Design, Implementation, and Evaluation of a Wizard Tool for Setting Up Component-Based Digital Libraries

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

Department of Computer Science, Federal University of Minas Gerais
31270-901 Belo Horizonte MG, Brazil
{rodrygo, pabloa, mgoncalv, laender}@dcc.ufmg.br

Abstract. Although component-based architectures favor the building and extension of digital libraries, the configuration of such systems is not a trivial task. Our approach to simplify the tasks of constructing and customizing component-based digital libraries is based on an assistant tool: a setup wizard that segments those tasks into well-defined steps and drives the user along these steps. For generality purposes, the architecture of the wizard is based on the 5S framework and different wizard versions can be specialized according to the pool of components being configured. This paper describes the design and implementation of this wizard, as well as usability experiments designed to evaluate it.

1 Introduction

The complexity of a digital library, with respect to its content and the range of services it may provide, varies considerably. As an example of a simple system, we could cite BDBComp (*Biblioteca Digital Brasileira de Computação*) [7], which provides, basically, searching, browsing, and submission facilities. More complex systems, such as CITIDEL (Computing and Information Technology Interactive Digital Educational Library) [3], may also include additional services such as advanced searching and browsing through unified collections, binding, discussion lists, etc.

Many of the existing digital libraries are based on monolithic architectures and their development projects are characterized by intensive cycles of design, implementation and tests [13]. Several have been built from scratch, aiming to meet the requirements of a particular community or organization [4].

The utilization of modular architectures, based on software components, beyond being a widely accepted software engineering practice, favors the interoperability of such systems at the levels of information exchange and service collaboration [13].

However, although component-based architectures favor the building and extension of digital libraries, the configuration of such systems is not a trivial task. In this case, the complexity falls on the configuration at the level of each component and on the resolution of functional dependencies between components.

J. Gonzalo et al. (Eds.): ECDL 2006, LNCS 4172, pp. 135–146, 2006.

In existing systems, in general, such configurations are performed manually or via command-line scripts. Both alternatives, however, seem inappropriate in a broader context of digital libraries utilization. Instead, higher level techniques to support the creation of complete digital libraries in a simple manner should be investigated [14].

The approach taken in this paper for simplifying the tasks of constructing and customizing digital libraries consists in segmenting such tasks into steps and in driving the user along these steps. This approach is achieved through the development of a digital library setup wizard running on top of a pool of software components.

Wizards are applications specially suited for assisting users on the execution of both complex and infrequent tasks, presenting such tasks as a series of well-defined steps. Though efficient as assistant tools, such applications are not useful for didactical purposes; on the contrary, they should be designed to hide most of the complexity involved in the task to be accomplished. Besides, they should provide a supplementary rather than substitutive way to accomplish the task, so that they do not restrict its execution by specialist users [8].

This paper is organized as follows. In Section 2, the architecture of the wizard is described in details. Following, Section 3 shows some usage examples. In Section 4, we discuss the usability experimental evaluation of the prototype developed. Section 5 discusses related work. Finally, Section 6 presents conclusions and perspectives for future work.

2 Architecture Overview

In this section, we describe the architecture of the wizard, which basically follows the MVC (Model-View-Controller) framework [2] with the addition of a persistence layer.

The *model* layer was primarily designed [12] based on configuration requirements gathered from the ODL (Open Digital Libraries) framework [14]. Later, it was extended in order to support the configuration of different component pools. Such extension was conceived inspired on the definition of a digital library from the 5S (Streams, Structures, Spaces, Scenarios, Societies) framework [6]. 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 framework. 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 [11] in Fig. 1 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) and/or repeatable (with cardinality greater than one).

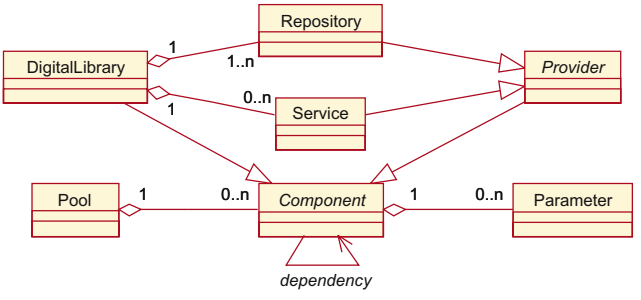

Fig. 1. Class diagram for the model layer

View and *controller* layers are integrated – a common simplification of the original MVC framework. They are responsible for handling user interactions, performing the corresponding modifications to the configuration model and 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.

The configuration interface is organized into steps, in a wizard-like fashion. 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 (Figs. 2, 3, and 4). 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 about the error.

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 and the current digital library identifier), the pool descriptor document also comprises a list of definitions to be used in path entries declaration. For example, in the listing below, 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.

```
<component id="library" type="model.pool.library.DigitalLibrary">
  <group>
    <label>General Configuration</label>
    <parameter id="libraryName" type="java.lang.String" mandatory="yes">
      <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>
    ...
```

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 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 the development project, we produced wizard versions for two component pools, namely, the ODL and WS-ODL frameworks.

3 Usage Examples

In this section, we show some usage examples of configuration tasks performed with the aid of the wizard developed.

The initial step welcomes the user and states the purpose of the wizard. The following step (Fig. 2) handles the digital library's configuration. At this step, previously configured digital libraries are listed by the wizard and the user can choose to modify or even remove any of them. Besides, he/she can choose to create a new digital library. Both library creation and modification are handled by a component editor dialog. For instance, selecting "BDBComp" from the list and clicking on "Details" opens this library's configuration editor dialog. This dialog comprises the digital library' general configuration (e.g., the library's home directory, name, and description), as well as its hosting information (e.g., the server name and port number for the library's application and presentation layers). Selecting a digital library from the list enables the "Next" button on the navigation bar.

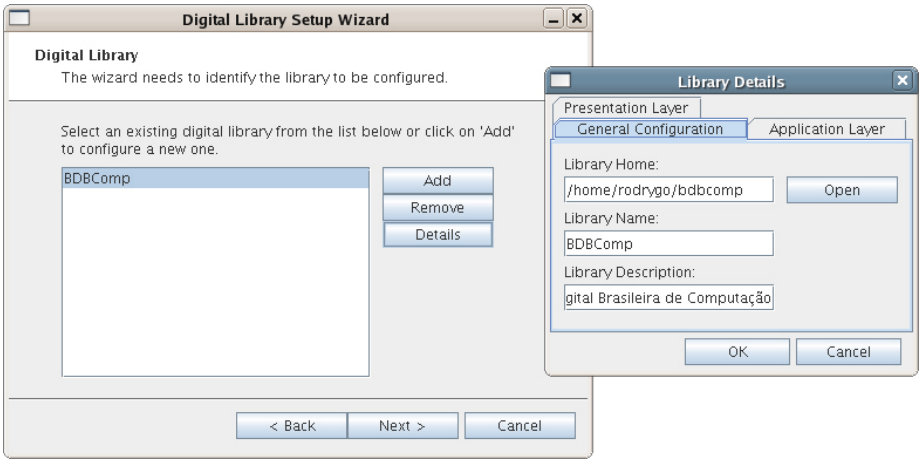

Fig. 2. Configuring digital libraries

Clicking on "Next" drives the user to the following step (Fig. 3), which handles the configuration of the digital library's repositories. Similarly to the previous step, this one shows a list of existing repositories under the currently selected digital library so that the user can choose to modify or remove any of them. As in the previous step, he/she can also add a new repository to the library. Clicking on "Details" after selecting "BDBComp Repository" shows its configuration editor dialog. Repositories' configuration parameters include administrative data (e.g., repository administrators' e-mails and password), hosting information and access permissions (e.g., the repository's server name and a list of hosts allowed to access the repository), database connection and storage paths (e.g., the JDBC driver used to connect to the repository's database and the PID namespace associated to records stored in the repository), etc. Since the whole configuration is performed on the currently selected digital library and is only saved at the end of the configuration task, clicking on "Back" warns the user that selecting a new library to be configured implies discarding the current configuration. If

there is at least one repository under the currently selected digital library, the "Next" button is enabled and the user can go forth.

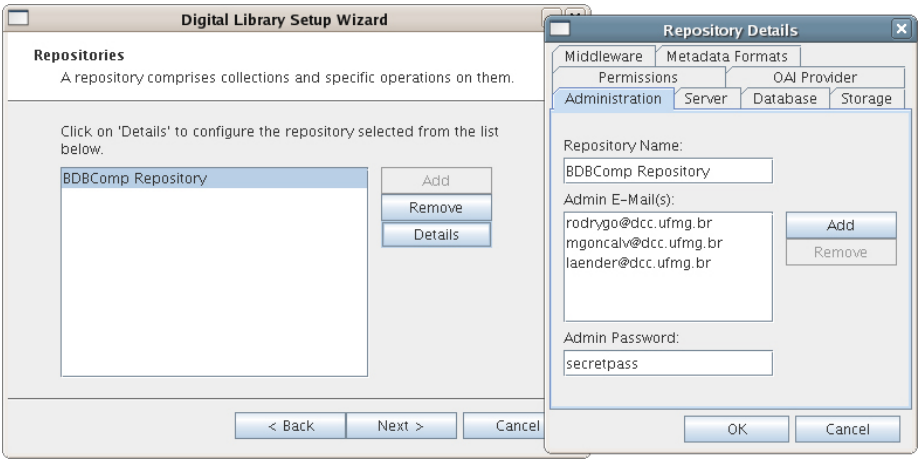

Fig. 3. Configuring repositories

The following step (Fig. 4) handles the configuration of the digital library's services. A list of all the services provided by the pool of components being used is displayed – those already configured under the current library are marked. Selecting any of the services displays its description on the right panel. Trying to unmark a service which is an instance of a mandatory component raises an exception, as well as trying to mark a service component which depends on other components or to unmark a service component that other components depend on. Selecting a service component which has additional parameters to be configured enables the "Details" button. For instance, selecting "Browsing" and clicking on "Details" launches this service's configuration editor. Its configuration includes navigational parameters, such as a list of dimensions for browsing and the number of records to be displayed per page, and presentational parameters, such as the XSL stylesheets to be used when displaying browsing results. As another example, the "Searching" service's configuration includes parsing and indexation parameters, such as lists of delimiters, stopwords and fields to be indexed, among others.

After configuring the services that will be offered by the digital library, the user is driven to the penultimate step. This step summarizes all the configuration performed so far, showing a list of repositories and services comprised by the library being configured. If anything is wrong, the user can go back and correct the proper parameters. Otherwise, clicking on "Configure" saves the current digital library's configuration and drives the user to the last step.

The last step (Fig. 5) notifies the user about the result of the whole configuration task. If no problem has occurred while saving the configurations performed, links to the digital library's services are made available to the user.

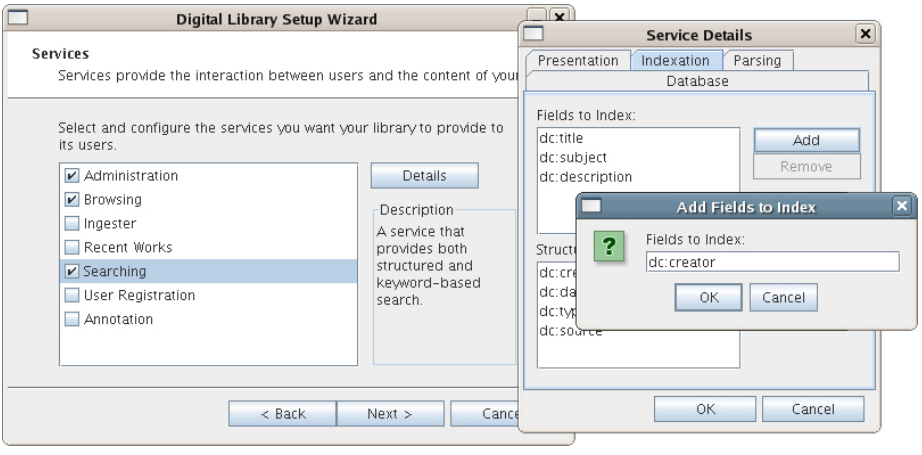

Fig. 4. Configuring services

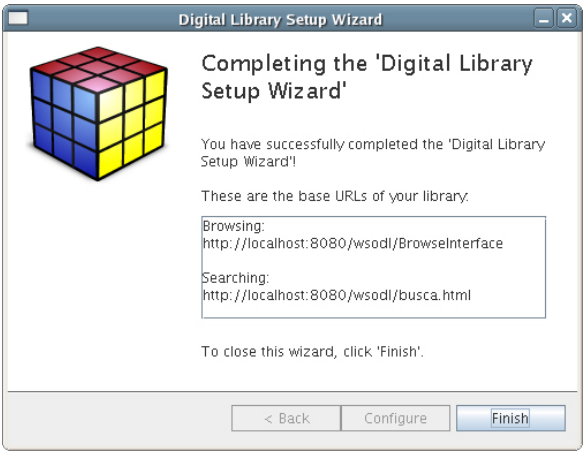

Fig. 5. Configuration completion

4 System Evaluation

In order to evaluate the usability of our tool, we have conducted a series of experiments involving four users from Computer Science (CS) and four from Library and Information Science (LIS). The experiments included performing two configuration tasks and filling in an evaluation questionnaire. Both tasks highly explore all interface elements of the wizard, 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. Since the wizard prototype we tested was running on top of the WS-

ODL framework [10], we designed this second task to be comparable to the one performed at a command-line installation test conducted with that framework. Though data insertion is considered out of the scope of our tool but is performed in the command-line installation experiments of WS-ODL, the comparison was still possible since they measured the installation time at distinct checkpoints, allowing us to discard data insertion time while comparing the overall times. Table 1 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 and 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.

Table 1. Completion time and correctness per 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

Comparing the wizard-guided and the command-line driven approaches for task #2 shows that configuring WS-ODL components with the aid of the wizard 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 wizard (hypothesis accepted by statistical analysis: t test with $\alpha = 0.05$). This is mainly due to the type-checking and component dependency checker systems of the wizard. Fastness and correctness attest the effectiveness of the wizard against the command-line driven approach. 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 approach, in terms of easing the configuration task, was rated 4.5.

The learnability of the tool was also derived from Table 1. For such, we devised two measures: configuration efficiency and expertise. Efficiency stands for the total number of items in the task divided by the overall task completion time. Expertise measures how close the user's completion time is to the expert's completion time. Table 2 shows the values for these two learnability measures. Efficiency is measured in terms of task items performed per minute.

Table 2. Efficiency and expertise per task

	Efficiency			Expertise		
User	Task #1	Task #2	Task #2[c]	Task #1	Task #2	Task #2[c]
CS #1	2.47	2.59	–	0.36	0.42	–
CS #2	1.74	1.59	–	0.25	0.26	–
CS #3	1.75	3.44	0.93	0.25	0.56	0.39
CS #4	1.65	3.05	1.24	0.24	0.50	0.51
CS Mean	1.90	2.67	1.08	0.28	0.43	0.45
CS Std. Dev.	0.38	0.80	0.22	0.06	0.13	0.09
LIS #1	0.81	1.36	–	0.12	0.22	–
LIS #2	1.62	1.61	0.93	0.23	0.26	0.39
LIS #3	1.45	1.73	–	0.21	0.28	–
LIS #4	1.15	1.40	0.94	0.17	0.23	0.39
LIS Mean	1.26	1.52	0.93	0.18	0.25	0.39
LIS Std. Dev.	0.36	0.18	0.01	0.05	0.03	0.00
Global Mean	1.58	2.10	1.01	0.23	0.34	0.42
Global Std. Dev.	0.48	0.81	0.15	0.07	0.13	0.06
Expert	6.90	6.15	2.41	1.00	1.00	1.00

From Table 2, we can see that, in most cases (CS #2 and LIS #2 are the only exceptions), configuration efficiency is increased (about 33%, on average) from task #1 to task #2. Here we regard all task items as equally difficult, what is quite reasonable once all of them consist of setting configuration parameters. Also, the few items that differ in difficulty (e.g., choosing a file in a dialog or adding an item to a list) are homogeneously distributed across the two tasks. Expertise – another learnability indicator – is also increased (about 49%, on average) from task #1 to task #2, what could show that the wizard is easy to learn. However, the hypotheses of efficiency and expertise growth from task #1 to task #2 were rejected by statistical analysis (t test with $\alpha = 0.05$), what suggests that perhaps task #1 was not enough for users to become familiar with the tool.

From the questionnaire filled in by the users who performed the wizard-guided configuration tasks, we devised other two metrics: didactical applicability and satisfaction, both measured based on 5-point bipolar scales, ranging from 1 (worst rating) to 5 (best rating). On average, in terms of understanding of the

concepts being configured (i.e., concepts pertaining to the domain of the component pool on top of which the wizard is running), the didactical applicability of the wizard was subjectively rated 3.75. This was an unexpected yet not unwelcome high value, since the design of wizards is not intended for didactical purposes. Satisfaction was measured in terms of comfort and ease of use. On average, users subjectively rated them 4.25 and 4, respectively.

5 Related Work

There are several works found in the literature that deal with component-based frameworks for building digital libraries. As far as we know, however, there are few works related specifically to the task of configuring such systems. In this section, we present four works that fall into the latter category.

5SGraph [15], a tool based on the 5S framework, provides a visual interface for conceptual modeling of digital libraries from a predefined metamodel. In the modeling task, the user interacts with the tool by incrementally constructing a tree where each node, picked from the metamodel, represents a construct of the digital library being modeled. Differently from the other works presented here, this one has a didactical goal: to teach the 5S theory.

BLOX [5] is a tool that hides most of the complexity involved in the task of configuring distributed component-based digital libraries. However, as occurs in 5SGraph, users interact with this tool in a flexible manner: its interface comprises a set of windows, each one representing the configuration of an ODL component.

The Greenstone suite [1] incorporates a wizard that allows non-specialist users to create and organize digital collections from local or remote documents. Driving the user step by step, this tool gets information such as the name and the purpose of the collection, administrator's e-mail, existing collections to serve as a model, base directories or URL's, etc. This tool, on the other hand, does not deal with the configuration of service provider components.

Finally, the OAIB application (Open Archives in a Box) [9], based on the COCOA framework (Components for Constructing Open Archives), provides a wizard for configuring metadata catalogs stored in RDBMS's. Its interface consists of a series of tabs where each tab presents different configuration options. Similarly to the wizard provided by the Greenstone suite, this one does not deal with the configuration of service providers.

Table 3 summarizes the characteristics of all these tools, comparing them to the ones present in our wizard.

Table 3. Wizard vs. related tools

	Wizard	5SGraph	BLOX	Greenstone	OAIB
task	configuration	modeling	configuration	configuration	configuration
objects	components	5S constructs	components	collections	catalogs
interaction	guided	flexible	flexible	guided	guided
didactical	no	yes	no	no	no

6 Conclusions and Future Work

This paper has presented a wizard tool for setting up component-based digital libraries. The tool is aimed at assisting users in the nontrivial task of configuring software components in order to build a fully functional digital library. The architecture of the wizard comprises a generic model layer for the purpose of supporting the configuration of different component pools upon minimal specialization.

The paper has also presented a usability experimental evaluation of a prototype running on top of the WS-ODL framework. Despite the relatively small number of users, the results (statistically meaningful) show that our approach is quite effective in easing the task of configuring that framework by hiding most of the complexity involved in the configuration task.

As future work, we plan to extend the wizard tool in order to support the customization of user interfaces and workflows. Though its comfort and ease of use have been well-rated, we plan to further enhance some interface aspects of the wizard based on users' suggestions and observations we made during the experiment sessions, in order to improve the overall learnability of the tool. Also, we intend to perform additional experiments in order to compare the guided and flexible interaction approaches, as provided by the wizard and the BLOX tool (for instance), respectively. In the near future, we plan to incorporate the wizard to the WS-ODL framework. Additionally, prototype versions for other component pools could be produced in order to test and expand the generality of the model layer.

Acknowledgments

This work was partially supported by CNPq funded projects I3DL and 5S/VQ. Authors would like to thank Allan J. C. Silva for his valuable help on statistical analysis of our experimental evaluation.

References

1. Buchanan, G., Bainbridge, D., Don, K. J., Witten, I. H.: A new framework for building digital library collections. In: Proceedings of the 5th ACM-IEEE Joint Conference on Digital Libraries (2005) 25–31
2. Burbeck, S.: Applications Programming in Smalltalk-80: How to use Model-View-Controller (MVC), tech. report. Softsmarts Inc. (1987)
3. CITIDEL. http://www.citidel.org, March (2006)
4. Digital Libraries in a Box. http://dlbox.nudl.org, March (2006)
5. Eyambe, L., Suleman, H.: A Digital Library Component Assembly Environment. In: Proceedings of the 2004 Annual Research Conference of the SAICSIT on IT Research in Developing Countries (2004) 15–22
6. Gonçalves, M. A., Fox, E. A., Watson, L. T., Kipp, N.: Streams, Structures, Spaces, Scenarios, Societies (5S): A Formal Model for Digital Libraries. ACM Transactions on Information Systems **22** (2004) 270–312

7. Laender, A. H. F., Gonçalves, M. A., Roberto, P. A.: BDBComp: Building a Digital Library for the Brazilian Computer Science Community. In: Proceedings of the 4th ACM-IEEE Joint Conference on Digital Libraries (2004) 23–24
8. MSDN. http://msdn.microsoft.com/library/en-us/dnwue/html/ch13h.asp, March (2006)
9. Open Archives in a Box. http://dlt.ncsa.uiuc.edu/oaib, March (2006)
10. Roberto, P. A.: Um Arcabouço Baseado em Componentes, Serviços Web e Arquivos Abertos para Construção de Bibliotecas Digitais. Master's thesis, Federal University of Minas Gerais (2006)
11. Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language Reference Manual. Addison-Wesley Professional (2004)
12. Santos, R. L. T.: Um Assistente para Configuração de Bibliotecas Digitais Componentizadas. In: I Workshop in Digital Libraries, Proceedings of the XX Brazilian Symposium on Databases (2005) 11–20
13. Suleman, H., Fox, E. A.: A Framework for Building Open Digital Libraries. D-Lib Magazine **7** (2001)
14. Suleman, H., Feng, K., Mhlongo, S., Omar, M.: Flexing Digital Library Systems. In: Proceedings of the 8th International Conference on Asian Digital Libraries (2005) 33–37
15. Zhu, Q., Gonçalves, M. A., Shen, R., Cassell, L., Fox, E. A.: Visual Semantic Modeling of Digital Libraries. In: Proceedings of the 7th European Conference on Digital Libraries (2003) 325–337