A Digital Library Component Assembly Environment

LINDA EYAMBE AND HUSSEIN SULEMAN
University of Cape Town

With the advent of the Internet came the promise of global information access. In keeping with this promise, Digital Libraries (DLs) began to emerge across the world as a method of providing structured information to their users. These DLs are often created using proprietary monolithic software that is often difficult to customise and extend. The Open Digital Library (ODL) project was created to demonstrate that DLs can be built as a network of components instead of as monolithic systems. Although the ODL approach has largely been embraced by the DL community, it is not without a few shortcomings. This paper introduces a graphical user interface and its associated framework for creating DLs from distributed components, consequently addressing a number of the limitations of ODL, as well as presenting a novel and generic approach for creating component-based systems.

Categories and Subject Descriptors: D.2.6 [**Software**]: Programming Environments – *Graphical environments*; D.2.13 [**Software**]: Reusable Software – *Reusable libraries*; H.3.7 [**Information Storage and Retrieval**]: Digital Libraries – *Systems issues*
General Terms: Design, Experimentation, Standardization
Additional Key Words and Phrases: Digital Libraries, Open Digital Libraries, Components, Graphical User Interface

1. INTRODUCTION

Recent developments in information and communication technologies, especially the Internet and the Web, have brought about significant changes in the ways we generate, distribute, access and use information. One of the most important contributions of Web technology has been the creation of digital libraries, which allow users to access high quality digital information resources from virtually anywhere in the world.

Traditionally, Digital Libraries (DLs) were built as monolithic and proprietary software systems that were complex and difficult to manage. In recent years, attempts have been made at decreasing the complexity of software systems by moving towards a modular approach. Several DL component models are now beginning to emerge, such as the Open Digital Library (ODL) [Suleman 2001] and OpenDLib [Castelli and Pagano 2002] projects.

Despite these models to facilitate the creation and extensibility of digital libraries and enhance inter-component interaction, building digital libraries still involves a fair amount of complexity. Current DL component models often rely on the manual configuration of each individual component, in order to produce a resulting DL.

This research investigates using a simple visual interface and associated framework in order to create a digital library from, but not limited to, distributed ODL components, thereby enabling inexperienced users to create Digital Libraries simply and quickly.

2. BACKGROUND AND MOTIVATION

2.1 The OAI and the OAI-PMH

Although digital libraries are ever growing in popularity, Arms [2000] explains one of the major problems they are facing is the issue of interoperability—connecting systems together in distributed digital libraries.

The Open Archives Initiative (OAI) was formed to address this need in a standardised manner by launching its Protocol for Metadata Harvesting (OAI-PMH) [Lagoze and Van de Sompel 2001]. The OAI-PMH promised to provide a simple mechanism for digital libraries to interoperate effectively. While the OAI-PMH can still be considered a nascent protocol, a large number of digital library projects are already working towards adding OAI capabilities to their systems [Breeding 2002].

Although the OAI has brought about interoperability solutions to support interaction among already existing digital libraries, simpler solutions are still needed to create digital libraries in the first place. To circumvent the pitfalls of

Author Addresses:
L. Eyambe, Rm 317, Department of Computer Science, 18 University Avenue, University of Cape Town, Rondebosch 7700, South Africa; leyambe@cs.uct.ac.za.
H. Suleman, Rm 317, Department of Computer Science, 18 University Avenue, University of Cape Town, Rondebosch 7700, South Africa; hussein@cs.uct.ac.za.

Name of component	Description of Functionality	Interface Protocol
DBUnion	To merge together the metadata from multiple sources	ODL-Union
IRDB	Search engine	ODL-Search
DBBrowse	To browse through metadata based on values of particular fields within the metadata	ODL-Browse
WhatsNew	To track and obtain, upon request, a sample of recent entries	ODL-Recent
Box	Dumb archive supporting submit and retrieve operations	ODL-Submit
Thread	Threaded annotation engine for discussion forums, guestbooks and resource annotation	ODL-Annotate
Suggest	Recommender system to make suggestions based on collaborative filtering	ODL-Recommend
DBRate	To manage the submission and access to ratings for individual resources	ODL-Rate
DBReview	Peer review workflow manager geared towards the review of journal and conference publications	ODL-Review

Table. 1. *ODL reference components, descriptions and protocols*

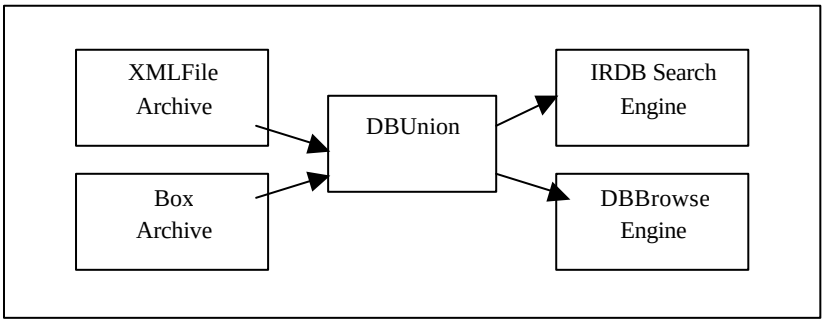

Figure. *1. A sample DLfrom ODL components*

monolithic systems, some researchers have used some form of component model—which is widely accepted as good software engineering practice—in constructing their DLs. However, component frameworks failed to be embraced by the DL community because, in several projects that adopted some form of component model, the components communicate using non-standard protocols, making modification a complex process.

2.2 DL Components and Systems

There is a large body of research dedicated to components developed for experimental purposes and an equally large amount for those developed for production DL systems, such as Arc [Arc 2004], which prides itself as being one of the first searching services based on the OAI protocol. There are also several pre-packaged DL systems on the market such as DSpace [Smith et al. 2003], Archon [Maly et al. 2002] and Greenstone [Witten 2003].

The key to a successful digital library is extensibility, but the above-named DL systems are all limited in that regard. With the framework presented in this paper, a new service can be easily added to the pool of components available to the graphical user interface, thus presenting the DL designer with more choice.

The Open Digital Library (ODL) project was initiated to create a lightweight framework based on the hypothesis that DLs can be built as a network of extended Open Archives instead of monolithic systems [Suleman 2002]. Because of the success of the OAI, the ODL approach is built upon the OAI-PMH. Suleman [2002] describes the ODL project as an attempt to infuse interoperability into all aspects of the digital library, and make the provision of services as simple as the provision of data, effectively facilitating the development, management and interoperability of DLs.

Table 1 tabulates some ODL components, their interface protocols as well as a brief description of the function of each component [Suleman et al. 2003].

Figure 1 illustrates how a digital library can be created by interconnecting a number of ODL components. The sample DL consists of a DBUnion component, which merges metadata originating from a collection of XML files (XMLFile component) and a simple database (Box component) into a single archive. The search (IRDB) and browse (DBBrowse) engines can then access this single archive to provide their respective services. Each of these components supports one of the ODL interface protocols outlined in Table 1.

During user testing of the ODL components, Suleman [2002] outlined a number of issues that arose. Mentioned

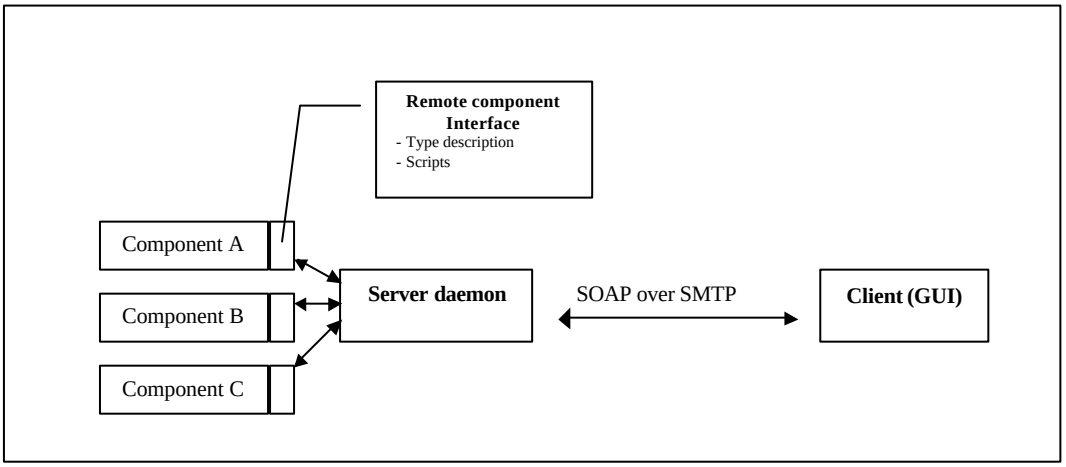

Figure. *2. The architecture of the system*

below are those that this research attempts to address with the introduction of a Graphical User Interface:

- Confusion over baseURLs: A baseURL is the link that makes a component accessible via the WWW. Some participants were confused regarding which baseURL to use since all the URLs were similar.
- Typographical errors: Entering URLs by hand resulted in many typographical errors.
- Connectivity errors: The user interface was sometimes connected to the wrong component, as the user interfaces were not in themselves components.

ODL's simplicity and uniform approach at applying its protocols to a wide range of components, improving interoperability at component level, makes it a suitable platform for experimentation. Furthermore, the ODL components are extensions of the widely accepted OAI-PMH standard for DL interoperability while other component models interact using non-standard protocols or resort to assembling components in a disparate or ad-hoc manner, resulting in problems of adaptability and interoperability.

In order for the ODL components to communicate, they are configured by means of command-line configuration scripts, which suffices for DL researchers with simple scenarios, but is inadequate for general and widespread use. With the introduction of a visual interface, a whole new group of potential digital library designers, not part of the digital library research field could adopt the technology and approach.

The extent to which visual development environments have evolved makes it rather surprising that, as of yet, very little research has gone into creating one of such systems for digital library components. This can probably be attributed to the fact that component technology in the digital library discipline is still fairly new and few standard components exist.

This research therefore arose in response to a need for simplified digital library creation in order to encourage the development of digital libraries by inexperienced users, as well as to address some of the problems encountered during the command-line configuration of ODL components.

3. APPROACH

3.1 Aims

The aims of this research were:

- To enable ordinary users (non-technologists or experts) to create a digital library from a suite of components. It will simplify the way digital libraries are created, effectively placing digital libraries within the reach of non-technologists or librarians.
- To aid in eradicating, or at least greatly reducing, several problems (e.g., typographical errors) that occur during manual configuration of DL systems. This painstaking manual process seems superfluous especially with the proliferation of visual development environments within and beyond the Web Services development community.
- To investigate the applicability of the visual composition environment to real-life, large-scale digital libraries.
- Because the system is designed to be generic, it can be applied to other development communities, for example Web Service developers, with little modification.

3.2 Architecture

A client-server architecture was used for several reasons. Firstly, the components were created for a Linux environment and given that MS Windows is still the most widely used operating system, several people will only have remote access

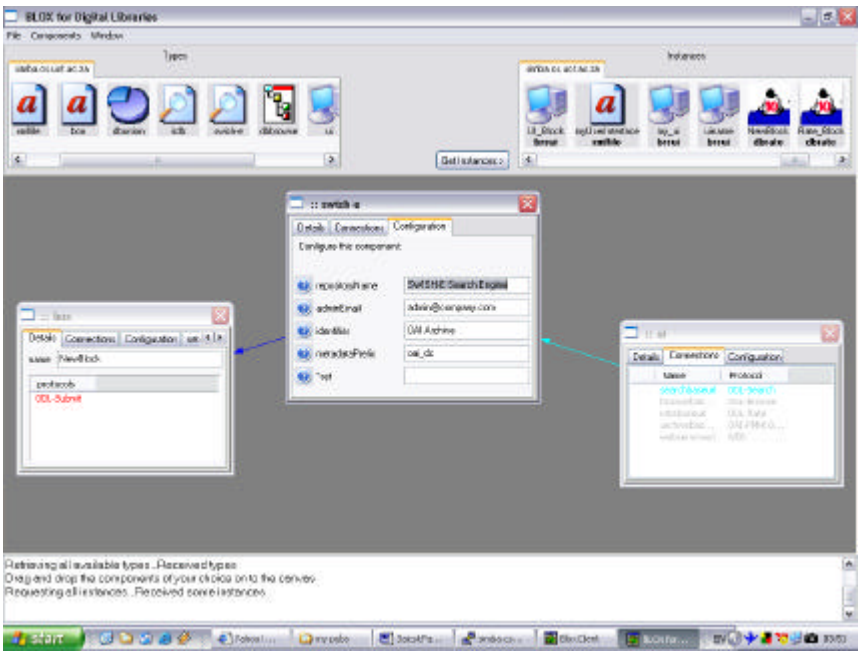

Figure. 3. *Snapshot of the BLOX Client*

to the components. In order for a graphical user interface to be widely used, it would need to work in a Windows based environment and access the components remotely. Because the GUI (named BLOX) was developed in wxPython, a Python port of wxWindows, it can also be run in a Linux environment.

Previously, to create a digital library using ODL components, one had to download and install all of the desired components then configure each one individually. This process may not be so straightforward if one's computer does not possess all the required software to run the components, such as a Web server, mysql and required Perl modules. But with a client-server architecture, one can connect to one or more servers hosting components without having to worry about installing them.

Furthermore, with a client-server approach, one can make use of component instances created by others. It should therefore be possible to use an existing DL comprising, for example, an archive and search engine instantiated by someone else and augment that DL with a different user interface.

In this client-server model, three distinct elements can be identified—a graphical user interface (the client) which will be the only part of the system directly visible to the user, a server daemon which handles all communication between the components and the GUI, and an interface to those components that both parties (the components and component server) understand. This is illustrated in Figure 2.

3.3 The Graphical User Interface

A Graphical User Interface was created to allow users to specify the components they would like to include in the digital library and the details of their configuration. This enables the user to interact with distributed components in an effortless manner by making use of point and click, and drag and drop capabilities. Components are assembled on a canvas and then sent off to be instantiated on the server. Each component is represented as a window. Arrows are dragged between windows to represent the connections between components, and all the configuration information is represented as a form in the component window. This is illustrated in Figure 3.

3.3.1 The Server Daemon

A Server was created to handle the interactions between the client and the components, and has the capability of handling requests from several clients simultaneously. The server is configured with a list of the root directories of the components and interacts with component scripts to obtain component information for the client, to create new instances of components, and to modify or delete existing instances. The server contains two separate parts which interface to provide the necessary functionality to the components it manages and the GUI – the server side communications component, and component-specific handlers. Handlers deal with the administration of the types of components they were designed to manage. That is, if the server has to deal with say, DL components and python components (python templates providing a specific functionality), a handler would have to be written for both component suite types and registered with the server. Handlers therefore enable diverse component-based systems to be constructed using the same GUI.

Script	Function
autoconfig	Receives an Instance Description (XML document) and uses that data to configure a component
getInstance	Takes an instance name and returns its Instance Description
GetType	Returns the Type Description of that component
listInstances	Returns all Instance Descriptions of a particular component (i.e., all instances of a component)
removeInstance	Deletes a specified instance

Table. 2. *The interface of a component*

The server was designed to transport XML documents from one or more clients to the components using SOAP over SMTP. This is achieved by listening for messages from the server-side communications component, interpreting these messages and forwarding them to the relevant handler. Although HTTP is probably the most common protocol for SOAP messaging, the request/response nature of HTTP made it an unsuitable transport mechanism due to its 300 timeout restriction, which is unsuitable for the configuration of certain DL components. The server and all its constituent components were implemented in Python.

3.3.2 Component Interface

Sometimes components have to be utilised in an environment other than the one they were initially created for. Such is the case with the ODL components. When the components are being configured manually, a Perl script is run and the user is prompted for configuration information along the way. This is clearly inappropriate for a visual environment, and so a different system had to be devised in order to configure these components. The matter is further complicated due to the fact that the components are distributed and the calling method may not possess the rights required to modify a component's interface remotely. To resolve this problem a new component interface was specified.

One element of this interface is the component's type description. A type description describes the information required to successfully configure a component. This is implemented as an XML Schema. Each instantiated component has an instance description. Since the type description is represented as an XML Schema, the instance description is an instance of that schema and is therefore an XML document.

In addition to the type description, the component interface supports five service requests, implemented as scripts, outlined in Table 2. The scripts were all implemented in Perl and stored in the component's root directory. All the components and their root directories are then registered with the server.

Non-ODL components can also be interfaced in this way in order to communicate with the server. This was demonstrated by interfacing the phpBB [2004] bulletin board, which was originally designed to receive its configuration information from an online form, and the SWISH-E [Rabinowitz 2003] search engine, which is an executable that was designed to receive its configuration information as command-line arguments. They both now interact seamlessly with the server.

3.4 The Component Connection Language (CCL)

A language was developed to represent the conceptual model of a digital library. This language, referred to as the Component Connection Language (CCL), contains the instance descriptions of the connected components, the server and port on which the components were installed as well as information required to reconstruct their graphical representation in the GUI. The CCL is an XML document, so when a DL is created with the GUI, the instance descriptions are stored in the CCL, sent to the server, the individual instance descriptions are retrieved by the server from the CCL and each sent off to the relevant component for configuration. The CCL can be saved and reloaded at a later date to continue creating the DL or for modification.

3.5 Additional Components Adapted/Developed

Although this research focused mainly on ODL components, to demonstrate its premise of extensibility, four other components were included in the research.

5.3.1 PhpBB

PhpBB [2004] is a popular PHP-driven bulletin board. The purpose of including phpBB in the component suite used in this experiment was not because it is imperative for a digital library to contain a bulletin board, but rather to demonstrate the possibility of augmenting the component suite with virtually any component that makes sense to include in a digital library.

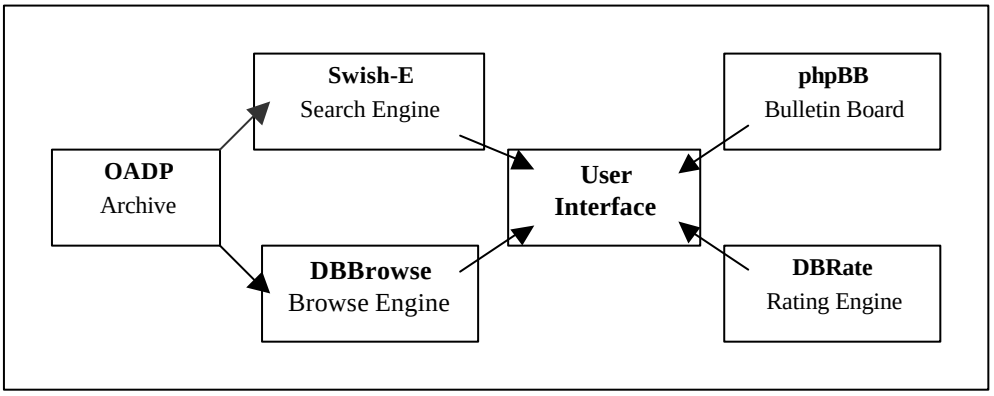

Figure. 4. *The DL users were required to build during the tests*

5.3.2 SWISH-E

The SWISH-E (Simple Web Indexing System for Humans – Enhanced) search engine, a descendant of SWISH, offers a unique combination of features that make it attractive for this DL component assembly research. Some of SWISH-E's offerings include: a fast and robust toolkit with which to build and query indices; good documentation; active development and bug fixes; and a Perl interface [Rabinowitz 2003].

Constructing a digital library by integrating non-OAI compliant components, such as the above two, with ODL components using a graphical user interface will enable a variety of complex and highly functional DLs to be created quickly and simply.

5.3.3 The Digital Library User Interface

The original ODL component suite contains a simple user interface component designed to work with the ODL-Search protocol. This component provides an online textbox in which search keywords can be entered. This suffices for your simple DLs, as depicted in Figure 1. However, users may wish to incorporate other services such as browsing and rating into a single user interface. Therefore, the basic ODL user-interface was modified to accommodate other services in a manner that requires no knowledge of the OAI or HTML standards. This is consistent with the aim of providing a simple way of creating DLs from distributed components.

5.3.4 External Archives

When configuring certain ODL components, such as the search or browse engines, using the command line, one would normally have to supply the baseURL of the archive is it trying to connect to. However, because the GUI was designed to prevent the user from having to type baseURLs, a new method was required to provide the baseURLs of external archives, i.e., archives located anywhere on the WWW. A new component called OADP (Open Archives Data Provider) was created, which contains the list of OAI-compliant archives available on the OAI website. The user simply selects the desired archive and links it to some other component such as a search engine. The OADP can be seen as a black box with exactly the same external interfaces as the other ODL components.

4. INITIAL TESTS

For the initial tests, 5 users were selected at random with 3 having no prior DL experiences, but all required to have some basic computer literacy skills.

4.1 Testing Format

Users were required to fill out a questionnaire in order to ascertain their background and knowledge of DLs, Web Services, ODL components, XML and other related technologies. The users then read a short handout containing explanations of DLs and the components used in the experiment as well as a brief description of the functioning of the DL visual composition system. The users were then given the task of building specific digital libraries. To have a control experiment on which to base any comparison, the users were instructed to build a simple digital library manually before attempting to use BLOX to build the more complex DL depicted in Figure 4. The task of creating a DL manually was somewhat simplified as the users were not required to install the components. Finally the users were required to fill out another questionnaire to obtain feedback on their experience. The entire test took approximately 1 hour including reading the handout and filling out the questionnaires.

4.2 Results

All the testers successfully completed all the tasks. It was apparent that the ease with which the tasks were completed by the participants was dependent on their past experiences. 1 of the 2 users who indicated in the preliminary questionnaire that they had limited or no experience in Linux environments struggled a bit with the manual configuration. The overall result of the test was very encouraging, as all the users agreed or strongly agreed that using the BLOX system was faster and easier than manual configuration. The major disagreement came from the choice of icons used to represent the different components. 2 of the users expressed that they did not always know what the system was doing in response to their input, in spite of the presence of the informational window as shown in Figure 3. 1 of the users expressed a bit of confusion as to what exactly they were doing, and they said it only became apparent after viewing the resulting DL. It may be worth noting that that particular user also indicated that they had no previous experience with visual development environments.

Despite having configured two digital libraries, one using the command-line and the other with BLOX, 3 of the users still felt that they did not fully understand the basic concepts of ODL or OAI-PMH. This can probably be attributed to the fact that the technical aspects of the ODL framework were shielded from the users, and the volume of information required to be assimilated in an hour.

All of the users agreed that they would consider using a system such as BLOX to create a digital library, should they ever need one. The generally positive feedback generated by the initial test suggests that this new remote configuration framework could be a viable approach for creating a DL simply and quickly. However, to confirm the results of this initial test, more comprehensive tests involving at least 30 users will have to be conducted.

5. CONCLUSION

This paper introduced the reader to the ODL components and presented a new framework for configuring remote distributed components using a Graphical User Interface. The results of the preliminary tests suggest the viability of a system such as BLOX for creating digital libraries. The test users, some with no prior DL knowledge, were able to get fully functional DL systems up and running with little effort. However, more extensive tests have to be carried out to verify if the visual approach can replace existing methods of DL creation.

Though the focus of this research was building a digital library from components using a GUI, the approach discussed in this paper can be applied to other types of systems as well, and thus can be regarded as a generic method for creating component-based data-flow-driven systems.

6. FUTURE WORK

The GUI needs to undergo a slight modification to resolve issues raised during the preliminary tests. These issues are predominantly HCI considerations such as choice of icons, aesthetics of the GUI, and giving the users a better sense of control.

A security infrastructure must be put in place in order to prevent malicious or inadvertent tampering of created instances. Furthermore, the host of a component may wish to limit the number of instances stored on their server. Currently, all users have equal access to all the component instances—should only the creator of an instance have modification rights on that instance? More research needs to be conducted in order to ascertain the security implications of using this framework.

Finally, DLs generally interact with users by means of a user interface accessible via the WWW. For this research, a user interface was created which accommodates the components being used during the experiment. However, if a new component was to be added to the component suite, the UI's source code will have to be modified to accommodate that component. What is needed is a more flexible user interface that can be fully configured to accommodate new components and different workflows.

REFERENCES

ARC. 2004. ARC - *A Cross-Archive Search Service.* http://arc.cs.odu.edu.

ARMS, W. 2000. *Digital Libraries*. Cambridge, MA, MIT Press.

BREEDING, M. 2002. Understanding the Protocol for Metadata Harvesting of the Open Archives Initiative, *Computers in Libraries,* Vol. 22, Issue 8, 24-30.

CASTELLI, D., PAGANO, P. 2002. OpenDLib: A Digital Library Service System. In *Proceedings of the 6th European Conference on Research and Advanced Technology for Digital Libraries*, ECDL, Rome, Italy, September 2002, M. AGOSTI, C. THANOS, Eds. Springer, 292-308.

LAGOZE, C., VAN DE SOMPEL H. 2001. *The Open Archives Initiative Protocol for Metadata Harvesting, Open Archives Initiative,* Available: http://www.openarchives.org/OAI/openarchivesprotocol.html.

MALY, K., ZUBAIR, M., NELSON, M., LIU, X., ANAN, H., GAO, J., TANG, J., ZHAO, Y. 2002. Archon - A Digital Library that Federates Physics Collections. In *Proceedings of the 6th International Conference on Dublin Core and Metadata for e-Communities (DC-2002: Metadata for e-Communities: Supporting Diversity and Convergence)*, Florence, Italy, October 2002. Firenze University Press, 27-34.

MOORE, D., EMSLIE, S., SULEMAN, H. 2003. *BLOX: Visual Digital Library Building.* Technical Report No. CS03-20-00, Department of Computer Science, University of Cape Town. Available: http://pubs.cs.uct.ac.za/archive/00000075/.

PHPBB. 2004. phpBB – *Creating Communities.*, http://www.phpbb.com.

RABINOWITZ., J. 2004 SWISH-Enhanced. http://swish-e.org/.
SMITH, M., BASS, M., MCCLELLAN, G., TANSLEY, R., BARTON, M., BRANSCHOFSKY, M., STUVE, D., WALKER, J.H. 2003. DSpace: An Open Source Dynamic Digital Repository, *D-Lib Magazine.* Vol. 9, No. 1. Available: http://www.dlib.org/dlib/january03/smith/01smith.html.
SULEMAN, H. 2002. *Open Digital Libraries*, PhD Dissertation, Virginia Tech. Available: http://www.husseinsspace.com/publications/odl.pdf.
SULEMAN, H., FOX, E., 2001. A Framework for Building Open Digital Libraries, *D-Lib Magazine*. Vol. 7, No. 12. Available: http://www.dlib.org/dlib/december01/suleman/12suleman.html.
SULEMAN, H., FOX, E.A., KELAPURE, R., KROWNE, A., LUO, M. 2003. Building Digital Libraries from Simple Building Blocks, *Online Information Review*, Vol. 27, No. 5, 301-310.
WITTEN, I. 2003. Examples of Practical Digital Libraries Collections: Built Internationally Using Greenstone, *D-Lib Magazine.* Vol. 9, No. 3. Available: http://dlib.org/dlib/march03/witten.