJS, the documentation organization, and the availability of technical support. Since OJS is an extremely sophisticated tool with numerous configuration options, a full account of all the features available in OJS is not within the scope of this report. Instead, it will focus on reviewing how to get OJS up and running and will walk through the minimum steps of creating a journal with journal managers, editors, reviewers, and authors.

Installing the OJS

(Note: This installation review was conducted using Mac OS X Leopard.)

The OJS installation file can be downloaded from the Public Knowledge Project website at http://pkp.sfu.ca/ojs_download. The system requirements for a web server running OJS are stated as follows:

- PHP 4.2.x or later with MySQL or PostgreSQL support
- A database server: MySQL 3.23 or later OR PostgreSQL 7.1 or later
- Unix-like OS recommended (such as Linux, FreeBSD, Solaris, Mac OS X, etc.)

At the time of this review, the OJS 2.2.2 was the latest version.

After downloading, the user is instructed to decompress the downloaded file and consult the README document. The README file presents a step by step list of instructions for installation, which are described below.

1. Extract the OJS archive to the desired location in your web documents directory.

After extracting the archive, the user needs to put the archive in the server's folder that is accessible from the internet. For OJS deployment, the user will have to get a system administrator to give them access to a web-accessible archive folder. Otherwise, the user will have to run a server which can handle significant traffic. However, for development purposes, the archive can be extracted into Mac OS X Leopard's Apache web server directory which will run the OJS locally. Note that in order to run the Apache web server, which is installed by default on Mac OS X Leopard, web sharing must be turned on in the system preferences panel in Mac OSX. Then, the archive needs to be copied to the */~[username]/Sites* folder, a location in Mac OSX specifically designated for web serving.

2. Make the following files and directories (and their contents) writable (i.e., by changing the owner or permissions with chown or chmod):

** config.inc.php (optional -- if not writable you will be prompted to manually overwrite this file during installation)*

** public*

** cache*

** cache/t_cache*

** cache/t_config*

** cache/t_compile*

** cache/_db*

chown and *chmod* are Unix commands which change or modify the owner of a file or folder and can only be executed by the UNIX OS superuser. The following commands were typed into the Mac OS terminal in the appropriate folder:

```
sudo chmod a+rw config.inc.php

sudo chmod a+rw public

sudo chmod -R a+rw cache
```

Note that the last command uses the -R recursion tag and changed the permissions all the subfolders of the cache folder.

3. Create a directory to store uploaded files (submission files, etc.) and make this directory writeable. It is recommended that this directory be placed in a non-web-accessible location (or otherwise protected from direct access, such as via .htaccess rules).

For this step, a folder can simply be placed in the user's Documents directory, not in the */~[username]/Sites* folder.

4. Open a web browser to <http://yourdomain.com/path/to/ojs/> and follow the on-screen installation instructions.

If these instructions are seen, then the archive and server are working correctly. The instructions themselves contain information already seen in the README document. In addition to the instructions, there is a set of basic configuration options that need to be set by the user. The most technically challenging configuration option is the database settings, and would probably require a system administrator if deploying on a university or deployment server.

Installation Problem and Technical Support

After setting all the desired configurations, the user pushes a button to perform the installation automatically. This is the first point at which the installation process hit a bug. The following error occurred:

> "*A database error has occurred: Can't connect to local MySQL server through socket '/var/mysql/mysql.sock' (20)*".

The Public Knowledge Project support pages are easily navigable and follow the standard format of a searchable bulletin board system. The solution to this problem is easily found. Using Mac OSX Leopard operating system, the user has to edit the */private/etc/php.ini* file and change a single line of code so that PHP can find the "default" socket file.

After the bug fix, the installation is successful and immediately takes the user to the OJS login page where they can create and manage a journal.

Site Administration and Management Options
Once logged in, the user is by default the site manager and is presented with a set of site management options and site administration functions.

Under the site settings tab, the site can be renamed. It is by default set to "Open Journal Systems". There is an option to upload a custom title image instead of specifying the title text. The user is also given the option of writing an introduction to the site, supplying a custom style sheet, etc.

Finally, there are numerous administrative options that provide several powerful tools for accessing data about the site (statistics, counts, user information, data formats for things like dates and time, version numbers, email settings, search settings, etc). The user can also see information about the server running OJS and can perform functions such as clearing the site's data caches, expiring user sessions (which requires them to log back in), and merging duplicate user accounts.

Creating a New Journal
To create a new journal, the user selects "create journal" and supplies the journal name and description. A path URL for the journal can be defined allowing for a direct link to the created journal. The user is also given the option of making the journal open to the public (for free) or to limit its access. The user can create as many journals as desired and can put them in any order. Once created, the journal can be managed under the "journal management" tab as described below.

Managing a Journal
There are numerous options available for managing journals. Overall, the managing options entail filling out web forms and templates and uploading files. One can also "enroll" additional users who are the editors, section editors, copyeditors, layout editors, proofreaders, and reviewers. The managing options also give control over default email forms, managing reading tools (for journal readers/subscribers) and allows the user to view journal statistics. Described here are a few of the most important functions:

Details: Here the user specifies the name of the journal, the ISSN, principle and technical support contacts, sources of funding, sponsors, publishers, and the physical mailing address of the journal. The user can specify email identifications and signatures. Finally, the user can supply the journal with a description, keyword list and tags for search engine indexing.

Policies: Here the user defines the focus and scope of the journal and the peer review policies (number of reviewers, criteria which reviewers are asked to judge submissions, time taken to conduct reviews, etc). This information appears in the "About" section of the journal. The user also has the option of specify two review models for peer reviewing submissions. OJS recommends the Standard Review Process because it steps reviewers through the reviewing process, takes advantage of automated reminder notification, and gives reviewers the standard recommendations for submissions (i.e., Accept, Accept with revisions, Decline, etc). The user also has a host of other policy options for each journal. One interesting option is the automatic journal archiving system which uses the LOCKSS (Lots of Copies Keep Stuff Safe) system which was developed by the Stanford University Library. With the LOCKSS system, one specifies 6-10 participating libraries that will register and cache their journal automatically.

Submissions: In this section, the user defines guidelines for submissions. Guidelines might include information about bibliographies, formatting, and what type of data formats for supplementary files. There is also a nicely laid out form for creating a numbered preparation checklist. There are also forms for defining copyright notices. OJS recommends using Creative Commons licenses but provides several sample copyright notice wordings for journals that offer different levels of access. Interestingly, there is a form for supplying information about competing interests and a form for allowing authors to tag their work for use in the Open Archives Initiative and a section for registering the journal for metadata harvesting within a global system of research databases.

Management: In the management section of the journal, the user defines the level access for the public. There are two levels of access. "Open Access" provides immediate and free access to all published content. With this level of access, the access policy can be defined. The user can also set their journal to subscription access. This option requires the user to fill out additional information in a separate Subscription Manager.

The management section also provides tools for setting the publication schedule. The journal can be set to use a number of systems for identifying different issues (volume number, numbered issues, year, title). The tool is flexible enough that the user could create a journal that has already existed in an offline mode and continue the issues online.

There is a field to state the journal's copyediting instructions and layout instructions. OJS provides an editable and extensive default copyediting instructions in HTML. For the layout instructions, the user has to provide their own guidelines. However, one has the option of uploading layout templates in any file format (for example, pdf, doc, etc.).

Finally there are options for specifying the journal's policies on proofreaders. As was the case with copyediting instructions, OJS supplies a substantial default proofreading policy.

The Look: The last part of the journal management toolset is the array of look customization options. Here one can change the homepage header, content, homepage image, and define the journal's logo, header, footer, and layouts. The layout manager is given as an intuitive GUI editor where one can build the right and left sidebars and choose a theme. There are about a dozen default themes.

Submitting a Work
In order to submit a new work to the journal as a prospective author, one simply clicks the "Author" option from the User Home section. Once the Author option is selected, one is presented with a list of submissions that the user has made to the journal. An ID number, the date submitted, which section of the journal, the list of authors, and the title is given for each submission in the list.

To make a new submission, one simply clicks the "New Submission" option. OJS takes the user through a five-step process for making new submissions. The first step asks a number of initial questions such as which section of journal for which the submission is intended (articles, letters, etc.). Then the prospective author is given a submission checklist (defined in the journal manager) that one literally has to check before proceeding. The checklist asks the prospective author if the submission has been published already elsewhere, that the submission is in the right format (Word, RTF, etc.), and that it is in the right format and adheres to the right bibliographic requirements. Then the prospective author is presented with the journal's privacy statement (again, defined in the journal manager) and is given the option of making an special comments for the editor.

The next section is a page where the author enters metadata about the submissions such as author information (for any number of authors), the title and abstract of the submission and the supporting agencies. The third and fourth sections provides a file browser to upload a submission file and supporting files. Supplementary files are not restricted in terms of file types and format and are intended to supply a prospective author with the option to submit data sets, research instruments, code, figures, tables, etc.

The final step in the submission process is the confirmation step where the author is given one last chance to review their submission. Once submitted, the principle contact is emailed to acknowledge that the submission has been received and is now in the editorial review process.

Reviewing, Editing, and Accepting Submissions
Once a work has been submitted by an author, it is entered into the OJS system as a submission which has an "unassigned" editor and reviewer. One can see the list of submissions going to the "Editor" page. The editor page allows the user to search submissions easily according to title and the dates submitted. One can also sort submissions according to whether or not they have been assigned and whether or not it is "in review" or "in editing".

The journal manager assigns an enrolled editor from the Editor section of the manager user home section. The editor will be sent an email taking them through the steps necessary to complete the editorial process.

As an editor, one can make one's decision about the submission by going to the Review section of the Submissions section of OJS. Once the decision is made, the author is notified by email. The editor also can make comments to be read by other reviewers and the author.

Once accepted, the submission can be moved to the proofreading stage and the layout editing stage. This is done in a similar way to the editor section though the complicated details of who is responsible for which tasks proved to be a little confusing. There is a final, and somewhat complicated, process for copyediting, where the editor and the author exchange drafts until they both agree on a final, mutually approved version.

Creating and Publishing Issues

Once submissions have been accepted, an issue for the journal can be created. Here one specifies the issue identification parameters and uploads a cover image, caption, and style sheet. Once an issue is created, submissions can be assigned to it and will then automatically show up in the table of contents section of the issue. The editor can "Publish Issue" whenever they are satisfied. After publishing, the journal issue automatically moves the issue to the Back Issues tab. At this point, an email (previously configured) is optionally sent to users who have registered to the journal stating that a new issue has been published.

Conclusion

Using OJS, the steps for creating a new online journal are surprisingly straightforward. The various management and editorial roles in a journal could be delegated to people with no technical knowledge of database management. Only moderate technical ability is required to install the journal and the technical support is more than adequate.

One exciting component of OJS is that it provides all the necessary tools to transition older, more established journals to an online version while simultaneously making it easy to create new startup journals from scratch.

Technical Analysis, Software Developer

This document investigates Open Journal Systems from a technical point of view. The goal is to examine, in broad terms, how OJS has been implemented and how this affects users and those looking to extend of the system.

There is a wealth of documentation on OJS available from the Public Knowledge Project website at http://pkp.sfu.ca/ojs_documentation. In particular the OJS Technical Reference document contains very useful information about how OJS functions "under the hood" and how to extend the system. It is available at http://pkp.sfu.ca/ojs/OJSTechnicalReference.pdf.

The OJS application consists of a complex directory tree with over 4,000 files. The application is implemented using PHP for the application code and Smarty Templates for the user interface. There are over 700 PHP files and almost 400 template files in the default OJS installation which demonstrates how complex the application is.

OJS Technologies

OJS relies on several different web technologies that each provide a piece of the implementation. Per the technical reference: "Open Journal Systems 2.x is written in object oriented PHP using the Smarty template system for user interface abstraction (http://www.smarty.net/). Data is stored in a SQL database, with database calls abstracted via the ADODB Database Abstraction library."8

[8] "Open Journal Systems, Version 2.1: Technical Reference, Revision 3," Public Knowledge Project, 2009, 6 Feb. 2009 <http://pkp.sfu.ca/ojs/OJSTechnicalReference.pdf>.

System Requirements
OJS is a dynamic web-based application so it requires a web server and related software to operate. Specifically, the following are required:

- Operating System: Linux, BSD, Solaris, Mac OS X, or Windows operating systems
- Database: MySQL (3.23.23 or later) or PostgreSQL (7.1 or later)
- Web server: Apache (1.3.2x or later) or Apache 2 (2.0.4x or later) or Microsoft IIS 6 (PHP 5.x required)
- PHP support (4.2.x or later)

Apache, MySQL, and PHP have been recently pushed as a complete web-application hosting package ('stack') and are available as a one-stop installation package: LAMP (Linux - http://www.lamphowto.com/), MAMP (Mac OSX - http://www.mamp.info/en/index.php), and WAMP (Windows - http://www.wampserver.com/en/). This approach makes it very easy to get a testing system up and running to evaluate OJS.

MySQL & PostgreSQL are relational databases that use Structured Query Language (SQL) to access and manipulate data. The database is used by OJS to store everything from user account information to comments and links to uploaded journal files.

Apache, or MS IIS, is the web-server software that runs on the server computer and is responsible for receiving page requests and sending back formatted data to the end user ('client'). Apache is free and open-source software maintained by the Apache Foundation while IIS (Internet Information Services) is published by Microsoft and is available with Windows Server operating systems and some end-user Windows Desktop Operating Systems. It is closed-source and proprietary.

PHP is the programming language in which the OJS application is written. PHP is a scripting language which means that source code is 'interpreted' at run-time and not pre-compiled. Thus the entire source code is available for viewing in the installation folder. The application consists of hundreds of .php files that contain the code needed to manipulate the journal data and assemble the OJS pages to be sent to the end user.

PHP: Hypertext Processor
http://php.net

"PHP is a scripting language originally designed for producing dynamic web pages. It has evolved to include a command line interface capability and can be used in standalone graphical applications."

-Wikipedia (http://en.wikipedia.org/wiki/php)

For web pages, PHP is run server-side before the web page is served to the client. PHP code can be scattered throughout a file surrounded by PHP tags ("**<?php ...... ?>**"). Code within these brackets is processed on the server and not returned to the client (end user). PHP code is often used to output/echo page content retrieved from a database as well as markup (HTML). The ability to output data retrieved on the fly makes PHP useful for creating *dynamic* websites.

Server-side processing is useful because it keeps private data secure, only sending exactly what is needed by each client, and also reduces the burden on the client. The downside is if the server is not very powerful, it can get bogged down having to process every page for every user. Many optimizations for PHP exist to increase page-serving performance site-wide as well as application-specific techniques like caching certain semi-static assets.

Note that if the PHP interpreter is malfunctioning or not installed the end user will see all the PHP code which could present a security issue if database passwords are stored in plain-text (a common occurrence).

Use of mature, powerful server-side, web-based language (PHP) for the application is good because:

- Clients (authors, editors, academics) do not need powerful computers to run it and can access it anywhere.
- It is well-supported so it is easy to find help if a local developer wants/needs to modify something or needs help with installation.
- Being server-side, the developer does not need to worry as much about end-user browser compatibilities.
- As it is mature rather than 'cutting edge,' it is less prone to breakages.

Smarty Templating Engine
http://smarty.net

Smarty is a PHP based presentation/templating engine, used for the visual interface of the application. The stated goal of the Smarty project is to be flexible, secure and to make it quick to develop user interface pages. Web developers use special 'smarty syntax', which resembles HTML, to develop the template pages.

Smarty Template files (.tpl) contain only presentation logic which is separate from the backend application logic. There is no PHP and only limited HTML present in the template files. This means a trusted team of experienced PHP programmers can maintain a 'tight, secure codebase' that is accessed and displayed in a multitude of ways by a different team of web designers. Web designers can be less-trusted because they cannot break the PHP code from within the templates. It is a separate issue to ensure less-trusted people do not have physical access to the PHP files which are easily editable.

Separating presentation/view functions from the core application code also makes the system flexible and can simplify development since work is segmented between the PHP and Smarty domains.

Smarty provides many template functions to generate commonly used HTML constructs like drop-down boxes and buttons. This further simplifies the user interface development process and reduces the chances that poorly-written HTML code will cause the user interface to break. Smarty supports caching and other performance enhancements.

The following simple example was adapted from the Smarty website and demonstrates how Smarty templates work and why they are useful. The page in question is accessed as index.php. This file retrieves content from a database or other source and saves it in variables that will be accessed in the template.

index.php

```
include('Smarty.class.php');

// create object
$smarty = new Smarty;

// assign some content. This would typically come from
// a database or other source, but we'll use static
// values for the purpose of this example.
$smarty->assign('name', 'george smith');
$smarty->assign('address', '45th & Harris');

// display it
$smarty->display('index.tpl');
```

The template file then contains the static HTML needed to display the page interspersed with Smarty tags which are surrounded by curly braces. When the Smarty template is processed only Smarty tags are affected.

index.tpl

```
<html>
<head>
<title>User Info</title>
</head>
<body>

User Information:<p>

Name: {$name}<br>
Address: {$address}<br>

</body>
</html>
```

HTML output

```
<html>
<head>
<title>User Info</title>
</head>
<body>

User Information:<p>

Name: george smith<br>
Address: 45th & Harris<br>

</body>
</html>
```

Use of a template engine is good because:

- A template engine separates core application code (should not be modified usually) from presentation code (may want to modify to make custom pages, site-specific looks).
- Comparatively less experience/skill is needed to design the look of the site since the developer is only dealing with the templating language.
- It enhances security since it reduces exposure of PHP functions to outside world and less skilled, possibly less-trusted users.
- Multiple view systems can theoretically use same back-end application and database.

Use of smarty is good because:

- Smarty is a stable, mature project. It used to be sub-project of PHP.
- It is optimized for ease of use, security, performance.
- It is well-supported with good documentation.

Database Abstraction (ADODB)
http://adodb.sourceforge.net/

OJS supports MySQL and PostgreSQL for the back-end database and future database types may be supported in the future. OJS uses ADODB for database abstraction to enable support for multiple database types.

Database Abstraction Layers provide a common interface for the programmer to use when writing code that will interact with a database. The Abstraction Layer is responsible for converting these commands into the proper format for whatever database is being used in practice.

ADODB is an open source database abstraction layer that has been continuously developed since 2000 and supports many different database types (far more than OJS itself). It is modeled on Microsoft's ActiveX Data Objects (ADO) concept (http://en.wikipedia.org/wiki/ActiveX_Data_Objects) but is a distinct product with unique non-ADO features.

Database Abstraction layers make it possible to use a database without formal SQL knowledge since it provides built in functions to for basic operations but can pass arbitrary SQL statements directly to the database which may not be cross-platform.

The code below was adapted from the ADODB website and demonstrates one way to use ADODB to access a database. This code assumes that the connection parameters are already stored in $server, $user, $pwd and $db.

```
include('/path/to/adodb.inc.php');
$DB = NewADOConnection('mysql');
$DB->Connect($server, $user, $pwd, $db);
# PEAR style data retrieval
$rs = $DB->Execute("select * from table where key=123");
while ($array = $rs->FetchRow()) {
 print_r($array);
}
```

Use of a database abstraction layer is good because:

- It makes it possible to support multiple types/brands of database; it puts burden on abstraction layer to handle platform-specifics.
- It provides possibility of not needing to use SQL to access the database.
- It can increase security/stability by stopping inadvertent misuse of database as well as malicious attacks by not always running arbitrary SQL statements.

Use of ADODB is good because:

- It is free, open-source with generous license for use in a wide variety of project types.
- It is mature,stable, well-supported.
- It is inspired by established ADO method of database access.
- It supports a wide variety of databases.

Overall Structural Design
From the technical reference: “The design of Open Journal Systems 2.x is heavily structured for maintainability, flexibility and robustness. For this reason it may seem complex when first

approached. Those familiar with Sun's Enterprise Java Beans technology or the Model-View-Controller (MVC) pattern will note many similarities"[9].

Model View Controller is a design pattern that separates data storage (Model), presentation (View) and application logic (Controller). The idea is the make the system modular and extendable by not burying elements from one domain (such as database queries) in another domain (such as the View files).

In general, OJS uses the database and Data Access Objects (DAO) for the Model, Smarty Templates for the View and PHP code for the Controller. In practice the OJS team has separated the dataflow into many categories that have a specific place in the hierarchy.

The major categories, roughly ordered from "front-end" to "back-end," follow:

1. Smarty templates, which are responsible for assembling HTML pages to display to users;

2. Page classes, which receive requests from users' web browsers, delegate any required processing to various other classes, and call up the appropriate Smarty template to generate a response;

3. Action classes, which are used by the Page classes to perform nontrivial processing of user requests;

4. Model classes, which implement PHP objects representing the system's various entities, such as Users, Articles, and Journals;

5. Data Access Objects (DAOs), which generally provide update, create, and delete functions (amongst others) for their associated Model classes, are responsible for all database interaction;

6. Support classes, which provide core functionalities, miscellaneous common classes and functions, etc.

Consistent file and class naming conventions are heavily stressed so that given a file deep within the directory structure, a developer can quickly determine roughly where it fits in the application hierarchy. The technical reference document has a detailed directory structure reference starting on page 8 and a class hierarchy reference on page 17.

Database Structure
OJS takes advantage of the power of relational databases to implement an efficient and complex back-end database. The database stores journal content like author information and article notations, but also stores operational information such as user account data. By default, the installation page creates the database automatically once it is given the database user credentials. Alternatively the page can print out SQL statements to be run manually.

[9] "Open Journal Systems, Version 2.1: Technical Reference, Revision 3," Public Knowledge Project, 2009, 6 Feb. 2009 <http://pkp.sfu.ca/ojs/OJSTechnicalReference.pdf>.

There are 77 tables in the OJS database after it is installed. Some tables have straightforward names like 'Journals' or 'Users'. Most tables have more complex names and purposes, such as ‘email_templates_data,’ However, naming conventions are followed that make it easy to figure out the purpose of each table. The technical reference has detailed table descriptions starting on page 13. The database schema is defined in an XML format to help maintain cross-platform support between database types. This file is located in “/dbscripts/xml/ojs_schema.xml.”

Request Handling
As part of the overall focus on flexibility, OJS implements a complex but powerful request handling procedure. All page requests start at index.php which invokes appropriate code based upon the request. Most actual code run is located in the classes folder. By default OJS uses a descriptive URL structure that helps inform the user of what is being accessed. Since it all goes through a file called *index.php*, it is useful to have the additional information directly in the URL. (See the example: http://www.mylibrary.com/ojs2/index.php/myjournal/user/profile.)

Everything after index.php is passed to a variable called PATH_INFO to help OJS to identify what to do next. Sometimes this is not supported by the server, in which case, disable_path_info option in config.inc.php can be set. This forces more traditional PHP URLs like:

http://www.mylibrary.com/ojs2/index.php?journal=myjournal&page=user&op=profile

at the expense of clarity to the end user.

“myjournal” identifies which journal to access since a single OJS installation can host multiple journals. “user” is the name of the Page class to load, so it will refer to pages/user/index.php. “profile” identifies the function in the page class to be called, namely to view a user profile in this case.

Because multiple files are called in succession before actual functions are found and executed, it is important to know exactly where to look for the actual code. By convention for a Page class designated 'user' the class is actually called UserHandler and the file is UserHandler.inc.php.

Inside the UserHandler.inc.php file are various functions accessed using the field after the class name in the URL. */user/profile* points to the “profile” function within UserHandler.inc.php.

Internationalization
The OJS team says that they designed the system with no assumptions for what language it would be displayed in so that it can easily be translated into new locales.

Each localization has a primary XML file that contains localized strings that are to be used. The Smarty template system makes it easy to implement localization since it can easily translate items when it is assembling the user interface. Because of the size of the OJS application, each primary XML file has over 3,000 lines of localized text. This ranges from simple messages like, “<message key="user.showAllJournals">Show My Journals</message>” to whole HTML-ized passages like default text for the copy-editing policy.

This separation of the non-application data from the view furthers the MVC pattern. All journal data is stored in the database or in uploaded files but also nearly all text content of any kind is further removed from the presentation via these localization files. What is left in the presentation is almost exclusively template design code specifying how to lay out the dummy elements that are later replaced with localized text. This is done within the Smarty templates using the directive: {translate key=”[keyName]”}. The primary XML file for the US English localization is at *locale/en_US/locale.xm*l. Additional localization strings are in *dbscripts/xml/data/locale* and *registry/locale* to control things like email template text.

Extending & Modifying OJS
OJS is designed to be extended and customized to meet the needs of installation. It is possible to perform quite a bit of customization without needing to modify core PHP application code which makes the system more accessible to institutions that do not have dedicated programming teams.

OJS Plugins
OJS is extendable using plugins available from the OJS website at http://pkp.sfu.ca/ojs_plugins. They provide various new features such as alternative user authentication methods (connect to an existing authentication system like a local network at the university, etc), as well as special data import and export formats.

The core PHP codebase is very complex, in part to make plugins possible. Throughout the code there are 'hooks' that enable alternative code to be used instead of the original OJS code. Many straightforward actions are implemented in multiple steps so that individual steps can be replaced with plugin code. Each plugin must belong to a single category. This is how OJS knows what capabilities the plugin has and which hooks can apply.

The initial categories hard-coded into OJS are (using internal names): *auth, blocks, citationFormats, gateways, generic, implicitAuth, importexport, paymethod, reports, themes*.

Each category has an abstract base class that defines the basic methods that a compatible plugin must implement to work with OJS. Plugins must extend the appropriate abstract baseclass.

As an example, an *importexport* plugin must extend the ImportExportPlugin class (located in classes/plugins/ImportExportPlugin.inc.php).

“Hooks” are pieces of code that, when called, check with the plugin registry to see if there are any plugins activated for the category of the hook. If the registry has an affirmative response the plugin code is accessed and run in lieu of the default OJS code. Hooks are scattered throughout the codebase giving plugins access to nearly all aspects of OJS behavior.

A plugin registry system is responsible for loading plugin files and making note of which plugins are to be accessed for each category. Each category has a folder in /plugins/ and the registry looks inside each category folder for subfolders that contain individual plugins. The plugin specifically registers each hook it has code for so that OJS knows exactly which points will involve custom code and which hooks can be ignored.

The OJS technical reference has a list of built-in hooks that spans 30 pages which further indicates how well OJS is 'wired' for plugin customization. Each hook has a description that says when in the code it is called. A plugin developer can then decide which hooks need to be registered to achieve the desired result. The OJS technical reference also has a detailed information on making new plugins including example code.

OJS Appearance Customization

In addition to plugins that add new functionality, the appearance is also easy to customize through Cascading Style Sheets (CSS) and template modifications. Due to the adherence to a Model-View-Controller paradigm it is possible to drastically alter the external appearance without needing to modify the behind the scenes behavior. CSS and Smarty templates are easier to understand for novice programmers and also benefit from being nondestructive -- an inexperienced web developer can experiment with the appearance without permanently affecting the operation of the journal system.

Templates are in charge of the physical HTML structure and how the journal data retrieved from the database is laid out on the page. By modifying the Smarty templates it is possible for the same information to be displayed in wildly different ways. CSS controls the physical appearance of page elements with control over color, text size as well as some layout attributes. CSS files are contained within the styles/ folder and are descriptively named. There is an extensive gallery of customization examples at http://pkp.sfu.ca/ojs-customization . This is very illustrative of what is possible with OJS but does not contain detailed information on how each modification was made so it is up to the end users and administrators to decide how to customize OJS to meet their aesthetic criteria.

Evaluation of Opportunities/Limitations for the Transliteracies Topic (and the Bluesky Group)

The document, “A Study of Professional Reading Tools for Computing Humanists” (http://etcl-dev.uvic.ca/public/pkp_report/), which describes feedback given by a number of users of the OJS reading tools, is a valuable source for understanding the strengths and limitations of OJS. Perhaps more importantly for the goals of the Bluesky Group, the document suggests a number of methods for improving and expanding the system. As the study used a small set of test subjects, the feedback was judged qualitatively, with in-depth interviews, rather than quantitatively as could have been done with a larger test group given a numerical survey.

The feedback provided in this study was primarily positive. The authors of the report specifically note that those interviewed felt that the system would save researchers time by including multiple resources on a single site and that the tools encouraged learning about topics related to an article, contributing to the reader’s understanding of the article.[10] The suggestions for improvement largely involve the streamlining of existing tools: improving search results, combining multiple tools for easier navigation, etc.[11] However, the report also suggests the possibility of adding social computing tools beyond the existing comment box and forum features, as the information provided by a journal only gains value through the communities that are developed around it.

[10] Siemens, Willinsky, Blake, et al., “A Study of Professional Reading Tools for Computing Humanists,” 2006, 13 Feb. 2009 <http://etcl-dev.uvic.ca/public/pkp_report/conclusion.html>.
[11] Siemens, Willinsky, Blake, et al., “A Study of Professional Reading Tools for Computing Humanists,” 2006, 13 Feb. 2009 <http://etcl-dev.uvic.ca/public/pkp_report/conclusion.html>.

The potential for additional social computing tools is, of course, a topic of interest to the Social Computing group of the Transliteracies Project and the Bluesky subgroup in particular. The current tools offered are relatively simplistic, providing a space for commenting that either lies below the text (in the case of the comment box) or on an entirely separate page (in the case of the forum). For example, a tool that integrates the social computing text with the articles, such as an annotation tool like CommentPress (http://www.futureofthebook.org/commentpress/) can be implemented and tested. This possibility is already being investigated, at least conceptually, by those associated with OJS and has described in a white paper written by Cara Leitch at University of Victoria entitled, "Social Networking Tools for Professional Readers in the Humanities." However, Leitch points out that there are currently no social networking tools that integrate the three reading strategies of evaluation, communication, and management.[12]

Further work in this direction is being developed by the Electronic Textual Cultures Lab at The University of Victoria led by Ray Siemens (http://etcl.uvic.ca/). They are developing the Professional Reading Environment (PReE), which uses an implementation of the Renaissance English Knowledgebase (REKn). PReE will implement a professional reading and research tool that uses a CommentPress style commenting system, sharable annotations, and socially-networked research groups.

However, each of these projects lacks a clear concept for the various roles, behaviors, or complex relationships (professional or informal) that a person might have in a social network. Bluesky might be able to contribute to these systems by focusing on developing a clear ontology of the "person" in a research context and developing a framework around this concept. For example, a person in a journal developed using OJS might play a single role or several roles: a contributor (e.g., author), an authority (e.g., editor), and a reader (e.g., subscriber). From these roles, not only should different actions and behaviors be available, but also different network associations and privileges. From this, it becomes possible to model more complex relationships, mirroring those found in real-life academic research.

A more complete ontology of the person (and their relations in a network) might also suggest novel tools which visualize the network, filter content, or improve content searching. Though meaningful data visualizations depend on the actual data, it's easy to imagine visualizations based on comparing user role/type/behavior to any number of other parameters, such as publications, comment volume, comment activity, link density, etc.

As an example of content filtering, one could choose to view only the comments from people who are authorities on a particular topic or one could see only comments made by graduate students.

A user could restrict searches to return content contributed by a particular user-role or by those with a particular relationship to the user. If a sizable community is using a particular journal (or similar professional research tool), search results might also be weighted and fine-tuned according heuristics derived from analyzing user-types and behavioral data.

Regardless of the details of a particular ontological scheme for a person, important questions

[12] C. Leitch, "Social Networking Tools for Professional Readers in the Humanities."

which need to be explored further are:

1. What is a professional reader in a research community?
2. How do we define authority?
3. How do we balance professionalism, expertise, and openness?

The possibility to extend OJS is facilitated by the powerful plugin architecture which makes it very feasible for OJS to be modified to represent new paradigms without materially affecting the core operational codebase. New conceptual models can be realized and released as OJS-compatible plugins which would allow interested users around the world to evaluate them without compromising their existing OJS installation.

Resources for Further Study

CommentPress. <http://www.futureofthebook.org/commentpress/>.

“Open Journal Systems.” <http://pkp.sfu.ca/?q=ojs>.

“OJS in an Hour: An Introduction to Open Journal Systems, Version 2.2.1.0.” <http://pkp.sfu.ca/files/OJSinanHour.pdf>.

“Open Journal Systems, Version 2.1: Technical Reference, Revision 3.” <http://pkp.sfu.ca/ojs/OJSTechnicalReference.pdf>.

REKn/PReE. <http://etcl-dev.uvic.ca/public/REKn/Site/>.

Siemens, Willinsky, Blake, et al. “A Study of Professional Reading Tools for Computing Humanists.” <http://etcl-dev.uvic.ca/public/pkp_report/conclusion.html>.

Willingsky, John. “Supporting the Reading of Research in Online Settings.” <http://pkp.sfu.ca/files/sshrc04.pdf>.