On RDBMS and Workflow Support for Componentized Digital Libraries

Pablo A. Roberto, Rodrygo L. T. Santos,
Marcos André Gonçalves, Alberto H. F. Laender

[1]Department of Computer Science - Federal University of Minas Gerais
31270-901 Belo Horizonte, MG Brazil

{pabloa,rodrygo,mgoncalv,laender}@dcc.ufmg.br

Abstract. *We present a new Web services-based componentized framework for building digital libraries. We particularly demonstrate how traditional RDBMS technology can be easily deployed to support several common digital library services. Configuration and customization of the framework to build specialized systems is supported by a wizard-like tool which is based on a generic meta-model for DLs. Such a tool implements a workflow process that segments the DL designer tasks into well-defined steps and drives the designer along these steps. Both the componentized framework and the configuration tool are evaluated in terms of several performance and usability criteria.*

1. Introduction

A digital library (DL) is one of the most advanced and complex types of information system, going far beyond search engines, since they offer many other valued services. They are normally designed for specific user communities, which must be involved with the digital library in all of its aspects, from specification to utilization, in order to guarantee its success.

Many of the existing DLs are based on monolithic architectures and their development projects are characterized by intensive cycles of design, implementation and tests [Suleman 2002]. Several have been built from scratch, aiming to meet the requirements of a particular community or organization [Laender et al. 2004].

One way to deal with these issues is through the creation of software component toolkits for DLs construction, in which each component is responsible for a small part of the DL functionality and integrates with other components in order to build a complete system [Suleman 2002]. Such toolkits offer a generic, extensible and reusable framework for building DLs, permitting, for example, to diminish the necessary effort to develop these systems.

In this work, we present WS-ODL, a new componentized framework to build digital libraries. The framework components operate on top of the Fedora architecture [Lagoze et al. 2005], which provides the framework repository and some basic infrastructure services. All communication among the components and between them and Fedora is done via Web services, using SOAP (Simple Object Access Protocol)[1], which provides advantages like enhanced interoperability and validation of input parameters.

[1]http://www.w3.org/TR/soap/

Besides Fedora, we make use of standard relational database technology to support some components functionality, mainly those providing more advanced services, e.g., searching using structured and textual data conjointly and multidimensional browsing. Since the framework configuration is not a trivial task, we have also developed a wizard-like tool, which guides the user through the installation process.

Our experimental evaluation demonstrates that this framework is feasible and that the components present superior performance when compared to those provided by the ODL framework [Suleman 2002], which served as a starting point for ours. The experiments also show that the wizard-based approach is very effective, allowing the users to install a DL system much more easily and rapidly than by doing it via command-line. We also discuss some drawbacks of the framework in its current version, which should be considered when choosing it as a solution to build digital libraries.

This paper is organized as follows. In Section 2 we describe related work. Section 3 overviews the architecture of all components that make up the framework. The wizard-based installation tool is described in Section 4. In Section 5 we discuss the experimental evaluation of our framework. Finally, Section 6 presents conclusions and perspectives for future work.

2. Related Work

The Open Digital Libraries (ODL) [Suleman 2002] project was one of the first efforts to advocate a componentized approach for the development of digital libraries. ODL proposed an extension to the Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) [Open Archives Initiative 2006] to support interactions among digital library components. This framework had some advantages, due to the use of an OAI-based philosophy and service componentization, like simplicity, openness and reuse. Nevertheless, it presents several problems with respect to performance, scalability and interoperability. Performance issues are due to specific implementation choices which led to scalability problems, mainly regarding the components that implement information retrieval functions. Interoperability issues are related to the use of the XOAI protocol, a not widely adopted OAI-PMH extension developed specifically for that framework.

To overcome some of the deficiencies mentioned above, mainly regarding interoperability, we chose to use the Web services technology in our framework. A Web service is a software system designed to support an interoperable interaction between applications in a network. It communicates with other systems via an interface described in WSDL (Web Service Description Language) and SOAP messages, that are transported using HTTP and XML. SOAP is a protocol for information exchange in a decentralized and distributed environment. Through the use of the XML technology, SOAP defines an extensible framework for message transfer on top of other low level protocols. The framework is designed to be independent of programming models and specific implementation semantics. More details on these technologies can be found at the W3C site[2].

Fedora [Lagoze et al. 2005] is an extensible repository for storage, management and dissemination of complex objects and their relationships. These objects can have local or distributed content, and can be attached to different "disseminations", which makes

[2] http://www.w3.org/

it possible to have various dynamically created representations of a given object. The relationship among the objects is expressed in RDF (Resource Description Framework). The architecture is implemented as a Web service and provides the basis for the construction of other applications, like a digital library, for example. Our framework supports such a construction on top of Fedora while improving part of its functionality, as will be shown. It should be emphasized that other repository infrastructures could also be used, provided they could explore Web services intra-communication purposes.

Regarding implementation choices, we chose to explore open source standards and languages such as Java and traditional DBMS technology, e.g., MySQL. In particular, we wanted to investigate how much support traditional DBMS technology could give to the implementation of the components functionality. In this context, Grossman et al. [Grossman et al. 1997] showed that it is possible to use the standard relational model to implement an information retrieval system and that such a system, in specific scenarios, presents good performance and scales well. Moreover, it makes it possible to integrate structured and textual data, which provides a more powerful search mechanism, comparatively to traditional search engines. This mechanism is very interesting to a digital library, where users have necessities more specific than those of a common Web user. We adopt, adapt, and extend these ideas by showing how several other DL common services can be implemented using the same technology.

With respect to the task of setting up components in order to build digital libraries, some tools have also been described in the literature. BLOX [Eyambe and Suleman 2004, Suleman et al. 2005] is a tool that hides most of the complexity involved in the task of configuring distributed componentized digital libraries. However, during the configuration task, users interact with this tool in an unguided manner: its interface comprises a set of windows, each one representing the configuration of an ODL component. In another approach, the Greenstone suite [Buchanan et al. 2005] incorporates a wizard that allows non-specialist users to create and organize digital collections from local or remote documents. This tool, on the other hand, does not deal with the configuration of service provider components.

3. WS-ODL Overview

In this section, we describe the WS-ODL framework developed to construct digital libraries from a pool of Web services-based components. This framework comprises three major parts:

- A data repository, based on the Fedora architecture, that supports basic infrastructure services, such as an OAI data provider and object versioning.
- A set of Web services to provide specific digital library services, such as searching and browsing.
- A client layer, responsible for generating user interfaces, for sending data to the Web services, and for treating the responses returned by these services. This layer is composed of a set of Java servlets, which execute XSL transformations (XSLT) on the data returned by the Web services.

The WS-ODL framework can be used in a distributed fashion. For example, the Fedora server can be hosted in one server, the Web services in another one, and the client layer in a third one.

3.1. WS-ODL Working Scheme

A typical component of our framework works like this (see Figure 1): (1) the client, via a Web browser, makes an HTTP request to the servlet; (2) the servlet requests to the Web service the relevant identifiers (pids), e.g., pids for recent works; (3) the Web service communicates with the database or the Fedora *risearch* service (Fedora Resource Index Query Service) to get the requested pids; (4) the database or the Fedora service returns the pids to the Web service; (5) the Web service passes the pids to the servlet; (6) the servlet sends the pids to the Fedora repository; (7) the Fedora repository returns the metadata corresponding to the pids; (8) the servlet executes an XSL transformation on the metadata; (9) the data transformed is shown in the client Web browser.

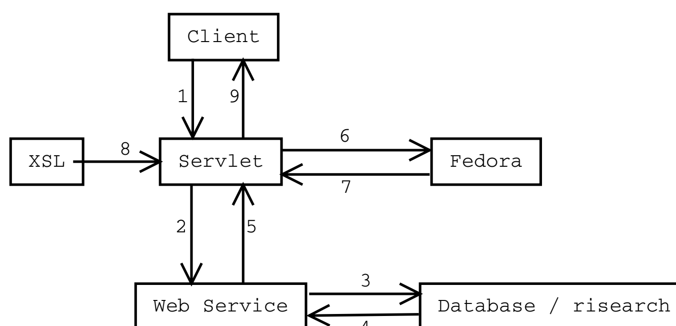

Figure 1. The WS-ODL framework general working scheme

Following, we describe each of the components that make up our framework.

3.2. Recent Works Component

The Recent Works component has the objective of stimulating users' curiosity by exhibiting the works most recently added to the digital library repository, in decreasing order of inclusion date. Users may additionally specify, as a parameter, the number of works they want to see. Otherwise, a default number of ten works is shown.

The pids for recent works come from a table (*doFields*) in the Fedora's relational database that stores (meta-)information about digital objects. The Recent Works Web service accesses this table and retrieves the fields *pid* and *mDate*, ordered by the latter, which contains the last modification date of each of the stored digital objects. The retrieved data is passed to the servlet, which in turn uses it to get the actual digital objects from the Fedora repository. Although based on the Fedora architecture, this component can work with other infrastructures as well with little effort, thus not depending on this particular architecture.

The option of making a direct access to the database and then to access the repository to get the actual data may seem a waste of resources, but this can be justified by the fact that Fedora does not have ordering capabilities in its search API. Therefore, the alternative option, i.e., to get all data and order them locally in the client, would be clearly much more inefficient than the adopted strategy.

3.3. Browsing Components

Our pool incorporates two browsing components with different features. The first one is composed of a set of Web services and servlets designed to provide a navigational

structure to the metadata stored in the Fedora repository. This structure integrates with the other components, allowing the user to have a rich experience in utilizing a digital library built from these components.

The Web services associated with this component are able to obtain: pids referring to a given author name, pids referring to the works of a collection, the name of a collection given its pid, and collection information given a work pid. In the context of this component, a collection is a related group of objects. For example, papers belonging to a conference proceedings form a collection. With this, we can have a collection for the proceedings of SBBD 2005, SBES 2005, SBRC 2005 and so on. This is similar to the OAI *set* concept [Open Archives Initiative 2006].

The servlets make use of the information obtained from the Web services to produce author's works pages and collection's works pages. The functionality of this component is similar to that described in Section 3.1. However, instead of accessing the database to obtain the pids, it uses the Fedora *risearch* service, which permits querying relationships (expressed in RDF) between the digital objects stored in the repository.

The Multidimensional Browsing component [Raghavan et al. 2005], our second component for browsing, provides a hierarchical navigation structure based on a metamodel expressed in XML and on a relational database. It allows for other navigation paradigms (e.g., by date) to be delivered to the user, by just altering its metamodel configuration and updating the database. It can also work as a substitute for author's works and collection's works pages. The metamodel basically contains dimension names and XPath expressions used to obtain the desired data from the digital object. This data is then stored in a database. This component has been created by the Virginia Tech DL Group[3] but we have modified its structure in order to accept multivalued attributes for digital objects (e.g., more than one author). An interesting feature of this component is that it is independent of the repository being used by the framework. This way, if Fedora is to be replaced by any other architecture, this component will demand no modification.

3.4. Searching Component

A searching service, structured or keyword-based, is an important part of any digital library, since it provides to the user a practical way of finding the information he/she desires. The Fedora architecture has searching capabilities in its API and provides a search interface to the end user. However, this search service has some limitations. For example, it does not support retrieval of query results ranked by relevance. In this sense, our Searching component can work as a substitute for the Fedora's native search API, adding the capability for processing relevance-based queries and the possibility of integrating structured and textual data in the same query.

Our Searching component takes three input parameters – the query keywords, the type of search and the number of documents to be returned – and produces an HTML page with the results satisfying the query. The search type can be either Boolean (AND / OR) or by relevance. In the latter case, documents are ordered according to their similarity to the query, which is calculated based on the standard vector space model [Salton et al. 1975]. The number of documents limits the size of the response, since the user is normally only interested in the top documents.

[3]http://www.dlib.vt.edu/

This component is also independent of the repository and works like the Recent Works component, using a relational database to return the pids for the objects satisfying the user query. This database includes relations that simulate an inverted file, and an SQL query can be issued to obtain the desired pids. So, there is a table representing a document (with its *doc_id* and *weight*), a table representing the terms (with the term identifier and *idf* – inverse document frequency), a table for the association document-term (with the term frequency – *tf* – within the document), and a table used by structured queries, indicating the document field in which a given term occurs. Figure 2 illustrates the database relations. This component is not just a simple SQL-based search service, since it is capable of ranking the retrieved documents by their similarity to the query. Moreover, matching between query and document can be partial, which in general does not occur in SQL.

Doc Table	
doc_id	weight
1	5,7
2	6,9
3	10,0
4	0,5

IDF Table	
term	idf
a	4,6
b	1,3
c	0,8
d	9,9

Doc_Term Table		
doc_id	term	tf
1	c	2
1	b	1
2	b	3
3	c	5
3	a	5
4	d	1

Doc_Term_Struct Table		
doc_id	field	content
1	date	c
1	title	b
2	creator	b
3	date	c
3	abstract	a
4	title	d

Figure 2. Search database example

The use of a relational database makes it possible to issue a query combining structured and textual data [Grossman et al. 1997]. For instance, we can search for the objects whose author's name contains the fragment *assis* (structured data) and containing the terms *based* and *zonas* (textual data). So, if the user enables the structured portion of the query, he can inform a data field (e.g., *author*), a relational operator (e.g., *contains*) and a value (e.g., *assis*), and this will work as a filter on the textual portion. The query produced is shown below.

```
SELECT D.fedora_pid, SUM(DT.tf * I.idf * I.idf) / D.weight AS rank
FROM doc D, doc_term DT, idf I, doc_term_struct DTS
WHERE I.term IN ("based", "zonas") AND D.doc_id = DT.doc_id AND
  DT.term = I.term AND D.doc_id = DTS.doc_id AND
  DTS.field = "dc:creator" AND DTS.content LIKE "%assis%"
GROUP BY DT.doc_id
ORDER BY rank DESC LIMIT 30
```

3.5. Administration Components

These components are used to insert data into the DL, as illustrated in Figure 3. The Data Inclusion component uses as input an XML file, in a format similar to that of a response to a ListRecords OAI request [Open Archives Initiative 2006], containing all records to be inserted in the repository. From this file, it generates a set of FOXML (Fedora Object XML) files (object0, object1, etc.), which are then inserted into the repository in the order they were created (based on their suffixes) with the aid of the Fedora API. The Source Database component is used to create a database that works as a middleware between Fedora and the browse and search databases. If it did not exist, the Browse and Searching components would have to communicate directly with Fedora to obtain the data they need. Since this communication (HTTP-based) is slower than a database communication, we chose to store the data in an intermediary database, to improve performance when creating the other databases. This way, the Source Database component also improves

the generality of the framework, since the Fedora OAI data provider could be replaced by any other component with the same functionality without affecting the creation of the databases. It basically issues OAI requests to the Fedora OAI data provider and stores the results in the database. These requests are incremental, meaning that only new content is retrieved from the repository. If the object has suffered any modification and it is versioned in Fedora, only the last version is retrieved. From the source database and a metamodel, the Browse Database component builds the browse database, used by the Multidimensional Browsing component. The creation of this database is also done in an incremental fashion. The last component (Search Database) creates the database used by the Searching component. It is possible to inform the delimiters between the words, the stop words and which fields are to be indexed to create the database. The final result will be a database similar to that depicted in Figure 2.

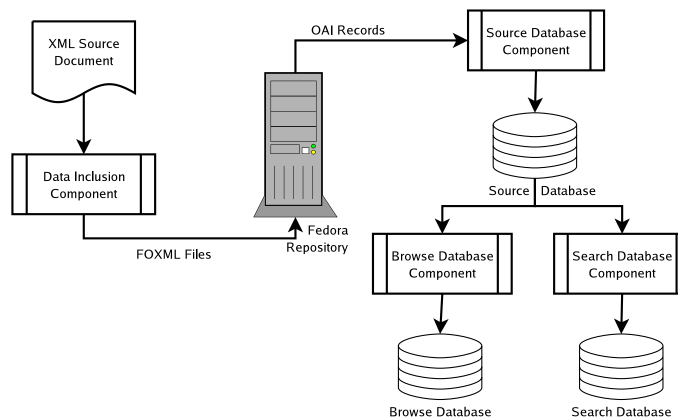

Figure 3. Administration components

4. Wizard Overview

In order to facilitate the configuration of the framework, a wizard-like tool is provided to guide the designer during the installation task. Its architecture basically follows the MVC (Model-View-Controller) framework [Burbeck 1987] with the addition of a persistence layer. The *model* layer was primarily designed [Santos 2005] based on configuration requirements gathered from the ODL framework. Later, it was extended in order to support the configuration workflow of different component pools. Such extension was conceived based on the definition of a digital library taken from the 5S (Streams, Structures, Spaces, Scenarios, Societies) model [Gonçalves et al. 2004]. Accordingly to 5S, a typical digital library is informally defined as a set of mathematical components (e.g., collections, services), each component being precisely defined as functional compositions or set-based combinations of formal constructs from the model. Our configuration model was devised regarding the components that make up a 5S-like digital library as configurable instances of software components provided by a component pool. By "configurable instances" we mean software components whose behaviors are defined as sets of user-configurable parameters.

The class diagram in Figure 4 shows a simplified view of the devised model. As shown in the diagram, a *digital library* is implemented as a set of configurable instances of *provider* components, among those supplied by the *pool* being used. A provider may

be typed either a *repository* or a *service*, according to its role within the library. For orthogonality purposes, the digital library itself is also implemented as a configurable instance of a *component*. Additionally, components may be declared mandatory, as well as dependent on other components. The configuration of each component is implemented as a set of *parameters*, semantically organized into parameter groups. For validation purposes, each parameter is associated to an existing Java type; they may also have a default value, in conformance with their defined type. Parameters may be also declared mandatory (not null) or repeatable (with cardinality greater than one).

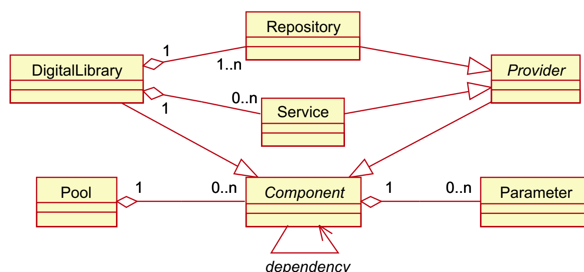

Figure 4. Class diagram for the model layer

View and *controller* layers are integrated – a common simplification of the original MVC framework. They are responsible for handling user interactions, for performing the corresponding modifications to the configuration model, and for displaying the updated model back to the user. Once user interactions are directly transmitted to the model, users modify a clone rather than the original configuration of each component. This allows users to cancel all the modifications performed to a given component at any time.

Since the configuration workflow is organized into steps, a wizard-like interface was a natural choice (Figure 5). In this interface, each step comprises a major aspect of a digital library: the library itself, its collections and/or metadata catalogs, and the services it may provide. In each of these steps, the parameters associated to each of the components they list are presented in dynamically created, tab-organized forms. Each tab corresponds to a parameter group. Form elements are designed according to the type of the parameter they represent: repeatable parameters are shown as lists, parameters representing file descriptors present a file chooser dialog, parameters with values restricted to an enumerable domain are displayed as a combo box, strings and integers are simply shown as text fields. The semantics of every parameter is displayed as a tooltip near the parameter label. Type-checking is performed against every value entered by the user; in case of an erroneous value, a corresponding exception is raised and the user is notified.

The *persistence* layer is responsible for loading and saving the components configuration. Besides that, it is up to this layer the tasks of setting environment variables and preparing databases that support the execution of some components. Its working scheme is based on two XML documents: a *pool descriptor* and a *configuration log*. The pool descriptor document details every component in the pool, including all configuration parameters associated to them. The description of each configuration parameter contains path entries of the form *document*:*xpath_expression* that uniquely locate the parameter in each of its source documents. Since some path entries are dependent on auto-detected or user-entered information, both only known at runtime (e.g., the base directory of the wizard tool and the current digital library identifier), the pool descriptor document also

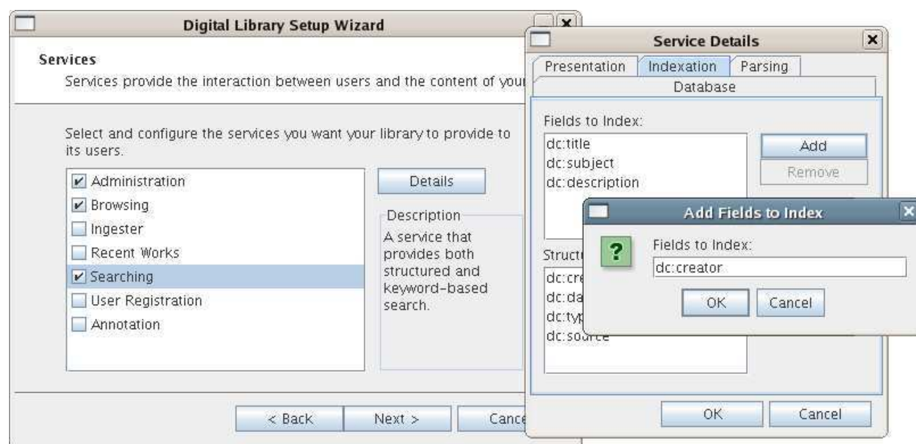

Figure 5. Configuring the Searching component

```
<component id="library" type="model.pool.library.DigitalLibrary">
  <group>
    <label>General Configuration</label>
    <parameter id="libraryName" type="java.lang.String" mandatory="yes" repeatable="no">
      <path>#wizardHome/res/libs.xml:/config/library[@id='#libraryId']/name</path>
      <default>My New Library</default>
      <label>Library Name:</label>
      <description>A human readable name for the library.</description>
    </parameter>
    ...
```

Figure 6. Fragment from the pool descriptor document

comprises a list of definitions to be used in path entries declaration. For example, in the listing in Figure 6, the path entry for the `libraryName` parameter is declared relatively to the definitions `wizardHome` (auto-detected) and `libraryId` (user-entered). The other document, a configuration log, acts as a cache for the persistence layer. It comprises information about the currently configured digital libraries running in the server.

Both XML documents are handled via DOM. Loading and saving of components are performed through XPath expressions. Based on the specification of each component (from the pool descriptor document), configured instances of them are loaded into the digital library model; besides, a template of each component is added to the model so that new instances of components can be added later. Loading is performed in a lazy fashion, i.e., objects are created only when needed. On the other hand, saving is only performed at the end of the whole configuration task, as well as some additional tasks, such as environment variables and database setup, performed via system calls.

Specializing the wizard tool to assist the configuration of different component pools can be done just by providing a description document for each pool to be configured, as well as eventual accessory scripts for performing system calls. In fact, during its development, we have also produced a version for the ODL framework.

5. Framework Evaluation

In order to verify the viability of our framework, we have performed several tests. These include an installation test, a data loading test and a performance test, all discussed below.

5.1. Installation Test

In the installation test, we wanted to measure how difficult it is to configure our framework, mainly by non-specialist users. Users were asked to follow an installation guide, containing instructions to configure all parts of the framework. This experiment was performed with 8 users, acting as digital library administrators, being 4 from Computer Science (CS) and 4 from Library and Information Science (LIS). The time taken by this task was measured at five checkpoints (A - install and configure Fedora; B - install Apache Axis; C - install the Web services on Axis; D - install the servlets; and E - insert data in the constructed digital library), comprising macro-phases of the installation process. The number of errors made in each phase was also measured. Tables 1 and 2 show the results. Times are displayed in the form *hh:mm*.

	Tasks					
User	A	B	C	D	E	Total
CS Mean	0:35	0:12	0:26	0:10	0:20	1:43
CS Std. Dev.	0:05	0:02	0:08	0:04	0:04	0:11
LIS Mean	0:43	0:14	0:30	0:08	0:20	1:53
LIS Std. Dev.	0:16	0:04	0:08	0:02	0:06	0:31
All Mean	0:39	0:13	0:28	0:09	0:20	1:48
All Std. Dev.	0:11	0:03	0:07	0:03	0:04	0:22

Table 1. Time spent per task

	Tasks					
User	A	B	C	D	E	Total
CS Mean	7	2	5	1	2	16
CS Std. Dev.	3	1	1	1	1	4
LIS Mean	11	4	12	2	6	34
LIS Std. Dev.	7	2	7	1	4	18
All Mean	9	3	8	2	4	25
All Std. Dev.	5	2	6	1	4	15

Table 2. Errors made per task

As we can see, the installation process is not trivial, with users taking almost two hours to complete the task. The main reason behind this is the fact that the process is all command-line driven. The most complex task is the installation of Fedora, followed by the installation of the Web services on Axis. This occurs because these two tasks are the most intensive in terms of using the command-line. Besides that, the installation of Fedora is naturally complex, due to the activities involved (e.g., configuring a database).

As expected, CS users tend to be more skilled in this kind of task, since they are used to dealing with computers. They completed the task just a little faster than LIS users (about 10%, on average), but made much less errors (about 53%, on average). This probably happened because CS users payed more attention to the installation guide. This way, their number of errors were lower, but the time they took was almost the same to that of LIS users, since they spent it carefully reading the guide. Nevertheless, the experiment demonstrates that our framework can be used by non-specialist users, and that they spend just a little more time than CS users, although making more errors.

In order to evaluate the effectiveness of the wizard-guided approach, we have conducted similar experiments also involving four users from Computer Science (CS) and four from Library and Information Science (LIS). These experiments included performing two configuration tasks and filling in an evaluation questionnaire. Both tasks highly explore all interface elements of the wizard tool, such as lists and file choosers. The first and simpler task, aimed at helping users to get familiar with the tool, consisted of modifying a few parameters of a pre-configured digital library. The second and more complex one consisted of configuring a whole library from scratch. This task was designed to be comparable to the one performed in the command-line driven installation test. Though data insertion is considered out of the scope of the wizard tool but is performed in the command-line installation experiments, the comparison was still possible just by discarding the time spent at checkpoint E while comparing the overall times. Table 3 shows the

completion time and correctness from the two experiments conducted with the wizard prototype (namely, tasks #1 and #2), as well as those for the users who also performed the command-line driven configuration experiment (task #2^c). For comparison purposes, the performance of an expert user – the developers of the wizard tool and of the WS-ODL framework – is also shown at the end of the table. Time is displayed in the form *hh:mm:ss* and correctness stands for the number of correctly executed items in the configuration task divided by the total number of items in that task.

	Completion Time			Correctness		
User	Task #1	Task #2	Task #2^c	Task #1	Task #2	Task #2^c
CS #1	00:05:16	00:10:48	–	1.00	1.00	–
CS #2	00:07:27	00:17:36	–	1.00	0.96	–
CS #3	00:07:26	00:08:09	01:36:00	1.00	1.00	0.78
CS #4	00:07:54	00:09:10	01:12:00	0.92	1.00	0.88
CS Mean	00:07:01	00:11:26	01:24:00	0.98	0.99	0.83
CS Std. Dev.	00:01:11	00:04:15	00:16:58	0.04	0.02	0.07
LIS #1	00:15:59	00:20:38	–	1.00	0.96	–
LIS #2	00:08:01	00:17:22	01:36:00	1.00	1.00	0.55
LIS #3	00:08:59	00:16:11	–	1.00	1.00	–
LIS #4	00:11:21	00:20:03	01:35:00	1.00	0.82	0.69
LIS Mean	00:11:05	00:18:33	01:35:30	1.00	0.95	0.62
LIS Std. Dev.	00:03:33	00:02:08	00:00:42	0.00	0.09	0.10
Global Mean	00:09:03	00:15:00	01:29:45	0.99	0.97	0.72
Global Std. Dev.	00:03:17	00:04:55	00:11:51	0.03	0.06	0.14
Expert	00:01:53	00:04:33	00:37:00	1.00	1.00	1.00

Table 3. Completion time and correctness per task

Comparison between the wizard-guided and the command-line driven processes for task #2 shows that configuring WS-ODL components with the aid of the wizard tool is much faster (about 500%, on average) than manually (hypothesis accepted by statistical analysis: t test with $\alpha = 0.05$). Configuration correctness is also substantially increased (about 34%, on average) with the aid of the tool (hypothesis accepted by statistical analysis: t test with $\alpha = 0.05$). This is mainly due to its type-checking and component dependency checker systems. Fastness and correctness attest the effectiveness of the wizard-based against the command-line driven process. Effectiveness was also subjectively rated by users who participated in both tasks and measured based on a 5-point bipolar scale, ranging from 1 (worst rating) to 5 (best rating). On average, the effectiveness of the wizard-guided process, in terms of easing the configuration task, was rated 4.5.

5.2. Data Loading Test

The second experiment was a data-loading test, in which we wanted to measure the time taken to put a whole collection within the framework. We have measured the time for six activities in this process: (1) transform the input file containing the records in a unique FOXML file; (2) split this file, in order to create one file per digital object; (3) insert these objects in the Fedora repository; (4) create the source database; (5) create the browsing database, and (6) create the search database. We repeated the experiment 10 times for each activity, and took the average execution time. Activities 1 to 3 are related to Fedora and can be different in case another repository is chosen. Activities 4 to 6 are related to our framework, no matter which repository is used.

We used three data collections: BDBComp [Laender et al. 2004] (4,142 objects),

LDB (29,480 objects) and Citeseer[4] (574,901 objects). The LDB collection is the union of three data sources (Lattes[5], DBLP[6], and BDBComp), with replicas discarded and scope constrained only to the Brazilian computer science community, since Lattes is a curriculum information system used by the whole Brazilian scientific community. The machine used for this experiment is a 32-bit Pentium 4 HT 3.2GHz, with 2GB of RAM and a HD of 400GB. The measured times are shown in Table 4.

Activity	BDBComp	LDB	Citeseer	Activity	BDBComp	LDB	Citeseer
A1	4.90	23.24	9,648.00	A4	49.30	294.82	-
A2	125.15	248.54	6,768.00	A5	117.00	3,333.32	-
A3	1,362.60	4,238.05	-	A6	42.34	668.60	-

Table 4. Data loading times (in seconds)

It can be observed that the most time consuming activity is the insertion of objects in Fedora. This happens because, for each object inserted, Fedora has to generate and store many RDF triples in the Kowari triplestore used to maintain object to object relationships. The number of created triples depends on the object´s size and on its relationships with other objects. In our case, about 30 triples are generated per object in average, and this has a major impact in the performance of this activity. Besides that, the Kowari version used in Fedora 2.0 (the version we used) has a memory leak problem, which also causes performance problems, especially with large volumes of data.

For the Citeseer collection, the whole activity could not be performed. One of the reasons is the memory leak mentioned above. The other is related to the collection size. Since this collection has more than 570,000 objects, the RDF triples generation process exhausted the 32-bit memory address space of the machine used, causing the Java Virtual Machine used to run the framework to crash with an out-of-memory exception. Since this activity is a pre-requisite for the others, they were not performed either. Note that these are problems specific to the chosen repository infrastructure, not the framework itself. Solutions to these problems include upgrading to Fedora 2.1, which comes with a bug fixed Kowari version, and using a 64-bit machine. Alternatively, we could replace the whole repository infrastructure.

For comparison purposes, we measured the data loading time taken by ODL for the BDBComp and LDB collections. We measured the time ODL took to generate its recent works, browsing and searching databases. To complete these tasks, ODL took 1,740 seconds for the BDBComp collection and 11,520 seconds for LDB. Regarding BDBComp, the two frameworks are practically identical, with a slight advantage of ours (1,701 seconds); but, concerning LDB, our framework data loading is 1.31 times faster than ODL, finishing the process in 8,807 seconds. One of the reasons behind this is the fact that ODL harvests the collection via OAI-PMH to each one of its databases, while our framework harvests only once (to the source database).

5.3. Performance Test

This experiment consists of a time comparison between the components developed and those from the ODL framework. To perform this test and to exercise the components

[4]http://citeseer.ist.psu.edu/
[5]http://lattes.cnpq.br/
[6]http://dblp.uni-trier.de/

utilization, we constructed a digital library prototype with functionality similar to that presented by BDBComp. Using Apache jMeter[7], we measured response times of HTTP requests issued to the components. We took two kinds of measure: the time taken for each component to return the relevant pids and the time taken for each component to show the response to the user. Table 5 presents the results for the LDB collection (mean time, standard deviation and gain over ODL). In the table, there are three kinds of entry: ODL entries, representing ODL components; WS-ODL entries, representing the components of our framework; and WS-ODL 2 entries, which represent a variation of our components, as explained below.

	PID Time (ms)		Page Time (ms)		Gain Over ODL	
Component	Mean	Std. Dev.	Mean	Std. Dev.	PID Time	Page Time
ODL Recent	232.55	5.96	1,748.18	144.48	-	-
WS-ODL Recent	31.59	3.58	3,312.18	120.52	7.36 faster	1.89 slower
WS-ODL Recent 2	31.59	3.58	465.63	43.20	7.36 faster	3.75 faster
ODL Search	229.81	17.29	1,068.64	43.65	-	-
WS-ODL Search	54.44	4.38	3,590.52	58.08	4.22 faster	3.36 slower
WS-ODL Search 2	54.44	4.38	447.68	7.29	4.22 faster	2.39 faster
ODL Browse Author	180.35	18.93	662.55	14.12	-	-
WS-ODL Browse Author	38.02	5.73	1,598.11	33.67	4.74 faster	2.41 slower
WS-ODL Browse Author 2	38.02	5.73	249.58	6.32	4.74 faster	2.65 faster
ODL Browse Collection	169.57	16.20	1,466.13	36.49	-	-
WS-ODL Browse Collection	60.11	3.87	694.97	32.35	2.82 faster	2.11 faster
WS-ODL Browse Collection 2	60.11	3.87	658.78	26.97	2.82 faster	2.23 faster

Table 5. Comparison of components response time

It can be noticed that our framework is much faster than ODL, in terms of retrieving pids. Regarding the time to produce the Web pages, our original framework is slower than ODL in all but one case. However, this happens because our components perform considerably more work than the ODL components. For example, if a retrieved object belongs to a collection, we call an additional Web service to obtain the name of this collection, in order to create a link to this collection's works page. We also insert links in each author name, leading to the author's works page. The Web service call, the additional data retrieved and the links penalizes our performance. If we do not perform all this additional work, our framework is again faster than ODL, as we can see from WS-ODL 2 entries. Our browsing collection page is created faster than ODL's in both cases, because there is not much extra work to do in this case. For this reason, the difference between our two versions is not significant.

Three main reasons explain the better performance of our framework: (1) the use of Java servlets and Web services technologies, which provides superior performance than that offered by CGI, the basis of ODL; (2) an effective use of the repository infrastructure, which offers means to achieve better performance (e.g., object´s caching); (3) a more effective use of the relational database technology, especially regarding the Searching component.

6. Conclusions and Future Work

This work presents a framework for the construction of digital libraries according to a componentized approach. The framework is divided in three layers: the data repository

[7]http://jakarta.apache.org/jmeter/

(in this case based on the Fedora architecture), the Web services, and the client. This architecture allows the distribution of the digital library components, which, by itself, leads to a greater flexibility of the system. Moreover, the use of Web services and open archives facilitates interoperability and standardized access to the system data, by human or machine users. The framework is accompanied by a wizard-like tool that implements a workflow process, driving the designer along the configuration of the framework.

Fedora utilization readily provides to the framework a repository to store digital objects, which can be simple, like the ones used in the examples, or complex, like multimedia objects. Also, Fedora provides some basic infrastructure services, like an OAI data provider. The use of the relational database technology provides support to create DL components with advanced functions, such as search combining structured and textual data and multidimensional browsing. It also improves the performance of the framework, since the above components get data from a database, instead of accessing the Fedora repository, which is slower. The generality of the framework is also improved, since the databases provide a separation between Fedora and the components. This way, Fedora can be replaced by another repository without affecting these components.

In the client layer, the DL administrator has the flexibility to create the layout that is most suited to the users, through the use of XML style sheets (XSL), which are informed to the components via XML configuration files.

Through an experimental evaluation involving Computer Science and Library and Information Science users, we showed that the framework is feasible and presents performance superior to the ODL framework, which inspired the development of ours. We also showed that providing a wizard-like tool to guide the framework installation is essential to facilitate its use by non-specialist users when deploying a DL.

Nevertheless, the framework presents some problems. A major problem comes from the Kowari triplestore used by Fedora. Its memory leaks and RDF triples generation slows down the data insertion process, or even makes it impracticable, as seen in Section 5.2. From the discussion above, it is important to analyze what will be the volume of data treated by the digital library being built, and whether large volumes are to be inserted at once. In these cases, Fedora, and therefore our framework, may not be a good solution in this current version. A change from Fedora to another repository is a possible way to solve the problem.

In the future, we plan to extend the wizard tool in order to support the customization of user interfaces and to enhance some of its own interface aspects based on users' suggestions and observations we made during the experiment sessions. We also plan to address the problems that come from the duplication of repository data in the databases. In addition, we want to upgrade to Fedora 2.1 and test the framework in a 64-bit machine as well as to investigate the use of other repositories. Besides these activities, we plan to use the framework to improve the services or even substitute the current version of BDBComp.

7. Acknowledgments

This work was partially supported by projects I3DL (MCT/CNPq/Protem grant number 680154/01-9) and 5S-VQ (MCT/CNPq/CT-INFO grant number 551013/2005-2).

References

Buchanan, G., Bainbridge, D., Don, K. J., and Witten, I. H. (2005). A new framework for building digital library collections. In *Proceedings of the 5th ACM/IEEE Joint Conference on Digital Libraries*, pages 23–31, Denver, CO, USA.

Burbeck, S. (1987). Applications Programming in Smalltalk-80: How to use Model-View-Controller (MVC). Technical report, Softsmarts Inc.

Eyambe, L. and Suleman, H. (2004). A Digital Library Component Assembly Environment. In *Proceedings of the 2004 Annual Research Conference of the SAICSIT on IT Research in Developing Countries*, pages 15–22, Stellenbosch, South Africa.

Gonçalves, M. A., Fox, E. A., Watson, L. T., and Kipp, N. (2004). Streams, Structures, Spaces, Scenarios, Societies (5S): A Formal Model for Digital Libraries. *ACM TOIS*, 22(2):270–312.

Grossman, D. A., Frieder, O., Holmes, D. O., and Roberts, D. C. (1997). Integrating Structured Data and Text: A Relational Approach. *Journal of the American Society of Information Science*, 48(2):122–132.

Laender, A. H. F., Gonçalves, M. A., and Roberto, P. A. (2004). BDBComp: Building a Digital Library for the Brazilian Computer Science Community. In *Proceedings of the 4th ACM/IEEE Joint Conference on Digital Libraries*, pages 23–24, Tucson, AZ, USA.

Lagoze, C., Payette, S., Shin, E., and Wilper, C. (2005). Fedora: An Architecture for Complex Objects and their Relationships. *Journal of Digital Libraries*. Special Issue on Complex Objects (to appear).

Open Archives Initiative (2006). OAI Protocol for Metadata Harvesting – v.2.0. On-line. Available: http://www.openarchives.org/OAI/openarchivesprotocol.html.

Raghavan, A., Vemuri, N. S., Shen, R., Gonçalves, M. A., Fan, W., and Fox, E. A. (2005). Incremental, Semi-automatic, Mapping-Based Integration of Heterogeneous Collections into Archaeological Digital Libraries: Megiddo Case Study. In *Proceedings of the 9th European Conference on Research and Advanced Technology for Digital Libraries*, pages 139–150, Vienna, Austria.

Salton, G., Wong, A., and Yang, C. S. (1975). A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620.

Santos, R. L. T. (2005). Um Assistente para Configuração de Bibliotecas Digitais Componentizadas. In *I Workshop em Bibliotecas Digitais*, pages 11–20, Uberlândia, MG, Brazil.

Suleman, H. (2002). *Open Digital Libraries*. PhD thesis, Virginia Polytechnic Institute and State University, Blacksburg, VA, USA.

Suleman, H., Feng, K., Mhlongo, S., and Omar, M. (2005). Flexing Digital Library Systems. In *Proceedings of the 8th International Conference on Asian Digital Libraries*, pages 33–37, Bangkok, Thailand.