BLOX: Visual Digital Library Building

Technical Report No. CS03-20-00

David Moore
Department of Computer Science
University of Cape Town
+27 21 424 3492
dmoore@cs.uct.ac.za

Stephen Emslie
Department of Computer Science
University of Cape Town
+27 21 788 6278
semslie@cs.uct.ac.za

Hussein Suleman
Department of Computer Science
University of Cape Town
+27 21 650 5106
hussein@cs.uct.ac.za

ABSTRACT

This paper describes a visual system which was created for connecting and configuring OAI/ODL digital library components. The feasibility of this approach was shown and results were encouraging.

Categories and Subject Descriptors

H.3.7 [**Digital Libraries**]: User Issues and Systems Issues.

General Terms

Management, Design, Human Factors, Languages, Theory.

Keywords

Digital libraries, BLOX, components, connection, ODL, OAI.

1. INTRODUCTION

Systems used for storing data electronically and allowing that data to be accessed through standard electronic means (such as the Internet) are often referred to as digital libraries. Software has been created which eases the creation of these systems [1,4]. The Open Archives Initiative (OAI) [2] provided standards for communication between digital libraries.

In particular, the Open Digital Libraries (ODL) project [3] provided components which can be combined to create digital libraries. BLOX was created to provide a visual interface for these components, in the hopes that using them would become easier.

2. BACKGROUND AND MOTIVATION

As more information becomes available on the Internet and corporations and academic institutions move towards recording all information digitally, digital library systems are gaining in popularity. It is important that these systems adhere to digital library standards and protocols, to ensure they can communicate with other information stores. These protocols can be complex and custom solutions are less likely to adhere to them strictly.

As an increasing number of end users attempt to create digital libraries for different purposes, it is also desirable for digital libraries to be easier to create. For these reasons, tools which aid the development of digital libraries are needed.

At present, there are a few digital library creation tools. Greenstone [4] is one of the more popular applications for creating digital libraries. It is simple to use, and research has recently been done in making it easier to use [Patel, Personal Communication].

Ease of use comes at some loss of functionality. Greenstone systems are limited to running on a single machine. They are also limited to storing and servicing information using their specific, inbuilt method.

The ODL project was an attempt to provide a more flexible approach to building digital libraries. It provided a set of components which covered different areas of digital libraries, such as data store, merging archives and searching and browsing archives. These components can be configured and connected together to create many types of digital libraries.

For example, in Figure 1, a simple digital library is shown. Two collections of XML files are exposed through a data provider. These are linked to an archive merger (DBUnion), which exposes both these data sources as a single archive of information. A search engine and a browse engine are connected to the merger, and these components provide discovery services which a user interface can then use.

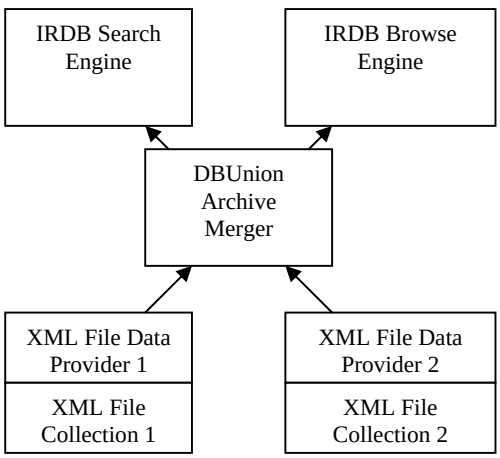

Figure 1: An example digital library using ODL components

Components connect to each other using protocols layered over HTTP. To connect to another component, a component is given the relevant source's URL.

The components remain difficult to configure. Although configuration scripts are provided, it is often necessary to edit XML configuration files manually to complete configuration.

The configuration files used to describe component setup can be complex. In particular, configuration errors are often introduced when defining connections between components, as the relevant URLs are usually long.

"The digital library in a box project was created to simplify and enable the creation of digital libraries" [7]. The current interface to the creation procedure uses a command line interface in a UNIX environment. Even skilled computer users have experienced considerable frustration before successfully making use of these components [8]. This lack of usability makes Digital Library creation using ODL components less feasible for novice users.

3. APPROACH

3.1 Aims

The aim of the BLOX system is to facilitate using the ODL components by abstracting them in a visual manner. Configuration information is separated from the connections between the components. BLOX provides an interface in which users can enter configuration data and connect components together in an intuitive manner.

3.2 System Structure

A visual environment for creating digital libraries needs a location or locations at which to create the necessary components. Creating a central repository on which to store these components is sensible for several reasons. For instance, more than one party might want access to the same data.

In addition, once components have been created, other parties might want to use the already created components for their own digital libraries. For instance, if a user wants to create a digital library, and another then wants to connect their own user interface to the already existing library, they should be able to use the visual interface to achieve this.

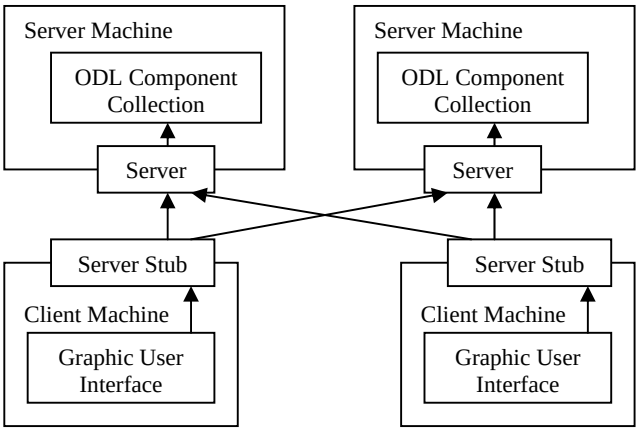

Figure 2: Overview of BLOX system

Alternatively, if two collections of components exist, a user should be able to access both collections and use components from either or both to create digital libraries.

For this reason, BLOX adopted a distributed client/server model. The client provides the graphic user interface and mechanisms for communicating with the server (Server Stubs in Figure 2). The server manages, configures and creates new instances of ODL components in certain locations. Figure 2 demonstrates this concept, showing two BLOX clients collaborating with two BLOX servers simultaneously.

Information needed to successfully configure components was represented using XML Schemas. These define what the configuration options are, which protocols a component can use (i.e., what type of components it can connect to) and what protocols it exposes (i.e., what types of components can connect to it).

3.3 Communications

A protocol was developed using the Simple Object Access Protocol (SOAP) standard [5] for all communication between client and server. This leaves the system open for external clients and servers to be developed. Due to potentially long configuration times, the SOAP messages were sent asynchronously.

3.4 Managing ODL Components

3.4.1 Automated Configuration

A system has been developed to ease automated creation and configuration of ODL components [Eyambe, Personal Communication]. The ODL components are stored in such a way that multiple instances of a component can be configured separately from a single installation.

A component which has not been configured can be viewed as a "type" of component, or potential component. A configured component can be viewed as an "instance" of its type. Using this model, root directories were assigned which held a specific type of component. Instances of that type of component were stored in subdirectories of the root directory.

Scripts written in the Perl scripting language, stored in the root directory, provide access to configuration of the components, to obtain information on the type of component and on what instances of that type currently exist.

The BLOX server uses this system to manage ODL components. It is configured with a list of the root directories of the types. It interacts with the scripts to obtain component information for the client, and to create new instances of components and change the configuration of already existing instances.

3.4.2 Manual Configuration

The system described in the previous section leaves the BLOX server loosely coupled to the components themselves. The server does not keep a register of the component types or instances – it relies on the scripts for that information at run-time. It is also robust in the face of non-existent scripts due to incorrect or out-of-date components.

Thus, components can be manually configured and deleted without special consideration for the BLOX server. Other systems or users could also use the scripts without conflict.

3.5 The Graphical User Interface

A Graphical User Interface has been developed for the creation and configuration of OAI/ODL components. A screenshot is shown in Figure 3. For comparison, a screenshot of manually configuring ODL components is shown in Figure 4.

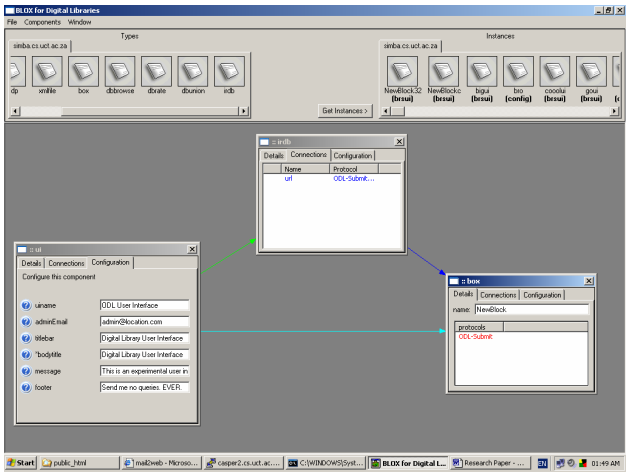

Figure 3: The BLOX user interface

This interface allows users to assemble components on a canvas. Components can then be configured and connected. Once configured, a set of components can be "published" as a digital library. This sends the configuration information to the server which creates an instance of the digital library.

Components are modelled as windows. The content of each window describes how the component should be configured. Arrows can be dragged between windows to represent connections between components. A form is used to capture all other configuration data.

```
-bash-2.05b$ ./configure.pl testsearch
+----------------------------------------+
| IRDB Search Engine Configurator        |
+----------------------------------------+
| Version 1.1 :: September 2002          |
| Hussein Suleman <hussein@vt.edu>       |
| Digital Library Research Laboratory    |
| www.dlib.vt.edu :: Virginia Tech       |
+----------------------------------------+

Defaults/previous values are in brackets - press <enter> to accept those
enter "&delete" to erase a default value
enter "&continue" to skip further questions and use all defaults
press <ctrl>-c to escape at any time (new values will be lost)

Press <enter> to continue



[DBI]
You need the DBI Perl module installed in order to use this script
Checking ... ok


[DATABASE DRIVER]
DBI requires special drivers to access different databases. Please
confirm that the list below contains the name of your database.
If it is not here or you are not sure, press CTRL-C and install it

ExampleP, Proxy, mysql

Press ENTER to continue
```

Figure 4: The current method of configuring components

3.5.1 Representing the Components

The user interface needs to provide the user with a representation of the available component types and component instances for use in creating a digital library.

This is done through the Type and Instance notepads at the top of the screen. Types and instances are represented by icons which can be dragged onto the canvas to create component windows. Types and instances are organised in separate tabs according to the server on which they reside.

3.5.2 The Component Windows

Critical to the whole system is the method of representing configuration information and possible connections in a way which separates these concepts.

Once a user has dragged a component type on to the canvas, creating a potential instance, he or she is presented with a window representing the component. The window consists of a tabbed interface. The tabs are "Details", "Connection" and "Configuration".

The decision to model components as windows is partly because windows are the primary container in the Windows operating system. This is similar to the concept of a component in BLOX, which is a container of configuration information.

Another point in favour of the use of windows is that it allows us to make use of prior user knowledge. In this respect windows are controlled in the same way in BLOX as they are in Windows. This includes resizing them, moving them around and closing them.

3.5.2.1 Details tab

The Details tab presents the user with the protocols which this component exposes. This will be discussed in conjunction with the Connections tab. This tab also allows the user to specify a unique name for this component instance.

3.5.2.2 Connections Tab

The Connections tab presents the user with a list of the protocols to which it is possible for the component to connect. Thus, if a protocol in the connections tab of this component corresponds with a protocol in the Details tab of another component, this component can connect to that other component.

For ODL components, connections are fields in the describing XML. Thus, they appear as optional or necessary fields in the Schemas which represent types. To differentiate connections from configuration information, the "appinfo" tag of the XML Schema description is used to convey information about which protocols a component can potentially connect to. Since configuration elements will not have the protocol information in this tag, presence of the information is enough to identify the relevant fields.

Connecting components is achieved fairly simply. A protocol is selected from the Connections tab, and dragged to the component to which it should connect. The dragging is represented by rubber banding an arrow across the canvas. If a connection is successfully made, a permanent arrow will be drawn between the two components. This can be seen in Figure 3. In addition, connections can be removed by selecting the relevant connection and pressing the "delete" key.

3.5.2.3 *Configuration tab*

The Configuration tab presents the user with a form consisting of all the available configuration fields. This form is created using the XML Schema representing the relevant type of component.

At present, two XML Schema Description (XSD) data constructs are explicitly supported, one is partially supported, and one is implicitly supported. Integers are represented in the form using a spin button. This consists of a text control, and two buttons which increment and decrement the integer in the text control. The text control can still be manually edited.

Enumerations, or a list of string values representing all possible values for a field, are also supported. They are represented with a drop-down list.

All other simple values are assumed to be strings. Thus, strings are implicitly supported. These are represented with a normal text control.

The partially supported data construct is the complex type. This data construct represents any collection of more than one other simple type. BLOX only supports complex types consisting of a sequence of simple types (complex types can also handle structures such as unions).

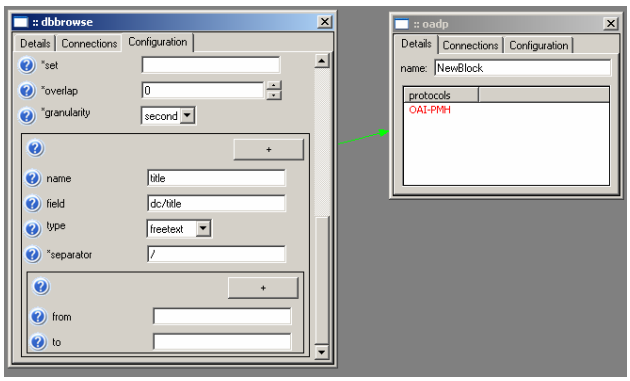

Figure 5: Repeatable fields in the configuration tab

Whenever a field can occur more than once, a button appears which enables the user to duplicate the relevant fields in the form. This also works for the supported complex types, which is shown in Figure 5.

3.5.3 *Canvas Control*

For users to feel comfortable with the interface they should have sufficient control over the canvas space. A number of functions can be performed in this respect. Users can:

- Delete component windows. This is done in the standard way of closing windows in Windows. Deleting a component window means that any connections to that component are also deleted.
- Change the size and position of windows. This is done in the same way that windows are moved and resized in Windows. If windows are dragging off the canvas then the size of the canvas is increased. This means that users have control over both component size and position and the size of the canvas they reside on.
- Windows can be automatically arranged in a number of preset ways: Cascade, Tile Horizontally and Tile Vertically. This is provided for ease of configuration when many component windows are on the same canvas. However this hides lines between components. It is probably simpler for a user to simply enlarge the canvas when more space is needed.

3.5.4 *Tools Used*

XML is used to communicate type and instance configuration information to the user interface. This medium is also used to communicate a completed digital library configuration back to the server.

The interface was implemented using the wxPython windowing toolkit and is therefore deployable on both UNIX and Windows operating systems. This was a design objective as existing digital libraries administrators, who are accustomed to using UNIX environments, should be able to switch to this interface without being required to change operating systems.

3.5.5 *Usability Methodologies*

A Graphical User Interface to the creation of digital libraries is not implicitly easier to use than the current command line method. To provide an interface that is easier to use BLOX applies certain usability criteria.

The GUI is intended to make design and creation of digital libraries easier. As such it is designed in such a way as to support the user's conceptual model of a digital library. An OAI/ODL digital library is a set of components that communicate to form a working structure.

Conceptually, this can be likened to a set of building blocks that work together. The BLOX interface aims at being consistent with this model in the following ways:

Components are modelled as windows. Windows are the primary container in the Windows Operating System. This is similar to the concept of a component in BLOX, which is a container of configuration information.

To make use of prior user knowledge in this respect windows are controlled in the same way in BLOX as they are in Windows. This includes resizing them, moving them around and closing them.

Connections are represented using lines with arrows showing the conceptual direction of connection (not necessarily the direction of flow of data). Connections are differentiated by assigning colours to the lines.

3.6 BLOX User Experience

A step-by-step description of a typical user experience with BLOX follows:

1. BLOX is loaded.
2. The user lets BLOX know what servers exist by adding servers to the server manager.
3. The user starts a new project using digital library components.
4. Types will start appearing in the types box. If more than one server supports the selected handler then a notebook tab will be created for each server. Each tab contains the types available on that server.

5. The user can now click on “Get Instances”, which was disabled till the project was started. If this is done then instances start to fill the instances box. Instances are organized according to the server on which they reside in the same manner as in the types box.
6. A user will now have a set of types (and instances if the “get instances” button was pressed). The user will drag types or instances onto the canvas. For each type or instance dragged in this way a component window is created.
7. Components are configured and connected. Names for components will be entered through the “details” tab. Configuration information will be entered in the “configuration” tab and connection information is supplied by dragging connections from the “connections tab”.
8. If the user is unsure what configuration fields represent, they are able to click on a question mark icon next to the field that supplies this information.
9. Once all components have been configured the user selects “publish” from the components menu.
10. The server attempts to create the digital library from the configuration given. If it succeeds then this is reported to the user. If it fails then an error is given that describes what the error was and where it occurred.

4. RESULTS

Preliminary tests have been performed on the system. Users were chosen for prior knowledge of digital library concepts, but the testers used were not familiar with all the ODL components used in the test.

4.1 Testing Format

Users read a short description of BLOX and a synopsis of the specific components used in the test. They were then required to follow a sequence of tasks which resulted in them first building a simple digital library, and then expanding the digital library to a fairly complex model.

The resulting digital library combined two archives with an archive merger, exposed this merged archive using a searching interface and a browsing interface, and connected these both to a simple Web-based user interface. The developed system is shown in Figure 6, with connection direction as specified in the BLOX user interface. The names given in brackets refer to the specific ODL components used.

The users were then given a questionnaire which tested reaction to aspects of the system ranging from usability of the graphic user interface to whether BLOX produced the digital library they expected it to. Questions were also asked concerning whether the user would want to use the system again. The tests ran for approximately 30 minutes, including reading time and answering the questionnaire.

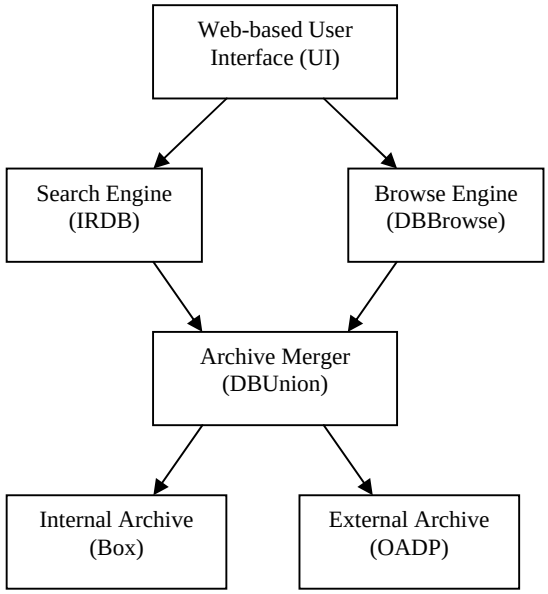

Figure 6: The digital library system created in the user tests

4.2 Test Results

All but one user successfully completed all the tasks in the test. The user who did not complete the tasks experienced some difficulty with creating the more complex digital library due to a limitation of the interface, but still managed to create the simple digital library.

The users’ impressions, as recorded by the questionnaire, were mixed. Some felt that BLOX was both faster and easier than manually configuring digital libraries, while others felt otherwise.

Users also had difficulty with understanding the components. There was confusion at times as to what the purpose of certain configuration fields was. Users occasionally had to be prompted with correct values for configuring.

The confusion arose when certain information configured the manner in which specific connections were made, rather than the internal state of the component.

However, the speed with which users developed digital libraries compare favourably with reported speed for building a digital library manually. Users building simple digital libraries from ODL components are reported to take anywhere from eight hours to three months to create their first digital library [8]. The testers managed to configure a fairly complex digital library in 30 minutes, including the time taken to understand the system and the relevant components.

One issue with the interface which arose consistently was the lack of sufficient feedback for user’s actions. The asynchronous model used, and the amount of time taken to run the scripts on the server, caused a time lag which caused many users to get frustrated with waiting.

However, positive feedback was uniformly given for BLOX as a visual component connection system.

5. CONCLUSION

The testing showed the viability of a system such as BLOX for creating digital libraries. All but one of the users successfully created digital libraries using BLOX.

It has been shown that the methods used by BLOX increase the speed of creating digital libraries. The tests demonstrated users who were unfamiliar with the components still developed digital libraries in a fraction of the time usually taken.

It cannot be said whether using BLOX to create digital libraries is easier than the alternative. Issues which users discovered concerning the system caused confusion for the more complicated tasks. Due to those issues, BLOX cannot be considered as a production-quality system. Instead, it has demonstrated a methodology which looks very promising.

6. FUTURE WORK

As mentioned in the results, there are still problems with the user interface which need to be resolved. In particular, methods for linking certain configuration information to connections need to be investigated. At present, the user interface represents them as fully distinct.

Preliminary tests have been performed with BLOX. More in-depth tests are needed to prove whether or not BLOX can fully replace manual configuration of ODL components.

7. REFERENCES

[1] Castelli, D., Pagano, P., Simi, M. Open DLib System. Accessed 12 April 2003. <http://opendlib.iei.pi.cnr.it/home.html>

[2] Lagoze, C. and Van de Sompel, H. The Open Archives Initiative: Building a Low-barrier interoperability framework. *JCDL '01*, June 17-23, ACM, 2001.

[3] Suleman, H. and Fox, E. A Framework for Building Open Digital Libraries. D-Lib Magazine Volume 7 Number 12, 2001.

[4] Witten, I., Bainbridge, D., Boddie, S., Don, K., McPherson, J. Inside Greenstone Collections. Greenstone Digital Library, 2003. Accessed 14 April 2003. <http://www.cs.waikato.ac.nz/~ihw/greenstone/inside.htm>

[5] Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H. F., Thatte, S., Winer, D. Simple Object Access Protocol (SOAP) 1.1. W3C, 2000. Accessed 12 April 2003. <http://www.w3.org/TR/2000/NOTE-SOAP-20000508>

[6] Mountain, H. M., Kopecky, J., Williams, S., Daniels, G., Mendelsohn, N. Experimental SOAP Binding to Email (RFC2822 – Internet Message Format). W3C 2002.

[7] Digital Libraries in a Box Homepage. Accessed 10 October 2003. <http://dlbox.nudl.org/index.html>

[8] Luhrs, E.. DL-in-a-Box. Rutgers University, New Jersey. 10 May 2003.