A Component Assembly Approach to Digital Library Systems

Linda Eyambe, Hussein Suleman

University of Cape Town

ABSTRACT

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 usually 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-like systems, as well as presenting a novel and generic approach for creating component-based systems. This system was subject to a user-based evaluation to confirm its utility and provide insights into possible extensions.

KEYWORDS: 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 library systems, 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) [1] and OpenDLib [2] projects.

Despite these models to facilitate the creation and extensibility of digital libraries and enhance inter-component interaction, building digital library systems 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 system from, but not limited to, distributed ODL components, thereby enabling inexperienced users to create DLs simply and quickly.

Email: Linda Eyambe `linda.eyambe@ericsson.com`, Hussein Suleman `hussein@cs.uct.ac.za`

2 BACKGROUND

2.1 The OAI and the OAI-PMH

Although digital libraries are ever growing in popularity, Arms [3] 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) [4]. The OAI-PMH promised to provide a simple mechanism for digital libraries to interoperate effectively at the level of metadata exchange. 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 [5].

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 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 produc-

tion DL systems, such as Arc [6], which prides itself as being one of the first searching services based on the OAI protocol. There also are several pre-packaged DL systems on the market such as DSpace [7], Archon [8] and Greenstone [9]. 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 [1]. Because of the success of the OAI, the ODL approach is built upon the OAI-PMH. Suleman [10] 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 [11].

Figure 1 illustrates how a digital library system 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 protocols in Table 1.

During user testing of the ODL components, Suleman [10] outlined a number of issues that arose. Mentioned 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. Thus, a visual interface may address the problems discovered in earlier ODL experiments by eliminating the need for textual configuration information wherever possible. In addition a visual interface may allow DL designers to simultaneously configure multiple components, thus enabling a degree of automation, especially where one component depends on another. Lastly, by enforcing a consistent strongly-typed model on components, it prevents the connectivity error previously encountered.

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 system 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 digital libraries.
- Because the system is designed to be generic, it can be applied to other development communities, for example Web Service development, with little modification.

3.2 Architecture

A client-server architecture was used for several reasons. Firstly, the components were created for a server (nominally Linux) environment and in order for a

Table 1: ODL reference components, descriptions and protocols

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

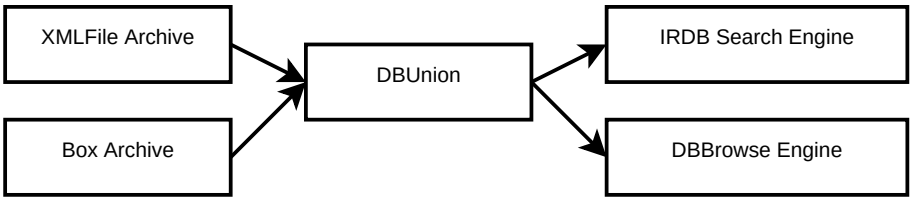

Figure 1: A sample DL from ODL components

graphical user interface to be widely used, it would need to work in a desktop (nominally MS-Windows) environment and access the components remotely. Because the GUI (named Blox [12]) was developed in wxPython, a Python port of wxWindows, it can be run in either a Linux or MS-Windows environment.

Previously, to create a digital library system 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.

Obviously there is a shift in initial effort from the designer of the DL to the administrator of the server where the components are located. This can be justified because it results in specialization of tasks – administrators install software components while DL designers design systems. The level of complexity of components also varies widely. While ODL components typically involve only unzipping, other families of tools may require additional work. In all cases, the designer of the system conceptualises the server as a provider of services, without having to know the details thereof.

In this client-server model, three distinct elements can be identified: a graphical user interface (the client) that will be the only part of the system directly visible to the user; a server daemon that 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.

3.2.1 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. The GUI initially communicates with the server to determine what types and instances of components are available (analogous to types and instances in object oriented programming). Once the GUI has obtained this information, the user is able to use existing instances or create new instances of existing types by selecting options from a toolbar above the canvas. This toolbar uses labels and icons to represent component types – for example, all search engine components have the same icon but are named differently (irdb and Swish-E in Figure 3, an example of a GUI session).

The user then interacts with these distributed components by making use of point and click and drag and drop capabilities. This method of interaction was chosen to mimic an artist's canvas where objects are assembled to create a final product. Each component (instance) is represented as a window. Arrows are dragged between windows to represent the data flow connections between components, and all the configuration information is represented as a form within each component window. Components are assembled on a canvas in this manner and ultimately instantiated on the server by clicking on a Publish button.

3.2.2 The Server Daemon

A Server was created to handle the interactions between the client and the components. The server is configured with a list of the root directories of its local components and interacts with a well-defined API supported by each component 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 that 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. Thus it is possible to support a different suite of component, not necessarily related to DLs, using the same GUI and server infrastructure.

The server was designed to accept XML documents from one or more GUI clients, destined for 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 timeout restriction, which is unsuitable for the configuration of certain DL components. The server was implemented in Python.

3.2.3 Component interface (API)

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. 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 developed, essentially wrapping components so that they support a standard remote management API.

One element of this interface is the component's type description. A type description describes the information required to successfully configure an instance of a component. This is implemented as an

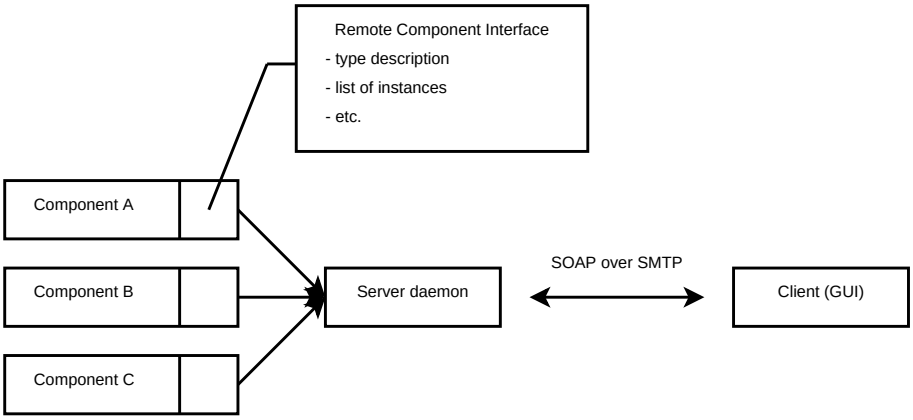

Figure 2: The architecture of the system

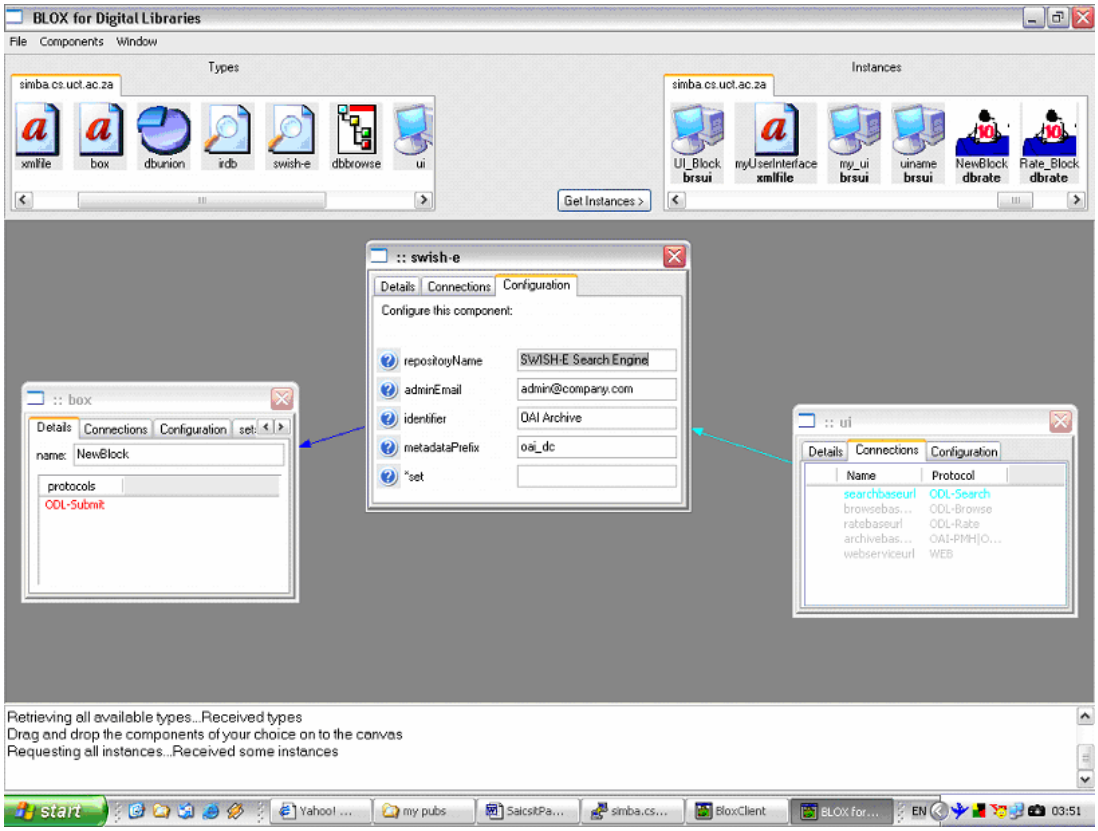

Figure 3: Snapshot of the BLOX client

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 conforming to that schema and is therefore an XML document.

In addition to the type description, the component interface supports four service requests, outlined in Table 2, to: list all instances; get configuration information for a single instance; set configuration information for a single instance; and remove a single instance from the server.

The component management API was implemented in Perl.

Non-ODL components can also be interfaced in this way in order to communicate with the server. This was demonstrated by interfacing the PHPBB [13] bulletin board, which was originally designed to receive its configuration information from an online form, and the SWISH-E [14] 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.

3.3 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.

3.4 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.

3.4.1 PhpBB

PHPBB [13] 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.

3.4.2 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 [14].

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.

3.4.3 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 trivially 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.

3.4.4 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 (the OAI term for Web service endpoint) 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.

While developing/wrapping each of these components required some programming in the context of this study, typical designers of DL systems expect only to use what is available. Organisations such as the OAI make sure that a variety of components are available at any point in time for DL designers to incorporate into their systems.

4 USER EVALUATIONS

Testing the system with a sample of users – some of whom had experience in DLs and software systems and other who did not – was considered necessary to validate the usefulness of the visual environment for composing DLs.

Table 2: The interface of a component

Service endpoint	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

4.1 Test Methodology

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 were briefed verbally on what DLs are, 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 tasks of building a simple digital library system and modifying an existing system to include one additional service component. To have a control experiment on which to base any comparison, the users were instructed to build a simple digital library manually in addition to using 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 (noting that installation equated to downloading and upzipping in all cases). Finally the users were required to fill out another questionnaire to obtain feedback on their experience.

4.2 Pilot Study

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.

All the testers successfully completed all the tasks, but they highlighted some recurring problems with the system. 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 user expressed a bit of confusion as to what exactly they were doing, and they said it only became apparent after viewing the resulting DL. These problems were addressed before a more complete user study was conducted, largely through the choice of more appropriate icons, better feedback to the users during the process and more logical default values wherever possible. The final system was appropriately renamed Blox+.

4.3 Population and Survey

34 students participated in the final experiment and the time it took them to complete it ranged from 25 minutes to over an hour. 33 of the 34 participants were undergraduates from various faculties, with a third of the students not having computer science as one of their majors. Half of the users had never used modelling software before, and a third had never used a Linux command prompt. Hence the stage was set to test the level of skill required to successfully use the visual component composition system.

Table 3 summarises the results obtained from the final questionnaire. This questionnaire required that users indicate the extent to which they agreed or disagreed with the statements presented according to a 5-point scale. The last four questions were free-form questions to obtain further comments from the users.

4.4 Results

On a qualitative level, all users succeeded in creating the digital library systems required of the test. In general, the overall feedback was positive since most users agreed that not only would they use Blox+ to create their digital library should they need one, but would also consider using Blox+ to create other component-based systems if it had support for them. After assembling the three-component digital library with the command-line and then with the GUI, users were able to better appreciate the differences between the two approaches, with the vast majority indicating that they thought it was a good idea to be able to assemble component-based systems with a graphical user interface. However, it must be noted that the results may have been different if the users were required to create highly complex DL systems – though such systems are rarely found in practice.

In terms of understanding, the distinction between types and instances affected the way some users were able to create the DL, as they were not sure when to select a component/instance from the types panel or from the instances panel. Most of the users who could not make this differentiation were not computer science students. This is an indication that the models well understood within one community for designing visual environments might not transpose to other

Table 3: Summary of responses from user survey

Questions	Responses				
	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
I understand how the Blox system functions.	2	16	12	4	
I understand the concept of types and instances.	3	19	9	3	
Based on previous knowledge, and the knowledge gained from conducting this experiment, I think it is a good idea to be able to graphically assemble systems from distributed components.	18	13	2	1	
I found the interface consistent with my previous experiences with Visual IDEs.	5	11	13	4	1
I would consider using ODL and OAI components if I need to build a system with requirements similar to the exercise.	13	13	4	4	
I would use a system such as Blox to create digital libraries if I had the need to create one.	12	16	6		
I would use a system such as Blox to connect and configure other types of components (e.g. other Web Services, or COM or CORBA objects) if it had support for them.	9	18	7		
Modifying an existing digital library was easy with the Blox system.	13	15	6		
When I created a digital library using Blox, it produced the digital library I expected it to produce.	19	10	5		
The interface responded as I expected it to.	14	13	6	1	
I found it easy to use Blox to accomplish the tasks.	11	17	6		
I found it easy to apply my previous GUI experiences with Blox.	16	8	7	2	1
I like the idea of representing a component as a window.	19	10	5		
I found using Blox to create a digital library easier than manually using OAI/ODL Components.	19	7	8		
I found using Blox to create a digital library faster than manually using OAI/ODL Components.	21	10	3		

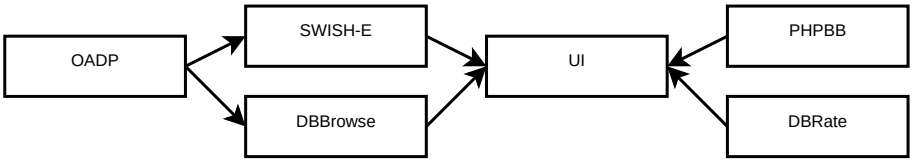

Figure 4: DL users were required to build during tests

communities of users.

When compared to the traditional textual configuration method, most of the users agreed that Blox+ was easy to use in accomplishing the tasks, with none disagreeing and 8 of the users choosing to remain neutral. When considered in isolation, most users once again agreed that Blox+ was easy to use and that the tasks were easily accomplished.

A number of users expressed their appreciation for the Blox+ system as is, with comments such as:

- "This is the new system to use in future."
- "I think that the Blox program is very good, personally I knew nothing about components and stuff before but with this program it has made me understand."

Also, a number of useful suggestions were made, including:

- "A built-in preview of the digital library as it appears in the browser".
- "A help button for troubleshooting, guidelines and stuff."
- "People should be told to create ui → irdb → xmlfile i.e., work from left to right (it's more intuitive)."

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 user 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 as digital library systems evolve into more complex interrelated information management systems.

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 dataflow-driven systems.

6 FUTURE WORK

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.

DLs generally interact with users by means of a user interface accessible via the WWW. For this research, a user interface was created that 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, possibly using a portal approach.

Ultimately, production DLs such as Greenstone are gradually moving towards a component model [15] – in the ideal case they also will eventually incorporate the ideas of visual design environments into their products.

ACKNOWLEDGMENTS

Thanks are due to David Moore and Stephen Emslie, who developed the first version of the Blox tool in 2003. Further, this project was made possible by funding from UCT and NRF (Grant number: 2054030).

REFERENCES

[1] H. Suleman and E. A. Fox. "A Framework for Building Open Digital Libraries". *D-Lib Magazine*, vol. 7, no. 12, 2001. URL `http://www.dlib.org/dlib/december01/suleman/12suleman.html`.

[2] D. Castelli and P. Pagano. "OpenDLib: A Digital Library Service System". In M. Agosti and C. Thanos (editors), *Proceedings of the 6th European Conference on Research and Advanced Technology for Digital Libraries*, pp. 292–308. Springer, September 2002.

[3] W. Arms. *Digital Libraries*. MIT Press, 2000.

[4] C. Lagoze, H. V. de Sompel, M. Nelson and S. Warner. *The Open Archives Initiative Protocol for Metadata Harvesting*. Open Archives Initiative, 2001. URL `http://www.openarchives.org/OAI/openarchivesprotocol.html`.

[5] M. Breeding. "Understanding the Protocol for Metadata Harvesting of the Open Archives Initiative". *Computers in Libraries*, vol. 22, no. 8, pp. 24–30, 2002.

[6] "ARC - A Cross-Archive Search Service", 2004. URL `http://arc.cs.odu.edu`.

[7] M. Smith, M. Bass, G. McClellan, R. Tansley, M. Barton, M. Branschofsky, D. Stuve and J. H. Walker. "DSpace: An Open Source Dynamic Digital Repository". *D-Lib Magazine*, vol. 9, no. 1, 2003. URL `http://www.dlib.org/dlib/january03/smith/01smith.html`.

[8] K. Maly, M. Zubair, M. Nelson, X. Liu, H. Anan, J. Gao, J. Tang and Y. Zhao. "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)*, pp. 27–34. Firenze University Press, October 2002.

[9] I. Witten. "Examples of Practical Digital Libraries Collections: Built Internationally Using Greenstone". *D-Lib Magazine*, vol. 9, no. 3, 2003. URL `http://dlib.org/dlib/march03/witten`.

[10] H. Suleman. *Open Digital Libraries*. Ph.D. thesis, Virginia Tech, 2002. URL `http://www.husseinsspace.com/publications/odl.pdf`.

[11] H. Suleman, E. A. Fox, R. Kelapure, A. Krowne and M. Luo. "Building Digital Libraries from Simple Building Blocks". *Online Information Review*, vol. 27, no. 5, pp. 301–310, 2003.

[12] D. Moore, S. Emslie and H. Suleman. "BLOX: Visual Digital Library Building". Tech. Rep. CS03-20-00, Department of Computer Science, University of Cape Town, 2003. URL `http://pubs.cs.uct.ac.za/archive/00000075/`.

[13] "phpBB – Creating Communities", 2004. URL `http://www.phpbb.com`.

[14] J. Rabinowitz. "SWISH-Enhanced", 2004. URL `http://swish-e.org/`.

[15] D. Bainbridge, K. J. Don, G. R. Buchanan, I. Witten, S. Jones, M. Jones and M. I. Barr. "Dynamic Digital Library Construction and Configuration". In *Proceedings of Research and Advanced Technology for Digital Libraries: 8th European Conference (ECDL2004)*, vol. 3232 of *LNCS*. Springer, September 2004.