PRACTICAL DIGITAL LIBRARY GENERATION INTO DSPACE WITH THE 5S FRAMEWORK

Douglas Gorton

A Thesis Presented to the Faculty of
The Virginia Polytechnic Institute and State University
In Partial Fulfillment of the Requirements of the Degree of

Master of Science
in
Computer Science and Applications

Thesis Committee:

Dr. Edward A. Fox, Chairman
Dr. Weiguo (Patrick) Fan
Dr. Shawn Bohner

April 13, 2007
Blacksburg, Virginia, USA

Keywords: Digital Libraries, 5S, 5SL, DSpace, Digital Library Specification, Digital Library Generation

Copyright © 2007 Douglas Gorton

PRACTICAL DIGITAL LIBRARY GENERATION INTO DSPACE WITH THE 5S FRAMEWORK

Douglas Gorton

Committee Chairman: Dr. Edward A. Fox
Computer Science

(Abstract)

In today's ever-changing world of technology and information, a growing number of organizations and universities seek to store digital documents in an online, accessible manner. These digital library (DL) repositories are powerful systems that allow institutions to store their digital documents while permitting interaction and collaboration among users in their organizations. Despite the continual work on DL systems that can produce out-of-the-box online repositories, the installation, configuration, and customization processes of these systems are still far from straightforward.

Motivated by the arduous process of designing digital library instances; installing software packages like DSpace and Greenstone; and configuring, customizing, and populating such systems, we have developed an XML-based model for specifying the nature of DSpace digital libraries. The ability to map out a digital library to be created in a straightforward, XML-based way allows for the integration of such a specification with other DL tools. To make use of DL specifications for DSpace, we create a DL generator that uses these models of digital library systems to create, configure, customize, and populate DLs as specified. We draw heavily on previous work in understanding the nature of digital libraries from the 5S framework for digital libraries. This divides the concerns of digital libraries into a complex, formal representation of the elements that are basic to any minimal digital library system including Streams, Structures, Spaces, Scenarios, and Societies. We reflect on this previous work and provide a fresh application of the 5S framework to practical DL systems. Given our specification and generation process, we draw conclusions towards a more general model that would be suitable to characterize any DL platform with one specification.

We present this DSpace DL specification language and generator as an aid to DL designers and others interested in easing the specification of DSpace digital libraries. We believe that our method will not only enable users to create DLs more easily, but also gain a greater understanding about their desired DL structure, software, and digital libraries in general.

Acknowledgements

There is a common saying that indicates that when going somewhere it is not the destination that matters, but the journey. There are many people who made this work possible throughout this journey by discussing, contributing, making suggestions, or even just being there.

I first must recognize the immense contribution and direction of my advisor and thesis committee chairman, Dr. Edward Fox. It has been he throughout my last three years at Virginia Tech who has sparked my interest in this field, encouraged me to contribute to research projects, and guided me through this work. His feedback, direction, and support have allowed this work to blossom into something of interest, and for this I am sincerely grateful.

My sincerest thanks must also go to my other committee members, Dr. Patrick Fan and Dr. Shawn Bohner. Their support, comments, and willingness to serve on my committee are much appreciated and have benefited this work by their expertise and suggestions.

Thanks also go to Will Cameron and his colleagues at Villanova University for their assistance with collections for the case study we provide to demonstrate and validate this work. I also appreciate the feedback and assistance from Dr. Marcos Gonçalves and his students at the Universidade Federal de Minas Gerais in Brazil. Special thanks go to all who allowed me to use their figures in this work as well.

It is friends and loved ones who help to get us through stages of our lives, including this one in mine. I thank the friendship and support of all those who got me through this work and graduate school at Virginia Tech, both my friends here at college and from high school alike.

I'd like to thank my friends and colleagues at the Digital Library Research Laboratory for their support, feedback, and friendship throughout my time here.

My greatest thanks go out to my parents who helped me begin this, and all journeys. Your love and support throughout my education has helped me get where I am and accomplish all that I have. Additional thanks go to my brother and sister for their encouragement throughout the years.

Last but certainly not least, thank you Susan for always believing in me, providing encouragement, and for your love and support.

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction

In today's ever-changing world of technology and information, a growing number of organizations and universities seek to store digital documents in an online, easily accessible manner. These "digital library" repositories can be powerful systems that allow institutions to store and maintain their digital documents and allow interaction and collaboration among users in the organizations.

Before we dig any deeper into these areas and the heart of this work, it is important to lay a common understanding for digital libraries and related technologies. In this introduction, we will give a brief overview of digital libraries and the generation of such repositories. These sections will provide quick overviews to each topic, and serve as an introduction and starting point to the related work discussion which covers in greater depth the important topics. We conclude this section with a road map to the work embodied in this thesis as well as a layout of the document itself.

1.1. What are Digital Libraries?

Very simply put, digital libraries are like traditional libraries; yet unlike traditional libraries' brick and mortar buildings and physical books, digital libraries contain electronic information in a much more ubiquitous nature than is possible within a physical building. Digital libraries have as much to do with the information stored within them as they do the technology that allows that storage to occur—the field of DLs is a hybrid of hypertext, multimedia, retrieval algorithms and technology, indexing, and many other facets. Due to the multi-disciplinary nature of DLs, a concrete, all-encompassing definition for a digital library is difficult. Many researchers in the field have tried to define digital libraries as something more than just the types of technologies and methods the field employs by attempting to describe the very heart of the concerns and aims of digital library systems. We investigate the nature of DLs in greater detail and provide a handful of definitions proposed by different researchers.

1.2. Digital Library Software

Since the advent of the field of digital libraries and the rise of DL systems in everyday use, there has been much examination of how to make the process of creating a

DL most efficient while still keeping a watchful eye on the overall goals, target content, and users. One common hindrance that can cause issues for DL creation is the time and resources needed to go through the various stages of planning, development, and execution of such an online repository. In the infancy of the field, virtually all DLs were custom developed from the ground up, causing these in-depth design and implementation processes. One attempt to address these issues and make digital libraries more available and reasonable for small organizations that do not have the resources or time necessary to build a DL from scratch was the creation of pre-built software packages. Using these DL software packages, a user could create a basic digital library that can accept, store, and serve out information to users with little or no custom programming. With the development of such tools, digital libraries became more accessible and used more widely. While much custom development had been done for prior digital library systems, which held large quantities of information and generally had large audiences, installable DL software systems allow users for any size collection—big or small—to easily create an online repository.

Such software is not without difficulty. While products like Microsoft Word will most commonly only be used by consumers on either Windows or Macintosh platforms, server based systems that continuously run DL software often run Linux, which can make consistency across platforms difficult and slightly complicate installation processes. Also, in making a generic DL software package, some nuances that occur in certain types of data can be lost or require custom development to get these generic DLs operating exactly as an organization would prefer. We will take a closer look at some of these issues later on in our discussion.

1.3. What is Digital Library Generation?

While digital library software packages enable a broader adoption of DLs, there is still a certain amount of configuration, customization, and data ingestion that must occur in such systems before they are truly optimally usable and set up to serve as many of the institution's needs as allowable. The generation of DLs attempts to abstract some of these processes into a simpler, clearer task where the nature of the desired digital library is described and the generator handles those details with regard to configuration,

customization, generation of pertinent code, etc. The intent is to automate these tasks in a way that the DL designer has an appreciation and understanding of the repository to be created but does not need to worry about the underlying technological layer as would be needed if the DL were created manually. Of course, given the wide variety of underlying digital library systems, the steps that a generator goes through and exactly what it can accomplish will vary from system to system. Similarly varying is the manner in which a user tells the generator what settings, features, and functionality they would like the generated DL to have.

1.4. Motivation

Digital libraries are by definition complex. Despite the continual work on DL systems that can provide out of the box online repositories to be created, the installation, configuration, and customization processes of these systems are still far from smooth and completely straightforward. These repository systems are invaluable to groups and organizations that want to store and use their documents and resources in an online nature, or even serve them out to the public. Still, those issues of installation and customization serve as roadblocks to some groups who would benefit from and be interested in such tools.

In our digital library generation process, we seek to make this process easier and to foster a greater understanding of the systems DL designers seek to create and use. With special focus on a common DL software product, DSpace, we look at a way to specify the details and structure of a desired DL, and provide an extensible toolset to generate such DLs from their specifications. We draw on previous work in DL generation to apply this area of research to practical, modern digital library systems. We draw on research that segments the concerns and functional areas of digital libraries into sections to bring greater understanding to the structure and meaning behind different areas of DLs.

Without this work, DL designers must go through the arduous tasks of researching a digital library platform, as well as learning about the details of installation and required configuration tasks, before they can go through an additional set of steps to add users, content, etc. to a digital library. Many universities and organizations have fulltime staff that only create and use DLs—it is not that we want to erase these positions, but there are

many menial and tedious tasks that can be automated in order to focus on what is important, the data and the library itself. Our work eases much of the initial overhead of running a digital library system: installation, configuration, and customization of the most common options and areas of DSpace.

Furthermore, if such a strategy is viable to allow for the specification of a digital library system for automatic generation, is a more general specification possible that can be used across DL platforms, and can we define a specification of our target DL that can apply to many types of DLs? If such a specification language were possible and available, the door would be open for the streamlined, greatly eased process of creating digital libraries from a single specification. If an organization were interested in using digital libraries to store and make documents available, they could define their DL once, and use that specification multiple times in order to experiment with different DL products.

1.5. Approach

Digital library generation has a few pieces that must work in sync in order to have a meaningful result arise. The digital library software itself must be able to be mapped to some sort of specification—a way of indicating the picture of the DL that is desired, a description of what is to be generated. Given such a declaration technique, a generator must be able to meaningfully use that specification and have adequate access to and knowledge of the DL system so that it may go through the process of configuring and generating the specified library.

For these distinct requirements, we build on previous research in order to have appropriate control and descriptive ability of our target digital library software system, MIT's DSpace software. We examine the nature of these software systems, the types of information needed for configuration, the structural makeup of live repositories, and the code base and APIs for the system. In order to make better decisions in developing the schema we would use for the metamodels for each, we looked at surveys of digital library software systems for comparisons, as well as read first hand accounts of users' experiences with these software packages [1], [2], [3].

We base our work on DL specifications on Fox and Gonçalves' work with the 5S Framework for Digital Libraries and its domain specific digital library declaration language, 5SL. We have defined a metamodel for the declaration of DSpace DLs through XML-based schemas and instances of those models that allows the description of a DL system. As with 5S and 5SL, we split the concerns of specification and generation into divisions based on their relation to DL structure, users, interfaces, services, etc. While pre-built DL software packages like DSpace are highly extensible through patches, plugins, and extensions, we look at the specification and configuration of only the most widely used features, assuming that if large changes to the source code are needed that perhaps automatic generation is not the best choice for those users.

For generation, we have developed an extensible, Java-based generator tool that accepts a target DL platform and XML specification, verifies its correctness and validity, and then uses APIs and custom code as it programmatically configures an instance of the desired DL software that matches the provided specification. Our process assumes the required software for the system has been installed (for example, DSpace requires Apache Tomcat and PostgreSQL) but our generation process takes the source code for the DL software system and along with a user provided XML DL specification outputs a working DL in the given software system.

Following results from work with 5SL, we believe that this separation of the nature of DLs will both ease the specification process and aid in understanding of the target software systems and digital libraries in general. From an architectural standpoint, we follow a similar pattern of design as the separation of concerns present in the XML based specifications for DLs, derived from 5S and 5SL. We believe the extensible nature of our tool will be useful, allowing it to possibly be used as the shell for future generators.

1.6. Research Contributions

Building on past explorations in the area, this work provides the following contributions:

1. a fresh application of the 5S framework for digital libraries to DSpace, one of the most commonly used digital library software packages currently available;

2. development of a 5SL DL metamodel for DSpace based on the most commonly used aspects of the software;
3. an extensible generating tool, dlGen, that provides automatic configuration and generation for DL specifications; and
4. a look towards what is required for creating a more general digital library model for specification that would be applicable to any digital library system.

1.7. Outline

- Chapter 1 serves as an introduction and lays out the goals of this work and the organization of the thesis.
- Chapter 2 discusses related work in digital libraries and digital library generation that we draw upon in this work.
- Chapter 3 discusses DL specifications.
- Chapter 4 discusses the DL generator tool.
- Chapter 5 describes a case study using our generation technique.
- Chapter 6 presents an examination of the dlGen work, and helps to analyze its contributions and shortcomings.
- Chapter 7 concludes this work, offering final thoughts, and suggests some future research in this area.

- Appendix A provides the full XML schemas for DSpace that our specifications and generator use.
- Appendix B provides the full schema created for our CITIDEL case study.

Chapter 2. Related Work

In this chapter we examine the nature of digital libraries and related technologies in greater depth as well as give examples of a few digital library software platforms to facilitate a better understanding of the types of content and structure that may be desired for automatic generation. We also discuss some past work in digital library generation and other works that are significant as well as related to our efforts.

2.1. Digital Library Definitions

Many definitions for the term "digital library" exist, and vary greatly, as do the conceptions and applications of relevant technologies. As we mentioned, the field is involved with a wide array of technologies, many of which are relatively cutting edge or novel in nature. A 2001 President's Information Technology Advisory Committee report describes the essence of these new digital libraries, saying they "will have features not possible in traditional libraries, thereby extending the concept of library far beyond physical boundaries [4]." Since one of the key aims of many digital library systems is preserving data over time and acting as archival solutions, the integrity and persistence of digital library systems is of key importance in many descriptions of DLs [5].

Marchionini and Fox succinctly show the differences between traditional and digital libraries—libraries connect people and information whereas digital libraries also may help strengthen and build on those connections [6]. Access and organization also play a big role. In traditional libraries, to find a collection of books that include information on a given arbitrary topic, we are limited by the categories that had been assigned to that book such as short summaries, or by characteristics of card catalogues. With the help of full text search technology, in digital libraries many works can be searched digital cover to digital cover, allowing for a much more complete knowledge gathering experience.

Leiner defines a digital library as [7]:

- the collection of services
- and the collection of information objects
- that support users in dealing with information objects
- and the organization and presentation of those objects
- available directly or indirectly

- via electronic/digital means.

Like Leiner's importance placed on the data storied in libraries, many researchers stress the importance of data and the ability to examine and represent data in multiple ways [7]. While in traditional libraries patrons are taught to be silent to let everyone work, in digital libraries the concept of community is vitally important and seen as interactions among readers, annotations on works, as well as collaboration [7].

In their 5S framework for digital libraries, Fox and Gonçalves describe DLs in terms of the fundamental building blocks that are associated with such repositories, into streams, structures, spaces, scenarios, and societies [8]. We examine their 5S framework in more depth in Section 2.4 due to its fundamental relevance to digital library generation and our technique.

The European Union's DELOS Network of Excellence on Digital Libraries DL Manifesto [9] also comments on the overloading of the term "digital library" given the field's relation to countless research areas such as data management, information retrieval, library science, document management, information systems, the Web, image processing, artificial intelligence, human computer interaction, and others. They provide additional input regarding the nature of such repositories calling them, "a tool at the center of intellectual activity having no logical, conceptual, physical, temporal, or personal borders or barriers on information." [9]

2.2. Digital Library Technology and Concepts

The field of digital libraries is a wide area of research in which many technologies and concerns have come into play when dealing with such repository systems. There is a set of concepts that are fundamental to DLs that are critical to understand much of the past research and work in this area. Among these, we highlight the concepts of metadata, web services, XML, and XML schema, to give a good foundation for the rest of this document.

2.2.1. Metadata

As we have discussed, stored within digital libraries are digital objects, often representing information such as PDF files, Word documents, images, really any sort of

data. While those documents may be the actual data stored within a DL, there is still another set of data for every object that describes the essence of that object, access rights and other administrative data, as well as hierarchical data that helps represent how that object is related to others in a DL. This additional, descriptive information is known as metadata—descriptive information about the real data stored in DLs. The three types of metadata [10] commonly in a system are:

- Descriptive metadata
- Administrative metadata
- Structural metadata

Each of these types of metadata has its own purpose in describing digital objects and helps a digital library in giving further information, context, and organization about documents aside from the document content itself. Descriptive metadata serves to define the content of the digital object itself—who authored the content, what is the title, what is its creation date, and so on. While in the case of textual information like MS Word documents or PDF files this content may be extractable, that is not always easy; in any case searching, browsing, and organization is greatly improved by storing these pieces of information separately. Administrative metadata helps with the management of digital collections with regard to access rights, preservation information, as well as more technical information about an object like copyright or publisher information. Structural metadata is focused on the structure among digital objects as well as within digital objects such as chapters, pages, or other sub-document divisions. This type of metadata helps collection organization as well as inter-document structural understanding [10]. While metadata standards and elements serve as a guideline for the different metadata records that should be associated with information, by no means are these standards inflexible, strict representations—many are properly extended and changed using qualifiers: subcategories that better define exactly what type of information a certain element record stores.

2.2.1.1. Dublin Core

The Dublin Core metadata standard was born out of the Dublin Core Metadata Initiative, an organization dedicated to the promotion and adoption of interoperable

metadata standards. Dublin Core can supplement existing metadata and help with browsing, searching, and indexing of web-based information [11]. The standard deals with mostly descriptive metadata, but also has some elements that can have other applications as well. Dublin Core includes fifteen properties that can be used for description of digital objects and other resources. We provide the listing of the fifteen elements in Table 1 below, along with the official Dublin Core descriptions [11] of each element.

Table 1 - Dublin Core metadata format elements

Dublin Core Element	Description
Contributor	An entity responsible for making contributions to the resource
Coverage	The spatial or temporal topic of the resource, the spatial applicability of the resource, or the jurisdiction under which the resource is relevant
Creator	An entity primarily responsible for making the resource
Date	A point or period of time associated with an event in the lifecycle of the resource
Description	An account of the resource
Format	The file format, physical medium, or dimensions of the resource
Identifier	An unambiguous reference to the resource
Language	A language of the resource
Publisher	An entity responsible for making the resource available
Relation	A related sequence
Rights	Information about right held in and over the resource
Source	The resource from which the described resource is derived
Subject	The topic of the resource
Title	A name given to the resource
Type	The nature of genre of the resource

2.2.1.2. METS

Another major standard dealing with metadata, METS (Metadata Encoding and Transmission Standard), encompasses all types of metadata including descriptive, administrative, and structural metadata [12]. METS is specifically designed for representation of textual and image data in XML Schema (which we discuss in Section 2.2.3). METS documents have elements and sections that relate to most aspects of a digital object and, in fact, can be used to represent entire objects including referencing external metadata in standardized formats. A METS document consists of seven major sections [12], which we detail below for clarity and consistency:

Table 2 - METS metadata format sections

METS Section	Description
METS Header	Contains metadata about the METS document itself such as creator, editor, etc.
Descriptive Metadata	May point to descriptive metadata outside the METS document or internally embedded metadata or both.
Administrative Metadata	Provides information regarding how the files were created or stored, intellectual property rights, source information, etc. which can also be internal or external to the document
File Section	Lists all files containing content which comprise the electronic versions of the digital object.
Structural Map	The heart of a METS document. It outlines hierarchical structure for the object and links elements of the structure to content files.
Structural Links	Allows creators to record the existence of hyperlinks between nodes in the hierarchy outlined in the Structural Map.
Behavior	Can be used to associate executable behaviors with content in the object. Each behavior has an interface definition that represents an abstract definition of the set of behaviors represented by a particular behavior section.

2.2.2. DL Services

While the main identifying characteristic of digital libraries may be their storage of documents, without meaningful ways to access and interact with stored information there would be little interest or use of such systems. DLs provide services which are ways of viewing, retrieving, and putting stored data to real use. The most basic information retrieval services involved in DLs are browsing and searching. Each of these services has many different flavors and can vary greatly from being simplistic to very complex with the functionality they provide. Other services are more oriented towards user interaction and value added functionality such as recommendations of items of interest based on those previously viewed. Also, notification services are common that notify users of newly added content that matches certain characteristics a user finds worthwhile. Similarly, functionality within a DL that allows users to “save” items of interest for future use or organization is also found in many DLs as another service. In digital libraries that use components which make up entire systems, services can be represented elegantly as individual units which work together [13].

2.2.3. XML and XML Schema

XML, or Extensible Markup Language, is a markup language used to describe data. Like other markup languages, e.g., HTML, XML describes how other information should be perceived [14]. While HTML markup tags determine how content is displayed, the markup that occurs in XML describes information and the structures and hierarchical organization inherent in such data. Unlike in HTML where tags are predefined and affect the display of data in predefined ways (like bolding a block of text), the power of XML lies in the lack of predefinition of tags [14]. For clarity, we provide a simple example in Figure 1.

```
<Fruits>
        <Fruit>
                <Name> Apple </Name>
                <Color> Red </Color>
        </Fruit>
        <Fruit>
                <Name> Lemon </Name>
                <Color> Yellow </Color>
        </Fruit>

</Fruits>
```

Figure 1 - A simple XML example

Here, we are describing fruit, so we create our own tags to hold information about individual fruits, their attributes (names and colors), and a top level tag to store our collection or fruits. While this is just a simple example to illustrate the nature of XML, the hierarchical nature of XML allows the description of nearly any information in a succinct, easily understandable fashion.

Although XML is a useful, completely user customizable way to represent information, to make such technology even more powerful it is necessary to pair XML instances with some way of specifying valid XML files for a certain application. In other words, having a way to specify data is powerful, but a corresponding way to validate the correctness of a data specification for a certain application is equally needed. XML schema support just that—a way of declaring a structure of data within the context of XML without specifying the data itself [15]. A good way of thinking about XML schema and XML is the relationship between needs and items that fit those needs. For example, if an individual is interested in a new car, they may have certain requirements, e.g.,

something fast, blue, and made in the USA. There are innumerable cars that fit those requirements, but all will fit into the specification of what is needed for an item to be acceptable. If an XML schema is our requirements for what is valid, an appropriate XML document is what fits those defined requirements. Again for clarity, we provide a XML schema in Figure 2 that matches the XML document in Figure 1.

```
<xs:element name="Fruits">
     <xs:complexType>
          <xs:sequence>
               <xs:element name="Fruit" maxOccurs="unbounded">
                    <xs:complexType>
                         <xs:sequence>
                              <xs:element name="Name" type="xs:string"/>
                              <xs:element name="Color" type="xs:string"/>
                         </xs:sequence>
                    </xs:complexType>
               </xs:element>
          </xs:sequence>
     </xs:complexType>
</xs:element>
```

Figure 2 - XML Schema example

Here, we define the top level elements for individual fruit to be contained in, specify that there is no limit to the number of individual 'Fruit' elements that can be defined, but specify that each must have a 'Name' and 'Color' sub-element, each of which must be strings. Our XML example in Figure 1 adheres to this schema, because it has a top level element of the same name, and is made up of 'Fruit' elements that each have 'Name' and 'Color' elements of type string. Again, this example is simple but illustrates what can be possible with any amount or type of textual data. With XML schema, we can program a computer to know whether or not a structure of data and the contents of that structure are valid for a given applications, a powerful mechanism that allows many systems to verify data before operating on datasets.

XML and XML schema are critically important in countless aspects of the Internet, web services, and other web based technologies. XML greatly aids interoperability concerns, providing a means of transferring data from one representation to another. In keeping XML information in a textual format, it is almost certain to be readable on any system.

2.2.4. Web Services

With the continued development of the Internet and organizations and businesses harnessing the power of such a distributed medium for communication of information, web services have arisen to become standards in web-based communication. Web services are typically built upon XML, which has become the de facto standard in information representation. The simplicity, extensibility, and systematic nature of web services have driven the development of more distributed, componentized systems and allowed for many systems and websites to become more open and interoperable to third party applications [16]. In web services, small pieces of information encoded in special, defined ways are transmitted across the web as a mode of information exchange. In such a methodology, machines send such data, and others listen for incoming requests and provide meaningful responses as required. Common forms of web services are SOAP (Simple Object Access Protocol) and REST (Representational State Transfer services). Perhaps the most common research oriented web services platform is Apache Axis, which runs on Apache Tomcat Java servlet container. With respect to digital libraries, web services play an important role in the communication of distributed, componentized DL systems [17].

2.3. Digital Library Software

The definitions and descriptions of digital libraries serve to describe the essence of these software systems which provide storage of and access to digital content. Like any type of software available, there are different ways to write a software product, different aspects that receive greater attention given the developers and goals for the system. For example, image editing software is similar in nature with regard to what such programs allow users to do, but looks slightly different, highlighting certain features more than others; some packages provide special functions that others may not. No doubt these various, similar products are developed and coded differently.

In the area of digital libraries and DL software, there are two main types of methodologies that are often seen—monolithic and distributed componentized systems [9]. A monolithic system is one that is entirely self contained, and usually resides on one server, where the code base and system functionality is tightly woven and intertwined. In more distributed componentized systems, instead of one large application running in one

location, the software is split into multiple software components that may likely be placed on different servers. This approach allows for distribution of required pieces of software for a system to multiple servers, perhaps in geographically distributed locations. Such a division can lead to more parallel processing and often better performance due to the separation of needs.

In this section, we discuss a few common digital library software packages, both out of the box solutions and extensible frameworks that can be built on to develop distributed digital library systems.

2.3.1. DSpace

The DSpace system is a digital research repository system that was built to address a very real need among academic institutions—combating the problem of increasing amounts of scholarly work generated by faculty and students that was had sparse viewing and suffered from occasional preservation issues. The system, a joint venture between the Massachusetts Institute of Technology (MIT) and HP Labs, aims at providing an online, electronic repository system that stores, can provide organization and preservation services to scholarly work, to provide for broader exposure and a longer, if not infinite lifespan to such work [18].

Here we provide greater detail about the DSpace digital repository software. We include architectural diagrams that have been reprinted with the permission of the DSpace project to better illustrate the software's object model, architecture, and internal processes.

2.3.1.1. DSpace Overview

DSpace is a digital repository system that allows users to submit, store, and allow others to read and use information that may have broad appeal. With a specific focus on the preservation of stored data, DSpace employs digital preservation functions such as the storage of checksums along with digital objects in order to keep track of and verify a file's conformance with the original. DSpace is an open source software project and is entirely written in Java [19]. DSpace was created breadth first, so that most functionality required by organizations seeking to use such a software product was covered, in a

simple and basic way [18]. DSpace has a developed underlying model that drives the way that users use the system, submit and use content, and how administrators can organize and configure the system. The software's underlying code base provides APIs that administrators and third party applications can use to interact with the DSpace system. In order to be more usable to different types of users, the software provides a configurable submission and workflow process that can be fit to any organization's policies and practices [18].

2.3.1.2. DSpace Object Model

While the functionalities of any software are important regarding its use, to have a full understanding of a software system it is necessary to look beyond the appearance of that software and get a glimpse at what happens on the backend. DSpace is no different. From simply browsing a DSpace repository a user can get a feeling for the structure of a DL, while the exact organization and division of its underlying object model remains relatively hidden. DSpace administrators, on the other hand, get a taste of DSpace's underlying structure no matter how much the built in, web-based administrative interfaces are used.

DSpace stores digital content, often referred to as digital objects; thus a very important part of the overall object model of DSpace systems is those objects themselves, called "Items". Items are organized in a hierarchy in which similar items are grouped and submitted into Collections of similar content. The highest level of content organization in the system is Communities, which are groups of Collections—in the model Collections are completely distinct from Communities, but make them up. As such, a Collection can be in more than one Community. Each Item stored in a DSpace repository is made up of a bundle of bitstreams, so as many files can be stored in a single digital object as needed [20]. Bitstreams adhere to the Bitstream Formats that the system knows about, and DSpace behaves in different ways with different types of objects—e.g., images may have their thumbnails displayed when browsing the system but .exe files cannot. The DSpace object model diagram is provided in Figure 3.

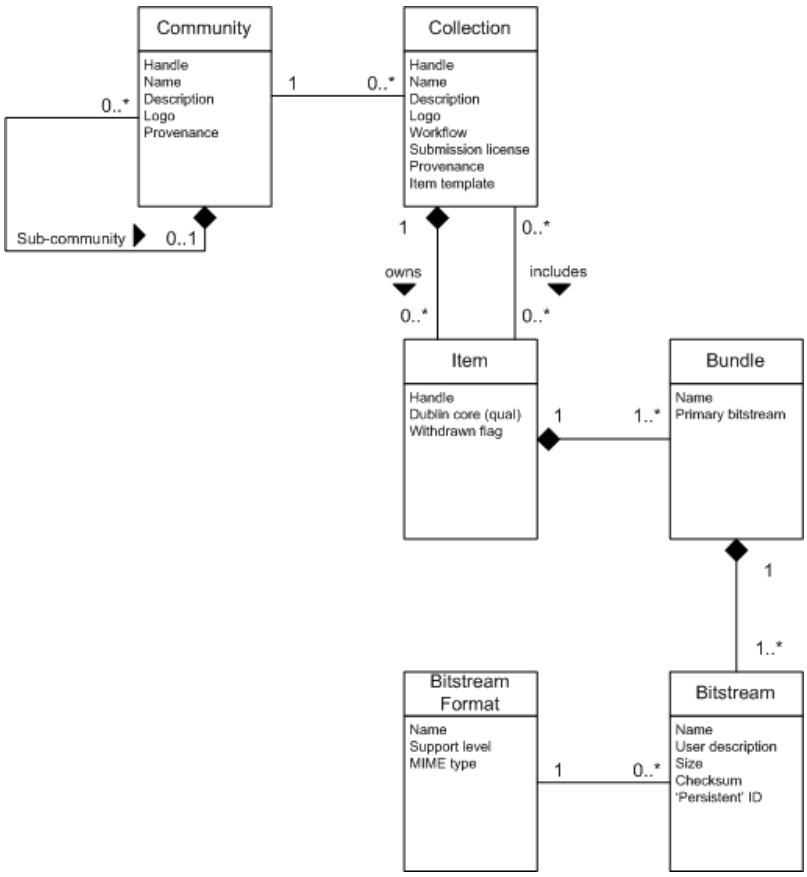

Figure 3 - DSpace object model [20]

2.3.1.3. DSpace Architecture

As detailed in Figure 4, the DSpace software is divided into a relatively common three-tiered architecture [18]. These three layers are the Application layer, Business Logic layer, and Storage layer. On the lowest layer of this architecture, the Storage layer, all bitstreams stored in the repository are stored as files on the system's file structure. References to these files and most other metadata, settings, and other information that drives the behavior of the system are stored in a relational database system, usually PostgreSQL. This marriage of techniques allows for the quick, relationally oriented access strategies for metadata and runtime data, while keeping stored documents in a regular file system. Together the Storage layer aspects of DSpace make up the Storage API. The Business Logic layer is made up of a set of classes or modules that embody the inner workings of many DSpace object types, including user related functions, browsing and searching related aspects, content management, and others. Business Logic classes

make up DSpace's Public API, which allows third party code to interact with DSpace in the same way that typical interaction within the software occurs. Lastly, the Application layer is the highest level layer of functionality in DSpace and brings together DSpace backend functionalities to provide the services and functionality that users see when they use the system. Included in the Application layer are the import/export functionality the software provides, statistics tools, and the web-based user interface. Given DSpace's open source nature, all of these software aspects have source code available to organizations using the system that can be tweaked and customized to more adequately meet their needs.

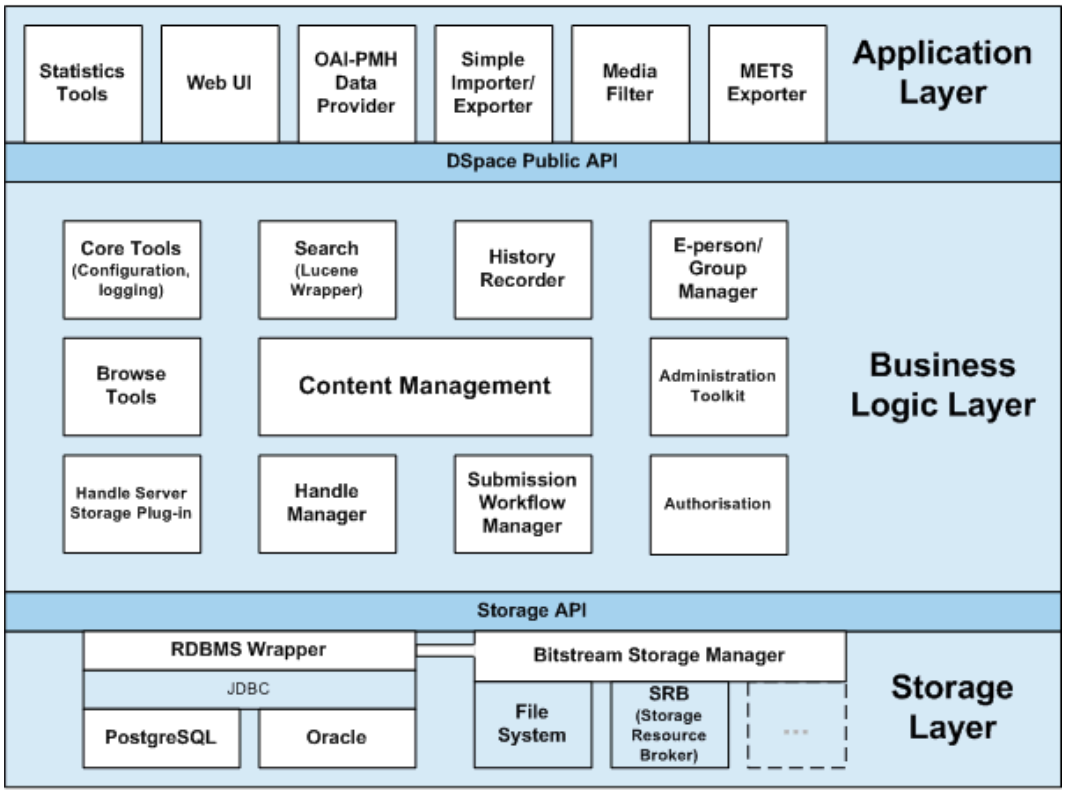

Figure 4 - DSpace architecture, split into Application, Business Logic, and Storage layers [20]

2.3.1.4. DSpace Ingestion

DSpace is a software system that serves as a repository which stores digital content. In a system with such a goal, perhaps the most critical aspect of the system is how that data enters the system. This occurs two main ways within DSpace. The web-based UI for the software allows a user to submit items to collections as long as they are logged in as a registered user. When users do such, they go through a configurable

workflow [21] where they upload and describe their submissions. (Workflows in DSpace are discussed further in Section 2.3.1.5.)

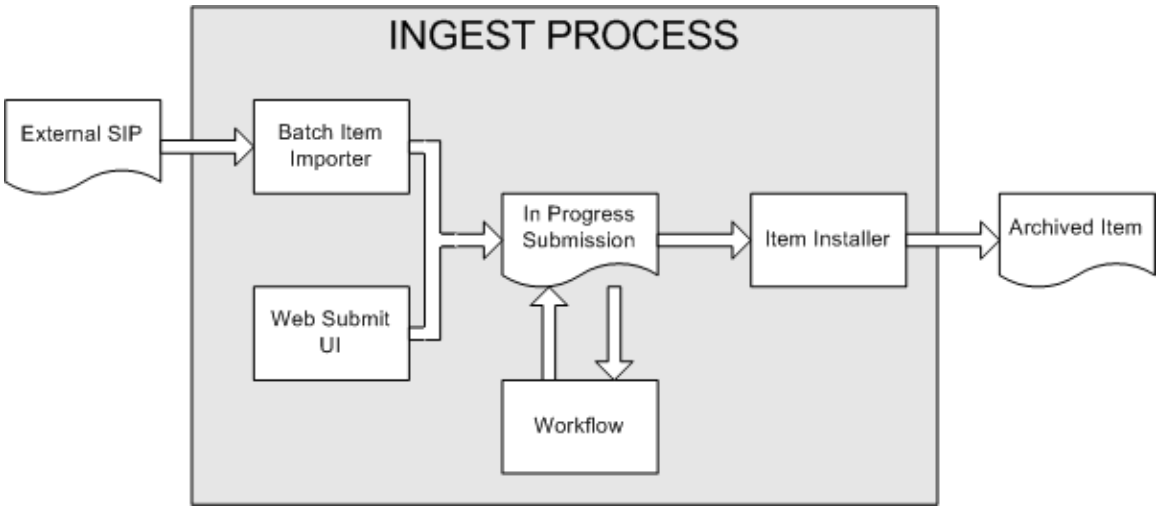

Figure 5 - DSpace ingest process diagram [20]

Alternatively, DSpace administrators who have a large amount of content to be batch imported may take advantage of the import/export functionalities of the system [20]. The Item Importer is a command line tool that comes bundled with the system and allows users to import collections of content into the system.

```
archive_directory/
    item_000/
        dublin_core.xml -- qualified Dublin Core metadata
        contents        -- text file containing one line per filename
        file_1.doc      -- files to be added as bitstreams to the item
        file_2.pdf
    item_001/
        dublin_core.xml
        contents
        file_1.png
        ...
```

Figure 6 - DSpace's simple archive format for importing and exporting [20]

The Item Importer uses DSpace's simple archive format, which is a simple directory structure that holds items for import into the system. (We provide an example of a simple

archive in Figure 6.) A top level archive directory contains uniquely named directories, each of which contains everything necessary to import a single item. Each sub-folder is required to contain two files, in addition to the actual content to be imported. The required file "dublin_core.xml" contains an XML representation of qualified Dublin Core element names and the textual content that those metadata records should contain, including author, title, and so on. A plain text "contents" file has one line containing the filename of each file that will be included in that digital object. Once this structure is put in place, the tool can simply be run and all content will be imported into the repository in question. The tool provides a "map file" after being run, which details all items that were imported and their new location within the system—this file can help with future exports or removal of groups of imported content [20].

2.3.1.5. DSpace Workflow

DSpace is one of the first open source repository systems to successfully combat the problems that lie in different requirements for submission of different types of information to different collections [18]. The DSpace submission workflow system is a critical part of the DSpace architecture that allows for the submission, processing, and final addition of content to the live repository. DSpace's underlying model includes EPeople, users who have registered with the system and have certain authorizations, roles, rights, and privileges that translate to abilities to complete certain tasks within the DSpace system. A typical submission begins with the system asking the user a couple of questions about the publication history of the item and the number of files involved in the submission. Then the system guides the user through the different steps of the process, which are outlined in Table 3.

Table 3 - DSpace submission workflow overview

Workflow Step	Description
1. Describe	User enters metadata about the document(s) they are submitting, including but not limited to authors, title, keywords, and a description.
2. Upload	The user selects and uploads the files on their local machine that they'd like to upload as part of the submission. Each file's type is identified by the system and the user verifies the file type.
3. Verify	An overview of all details of the submission are given including a summary of the entered metadata and the files involved in the submission.
4. License	The user is shown and must agree to the license the system administrator has assigned to submitted content for this collection.
5. Complete	The user's actions in the submission process are complete. Based on the workflow steps set for the collection, the item may immediately be added to the collection or have to be reviewed by system administrators before its addition to the collection.

2.3.1.6. Manakin for DSpace

DSpace is a very mature project. Since its inception in 2000 there have been many developments and iterations which have strengthened the overall feature set, interaction, and configurability of the software. Although relating to the separation from the JSP / Servlet differentiation of presentation vs. business logic layers, one of the greatest issues users of the software experience is a difficulty in branding, or creating a customized look and feel, in DSpace's web-based interface.

One project to combat this problem is what Texas A&M refers to as their DSpace XML UI project, the current second iteration of which is called Manakin [22]. Manakin acts as a layer above the traditional JSP presentation layer of DSpace and allows for each community or collection within the repository to have its own look and feel. The software splits up its realization into Aspects and Themes [22]. Aspects are different parts of a DSpace website that can be customized or toggled on and off in a DL instance, such as the login mechanism. Themes are the visual character of these different Aspects that make up DSpace websites. When combined, these two logical parts of Manakin work together to provide a more customizable and attractive web-based interface for DSpace. Manakin is an "add on" of sorts to DSpace, so before it can be used a working copy of DSpace must be installed and configured.

2.3.1.7. DSpace Future Roadmap

Many drawbacks of DSpace that have been identified by the user community in the past are things that are included to be addressed in the published roadmap for DSpace 2.0, which is beginning to be formulated and will likely have initial versions released in 1½ to 2 years [3]. The new design will have particular focus on making the system more scalable by addressing issues to allow greater capacity of items, improving the ingestion rate of objects in bulk add situations, and making the system's processing more concurrent. While some of the largest known DSpace repositories contain hundreds of thousands of digital objects, the goal of the new system is that the architecture should be able to store and support up to 10 million items [3]. There is desire to make the system more interoperable, and work in this area will include a much more concrete and descriptive object model and core interface, both of which would be clearly published in order to better facilitate interoperation for administrators and third parties. The item model will be revised to include versioning of items plus other changes. To make the nature of the software more plug and play oriented, an extension model is being investigated that would greatly ease the difficulty in creating and using components that can easily snap into a DSpace instance [3].

2.3.2. Greenstone

Greenstone is a suite of software for building and distributing digital library collections. The software, created by the New Zealand Digital Library project at the University of Waikato, is completely open source and is compiled C++ in nature [23]. Unlike some other software packages for DLs, Greenstone is specifically aimed at providing DL services to organizations, including those in developing nations. To further this aim, their software has been distributed on CD-ROM through UNESCO channels and organizations and can also create CD-ROM based DLs which include built in web servers and everything needed to run the DLs created [23]. The software is highly interoperable, supports importing from and exporting to DSpace collections, and supports multiple metadata schemas including a couple of New Zealand metadata standards.

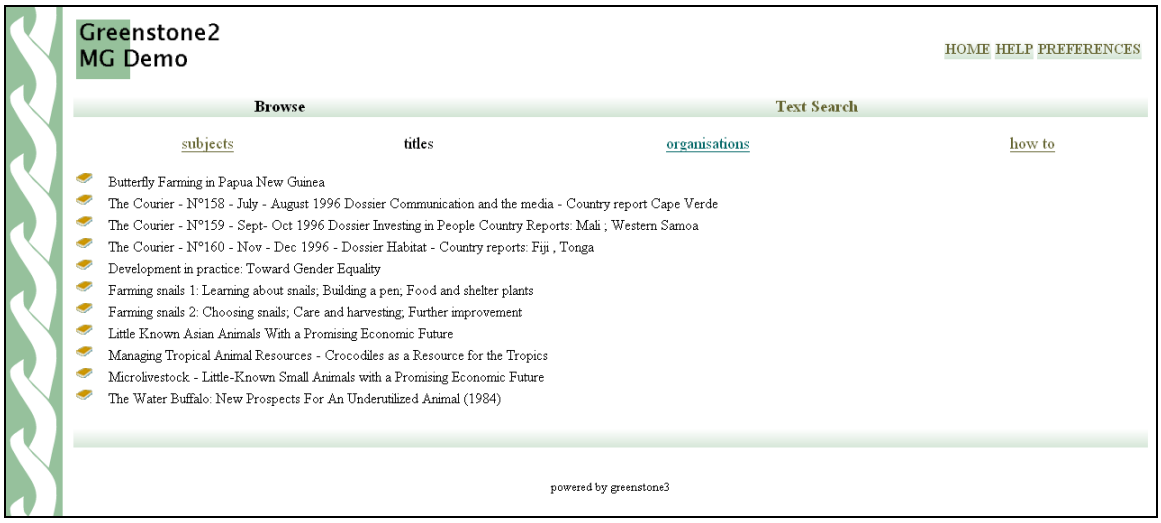

Figure 7 - Screen shot of Greenstone sample collection browse interface, browsing by titles

Like most digital library systems, Greenstone has facilities in place that allow the searching and browsing of documents in collections. Although it can hold virtually any type of document, Greenstone is specifically oriented towards textual information, and can support and use divisions such as sections and chapters in text while searching and browsing for information [23]. Unlike some systems that require user interaction to identify metadata information for documents, Greenstone has a plug-in based architecture that provides automatic metadata extraction and processing for various document types. While plug-ins for most popular document formats come paired with Greenstone, they can also be authored by those with some programming experience.

Greenstone 3, a completely redesigned version of the software, has recently come into beta after years in the making. Motivated by greater customization and modularity in a distributed, service-based architecture, Greenstone 3 adds greater, more flexible functionality while remaining backward compatible [24]. This new generation of the Greenstone software has been completely rewritten in Java, abandoning the C++ construction previously used. Also, to allow for distributed DL systems the new software uses Apache Axis' SOAP (Simple Object Access Protocol) that helps to allow services and content from separate and geographically distributed servers [24].

2.3.3. Fedora

The Fedora project (Flexible Extensible Digital Object Repository Architecture) is perhaps one of the largest digital library efforts currently in development that focuses on a componentized framework for allowing the creation of distributed digital libraries. Born from a 1990 DARPA project that first defined the notion of digital object, the Fedora project is a continuing research effort to develop a rich framework for the creation, management, and preservation of digital content [25]. Very different from efforts like DSpace and Greenstone that provide a monolithic, installable digital library system that has great functionality out of the box, Fedora serves as a framework that can facilitate the building of DL systems. Out of the box, Fedora provides resource storage, dissemination, and a series of web service based facilities to help accommodate such behaviors. All of Fedora's APIs are defined using the Web Service Description Language (WSDL) for added extensibility and interoperability. Fedora follows the increasing use of web services in business and research efforts alike [16] in using web services to provide greater flexibility with regard to geographic location and network structure.

Unlike some other DL projects, Fedora's object model is flexible enough to support a plethora of structures, hierarchies, complexities and relationships among digital objects. The Fedora object model can be understood or examined in two different abstractions—representational and functional [25]. The representational perspective views the object itself as a black box, abstracting the actual meat and bones within that object. The focus in that case is on the disseminations, the different ways that an object can represent itself or different means of presentation of the nature and data that is stored in a Fedora complex digital object. The functional perspective that Fedora objects can also be seen in is much lower level, paying much greater attention to the different data elements, functionality, etc.

2.3.4. Componentized Research Efforts

Unlike the software systems created by large groups of individuals that are widely used publicly, there is much work that occurs in smaller research efforts that try to push the state of the art in digital libraries. Due to the direction digital library technology is headed, most of these efforts are componentized efforts that attempt to separate DL

aspects into pieces that may be run separately and work together to build a DL system. Fedora could be included in this section, but we separate it given its maturity and amount of use. Here, we describe a few of these more research oriented, component based systems.

2.3.4.1. Open Digital Libraries

Work on Open Digital Libraries (ODL) centers on the need for digital library system architecture to continue to evolve given the constant flux of related fields such as library science and network architectures [26]. Using developed standards like Dublin Core metadata and the Open Archives Initiative's Protocol for Metadata Harvesting (OAI-PMH), Suleman and Fox develop a framework that allows for the composition of DLs using a combination of local and distributed content joined together to make a single DL system [26].

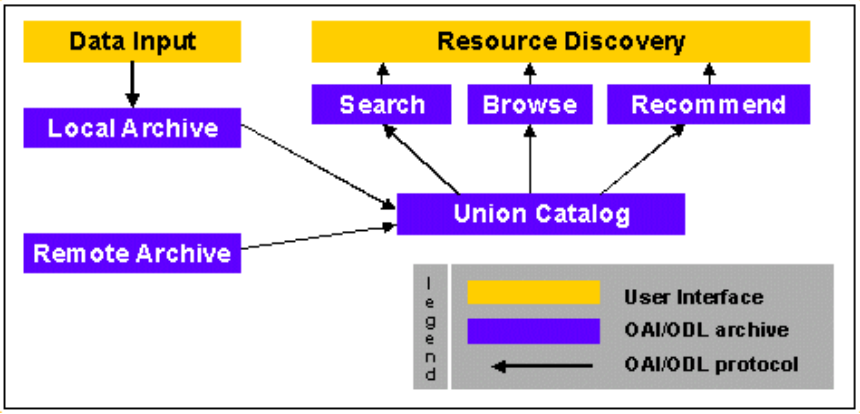

Figure 8 - Example of network Open Digital Library [26]

In ODL, a componentized strategy is used to build a digital library system. It suggests the concept of a Union Catalog, a component which brings together sets of metadata from multiple sources and appears to the digital library as if only one set of objects is present [26]. Based on this catalog, that body of information drives components that provide typical DL services such as searching, browsing, and recommendation. These components are envisioned to be self-contained so that they may be added or removed arbitrarily in creating DL systems.

In addition to the ODL framework, related work created the Blox DL creation tool [27]. Blox uses a visual interface that represents different components in componentized DLs, targeted to ODL components but supporting others as well, in order to allow users with little or no DL or Unix experience to create digital libraries [27]. The technique uses client server architecture over SMTP for DL creation in Windows and away from the server actually housing the components.

2.3.4.2. WS-ODL

Given the freshness and success of ideas proposed in Suleman's work with ODL [26], that work was later extended to make use of web services to build WS-ODL (Web Services Open Digital Libraries) [28]. WS-ODL operates on top of the Fedora Architecture, which includes Apache Tomcat and Apache Axis web services for communication among components. The WS-ODL framework includes a data repository based on web services to provide DL services, and a client layer that provides a user interface and acts as a mediator between the components that make up the DL. Using this methodology, WS-ODL provides a variety of components including those for searching, browsing, recent works, annotation, registration, and administrative tasks [28]. Through evaluation they show that such an architecture can be used to create a distributed digital library and can outperform the original ODL work.

Like many digital library systems, the installation and setup procedures to the WS-ODL work are not simple. To combat these difficulties a wizard tools was created to streamline the process of configuring DLs created using WS-ODL [29]. Using the Model-View-Controller architecture (MVC), the tool uses a graphical interface to allow users to step through the different steps of DL configuration.

2.3.5. Digital Library Software Comparison and User Evaluations

With such a body of work in digital library systems, both production and research efforts alike, these systems are not only likely to get much use from the worldwide user community, there are sure to be many opinions of all systems. Here we provide some brief comparisons and views from the user community of production systems like DSpace, Greenstone, and Fedora.

2.3.5.1. General Digital Library Software Needs and Reflections

One of the largest barriers to a system's widespread acceptance is the wide array of organizations who wish to create, publish, and maintain digital content. Nevertheless, the ways they want to use and present that data are even more dissimilar. In a recent DSpace roadmap document, authors admit that there may be no perfect solution that fits all users needs [3]. DL systems that provide out-of-the-box functionality can never meet all the needs of all users; they are just not that feasible. This is why open source, expandable, and customizable solutions are so important in the field of digital libraries.

Some groups have done work in assessing digital library systems and how DL systems fulfill users' needs. One group defined nine factors leading to users' acceptance of DL systems including terminology used, the screen design, navigation systems, relevance to users' needs, system accessibility and visibility, usability to new users, the overall experience of using a system, and its relevance to the users' domain knowledge [30]. Other research has said that users' views of digital libraries still lie largely within the original conception of a traditional library, and that actions in a digital library should, at least on some level, mirror actions and experiences seen in traditional libraries [31].

2.3.5.2. DSpace Reflections

Especially with large production grade digital library implementations the number of digital objects in a system may reach the hundreds of thousands or millions. With size and technological stresses that such a large number of digital objects can put on a system, system performance becomes an important aspect of a system's attractiveness or lack thereof. There have been repetitive complaints of DSpace's performance when large numbers of objects are in a live system [3]. The DSpace project's relationship with HP has led them to suggest a number of HP branded systems that meet the needs of different levels of DSpace repositories [32]. The lowest level system, their "entry-level" suggestion, has a hefty $40,000 price tag—likely more money than many if not most of DSpace's users would like to put into their system. Performance is a typical bottleneck in monolithic systems that share common architecture for a multitude of related services required to build a digital library system. Those performance problems explain some of

the benefits of distributed systems like Fedora, where the required servers and systems can be distributed to provide a separation of required processing over multiple machines.

In other evaluations, the most widely cited plusses of the software were its internationalization ability, security, and especially the user community that provides help with troubleshooting and other technical issues. The greatest disadvantages commonly discussed are the ease of deployment, ease of working with the code base, and the ease of system administration [33]. Other comments were made concerning DSpace's lack of personalized collections and digital rights management integration, though these aspects are not quite as common in research DLs as with more commercial ones [34].

2.3.5.3. Greenstone Reflections

Greenstone impressions are less well represented in the literature despite the maturity and wide use of the software. Users indicated that one of the strengths of Greenstone is its documentation and manuals, as well as a supporting book [35], which are also the most commonly used forms of support and learning materials for the software, in addition to email assistance or website use [2]. Despite the strong documentation, some individuals note that written information is too technical in nature, and even more in-depth documentation is needed. A surprising number of Greenstone collections are published to CD-ROM, according to a recent survey which indicated that nearly one-third of organizations use this publication method. The software is widely used and many collections have been created internationally that showcase its usefulness and ease of use [36]. Few negatives are given for the Greenstone software, but that also may be a function of the limited sources of user opinions.

2.3.5.4. Fedora Reflections

According to the literature and reports of user impressions, Fedora is consistently rated as one of the top solutions for the creation of digital library systems [33]. Perhaps due to its nature as an extensible framework used to create DLs instead of a pre-built software system that works out-of-the-box, Fedora serves as a good starting point to create many DL systems that meet users' needs. The best reviewed aspects of the software tend to be interoperability, support for working with the code base, and ease of

deployment [33]. Lowest marks on the software go to the strength of the community knowledge base, which is relatively limited for Fedora.

2.4. 5S, 5SL, and Related Tools

Given all the elements, concerns, and complexities that are involved in digital libraries, there have been many efforts to more formally describe the nature of such repositories with regard to their structure, organization, user concerns, scenarios, etc. Gonçalves, Fox, et. al. devised the 5S framework for digital libraries, a complex, formal representation, initially of the elements that are basic to any minimal digital library system.

2.4.1. 5S Model

Digital libraries are immensely complex systems which allow information to be stored in an intelligent, usable, and easily retrievable fashion. Due to their complexity, and the relative maturity of the field, although there has much experimentation and implementation of digital library systems, there has been little theoretical basis for design and implementation. In order to address the complexity of digital libraries, Gonçalves, Fox, et. al. proposed the 5S Model for Digital Libraries [8]. This model serves as a basis to understanding and classifying digital libraries by addressing the many interdisciplinary components to such systems, including aspects of system architecture, actors and societal implications, as well as interface elements. According to the model, the nature of DLs can be described using five types, or S's, which are Streams, Structures, Spaces, Scenarios, and Societies [8]. When proposing the model, the authors lay out a taxonomy of terms related to DLs and where they fit into the 5S classification scheme but are quick to point out that the field will change over time and thus such a taxonomy must also evolve as the climate of the field changes. To bring further validity to the model a set of formalisms are provided which solidify the 5S's and related terms.

This work is important because it defines a set of constraints and way of thinking about digital libraries which brings some order to the chaos of different research and commercial digital library work that occurred in the infancy of the field. Having a solid foundation of the nature of digital libraries, or any breed of system, is the first step to

categorically defining the essence of what is to be programmatically created. While different groups may come up with different theoretical models for DLs, once a concrete way to describe systems is developed they are able to be described and interacted with universally with greater clarity.

2.4.1.1. Streams

Streams [8] are sequences of data of any type, for example bits, bytes, images, or video data, that can represent either static documents or dynamic real time data such as a patient's heart rate or other vitals. Though each separate stream may have a different nature, subject matter, or type, they all share the quality of storing/providing information. Given the importance of the Internet and digital libraries' tight relationship with it, the validity of a basic concept of streams within DLs is clear.

2.4.1.2. Structures

Structures [8] refer to the fundamental organization that is seen through digital libraries, and any electronic system. Examples include structures that represent markup languages such as XML and HTML, taxonomies, and user relationships. Structures can refer to the indexing structure used to organize and quickly find documents or parts of documents in modern information retrieval systems like databases or text indexing. Overall, anything that is related to the structure of a digital library belongs to this category.

2.4.1.3. Spaces

The authors define Spaces [8] as "a set of objects together with operations on those objects that obey certain constraints". Spaces can vary greatly from being very real, such as windows, views, and projections of data, to being very abstract such as the definitions of spaces for information, including what constraints apply to the information, as well as to ensure relevance. Spaces refer to physical locations, user interface layout and behavior, as well as the underlying details of information retrieval methods (e.g., using distance in a space to estimate likelihood of relevance).

2.4.1.4. Scenarios

Scenarios [8] are ways of using a given system, the sum total of which can help to define the overall functionality required of or provided by a system. Scenarios are perhaps the single best representation of the functionalities that are coupled with the data and infrastructure needed by a system, and can help provide that information to developers for an excellent picture of a system. Different states and events that occur in a system throughout user interaction in scenarios are important to the nature of scenarios, as more than one scenario can arrive at a common state, influencing the development, code structure, and reusability of code or components/classes. The flow of data within a system of workflow in submissions to a system can also be easily modeled with scenarios.

2.4.1.5. Societies

A Society [8] is "a set of entities and the relationships between them" and can include both human users of a system as well as automatic software entities which have a certain role in system operation. The relationships between users and computer entities are important, as the users themselves. For instance, a system may allow a group of individuals to sign up together, but the group leader may have different permissions or abilities from other regular users. Thus, though all the users are related, i.e., they are from the same group, nevertheless the leader has additional rights, to perform administrative functions on other user accounts. Such a group of users could be considered a society, just as the entire user base of a system could be a society. Societies may define mutual interest in a topic, similarities in real world group membership, or similar characteristics involving software components of a system. Society membership also may define information access rights regarding the use of a system.

2.4.2. 5SL

Given the complexity of today's digital libraries, the organization of DLs proposed in the 5S model is a needed step in describing the architecture and qualities of a digital library. 5SL [37], a language for declarative specification and generation of digital libraries, enables DL designers to harness the power of 5S for high-level specification of

digital libraries in the 5S areas—streams, structures, spaces, scenarios, and societies. Using the model set forth in 5S, 5SL can assist specification of how content is stored; how that content is organized, structured, described and accessed; which services are offered by the library; and how users use those services. According to the seminal work on 5SL, Gonçalves and Fox attribute the lack of encompassing design patterns for DLs to the homegrown nature of DL systems [37]. Smaller pieces of systems are envisioned and developed as needed and bigger plans for systems are far less often utilized. 5SL provides the tools to do just this, and enables DL designers to raise the level of abstraction and modeling available to plan and design their online repository systems.

Table 4 - 5S Framework Model overview [37]

Models	Primitives	Formalisms	Objectives
Stream Model	Text; video; audio; picture; software program	Sequences; types	Describes properties of the DL content such as encoding and language for textual material or particular forms of multimedia data
Structural Model	Collection; catalog; hypertext; document; metadata; organization tools	Graphs; nodes; links; labels;	Specifies organizational aspects of the DL content
Spatial Model	User interface; index; retrieval model	Sets; operations; spaces; vector space; measure space; probability space	Defines logical and presentational views of several DL components
Scenarios Model	Service; event; condition; action	Sequence diagrams; collaboration diagrams	Details the behavior of DL services
Societies Model	Community; managers; actors; classes; relationships; attributes; operations	Object-oriented modeling constructs; design patterns	Defines managers, responsible for running DL services; actors, that use those services; and relationships among them

5SL is an XML based language; the widespread usage of XML in the field of computing helps to make the format more understandable and usable. Different past research ([38], [39]) endeavors have produced work that aligns nicely with some of the perspectives that 5S provides and 5SL uses. Within 5SL, MIME types are used to specify the types for file types, XML or RDF Schemas are used to define structure, and UIML (User Interface Markup Language) [40] defines user interfaces and other aspects of DL

design. Societies and Scenarios are defined by UXF [41], an XML version of UML (User Modeling Language) [42].

5SL is only a language to define digital libraries which fit inside the constraints allowed; while generation of digital libraries can make use of 5SL in the process, generation itself requires added tools.

2.4.3. 5SGraph

5SGraph is a domain specific digital library modeling tool that was conceived in the original work with 5SL as a helper tool when building digital libraries. One issue that is seen in the use of 5SL was that at times specifying digital libraries in 5SL was time consuming and could get tedious when working in the native textual sense in which 5SL is envisioned. To try and lessen this burden, 5SGraph offers a visual way to compose DLs and their related pieces, e.g., covering the sub-models for streams, structures, spaces, scenarios, and societies [43]. Using the tool helps a DL designer to walk through the process of specifying a digital library, through the different relevant aspects of 5S, and provides increased usability not only in specifying such a model but also in interacting with the application itself, that operates as a standard Windows/Java visual application. In experimentation with users creating DL models with 5SGraph, not only were users able to complete specification tasks in record time, users also reported a greater understanding of the 5S theory of digital libraries after using a tool in which those concepts are so essential and critical [43].

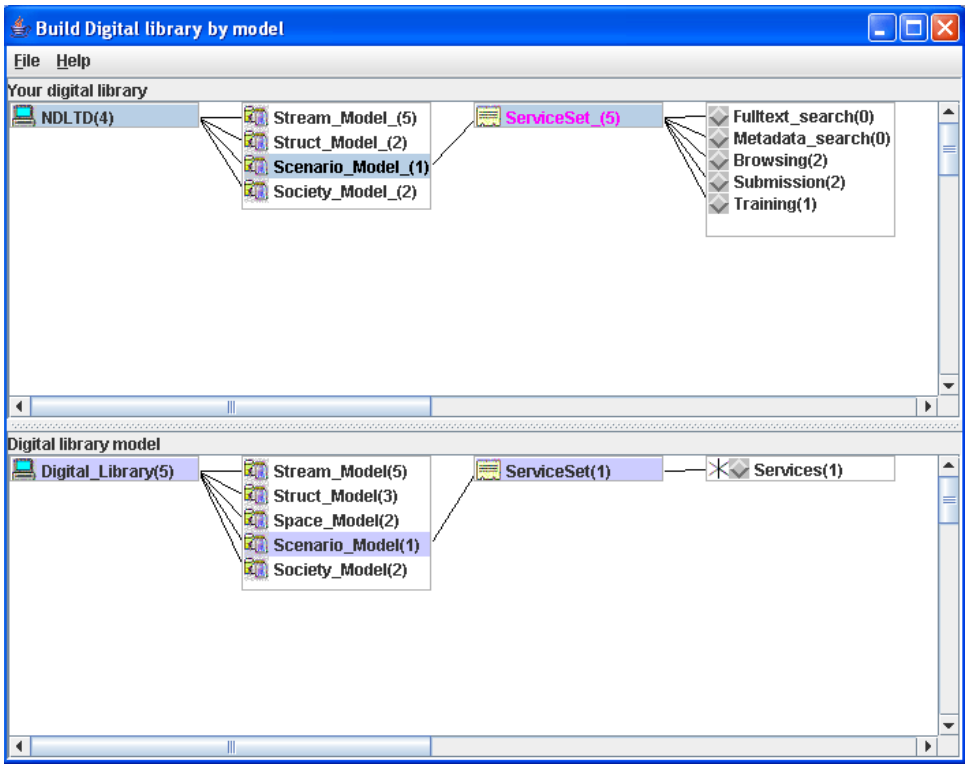

Figure 9 - 5SGraph DL modeling tool

2.4.4. 5SGen

The authors of the work with 5SL provide proof of concept of the DL definition language by pairing it with 5SGen, a tool which accepts a digital library defined in 5SL and using a pool of components, generates much of the code necessary for a DL implementation—to handle the described data, structure, and details [37]. The generator is service-oriented and takes advantage of the scenario aspects of 5SL to allow users to specify services at a very low level and have those services created in the resulting classes. The design also handles workflows in order to give a more tailored control over DL creation [44]. The generator was used in a pilot study to generate components in the MARIAN Digital Library, developed at Virginia Tech, and generated a set of class managers, indexing classes, an analyzer, and user interfaces—in total about 1600 lines of code. The code is aimed to be reusable and extensible in nature so that it can be built on in the future to provide greater functionality. The authors provide multiple case studies ([44], [45]) to demonstrate the results of their work.

Chapter 3. Digital Library Specification

The first step in the programmatic generation of digital library systems is a mechanism for the specification of what is to be generated. Based on the most commonly used and configured aspects of the DSpace digital library software we develop an XML-based specification language for modern, practical digital library systems drawing upon existing work in this area. Our approach allows for the specification of DSpace digital library instances using XML. In order to create a more meaningful way for users to specify their desired DL systems, we split the concerns of DLs according to the 5S framework for digital libraries.

3.1. Overview

Our technique for digital library specification borrows much from the 5S and 5SL work done at Virginia Tech by Gonçalves and Fox. We adapt their XML-based generic DL specification language to a more specific purpose—modeling DSpace instances. While originally their model was very complete, the abstraction was at a level that was more suited to theoretically describe a DL. In this work, we move towards a practical DL metamodel [46]. Such a model can be applied directly to modern, powerful DL systems and describe their nature, structure, and functionality. Using their original specification language as a guide, we pick and choose aspects of that language that have relevance to DSpace's configurable behavior. While DSpace is a broad, open source project that has much built-in capability as well as the obvious manipulation and customization that can be achieved by editing the source code, addressing and enabling our generation technique to allow all possible code changes is out of the scope of this work. Thus, we focus on only the behaviors and customizable aspects of the software, in our specification and generation techniques.

Like in the original 5SL work, we create our specification for DSpace digital libraries using XML. We use XML schema in order to verify the validity of XML DL specifications with regard to existence of required elements as well as type checking of specification elements. Like in the original 5SL [37], we heavily rely on the separation of

concerns of a digital library into the 5S's that 5SL is based on—Streams, Structures, Spaces, Scenarios, and Societies.

3.2. 5S and DSpace

In order to continue to use a 5S driven organization and separation of the concerns of a digital library, it is necessary to examine the DSpace functionality and architecture in the context of the 5 S's. Thus, we decompose the functionality, structure, and services of DSpace into the aspects that the 5S framework suggests.

The 5S framework is intelligently laid out to segment the different aspects of digital libraries—including DSpace. In Table 5, we detail how the different parts of DSpace fall into the 5S framework.

Table 5 - 5S model aspects of DSpace

5S Model	DSpace Objects
Stream	Bitstreams, Known Bitstream Formats
Structure	Collections, communities, authorizations for them
Space	UI: Interface issues like logos, text blocks, and items to show when browsing. IR: indexing fields
Scenario	Workflow, toggle features like RSS
Society	Admin Users, regular users, groups

3.2.1. Streams in DSpace

In DSpace, all files are considered bitstreams—they represent structured data that is usable within the context of digital objects that are stored within DSpace. All documents that are associated with digital objects, for example JPEG files for image digital objects, or PDF documents for electronic theses, are considered bitstreams to the DSpace system. However, for our specification and generation we are not interested in bitstreams in that context. DSpace's Bitstream Format Registry is a registry of all of the formats that DSpace knows about and can identify as a certain type of content. When a user submits a document to DSpace, the system identifies their submission as a certain

sort of file which gets recorded with the submission. This becomes important for files that can be indexed, like text files or PDF submissions, so that a file can be included in searches as appropriate. While DSpace includes the most commonly used file types, in many applications individuals may want to create repositories of customized file types. DSpace understanding these files as their actual types can greatly increase the usefulness of certain aspects of the system.

3.2.2. Structures in DSpace

The main aspect of structure with regard to 5S is of course the overall organization and hierarchical layout of the digital library and all the documents and groupings of documents that exist in each DL. Within DSpace, the main organizational structure involves Communities and Collections. Communities are targeted towards types of users or groups of users with similar interests. Initially they were modeled around the sort of organization that most institutions of higher learning have—departments with separate interests, say geology vs. computer science vs. literature, and gathering together all the collections of documents that are of interest to those groups of people. Communities serve only to group together collections for different audiences, while collections are the groupings that actually hold digital objects. Also, collections need not only be in one community; a given collection can exist in one or more different communities for instances where there is overlap of interests between various communities of users.

For the specification of DL structure within our specification method, we keep this hierarchy of collections and communities intact. The specification of each community and collection holds details that are necessary for DSpace to have, that provides structure for its running instance, including collection names, subjects, and other text fields that are used throughout DSpace's interface. Collection and community logos are also specifiable in the same manner.

Although perhaps not as it was originally intended in 5S or the original 5SL, we have also included authorization related policies in this section of our specification language. We felt authorization policies, that detailed which users or groups could perform certain actions within specific collections, fit best into the structural category. By

including these policies within the specification of the collections they describe and relate to, we can alleviate some of the typographical errors or inconsistencies that can occur when separate, distinct parts of a specification relate to one another. In DSpace, users are part of authorization groups. Every aspect of structure, collections, communities, even individual digital objects, can be assigned different authorizations based on groups of users. We automate the assignment of authorizations to the collection level of this structure. Community level authorization policies only really apply to the ability to list collections they contain, so we ignore that level of authorization for our purposes. Similarly, items can have authorizations attached to them about which groups of users may view or complete other actions on individual items. While some users may be interested in doing this, assigning individual access policies on items, it is functionality that an administrator will probably manually install into DSpace if they are interested in that kind of control. We focus on only the collection aspect of structure in our specification, as it is a medium sort of granularity of authorization that is most commonly used if creating collections through DSpace's web based administration interface.

3.2.3. Space in DSpace

The two main areas we cover in the Space model are user interface and information retrieval related issues. Since DSpace is a web-based system, there are many user interfaces which fall into the category of spaces in one way or another. DSpace includes an implementation of Apache's Lucene search engine technology for indexing and searching over documents in the digital library, which fit nicely into the information retrieval facet of 5S's Space model.

Like most web-based systems, the DSpace interfaces are numerous and occasionally relatively complex. Allowing full control of specification over all of these pages would be an arduous task, in its own right requiring wrk comparable to that needed for a thesis. As such, we have looked at the most easily configurable and most useful modes of configuration and specification of interface elements for DSpace. Perhaps the simplest and most useful aspect of DSpace's facilities for UI changes is the use of CSS for the styles used throughout the web pages. The types of metadata that are displayed while users are browsing and searching for items also can be easily configured, and we

include that functionality as well. In addition, overall logos that are used to personalize a DSpace instance can be relatively easily changed as well as the textual name of the repository, “PhotoSpace” for example.

The most prevalent information retrieval issue that is seen in DSpace that is not hard coded within its code base is the indexing options and details that drive the searching behavior of DSpace. Manipulating these options helps to determine what gets retrieved when DSpace searches are made, as well as which metadata fields are indexed to make searching possible.

3.2.4. Scenario in DSpace

Scenarios are different from much that we have discussed with regard to specification thus far since they are not simple, identifiable elements that can be seen in the interfaces or architecture of a software system. Scenarios show up in multitudes in many software systems. In DSpace, anything from an administrator adding a new authorization policy to allow a normal user access to a community to a user searching for records that contain specific words can be considered a scenario. While the Scenario model is immensely important when describing the nature of a digital library system, in a monolithic system such as DSpace a certain set of functionality is included out of the box. Since the software is open source, individuals and organizations can change or add functionality as needed. Also, there is no easy way within the software to “turn off” certain services while leaving others. Without getting dirty in the DSpace code base and making changes to several files that work with one another, the functionality that DSpace provides out of the box is rather set—you get what is there.

One instance when this is perhaps not the case is in managing the submission workflow. DSpace does have a configurable workflow system [21] for users to submit materials to the repository where documents can be routed to individuals who will verify submissions and accept them for inclusion in the DL. Out of the box, DSpace provides for minor changes to this workflow, while there has been work to extend that process into a much more open and extensible architecture.

Another aspect of DSpace that can be easily turned on and off is a rather simple one—the ability to enable or disable the RSS syndication feature of DSpace. When

enabled, each community and collection has links for users to view new additions via an RSS feed. By default this functionality is not enabled, but RSS feeds are becoming increasingly popular throughout the Web for keeping up to date with varied topics.

3.2.5. Society in DSpace

The 5S Societies model [8] deals with perhaps the most important aspect of DLs, and any software system perhaps, the users and communities of users. In DSpace, users are known as EPeople and can be created either by a user creating an account on a DSpace site, by connecting it to some other authentication system like LDAP, or by an administrator creating a new user through the administration tools on the site. There are two types of EPeople in DSpace, regular users and administrative users. Within the context of the DL software, these users are both of generic EPerson types but have different authorizations and access rights assigned to them. Multiple users who share common interests, attributes, or authorization needs are grouped together using DSpace's Groups concept. Group membership is either assigned in a blanket fashion when accounts are created, or membership can be assigned manually through the DSpace administrative interface.

As for specification, we maintain the separation between the definition of regular users and administrative users due to the semantic differences between those types of users, no matter how similar the underlying architecture presents them. We represent Groups as well, keeping their specification very simple and true to their nature in DSpace, sub-collections of users who have commonalities and would benefit from being referred to with some singular name. Like in DSpace, group names in our specification also are used in our Structure model in assigning authorization policies to groups of users.

3.3. DSpace DL Specification

Based on the different aspects of the DSpace digital library software system that relate to the different areas of the 5S model for DLs, we develop a specification language that enables users to define digital library systems they would like to create using DSpace. The manifestation of this language is a model in XML schema for a DSpace

driven DL, which allows us to repurpose that model for both our digital library generation technique as well as other tools that support XML and XML schema. Also, because the model is manifested as an XML schema we can use it to validate instances of the model, thus further harnessing the power of XML for those purposes.

In the following sections, we detail the makeup of our model for DSpace specification by providing discussions about the makeup as well as showing diagrams of the model for each of the sub-models we use that are derived from 5S. For reference, we provide a full copy of the XML schema and an example in Appendix A of this document.

3.3.1. DSpace Specification Top Level Overview

In previous versions of 5SL metamodels, the top level of the Digital Library specification has only included the umbrellas of the 5S's that the model is built on. We largely model that philosophy, and the majority of our model lays under the encompassing models for Streams, Structures, Spaces, Scenarios, and Societies. However, unlike the original conception for 5S, representing the structure and concerns of a digital library system, our aim is to take it a step further and include higher level configuration issues of the software itself, not only the digital library that the software embodies. Therefore, at our top level, in addition to having the element groups for specification of the 5S areas, we also include a series of elements for the declaration of a number of configuration related options. These parameters are overviewed in the table below, followed by some additional discussion.

Table 6 - DSpace specification top level elements

Specification Element	Description
SourceDirectory	Path to directory where DSpace source resides
InstalledDirectory	Path to directory where DSpace is to be installed
JDBCPath	Path to JDBC Jar file for DB software version used
URL	Full desired URL to where DSpace will be
Hostname	Hostname of DSpace instance
Port	Port of DSpace instance
DBType	Database type (postgres / oracle)
DBDriver	Name of Java database driver
DBHostname	Database hostname
DBPort	Database port
DBUser	Database username
DBPass	Database password
SMTPServer	SMTP server to use for outgoing emails
SMTPAuthUser	Username for SMTP Auth, if required
SMTPAuthPass	Password for SMTP Auth, if required
FromEmailAddr	From email address for communications
FeedbackEmailAddr	Feedback email address
AdminEmailAddr	Administrator email address
HandlePrefix	Handle prefix (default is 123456789 if none setup)
PatchPath	Path where optional patches to include reside

The 'SourceDirectory' and 'InstalledDirectory' elements are relatively straightforward, and represent the locations that will be used in the generation process, including as the compilation of the DSpace source proceeds. The 'JDBCPath' element must hold the path to a JDBC connector Jar file so that our code can communicate with the database through Java. The 'URL', 'Hostname', and 'Port' elements represent the location that the repository will be accessible on from the Internet, once it is live. These also are used in the DSpace configuration process, separate from the structure or character of the finished DL itself. DSpace can currently support both PostgreSQL and Oracle in the data storage layer in the architecture, though PostgreSQL is preferred and strongly recommended. The selection of database types to use, as well as the location and other configuration options that are database related, are specified in 'DBType', 'DBHostname', 'DBPort', 'DBUser', and 'DBPass' – based on the settings used when installing and configuring the user's database of choice. Several parts of DSpace, including user creation, include situations where the installed DSpace system sends out

emails programmatically. For consistency across platforms and ease of use, this is done through the specification of an SMTP server. The 'SMTPServer' element requires the hostname of a mail server capable of sending email. In cases where that mail server uses SMTP authentication, we mirror the DSpace configuration by providing the 'SMTPAuthUser' and 'SMTPAuthPass' which represent the username and password so that emails may be successfully sent. Emails sent programmatically will have the 'from' email address set 'FromEmailAddr'. Other places on running DSpace instances and the emails they sent mention the email addresses of system administrators and a feedback email address, specified in our top level model as 'AdminEmailAddr' and 'FeedbackEmailAddr', respectively. If the organization is registered with the handle system at handle.net, they would specify their handle identifier in the 'HandlePrefix' element, otherwise leave the default of '123456789' intact. Lastly, if the user has obtained any patches they would like to patch into the source code before the generator's DSpace compilation stage, they can provide a path to where such patches are available on the local system (though we do not implement this functionality).

Besides configuration details which assist in the compilation and installation of DSpace, the only top level details that are provided are for the 5S models we use, 'Stream_Model', 'Structure_Model', 'Space_Model', 'Scenario_Model', and 'Society_Model'. We detail these aspects of our specification model in the next few sections.

3.3.2. DSpace Streams Specification

With regard to streams, or the actual data or information that makes up the heart of most digital objects, in 5SL's initial incarnation sub-elements within a Stream Model declaration were titled by stream type—text, image, etc. While from a purely theoretical approach this differentiation works well to denote the different types of documents possible, in actuality that sort of constraint brings an added layer of difficulty if a specification is to be made in a practical, real world context. Especially with an application of this practice to DSpace—which is more focused towards storing and accessing documents than actually displaying them in an attractive, web-based context—stored digital objects may be of many types, not just following the generic breakdown

into text, image, sound, etc. If this original naming strategy for element names were to be used in our context here, countless object names would be needed—text, image, audio, video, program, checksum, and so the list goes on. While illustrating with those classes of file may be useful, in a practical DL generation task overloading those names to such an extent would likely make the task more difficult and not ease the inherent difficulty already present in creating a digital library.

In our approach, we make the DL specification accepted by our generation use a generic name for streams, 'Stream', and leave the rest of the definition of what makes defined streams unique to child elements within 'Stream' elements. For the stream aspect of our DSpace oriented model, we define the stream types that DSpace knows about as a set of 0 to n Stream elements. Each stream is named, given a short textual description, as well as has a content-type associated with it, which is also known as its MIME type. Each stream may have 1 to n extensions which are associated with that particular type. In this model, all items are required to be filled out in order for DSpace to consider it a valid Bitstream format. As such, our XML schema requires that all elements listed are existent in an instance of this model. In Figure 10 we provide a graphical representation of the schema we've developed for the Stream model for our DSpace specification. Throughout the rest of this chapter we provide similar diagrams to show the structure of the remaining sub-models we use in our DSpace specification language.

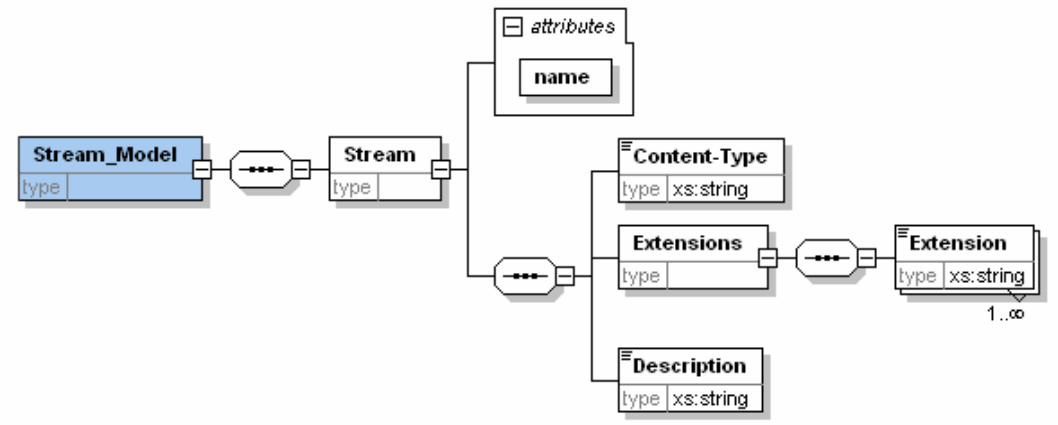

Figure 10 - DSpace specification Stream model structure

Modeled after DSpace's concept of Known Bitstream Formats, in any default DSpace installation there are already Bitstream formats DSpace knows about that come

preloaded. As such, DSpace by default knows about the most commonly used document formats including image, textual, audio, and video formats. These formats are considered to be default and ingrained in a system and cannot be removed through specification, only added to.

3.3.3. DSpace Structures Specification

The structure definition in our model for DSpace specification is relatively involved to cover the amount of structure that is seen in DSpace instances, as well as similar to the structure found in many modern day digital library systems.

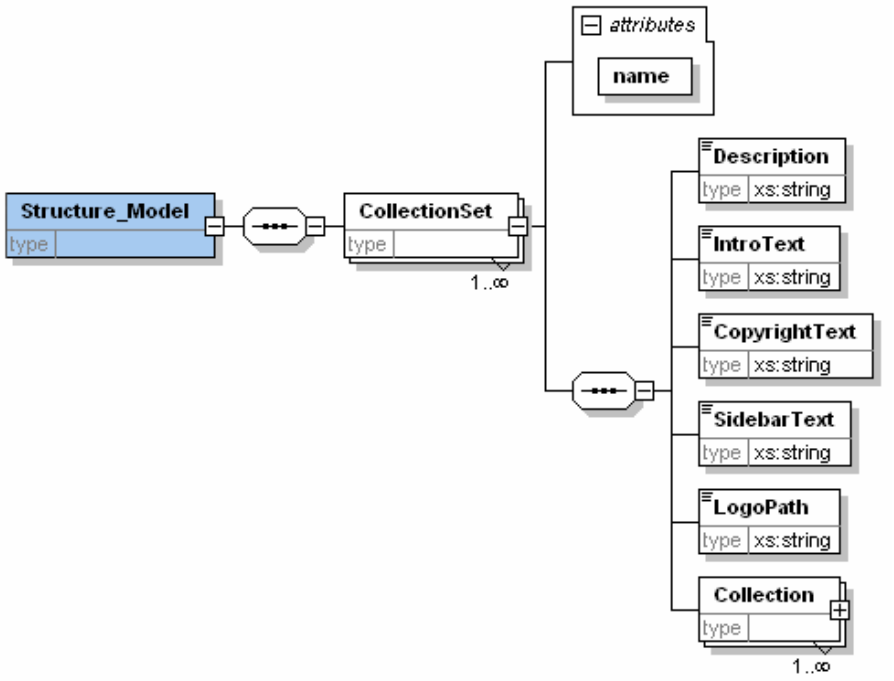

Figure 11 - DSpace specification Structure model structure

Borrowing from 5SL, we split the structure model into the main divisions of ‘CollectionSets’ and ‘Collections’ which serve to represent Communities and Collections in DSpace. The top most level of that structure is ‘CollectionSets’, each of which represents a Community, or set of Collections in the DSpace architecture. At that level of the specification, there is required to be a name field for each CollectionSet which uniquely identifies a Community by name and also is displayed prominently in DSpace while browsing the system. Associated with DSpace communities there are also a series

of textual fields, some of which can support HTML content as well. Among these, our model allows descriptions of communities, the specification of introductory text to be shown for a community, text blocks that detail the copyright issues of community content, and text to be shown in the browse interface's sidebar for each community. Also specifiable for each 'CollectionSet' is the logo that will be shown for that community when users browse to that community homepage.

For each 'CollectionSet' in the system, there are 1 or more Collection nodes that must be specified; these represent collections in DSpace—the bodies that store the actual content of the repository, the digital documents. Collections have a similar makeup with regard to needed explanatory text and resources to Communities in DSpace, and thus the similarities are also seen between our 'Collection' and 'CollectionSet' specifications in our model.

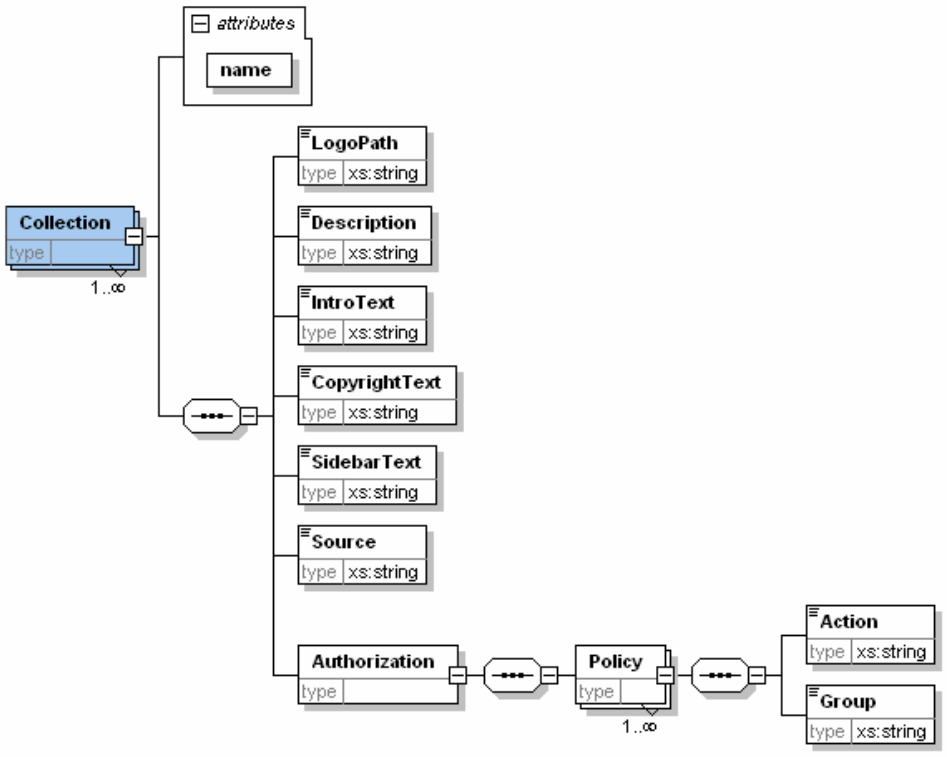

Figure 12 - DSpace specification Collection submodel, from Structure model

Similar to CollectionSets, each 'Collection' element must have a name attribute which uniquely identifies each collection in DSpace. The same 'Description', 'IntroText',

'CopyrightText', and 'SidebarText' elements that are used in the 'CollectionSet' model are also used here. They represent the same textual information that is displayed when browsing DSpace collections as their counterparts represented for browsing DSpace communities. Likewise, the logo displayed for DSpace collections also can be specified within the 'LogoPath' node in our Collection specification. Since collections in DSpace are the logical place where digital documents exist, we provide a facility in our specification to specify a directory where source documents are located that fit into the specified collection.

In the Collection piece of the Structure model is the place where authorization policies are defined that help to describe how users and groups of users may access and manipulate different levels of the hierarchical structure in DSpace repositories. This manifests itself as a set of Authorization Policies, each of which is comprised of an action and a user group who should have rights to execute that action on the collection in question. The 'Group' element specified under each 'Policy' element is a textual string that refers to the name of one of the groups. In addition to user generated groups, suitable names for this element (and DSpace in general) are the groups that DSpace automatically generates—"Anonymous" for regular users who are not signed into DSpace as a registered user, "Registered" for users who have registered with the DSpace instance in question, and "Administrator" for the set of all administrators for the system. The 'Action' element refers to any DSpace internal action, some examples of which are "Read", the ability to view a collection or item, "Remove", the ability to remove something from DSpace, and "Add", which signifies a user's ability to submit to a collection.

3.3.4. DSpace Spaces Specification

We use the Space aspect of our model to define the most commonly and easily modifiable user interface elements that do not require editing the countless JSP files which handle the final rendering of most DSpace web pages. We follow the example of the original work on 5SL, which split this aspect of digital library specification into two parts, User Interface related issues and Information Retrieval details [37].

As mentioned in prior discussion, perhaps the most important functionality DSpace provides is to define how rendered HTML documents appear using CSS. For simplicity in modeling, instead of incorporating these extensive declarations as part of the DSpace DL specification, we simply accept a path to a CSS declaration that will be used in this generation. DSpace's architecture encapsulates this declaration in a JSP file, which we provide as part of our generation package for additional user specification and manipulation. DSpace also makes use of a series of HTML or plain text files that get used to build the interface users see when they browse a DSpace instance website. Among these, we accept paths to text files which contain content to be used as "news" near the top of DSpace's website as well as a news section used on the side of the main page. Similarly, the regular main page sidebar is user modifiable by specifying a path to the content the user would like displayed. The page header used at the top of each page, including the logo image to be used, is also user specifiable in our model. Further, we also include optional support for Manakin DSpace, the Texas A&M's DSpace user interface enhancement project, which includes additional details such as Manakin's own overall DL description as well as Aspects and Themes (which are discussed further in Section 2.3.1.6 above). The last item listed in the UI section of our Space model is 'BrowseItemsToShow' which is a collection of Item elements, each of which contains a Dublin Core field name that should be displayed in the item view when users are browsing and viewing documents in the repository.

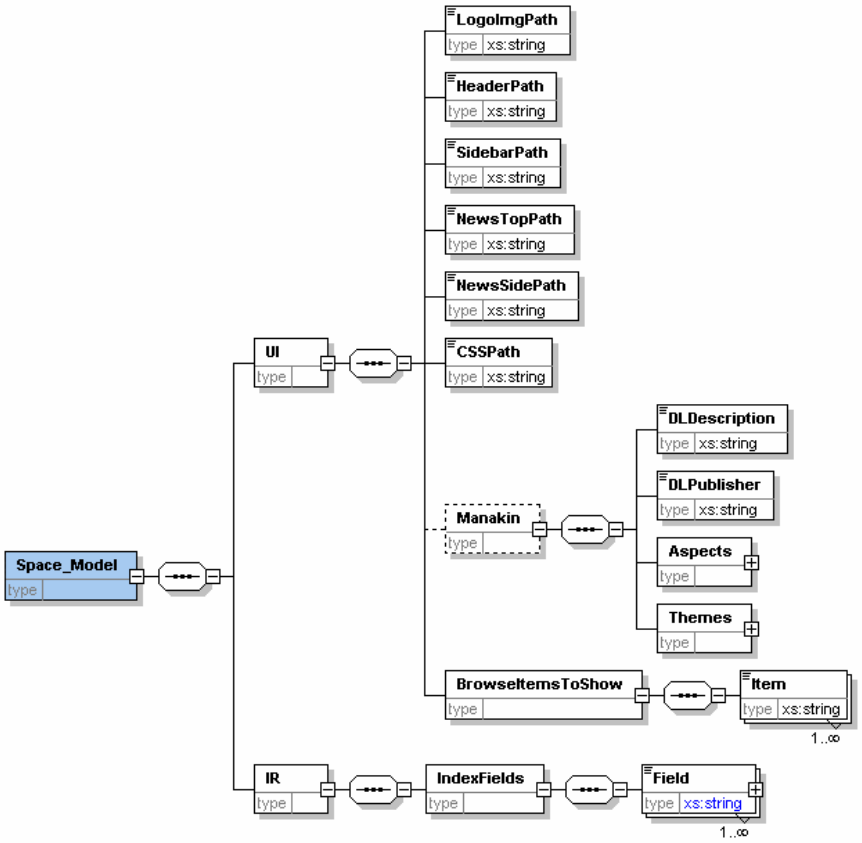

Figure 13 - DSpace specification Space model structure

The IR section of our Space model is rather sparse, only including a definition of which fields should be indexed. Here, we add support in specification for the indexes that are created by DSpace in order to be used with the Apache Lucene search implementation that is included in DSpace. Within the 'IndexFields' element is the specification of 'Field' elements, each of which has a name attribute that indicates the index to which the specified metadata element should be added. Changes here affect the behavior of searches but do not affect the actual visual display of the DSpace search interfaces—another instance of the loose coupling of the user interface with the underlying business logic of DSpace.

3.3.5. DSpace Scenarios Specification

While originally the Scenario aspect of the 5SL specification language dealt more with the system interactions that made up DL services, in applying this to existing DL platforms we abstract those details out to representing services that can be toggled on and

off. Unfortunately, DSpace doesn't provide turnkey control of many of its services, most are tightly built into the system. The one aspect that we do provide is RSS support, which provides syndication of newly added items to users who subscribe to the automatically generated feeds available from each collection and community in DSpace DLs. Within the main 'Scenario_Model' element, there is a sole element, 'EnableRSS', which can be either true or false.

3.3.6. DSpace Societies Specification

The important characteristics related to users and groups of users which show themselves in the Societies model of the 5S model are also present in our model for DSpace specification. Our 'Society_Model' top level element contains only sub-elements, each of which group together user-related issues that are important to DSpace and digital libraries in general.

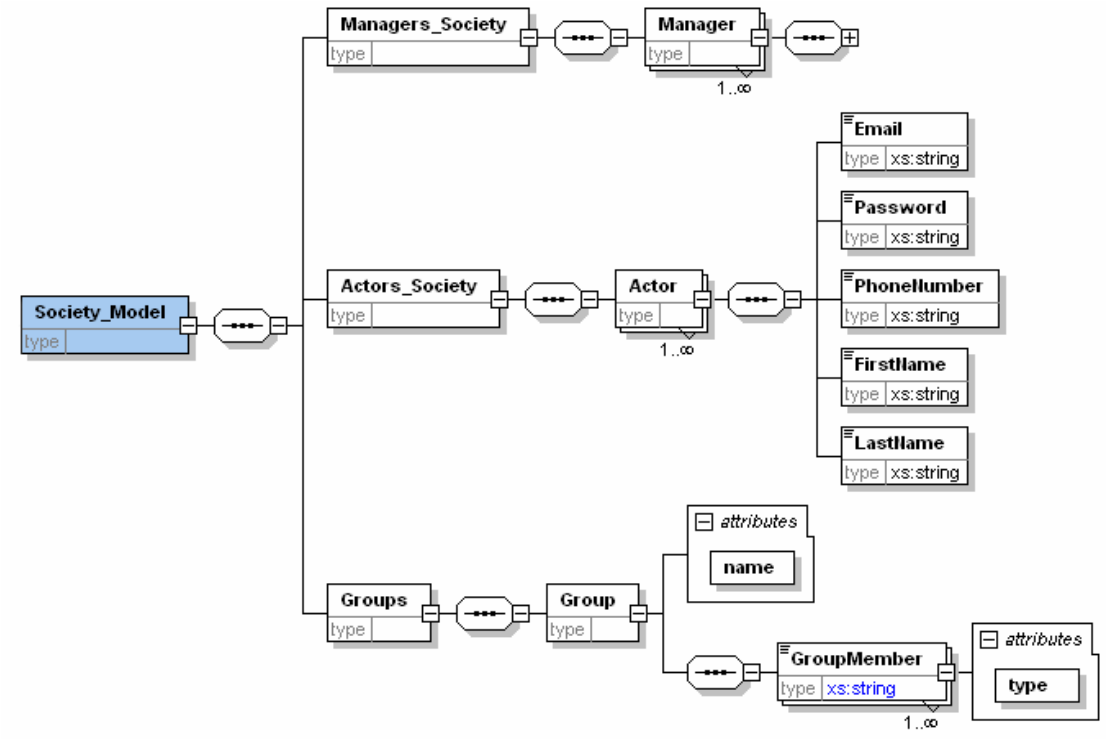

Figure 14 - DSpace specification Society model structure

Within DSpace, both administrators and regular users have the same data elements that relate to them and have the DSpace realization as EPeople, with different authorizations attached to them. Therefore, in our model, the Managers (DSpace administrators) and Actors (typical users) have the same information associated with them. For the purpose of brevity, we expand only the 'Actor' element in Figure 14, as the 'Manager' element contains the same metadata. As such, for each user in DSpace a set of details are required including an email address, password, and phone number, as well as a first and last name.

Another important part of the societies evident in a digital library are groupings of users based on common interest, level of access, etc. Within DSpace, these groupings are fittingly called Groups. Groups are made up of one or more users and/or other groups. In our model, within the 'Groups' element there are zero or more 'Group' elements, each of which represent one DSpace Group. DSpace Groups each have a unique group name, which appears in our model as an attribute on each XML Group element. Each 'Group' element has a set of 1 to n 'GroupMember' elements, each of which is a member of that DSpace group. To specify whether a 'GroupMember' element represents a group or user, the 'type' attribute may be set to either "group" or "user".

Chapter 4. dlGen Digital Library Generator

Given the model we have devised for the specification of DSpace repositories, in addition to the ability to map out a digital library system, it would be useful to be able to apply such a model to actually create DL installations. In this chapter we discuss our generator for DSpace digital libraries by providing a brief overview discussion, plus more in depth descriptions of the design, architecture, and implementation of the tool.

4.1. Generator Overview

The concept of programmatic digital library generation is not a new one. In Chapter 2 of this document, we describe in some detail some of the past work in programmatic digital library configuration and generation. Most of this past work has dealt with relatively research oriented DL systems, and there has been little work in attempting to generate a working digital library of any widely used DL system. Some work also varies with regard to the interaction mode the process runs in, DLs created through a prewritten textual specification, versus a more interactive, graphical DL generation processes. In Figure 15 we provide an overview of the specification and generation process.

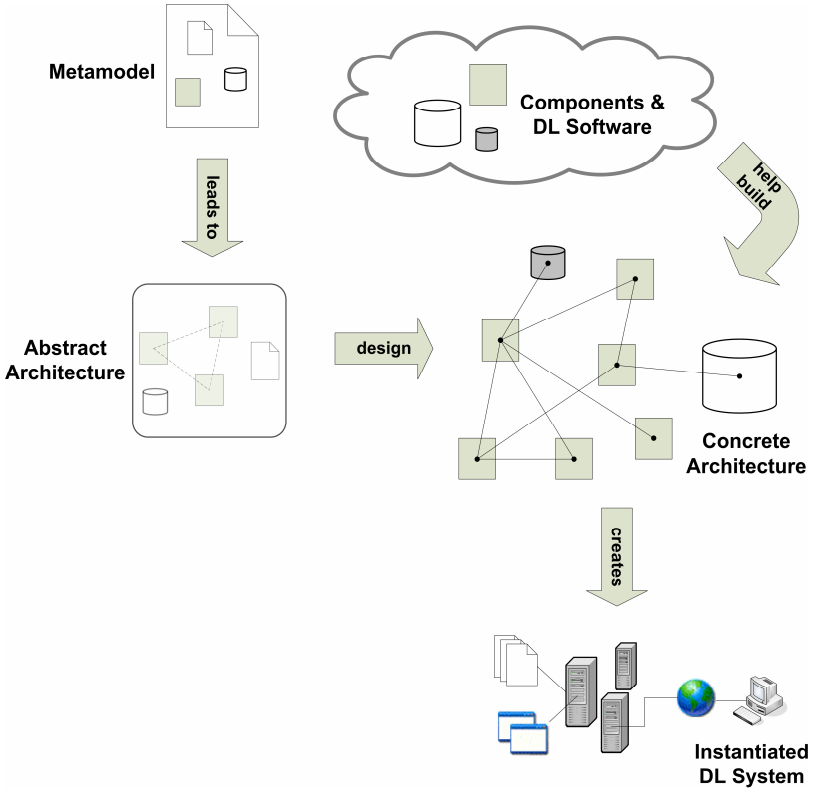

Figure 15 - Overview of our generation process. This representation of a DSpace specification serves as a metamodel for which specific instances can be derived that represent a user's desired DL system and make up an abstract DL architecture. Based on the declared DL and DSpace software, a concrete architecture gets created for that DL, which finally builds a working DL.

Our generation process uses a textually based, XML specification that details the digital library a user wants created. While our main work here deals with the specification and generation of repositories using the DSpace software, the generator has been built in an extensible nature that would allow for the use of generators for other DL systems that can be snapped into our main application if the DL software specific coding has been done for repeat use.

In the next few sections, we describe some of the requirements that influence the organization and design of the generation tool, an in-depth look at the architecture of the generator, as well as a discussion of the implementation of various parts of the generation process.

4.2. Requirements and Design

There were a loose set of aims which we sought to meet in creating a generation tool for digital libraries. First of all, and perhaps most critical, was the general purpose of programmatically creating a digital library system with no user interaction whatsoever. There are, of course, plusses and minuses to that decision. Such a system could be started and set aside to run through its process without the need for a user to continually watch and interact with it. On the other hand, when errors occur in the process due to technical problems or bad specifications, there is no way to react to those problems at generation runtime, so errors are reported, that must be fixed, and the generation tool is rerun.

Though we focus on DSpace generation in this work, we also wanted to design the architecture of the generator so that it can be easily extended to work with other generation components that have not yet been authored. To meet this aim, we dynamically load the generator classes based on a separate configuration file which is read at runtime and specifies components that will be used for generations. It is true that the generator is only a shell without the DSpace specific classes; we still believe having that extensibility is beneficial and makes the tool more effective in the long run.

In this work we specifically aimed to have some of the overhead of DSpace installation covered and executed by the generator itself in order to lessen the load on the DL designers and system administrators. In our process although users must have Apache Tomcat and PostgreSQL installed before they can run our generator, the generator covers all DSpace related installation tasks with regard to configuration and compilation.

4.3. Generation Process Overview

Based on the way the DSpace software is set up and likely true with other software packages as well, the order in which configuration and generation processes occur, matters deeply. Different tasks require others to be completed first as the presence of certain details, pieces of code, and data may hinder the ability of other aspects of the generation to properly occur. This fact is understood and reflected in our system, and the generation process follows an ordering in which all important steps occur in proper succession.

Due to the multiple facets of our DSpace generation process and dependencies between different parts of the process, the ordering of generation tasks needed careful attention in order to determine the necessary sequencing. Some aspects of the process, like Apache Tomcat installation, must occur before any DSpace code gets dealt with at all, while others require that Tomcat has already been installed and DSpace has been compiled. Overall, it is important for the end user of the tool to run the generator as root, so that all permissions are available that are needed to setup Tomcat, PostgreSQL, and DSpace.

First, before our generator may be run, the user must install Apache Tomcat, the servlet container server that runs DSpace. Minor configuration options must be set for Tomcat to work properly, but for the most part the installation is straightforward. Because DSpace requires a database server for backend metadata storage, before the repository software can be installed and used, a supported database engine must be put in place. We use PostgreSQL, one of the common open source database software packages available today. (DSpace has supported Oracle in the past, but today's implementations all but require PostgreSQL. Some work is being done by contributors to allow DSpace to again work with Oracle, and other database software packages, but that is looking ahead.) PostgreSQL's installation procedure requires the initialization of the database, user creation, etc., but is as easy as following a set of steps. From this point on the remainder of the installation of DSpace is handled by our generator.

Next, DSpace's code base is uncompressed and some initial configuration must occur such as setting hostnames, database information, as well as other settings DSpace requires before compilation. Much of this configuration occurs through DSpace's dspace.cfg file, which we revisit later for additional configuration. Using Apache's Ant compilation assistance tool, we can compile and build the DSpace .war archives, which will be copied into Tomcat's web application directory and expanded when Tomcat is restarted. As mentioned above, this build process also initializes DSpace's database structure and assuming all went well, DSpace should be in working order and serving out web pages on the port and path specified in the DSpace configuration. According to the DSpace installation guide [20], most problems that occur up until this point are database related settings, which should be much less of a problem in this generation process. Now

that DSpace is compiled, installed, and running, we can make use of the DSpace APIs to create and manipulate the hierarchical structure of the library, content, and user related aspects.

4.4. Architecture

The architecture of a system such as this is something that should be given some careful thought and examination for the best way to structure the organization and interaction of the classes that make up the program. The bulk of the generator tool that this work encompasses involves the dynamic creation of a DSpace instance based on the specification of a digital library in the 5SL-based DSpace specification model that we have described. To meet some of the goals of the tool, we have also designed a relatively simple, higher level architecture that dynamically loads classes required for generation. In such a way, the main generator class itself need not be recompiled for additional DL generators to be added to it, which greatly increases the life and scope of the tool itself.

The overall path of execution includes a few different stages and throughout the following sections we detail all levels of this process. First of all is the existence and validity checking of all input files required for generation of a specific DL in a given DL platform. Next, semantic checks of the input model ensure all aspects of the model agree with one another. Lastly, the real heart of the generation occurs as each 5S-related class has its own generation fired off sequentially to configure and generate the DL.

4.4.1. Input Files

To drive the way in which the dlGen application works, the tool uses a configuration file, supportedDLs.cfg, which details the different DL software products that the generator currently supports. This file is made up of key value pairs for the name of supported DL systems and the schemas and classes that are used for that type of generation. In addition to this configuration file, the XML schema for the DL platform desired is needed by the system as well. We provide an example of this configuration file in Figure 16. At runtime, an input file also is required that must be an XML file which adheres to the constraints and structure specified in the schema for the DL platform in question.

```
# configuration file for generation process
#
#  - denotes the supported platforms for the generation process
#  - denotes the classname to use for that platform
#  - denotes schema to be used to each supported platform
#
#  ex: dspace = DSpaceGen
#      dspace.schema = dspace_gen.xsd

dspace = DSpaceGen
dspace.schema = dspace_gen.xsd
```

Figure 16 - dlGen generator configuration file example

4.4.2. File Existence and Validity Checks

In order to be as stable and complete as possible, we include file existence verification to ensure that all files referenced and needed in this process, like the imputed DL model, DL schema, and configuration file, will exist at runtime. Error messages are outputted and the generation process ceases if one of the files specified do not exist.

Assuming that the input files exist, the schema is checked to be well formed, and if so, the DL model passed into the tool is checked to be well formed and be an instance of the DL schema provided. All of this processing and checking occurs in the main dlGen application, and remains applicable to any future platforms that are specified in the configuration file, no matter the content. Once all these conditions are met, the path of execution changes from being in the main application to a main SpecificDLGen class (which is a generalization of instances like our main DSpaceGen class).

Following a main class that coordinates generation of a particular DL platform, there are additional classes that handle generation tasks for the 5S-related aspects of Streams, Structures, Spaces, Scenarios, and Societies. Although the main dlGen application checks for a well formed XML DL model and compliance to the DL schema being used, there are some semantic aspects that cannot be easily checked using a schema. These sorts of issues do not include type checking, which the schema does enforce, but instead include issues related to consistency between different sub-models that make up the DL definition. One example of such a situation is that while Groups are specified within the Society piece of our specification model, the Structural model has authorization related details, which include referencing a Group that should be given certain authorizations. While the presence of elements can be easily checked by

validating an instance against a schema, this level of checking is more complex and we decided to abstract it out to programmatic checks instead.

4.4.3. Classes and Structure

The main generator class, SpecificDLGen, has two main purposes. First, it coordinates the execution, configuration, and generation of the different 5S-related aspects of DLs, which are handled by additional classes. Second, the class completes generation tasks that are not specific to the digital library architecture, structure, or nature itself but involves the installation of the DL software itself, which is the canvas upon which the rest of the generation works.

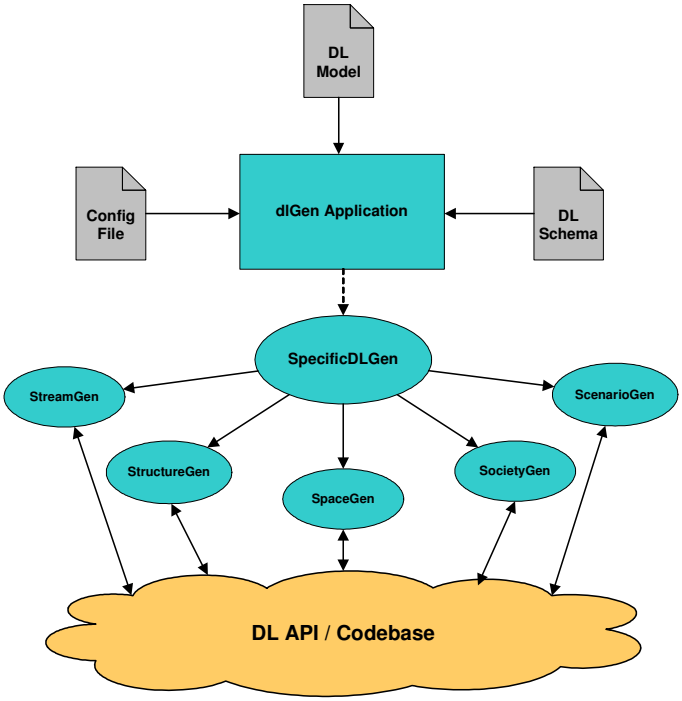

Figure 17 - Our generic generation architecture

In our case here, SpecificDLGen handles the customization, compilation, and installation of the DSpace software from the original source code. We organize functionality into the remaining main classes of our generator following the organization of the original work with 5SGen [44], making one class to handle generation for each 5S related area—a StreamGen, StructureGen, SpaceGen, ScenarioGen, and SocietyGen. Each of these classes covers tasks that are specific to their area of DL concerns. For

example, the StructureGen class handles DL structure and organization related configuration and generation; the SocietyGen class covers user and group related tasks. Looking even closer, each 5S-related generation class is split up into a number of functions which carry out major generation tasks. Two such examples functions are `createUser()` and `createCollection()`, which are singular functions that are greatly reused to help generate the overall library as specified by the user. Much of the interactions in functions such as these occur through use of the DL specific code bases and APIs. Many widely used DL systems are now developed enough so that their underlying code can be used externally from the DL itself so that functionality that occurs within the DL product itself can be accessed by code running on the same machine. In our DSpace generator we rely heavily on DSpace's backend API that allows us to interact with and programmatically affect how the DL functions.

4.5. Implementation

In implementing a system such as the dlGen tool to generate practical digital libraries, we attempt to use technologies that would give the tool the best possible functionality and longevity. Since DSpace is coded almost entirely in Java, our using Java for our system as well, simply made the most sense, to avoid cross programming language problems.

4.5.1. dlGen Main Application

The main dlGen application is made up of a single class that, in reality, does little more than begin the generation process and ensures that the application is run with all required parameters. The initial process checks the validity of all input files, ensuring they exist, that XML files are well formed, and that the DL specifications passed into the tool adhere to the model for the DL platform in use. To ensure the existence of all files referenced by user specification or at runtime, we use Java's standard IO packages. Normally, the application will be run from the command line with two command line parameters; the first, specifies a DL platform to generate the DL specification, while the other denotes the XML specification that should be used. Here is an example:

```
java dlGen dspace dspace_instance.xml
```

In this case, the generator would be running using the "dspace" schema and generator class specified in the generator configuration file. One method, `loadConfigs()`, opens the generator config file and ensures that the DL platform specified at runtime exists, and so does the schema listed for that DL type. Another method, `validateXML()`, validates the XML DL specification. To ensure XML is well formed, we use Java's standard API package, `javax.xml`. We use the same package's schema functionalities to verify the compliance of XML instances to the corresponding model.

Being designed specifically to be a frame of a DL generating program that would be extensible without the need for recompilation, we use specific technology that allows us to dynamically use different generation classes for digital library software systems. To meet this aim, we make use of Java's Reflection API, which allows for the dynamic loading of classes and instantiation of objects defined therein. Once classes are dynamically created, objects that represent the individual methods that are used also are in order to call member functions of the base generator.

Throughout our digital library generator we use two common method names to complete the two main goals of the generator—ensuring the specification is syntactically and logically valid, and actually generating the digital library system. In Section 4.4.2, we mentioned the logical inconsistencies between different sub-models of DL specifications that we check as well as conformance to a schema. For these processes, both within the top level dlGen application and in lower level classes, we call such methods `checkSpec()`. This method is first called on the dynamically loaded main generation class for the DL platform in question, DSpace for us now, and then those calls cascade throughout the rest of the generation classes the generator uses. Each call returns a Boolean true or false according to the specification's logical validity for each generator class. While we assume that the sub-structure of generator classes adheres to the 5S model and one generator exists for each of the S's, this need not necessarily be the case. When dlGen's `checkSpec()` call finally returns, a true value indicates the DL specification is completely valid and generation can begin. The other method name we use throughout our class structure is `generate()`, which is called in the dlGen class to set off a cascade of generation tasks. Each `generate()` call triggers the generation

process for each 5S related generator our method uses, and returns a Boolean true or false to indicate success or failure of the particular class's tasks. If any class's generation process fails, the process will halt entirely and the user will be given an appropriate error message giving the task that failed. We outline this process in Figure 18 below.

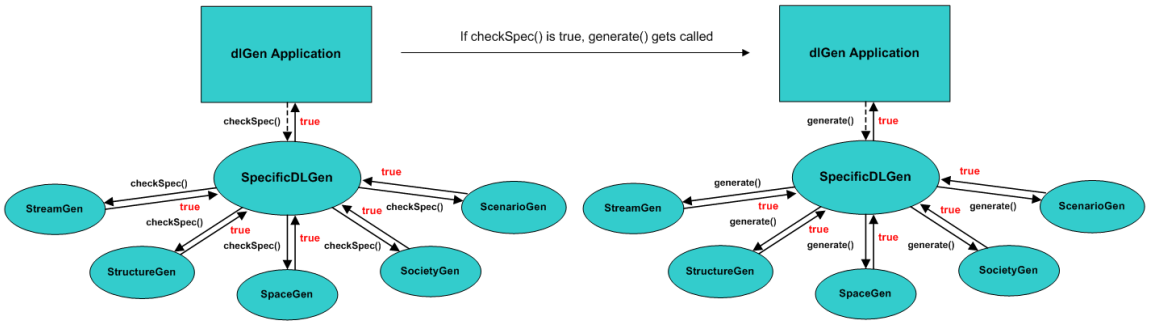

Figure 18 – Overview of successful dlGen application execution; if checkSpec() succeeds, then generate() is called throughout the same class structure.

4.5.2. DSpaceGen Class

Just as the main dlGen class drives the execution of the entire generation process, our DSpaceGen class drives the generation with regard to the particular DL type for which the tool is being run. The class is split up into three main processes. As with all generation related classes in our architecture, DSpaceGen has `checkSpec()` and `generate()` functions which respectively check the specification in use for logical consistency and generate the DL. Thirdly, this main generation class handles the configuration tasks central to the DL software and not the DL instance itself. Details such as the DSpace source and installation directories, port, and hostname Tomcat runs on, and others, are specified at the top level of our DSpace DL specifications and are therefore handled on this level of the process. The `checkSpec()` function simply calls the `checkSpec()` method on each 5S generation class—StreamGen, StructureGen, SpaceGen, ScenarioGen, and SocietyGen. The `generate()` function is similar, simply calling the corresponding `generate()` method in each lower level generation class.

Since DSpaceGen handles details specified at the top level, there is set of elements that must be checked for logical correctness. While the class does have a `checkSpec()` function used to execute 5S generation classes, it would be bad practice to

lump the same behavior explicitly in the same method. For clarity and a good separation of code, we define a `checkRootSpec() function` that does the same logical checks for top level elements that are not tied to a specific 5S area. In reality, the only checks that occur at this level are to ensure that all paths specified are valid and exist; since these elements are separate from any 5S area, they do not reference content in any of those groups of elements. In the same way, we have a `generateRoot()` function that completes configuration and generation tasks associated with the DL software itself, not the nature of the digital library instance we are generating. Tasks that occur on this level are the configuration of DSpace configuration options in DSpace's dspace.cfg file and compilation of code. In Figure 19, we provide an excerpt from a DSpace configuration file for reference. These specialized root functions are called first before other generation classes when the specification checks and generate functions are called.

```
#
# DSpace Configuration
#
# Revision: $Revision: 1.78 $
#
# Date:     $Date: 2006/05/26 14:26:12 $
#
##### Basic information ######

# DSpace installation directory
dspace.dir = /dspace

# DSpace base URL.  Include port number etc., but NOT trailing slash
dspace.url = http://localhost:8090/dspace

# DSpace host name - should match base URL.  Do not include port number
dspace.hostname = localhost

# Name of the site
dspace.name = PhotoSpace
```

Figure 19 – Example excerpt from DSpace's main configuration file, dspace.cfg

4.5.3. 5S-Related Classes

Though any architecture is acceptable for DL generation classes to be used with dlGen, we decided to divide the generation aspects of our DSpace generation classes into the logical areas that are described in 5S. These classes, StreamGen, StructureGen, SpaceGen, ScenarioGen, and SocietyGen, run and handle the generation of DSpace digital library instances within our technique. It is in these classes that the heart of generation occurs—the higher level classes mostly drive the execution and prepare DSpace to hold and support the user defined DL. Given that the specifications we use are

in an XML based specification language, there is much XML parsing that must occur, so the user-defined characteristics can be read and processed. For simplicity in coding, we use JDOM as an XML parsing package which aids in this process.

Given the many different ways that these generation classes use and interact with the DSpace code base, an intimate understanding of how DSpace works on the backend is critical in order to allow our classes to affect DSpace. Each individual generation class makes DSpace calls to what are commonly referred to as the Public and Storage APIs. In order to make use of any DSpace objects or functionality, the following system call must be made to set the location of the DSpace config file (where the second argument is a full path to the DSpace configuration file):

```
System.setProperty("dspace.configuration", dspaceConfigFile);
```

In Figure 20, we show how our main DSpaceGen class and 5S-related generation classes fit into the overall DSpace architecture.

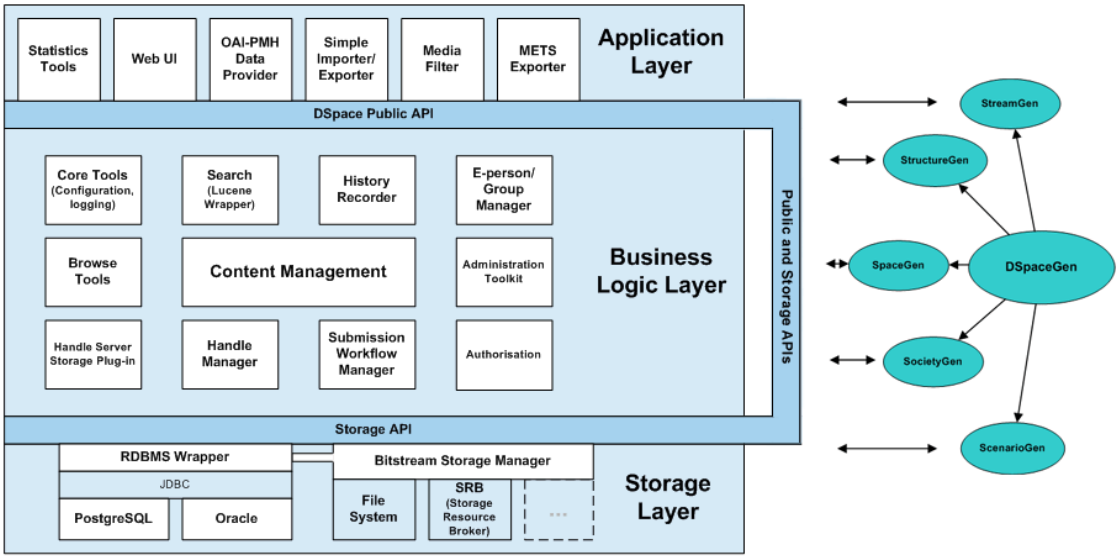

Figure 20 - DSpace architecture diagram with our generator architecture

4.5.3.1. DSpaceStreamGen

The main Stream representation in DSpace that we use in our generation process is the concept of Known Bitstream Formats, which designates the possible file types that DSpace knows about that can be used in stored documents. Specified bitstream formats

within our specification include a short name, description, content-type (MIME type), and one or more extensions that MIME type uses. As we mentioned earlier, there is a set of pre-existing common bitstream formats within DSpace out of the box, so we take special care to avoid any repetition that could confuse the way DSpace operates. To account for this, we have three main methods that we use to set the proper Known Bitstream Formats in DSpace—`checkForMime()`, `createBitstreamFormat()`, and `addExtentions()`. As we process each specified bitstream format, `checkForMime()` is called to determine if the MIME type for the format currently being processed already exists within DSpace. If it does not, `createBitstreamFormat()` is called and that bitstream format is created. If the MIME type exists already, the extensions specified for the format in question are collected and `addExtensions()` is called and adds only extensions not already associated with that MIME type. When all bitstream formats in the DL specification have been processed, generation for the stream generator is complete.

4.5.3.2. DSpaceStructureGen

Structurally related characteristics of the DSpace software that we have decided to include in the StructureGen aspect of our DSpace generation are completely hierarchical in nature. DSpace's main structure is divided into collections and communities, which are sets of collections. The Stream model within our DSpace specification is made up of one or more 'CollectionSet' element, which is our abstraction of DSpace communities. Each 'CollectionSet' element has other configuration details that DSpace communities must include such as title, description, etc. Inside each 'CollectionSet' is one or more collection, which represents DSpace collections. They, too, contain details that DSpace requires collections to have such as title, as well as other metadata.

Our DSpace Structure specification is processed one 'CollectionSet' at a time. In order to slightly abstract the underlying DSpace related code, we create four functions which do the bulk of the work for this sub-model's generation—`addCommunity()`, `addCollection()`, `addCollectionPolicy()` and a helper function that obtains a collection's internal ID based on its name and community. All metadata for each 'CollectionSet' is read, and a `addCommunity()` function call is made to create that community in DSpace. Next, for each collection that is specified for the 'CollectionSet'

in question, metadata for that element is read and a `addCollection()` call is made, which in turn creates the collection within DSpace. Within our Structural model for DSpace we include authorizations for the collections declared, which give groups of users the ability to complete certain tasks within DSpace. Within each collection element in our specification, there is an 'Authorizations' element that contains individual policies that denote an action that a group of users has access to do. For each policy specified in each collection, the `addCollectionPolicy()` method is called which gives permission to a particular group to do a certain action. Once all policies for each collection are added, the groups in question in DSpace will have permission to do what was specified. Also as part of the process of each collection being added, a specification may include a 'Source' element that a user can use to import content into a specified collection. After all 'CollectionSet' and 'Collection' elements have been iterated through and processed, DSpace includes the structure and content shown in the DL specification currently being generated.

4.5.3.3. DSpaceSpaceGen

The two main aspects of the Space model from our specification are UI and information retrieval oriented details. User interface characteristics that we generate and set largely involve files, HTML files representing headers, and image files for logos. The generation of these is really rather straightforward, and we can simply take the files the user provides for those elements and copy them to the correct locations within the DSpace source code. In cases where the destination location already exists, the original file is saved with the original filename with ".orig" appended to the end. The other piece that comes into play here is our allowing users to specify the list of metadata fields that they would like to be shown when an individual item is displayed. Inside the 'BrowseItemsToShow' element, items to be displayed are each simply specified inside an 'Item' element. The list of items that should be shown when viewing objects in DSpace is set in the DSpace configuration file, dspace.cfg, on an item `webui.itemdisplay.default` that consists of a comma separated list of Dublin Core elements to show. For example, if a user wanted to only display the title and subject of items when they were displayed, the corresponding configuration line for DSpace is:

```
webui.itemdisplay.default = dc.title, dc.subject
```

All items specified are checked against a file in DSpace's configuration directory, dublin-core-types.xml, that lists the Dublin Core elements and their qualifiers. We simply read all specified items from the specification and concatenate them together with commas and set this item in DSpace's configuration.

We also support using Manakin, the DSpace interface extension project, as an aspect of the Space model that can be specified, although due to constraints the generation has not been implemented. Manakin operates as an entirely separate website interface using DSpace's same backend code base and storage sources, so it still contains the same content. To make Manakin active, we must have access to the Manakin source code, which must be linked against the source code path for an existing DSpace. Changes are required to Tomcat's configuration to denote where the new servlet system will be running in its context. Lastly, a few additional configuration items must be added to DSpace's configuration file so that Manakin can use them at runtime. The `xmlui.repository.identifier` item denotes the short name that Manakin will use to describe the DL it displays. When we set this value in the configuration file, we use the same value as specified in the 'name' attribute of the top level 'Digital_Library' element. We use the same source from our specification for the title item as well. The `xmlui.repository.description` item is specified within our specification in the 'Manakin' element as the 'DLDescription' element. The `xmlui.repository.publisher` item is specified at the same level as the 'DLPublisher' element. The similar subject item is specified as our 'DLSubject' element in the same location.

```
##### Manakin Specific ######

xmlui.repository.identifier = PhotoSpace
xmlui.repository.description = Our photography collection
xmlui.repository.publisher = Doug Gorton
xmlui.repository.subject = Photos
xmlui.repository.title = PhotoSpace
```

Figure 21 - Example Manakin details in DSpace's configuration

The information retrieval aspects that we take into account are somewhat similar in their manner of specification. In our specification, inside a 'IndexFields' element, there

are a collection of 'Field' elements, each of which has a 'name' attribute indicating to which index the specified Dublin Core element should be added. The fields that DSpace uses to index are also specified in the DSpace configuration file. In that configuration file, items `search.index.1` and up determine which DC elements are indexed to influence searching results. An example of one such line in the configuration file is:

```
search.index.1 = title:dc.title.*
```

This example basically indicates that the 'title' DC metadata element and all qualified versions of the element as well should be added to the title index, so that when a user searches by title it is included. In order to include this aspect in our process, we programmatically read the specified items to be indexed and which index they should fall in. Then, we set corresponding items in DSpace's configuration as the user specified. Normally after these types of changes are made, DSpace must be re-indexed so that the changes are effective in previously ingested documents. However, since our generator creates the DSpace instance, it is initially empty and thus no re-indexing is necessary. When this process is completed for the Space model, all aspects specified by the user will have been completed including putting the correct headers, logos, and other elements in place so that they'll exist in DSpace's interface.

4.5.3.4. DSpaceScenarioGen

As we mentioned, the extent to which we include scenario-related configuration and generation aspects in our model is minimal. The only feature configured here is the RSS feed generation that DSpace provides to allow users to view new items through a different medium. Within DSpace, this option is set within the dspace.cfg configuration file. While there are custom settings that allow users to change the ways that RSS feeds are generated, we simply provide the ability to use default RSS functionality or to disable it. The setting is found as `webui.feed.enable` in that configuration file and it is set to true or false as is specified by the user. We use the same processing code here as in the root DSpace configuration, which sets or overwrites key value pairs in the DSpace configuration.

4.5.3.5. DSpaceSocietyGen

The society aspect of our specification model is where we deal with all user-related details and configuration. As we mentioned earlier, the main aspects of DSpace that we generate here are administrative users (Managers in our specification), regular users (Actors in our specification), and groups of users (also groups in our model). Each section is enclosed by an encompassing element that groups all similar details together, 'Managers_Society', 'Actors_Society', and 'Groups'. Within DSpace, the nature of administrators versus regular users is only that administrators are part of an Administrators group, which gives them all the necessary rights, privileges, and authorizations within the system. The same set of parameters is required for each type of user, and thus the generation techniques for administrators versus regular users do not differ greatly. Every user in DSpace is unique based on their email address, which is the main login that is used in conjunction with a password to access the system. In the code for this generator class, we again organize the functionality into higher level classes that abstract out DSpace specific code. We create a function `createEPerson()` that creates a generic user in the system based on a type parameter, which determines if the passed in details are for a regular user or administrator. If the user is an administrator, the user is added to the Administrators group in DSpace.

Starting with Managers, as we have in similar situations in the implementation, we use JDOM to parse through the XML and extract each 'Manager' defined in its own element in the 'Managers_Society'. For each manager, we obtain the specified email address, password, first and last names, as well as a phone number. To add these users into the system, we call our `createEPerson()` method specifying we want to create administrators. The creation of regular users specified in the 'Actors_Society' works in a similar way. After parsing out each actor's declared details, a series of calls to `createEPerson()` occur, specifying that the users are regular users. In reusing this user generating code, we are able to create all the users necessary for the system with one function. Because we are starting from a blank DSpace instance, there is no need to ensure users do not already exist in the system—the `checkSpec()` function has already made sure that there is no overlap between users' email addresses in the specification.

The other main type of information that is dealt with here is DSpace group creation. Inside the 'Groups' element, each 'Group' element represents one group that should be created in the system. Each 'Group' element has a name attribute, denoting the unique name of the group to be created. For group creation we have created a `addGroup()` function which abstracts out the DSpace-related group code and programmatically creates a group in the system with the given name. For each group specified, we call this method so that each group is created. Groups are no good without users in them, thus we also have created an `addMemberToGroup()` function which does exactly what the name indicates. Technically groups may hold users or other groups, so in our function we take three input parameters, the type of addition that is to be made (group or user addition), the email address or group name of what is to be added, and the group name to which the member should be added. Within our 'Group' element in our specification model, there are one or more 'GroupMember' elements, each of which has a 'type' attribute which denotes either "Group" or "User" as necessary. The element itself contains the member to be added, whether email address or group name. For each 'GroupMember' element we parse the required information and make calls to `addMemberToGroup()` as needed to create the details specified by the user. Once all specified groups, managers, and actors specified have been created, DSpace will be ready for those users to use the system and have authorization to complete tasks based on what authorizations have been given in the Structure model of our specification.

Chapter 5. Case Study – CITIDEL CSTC Collection Specification and Generation

Given the work that is embodied in this thesis, a perfect complement to the methods and software described is an example of that work put to use in a real world context. To do this, we have obtained from colleagues at Villanova University one part of the CITIDEL (Computing and Information Technology Interactive Digital Educational Library) digital library, the CSTC collection. Here we use our method to create an XML specification for the CSTC CITIDEL collection and use our dlGen generation tool to generate a working DSpace implementation based on the provided specification. In this chapter, we will explain a little bit about CITIDEL and the CSTC collection, provide a discussion of the process of generating the DL with our technique, and include a reflection on the process.

5.1. About CITIDEL and the CSTC Collection

CITIDEL is a joint venture between a number of universities that was created to serve the computing community by providing a repository of electronic materials, specifically targeted to aid computing education. It serves as a repository to bring together resources from a number of different sources. Originally created using a home grown architecture based on Open DLs and building upon the MARIAN DL software developed at Virginia Tech [47], the DL is now being ported over to use DSpace [48]. The CSTC collection is the Computer Science Teaching Center's peer-reviewed content for teaching in the computing sciences. Given the heterogeneity of the sources CITIDEL draws from, it was necessary to use a specialized schema to which all source collections' metadata can be mapped.

In Figure 22 we provide a screen shot of the existing CITIDEL home page.

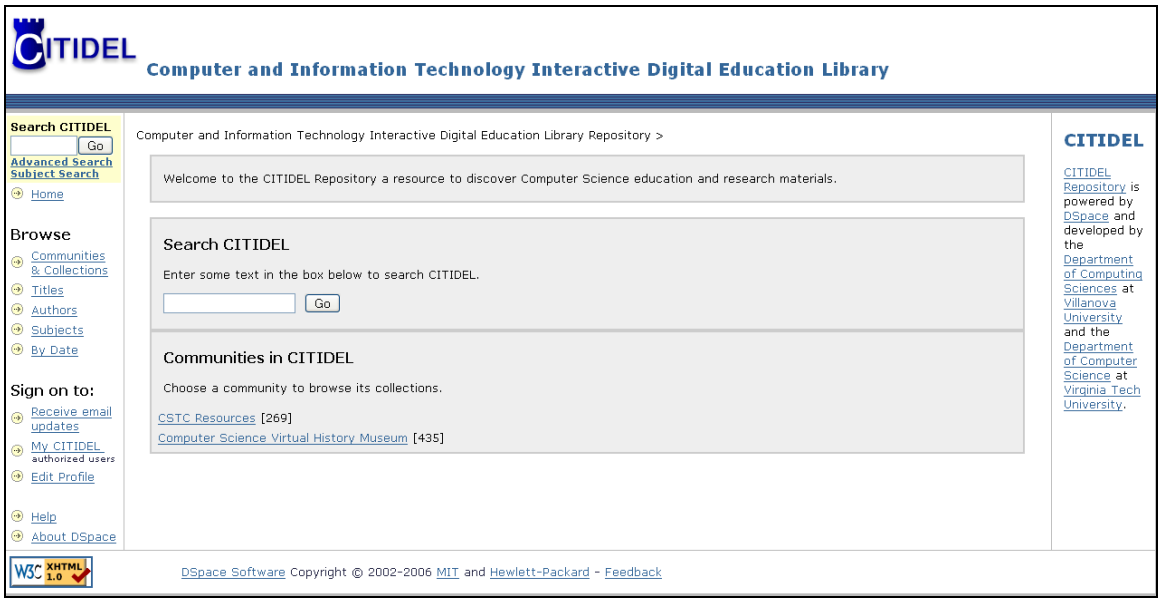

Figure 22 - Existing CITIDEL homepage

5.2. Specification

Though we do have the CSTC collection from CITIDEL, beyond interface design and details that we can determine visually, we have little picture of the inner-workings and settings that the live CITIDEL site is using. Thus we choose these settings intelligently, as is available for our specification model. The existing DL included a community entitled “CSTC Resources” with a collection by the same name with 269 digital objects. As seen in Figure 22, the original DL was branded with the CITIDEL logo and all descriptive text throughout the interface referred to the DL as “CITIDEL”. The site was using the RSS functionality of DSpace to provide syndication of content. To make up for holes in our understanding of the existing DL, we also decided to create an administrative user, and a set of five regular users, all part of a ‘student editor’ group which has rights to add content to the collection. We also chose to use default browsing options as well as indexing options for searching. While we authored this specification manually, we have also successfully used the 5SGraph tool discussed in Section 2.4.3 to visually create a specification using our model. For completeness, we include the entire specification we created in Appendix B and will provide a brief discussion of two of the sub-models in this section.

5.2.1. Structure Specification

The structure specification for the current live CITIDEL DSpace instance is relatively simplistic and was therefore easy to represent in our specification. We need to represent a community called "CSTC Resources" that has only one collection in it, also called "CSTC Resources". We can do this with a 'CollectionSet' element for the community and a 'Collection' element for our "CSTC Resources" collection. While DSpace has metadata on collections and communities for description, sidebar text, etc., the existing DSpace instance did not make use of any of these. We leave these fields blank in our specification, but do include them as they are required to adhere to our model.

The collection for "CSTC Resources" includes some additional information that helps us know how to generate the collection. Authorization policies for the collection are listed within the 'Authorization' element, which give DSpace understanding of who should be able to have what access rights on this collection. Also, the collection specification includes a 'Source' element, which our generation process uses to import existing content into DSpace. For this element we provide the local location of the content we were provided by the CITIDEL team.

5.2.2. Society Specification

The society aspect of our DL model always represents the users that are involved in the system. Here, as mentioned, we did not have much information about the types of users or number of users that the system used. To give an example of such generation, we decided to generate one administrative user and five regular users. In specification, administrative users come across in the 'Managers_Society', each with their own 'Manager' node which encapsulates all relevant data elements needed for such users in DSpace. Similarly, each regular user we wanted the generated CITIDEL site to have is represented using 'Actor' elements within the 'Actors_Society'. The regular users who we decided would be student editors in our CITIDEL instance are denoted as 'GroupMember' elements referenced by email address within the 'Group' specification for "studentEditors".

5.3. Generation

The generation process given our specification for the CITIDEL CSTC collection, went very smoothly and when finished left a working DSpace installation that met all the criteria specified. The generation process began by the generator tool being called to generate a DSpace instance with the passed in XML specification:

```
./dsrun dlGen dspace dspace_instance.xml
```

The `dsrun` command is a helper script derived from a DSpace script by the same name that calls the specified Java class with the appropriate class path for DSpace code to properly link and execute. Once the tool began and all aspects of the specification were checked for syntactic and logical correctness, the actual generation process began. The Scenario and Space aspects of the model were set by classes by the same names. The process prompts the user that compilation of DSpace is about to begin and warns that the process may take a few minutes.

The process continued. As DSpace compiled, the output of that compilation gets displayed to the user on the screen. After compilation was complete, the user was informed that the compiled WAR files are being copied to Apache Tomcat's web applications directory. When this was complete, the user was told that Apache was restarting to install those web applications. After the restart completed, DSpace was then live. The rest of the generation process ran through Stream, Society, and Structure generation which make use of DSpace API code to manipulate and create content within the DL. As users were created, users added to groups, and communities and collections were added, the user was given notification of the current progress. As each collection was added, any importing of content occurs at that point, by far the longest part of the generation process. In total, from specification to DSpace being live and the generation complete, the process took a little over 50 minutes. Table 7 lists approximate times that each part of our generation took to complete.

Table 7 - Generation Process Elapsed Time

Generation Aspect	Elapsed Time
Create Specification	15 minutes
Scenarios and Spaces	10 seconds
DSpace configuration	5 seconds
DSpace compilation	4 minutes
Streams, Societies, Structures	15 seconds
Import Collection Content	30 minutes

5.4. Reflections

There are a few differences in the original DL and our generated version, which is not surprising given that in our generator we have implemented only the most widely used areas of DSpace for generation. On the existing CITIDEL DSpace instance, they have manually suppressed the RSS feeds area from showing in the bottom of the right hand sidebar whereas they appear on our generated version, given it is the typical behavior of DSpace with RSS feeds enabled. Also, the existing CITIDEL version has a patch installed that shows the number of items in each community listed on the home page, whereas our generated version did not include this patch. The live CITIDEL site makes use of the Subject Search feature, which is shown on the left hand sidebar just under "Advanced Search"—our generator does not include all that is required to enable this functionality. Lastly, given their reliance on subjects in the existing CITIDEL implementation, they have adjusted the browse listings to include a column for subjects. While we allow the specification of items to be shown while browsing, this only applies to the view of individual items, not when multiple items are shown.

In general, this case study has successfully generated a working DSpace instance (which we show in Figure 23) with all the content from the CSTC collection. As described, there are some differences between the DSpace instances due to the use of items that we cannot automatically generate or aspects that were custom coded by the existing CITIDEL team. While in this case we were copying the nature of an existing DL using our process, this would likely not always be the case. On the contrary, we envision the tool would more likely be used either as a means to quickly get off the ground with a

new digital library or as a method to explore and learn about DSpace and digital libraries in general.

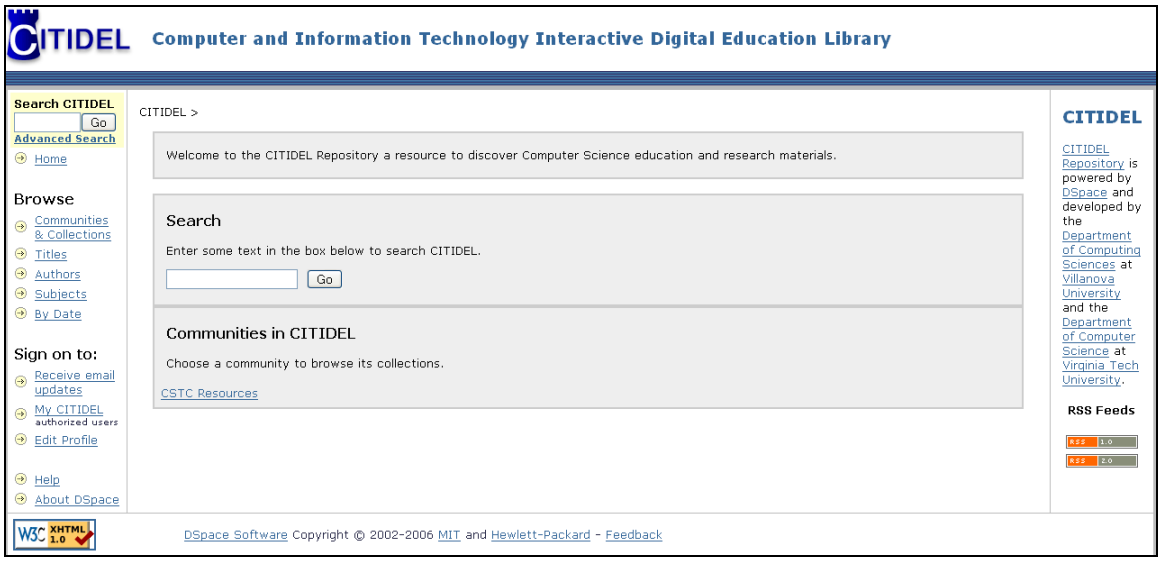

Figure 23 - Home page of our generated CITIDEL CSTC collection

Chapter 6. Analysis

Here we have described a way of specifying digital library systems for the DSpace repository software system. We have drawn much on previous work in digital library specification and generation and incorporated much in our model for DSpace generation. In this section, we provide some reflections on our efforts in DL specification and generation and a glimpse forward at some of the broader directions that these areas are moving towards based on this and other work.

6.1. General Reflections

Throughout our work here with DSpace, we have continually applied past work and our own perceptions about specification and generation which has come together to build a working generator that accepts DSpace DL specifications. However, based on the choices that were made throughout this work, our model has taken on a specific character. Given a separate look at DSpace, there may have been different aspects of the software that were given greater focus or perhaps a different division of the aspects of DSpace with respect to 5S. Given other case studies that focused on different aspects of the software, those functionalities may have been more likely to be included in our initial model here. From this standpoint alone, there is much possibility for future work with both DSpace and other platforms and architectural concerns. We examine some possibilities for future work in Chapter 7.

The architecture we use in this work uses a simple division into classes based on the 5S's described in the 5S framework for digital libraries. That class structure for DSpace generation, paired with the extensible nature of our dlGen tool, provides a combination of functionality that not only can generate DSpace instances but also can be used as a basis for future generation with other DL platforms. One goal in creating this method of specifying and generating DLs was to not only provide further exploration related to 5S and applying that framework and related work to a mature, popular DL software platform but also to provide something meaningful to the DSpace community. We believe that the work completed here that helps ease some of the overhead of

installing and creating repositories in DSpace will be of use to DSpace users, and we will get a better picture of that when we release the work to the DSpace community.

One of the challenges faced when researching and completing this work could almost be considered as information overload from the DSpace community. The software is so grand with all its facets and functionalities that as we examined its nature it was continually difficult to decide which aspects we should try to include in our model. Though added functionality could come out of it, we are still quite hesitant to include low level tasks that involve drastic changes to source code due to the ever changing aspects of computer code, especially in an open source project such as DSpace. Similarly, the average of five listserv digest emails from the project daily with user problems, perceptions, and questions only added to this nature of compounding amounts of information. At some point we needed to decide on a set of aspects of cover in our specifications and generation and leave others to future work. While we do believe we've covered the most commonly used and most important aspects of the software, yet since no DL platform provides adequate functionality for all users, our choices may also not fit all users' needs best. We detail some of these options under future work in Chapter 7.

Throughout this research we were constantly planning, designing our way of specifying DLs, and coding pieces of the generator to add to the greater tool. While through our testing, as we completed aspects of generation, we learned that our tool functioned properly, the results of the case study we provide in Chapter 5 were both interesting and surprising. While in testing we saw small pieces of the generator creating proper structures and elements in DSpace, it was only when the technique was used to specify and generate a more real life repository that the success of the work was truly revealed. All the planning and work on generation tasks actually yielded something of use when the finished tool was run against a DL specification. In that period of excitement, another result was also clearer—overall the generation process is not fast. Given the small size of the collection included in our case study, the import time for that content was near 30 minutes, an astounding length of time for such a small collection. Compilation, too, was a surprising four minutes, a testament to the size and complexity of DSpace. It is true that the time consuming aspects of the process were out of the scope of our own generation classes. However, that length of time speaks to the need of further

refinement in the software that, if nothing else, provides much faster import throughput, which will greatly speed up the overall process.

6.2. Towards a More General DL Specification

DSpace is not the only system available that we can use for the specification and generation of DLs. While it is exciting work to have a way of laying out a library for automatic generation in DSpace, not everyone is interested in using DSpace and naturally there are limits to its usefulness in a broader context. Greenstone is somewhat similar to DSpace with regard to general functionality, maturity, and user base so it is a perfect candidate for the same kind of work. In examining such a task broadly at this point, we have noticed much similarity between the two systems, especially given the up and coming Greenstone 3 which moves away from their traditional C++ implementation toward a Java version that also runs on Tomcat. Although they are referenced differently on the backend, one of the benefits of a specification strategy is that the implementation details can be hidden while showing the full nature of a DL logically. At this level, there are commonalities that can be represented in the same way, leading to a more generalized and more applicable DL model.

Optimally, a model for practical digital libraries is not one that can generate a specific software system, or group of software systems. Though we greatly draw from the previous work involving the specification of DLs in 5S and 5SL, that work is very formal in nature, and all encompassing while our work with DSpace specification and generation is much narrower in scope. We chose a sampling of the most commonly used aspects of DSpace that are relatively easy to implement, through programmatic generation of structures and content within the DL. This work provided a proof of concept that a modern, practical digital library system could be specified in a textual, XML-based way and subsequently be programmatically generated. Given the nature of a pre-made, out-of-the-box solution for DL creation like DSpace, flexible control over functionality and the creation of new functionality (Scenario-related aspects) is difficult. Our model includes no aspects of this important part of 5S, largely driven by the constraints of pre-made functionality. Optimally, a specification could include details that include services to turn

on or off within the target DL system or even perhaps a way to create new services simply by specifying what is desired.

Why is this difficult? There are lots of different types of digital library systems in existence, between research and commercial products. Many different systems lead to much division among how systems are oriented, what functionality each provides, what does the system architecture entails, or even how open are the implementations. Perhaps the largest barrier is the difference between monolithic versus distributed, componentized systems. The architectures of these systems are very different; the systems operate very differently to provide functionality to users. How does a specification model get developed that is flexible enough to represent these varying types of DL systems? How do you intuitively model a system with twenty servers with the same language as an architecture that uses one? How can the nuances of very similar concepts between two DL systems be abstracted so that they can be specified in a common way while providing each individual system with enough information to generate those objects? We cannot provide answers to these questions. But we can provide a general direction.

One method to tackle these hard questions is the building of a model that is so expansive, so in-depth, that the nature of every individual DL system is included in its entirety, while still staying as generic as possible. It must be built as an ontology would be, examine one DL system, such as we have with DSpace, and note its nature, intricacies, architecture, and all other aspects. Create a model for that behavior and structure that captures all required details to configure any aspect of that software. Pick another software system, and examine its composition. Then map the second DL's nature to the first model, using already existent elements as much as possible but creating new elements when necessary. Continue repeating, over and over again, until all DL systems have been cataloged and a massive schema has been designed that can, theoretically, represent any digital library system in existence. After a while there will be fewer and fewer new elements to add due to the breadth and depth of the model that has been developed.

Another possible avenue to pursue in moving towards a more general means of specifying DLs is to rely heavily on the 5S framework for digital libraries [8], as we did in this work. 5S examines the nature of DL systems and classifies the different aspects of

DL systems into five groupings, all with separate but important characteristics. Instead of building an all encompassing global schema, as described above, we can use 5S as a common ground to map all DL systems. 5SL [37] and this work have provided small steps towards this goal, but this is still far from a complete means of specifying any DL system. If using these tools for DL specification we are able to create separate representations for many or any DL platform, to be useful for generation they must all follow a general pattern or adhere to a single schema, wherein lies much of the difficulty of such a task. The differences between details and configuration options that these different platforms require makes it difficult to anticipate a way of specifying any and all possible configurations, which further complicates the task.

6.3. Towards a More General DL Generation Framework

Not only can a more general specification for digital library generation be derived; a more general way of generating digital libraries would be similarly useful. This could happen on two levels, either within a particular generator for a DL platform, or on a more general level that can be applied to generators for all DL systems as well. In this work the way our generator is authored the low level implementation is aimed at successfully completing the tasks within DSpace and much less at elegance. Though we reuse code and functions within classes, there are similarities between certain aspects of different generation classes, like parsing elements and adding key value pairs to configuration files. A better way of handling these details would be to use a global class which contains helper functions for common tasks. Commonalities exist across DL platforms as well with regard to how things must be generated. Configuration files are often used with DL software, like many programs, which can also perhaps be abstracted to a higher level. DSpace's configuration files are a mix of plain text, key value pair oriented configuration files and XML ones; our examination of Greenstone has yielded a similar nature of configuration with both unstructured text oriented configuration and XML based ones. Perhaps the processing of generation tasks within a given system can be driven by a meta-definition declaration of sorts, marrying the importance of configuration files and using global functions for similar tasks. For example, with a

function that sets values in a system's configuration file we can envision a declaration of this to be:

```
ADDCONFIG itemName valueLocation
```

(where itemName is a named configuration item needed in a configuration file, in a known place in a DL specification denoted by valueLocation) If each configuration option that must be set for a DL platform is defined above with its corresponding location in a DL specification, we can even abstract out the code itself to a more general approach. In this way, if new configuration items are added to software in subsequent versions, they can be added to a textual file containing ADDCONFIG directives without the need to even recompile the generation code.

Digital libraries and DL generation aside, such a generation tool is simply an application like any other program, on a certain level. Although the eventual review of this type of programming must have a certain leaning towards DLs, more general architectural tools and techniques can be applied as well to give DL generation a more formal, generally applicable, and directed architecture. One such possible work is Kruchten's 4+1 view architectural model [49]. Just as the original DL specification work in 5SL used known standards in order to be more broadly applicable, the 4+1 view model defines four views plus the scenarios that make up a system and defines them graphically by using standard graphical architectural modeling techniques. Joined by scenarios to show how the different architectural views interact, the model gives a clear picture of the system's overall functionality, interactions, and process. Perhaps DL generation can use a similar technique to eventually model actual generator behavior graphically in addition to tools for the graphical specification of DLs. Similarly, the overall workflow of a generation process can be modeled using a language like XPDL (XML Process Definition Language) [50], which uses formal constructs to define the workflow process that is involved in software systems. Also, given its XML realization, XPDL may integrate well with previous and our work with XML based DL specifications. In the same light, while in this work we apply 5S mostly to the nature of the actual digital library itself that is specified and created, there is little reason why it cannot also be applied to the overall generation process.

Again, a more generalized look at the generation of DLs is not easy; it requires a broader and unbiased look at the tasks and processes involved in such programmatic configuration and requirements. Those concerns must be abstracted so that generation can be understood as the nature of what is to be done, not just how it is down on a lower, programmatic level. Whether a logical model to be used as a guide for such programmatic tasks, or a more directly model driven generation architecture, a more formal look is needed at DL generation.

We do not provide any answers here; we only provide a little direction and a glimpse at the big picture of DL specification and generation. Certainly a more directed approach for specification such as the 5S directed one we mention here is more desirable over the chaos of a global schema which simply gathers together all possibilities. At the same time, such an approach is costly by not merely joining individual groups of details for separate DL platforms together into a single, global schema; rather all aspects must be examined in the context of how they fit into 5S. Just like the overall direction of specification refinement, a similar need lies in the need for a more generalized understanding and framework that describes DL generation. Such a process should follow some known pattern or model that provides some more global understanding of the tasks that need to be processed instead of a structure based only on necessity.

Chapter 7. Conclusions and Future Work

In this work we presented a method for specification of the nature of digital library systems based on past work with 5S and 5SL for describing and specifying DLs. Not only does our system provide a proof of concept for generation of practical digital library systems with the DSpace software, it also can be invaluable to the DSpace community for the installation, configuration, and generation of DLs created with DSpace. For new users, such a tool would allow DLs to get off the ground much more quickly than if users needed to spend time rigorously investigating the software structure, configuration, and other nuances. Hopefully, this work is of use to DSpace users and those interested in using that software. Hopefully, this work also will stimulate new investigation of the generation to other DL platforms from textual specifications.

7.1. Contributions of This Work

- A fresh application of the 5S framework to the specification and generation of practical DLs
- A specification model that can be used to specify the nature of digital libraries in DSpace
- The creation of an extensible generation tool that can use specifications for DL systems to generate working instances of these systems
- The creation of generation classes to generate instances of DSpace digital libraries based on DSpace DL specifications matching our model
- A look towards what is required for a more general model for practical DLs
- An application of this work to a DL previously running in DSpace
- A metamodel for 5SGraph that can be used to specify DLs using our DSpace model

7.2. Future Work

While we have accomplished much in this work, there is still much to accomplish related to the specification and generation of digital library systems. Below we touch

briefly on four main classes of work that can be done to forward the area of DL generation.

7.2.1. Further Work with DSpace Generation

With regard to DSpace, although we covered the most common functionality there is still much that we did not touch. While we provided one case study showing the use of our tool, additional case studies and evaluations with different levels of DL designers would be beneficial. There are likely trends in the way the software is coded that would allow more low level modification that would better allow for the generation of more dynamic, specific DL implementations. Also, our strategy only targets first time installers of the software; if a user wants to make changes to their installation after the tool has been run once, it cannot be done using dlGen. Extension of our generation classes to allow incremental re-generation of DSpace DLs would be a nice addition to this work. A much deeper ability to specify UIs using Manakin would also be a useful area of work as well. The next generation DSpace software is slated to be released in the next couple of years and with it comes not only great added functionality, but also added generation tasks that can be tackled.

7.2.2. More Work with Greenstone, Fedora, and Other DLs

DSpace is simply the tip of the iceberg. While we have done some investigation regarding Greenstone generation, the software is ever-changing and will need repetitive evaluations in order to ensure all needed functionality can be generated. Perhaps a good middle ground is to develop one model that can define DL specifications that can accurately be applied to both DSpace and Greenstone generation equally well. Given these two products' place in the community, not only would it be a good first step towards a greater, more expansive model, but it would likely cover a reasonable percentage of DL developers' needs. However, since both Greenstone and DSpace are relatively monolithic systems, a compelling additional software product to investigate regarding generation of DLs would be Fedora, which is entirely componentized and relies heavily on web services. Many would say that Fedora is the next up and coming product

to hit the community and there would be certain benefits for generation research for the software.

7.2.3. The Big Picture for DL Specification

As we discussed, the eventual development of an all encompassing framework for the specification of any DL system would be the crowning achievement for this type of work. In addition to this proposed work being developed from the ground up, it could also be considered a survey of all work in DL generation in the future for which trends and conclusions could be drawn. However, in any way it might be approached, this is not an easy task.

7.2.4. The Big Picture for DL Generation

Just as there needs to be eventual movement towards a broadly applicable model for digital library specification, a more framework oriented approach for the generation of DLs based on specifications is also a direction that would allow for easier, more consistent generation. While DL generation involves different aspects of the target DL and DL platform, similar elements exist throughout various platforms that suggest a more organized, generalized approach to DL generation based on similar generation tasks.

Appendix A. Full schema for our DSpace model

In this section, we provide the full schema representing what we consider for generation in the DSpace software. All specifications that are made to be generated with our system must adhere to this schema with regard to element structure as well as required elements and attributes. Diagrams for the different sub-models in our schema are provided in Section 3.3 of this document. (Note: Due to the length of lines often in XML documents, the presentation of this schema here is difficult to view. In some cases we left align elements and their children that would not normally be left aligned given their place in the XML structure, but it aids in making the schema slightly more readable.)

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<!— Top level element representing a model for a DL -->
<xs:element name="DLModel">
      <xs:complexType>
      <xs:sequence>
            <!— Digital library element -->
            <xs:element name="Digital_Library">
                  <xs:complexType>
                        <xs:sequence>
                        <!— DSpace configuration details are denoted
                        here and used pre-compilation -->
                        <xs:element name="InstalledDirectory"
                        type="xs:string"/>
                        <xs:element name="SourceDirectory"
                        type="xs:string"/>
                        <xs:element name="JDBCPath" type="xs:string"/>
                        <xs:element name="URL" type="xs:string"/>
                        <xs:element name="Hostname" type="xs:string"/>
                        <xs:element name="Port" type="xs:integer"/>
                        <xs:element name="DBType" type="xs:string"/>
                        <xs:element name="DBDriver" type="xs:string"/>
                        <xs:element name="DBHostname"
                        type="xs:string"/>
                        <xs:element name="DBPort" type="xs:integer"/>
                        <xs:element name="DBUser" type="xs:string"/>
                        <xs:element name="DBPass" type="xs:string"/>

                        <xs:element name="SMTPServer"
                        type="xs:string"/>
                        <xs:element name="SMTPAuthUser"
                        type="xs:string"/>
                        <xs:element name="SMTPAuthPass"
                        type="xs:string"/>
                        <xs:element name="FromEmailAddr"
                        type="xs:string"/>
```

```
                        <xs:element name="FeedbackEmailAddr"
                        type="xs:string"/>
                        <xs:element name="AdminEmailAddr"
                        type="xs:string"/>
                        <xs:element name="HandlePrefix"
                        type="xs:string"/>
                        <xs:element name="PatchPath" type="xs:string"/>

<!— Stream sub-model, stores file stream related details -->
<xs:element name="Stream_Model">
      <xs:complexType>
            <xs:sequence>
            <!— Streams represent DSpace Known Bitstream Formats -->
            <xs:element name="Stream">
                  <xs:complexType>
                        <xs:sequence>
                        <xs:element name="Content-Type"
                        type="xs:string"/>
                        <xs:element name="Extensions" maxOccurs="1">
                              <xs:complexType>
                                    <xs:sequence>
                                          <xs:element name="Extension"
                                          type="xs:string"
                                          maxOccurs="unbounded"/>
                                    </xs:sequence>
                              </xs:complexType>
                        </xs:element>
                        <xs:element name="Description"
                        type="xs:string"/>
                        </xs:sequence>
                        <xs:attribute name="name" type="xs:string"
                        use="required"/>
                  </xs:complexType>
            </xs:element>
            </xs:sequence>
      </xs:complexType>
</xs:element>
<!— Structure sub-model, represents the content holding structure of
the DL -->
<xs:element name="Structure_Model">
      <xs:complexType>
            <xs:sequence>
            <!— CollectionSet element represents DSpace Communities -->
            <xs:element name="CollectionSet" maxOccurs="unbounded">
                  <xs:complexType>
                        <xs:sequence>
                        <xs:element name="Description"
                        type="xs:string"/>
                        <xs:element name="IntroText" type="xs:string"/>
                        <xs:element name="CopyrightText"
                        type="xs:string"/>
                        <xs:element name="SidebarText"
                        type="xs:string"/>
                        <xs:element name="LogoPath" type="xs:string"/>
                        <!— Collection elements represent DSpace
                        collections -->
                        <xs:element name="Collection"
```

```
                        maxOccurs="unbounded">
                              <xs:complexType>
                                    <xs:sequence>
                                    <xs:element name="LogoPath"
                                    type="xs:string"/>
                                    <xs:element name="Description"
                                    type="xs:string"/>
                                    <xs:element name="IntroText"
                                    type="xs:string"/>
                                    <xs:element name="CopyrightText"
                                    type="xs:string"/>
                                    <xs:element name="SidebarText"
                                    type="xs:string"/>
                                    <xs:element name="Source"
                                    type="xs:string"/>

<!— Holds authorization policies -->
<xs:element name="Authorization">
      <xs:complexType>
            <xs:sequence>
            <xs:element name="Policy" maxOccurs="unbounded">
                  <xs:complexType>
                        <xs:sequence>
                        <xs:element name="Action" type="xs:string"/>
                        <xs:element name="Group" type="xs:string"/>
                        </xs:sequence>
                  </xs:complexType>
            </xs:element>
            </xs:sequence>
      </xs:complexType>
</xs:element>

                                    </xs:sequence>
                                    <xs:attribute name="name"
                                    type="xs:string" use="required"/>
                              </xs:complexType>
                        </xs:element>
                        </xs:sequence>
                        <xs:attribute name="name" type="xs:string"
                        use="required"/>
                  </xs:complexType>
            </xs:element>
            </xs:sequence>
      </xs:complexType>
</xs:element>
<!— Society sub-model, holds user and group related details -->
<xs:element name="Society_Model">
      <xs:complexType>
            <xs:sequence>
            <!— holds DSpace Administrator users -->
            <xs:element name="Managers_Society">
                  <xs:complexType>
                        <xs:sequence>
                        <xs:element name="Manager" maxOccurs=
                        "unbounded">
                              <xs:complexType>
                                    <xs:sequence>
```

```
                              <xs:element name="Email"
                              type="xs:string"/>
                              <xs:element name="Password"
                              type="xs:string"/>
                              <xs:element name="PhoneNumber"
                              type="xs:string"/>
                              <xs:element name="FirstName"
                              type="xs:string"/>
                              <xs:element name="LastName"
                              type="xs:string"/>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <!— holds DSpace regular users -->
      <xs:element name="Actors_Society">
            <xs:complexType>
                  <xs:sequence>
                  <xs:element name="Actor" maxOccurs="unbounded">
                        <xs:complexType>
                              <xs:sequence>
                              <xs:element name="Email"
                              type="xs:string"/>
                              <xs:element name="Password"
                              type="xs:string"/>
                              <xs:element name="PhoneNumber"
                              type="xs:string"/>
                              <xs:element name="FirstName"
                              type="xs:string"/>
                              <xs:element name="LastName"
                              type="xs:string"/>
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      <!— represent DSpace groups -->
      <xs:element name="Groups">
            <xs:complexType>
                  <xs:sequence>

<xs:element name="Group">
      <xs:complexType>
            <xs:sequence>
                  <xs:element name="GroupMember" maxOccurs=
                  "unbounded">
                        <xs:complexType>
                        <xs:simpleContent>
                              <xs:extension base="xs:string">
                              <xs:attribute name="type"
                              use="required">
                              <xs:simpleType>
                                    <xs:restriction
                                    base="xs:string">
```

```
                                    <xs:pattern
                                    value="User|Group"/>
                                    </xs:restriction>
                              </xs:simpleType>
                              </xs:attribute>
                              </xs:extension>
                        </xs:simpleContent>
                        </xs:complexType>
                  </xs:element>
            </xs:sequence>
            <xs:attribute name="name" type="xs:string"
            use="required"/>
      </xs:complexType>
</xs:element>

                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      </xs:sequence>
</xs:complexType>
</xs:element>
<!— Scenario sub-model, allows services to be enabled or disabled -->
<xs:element name="Scenario_Model">
      <xs:complexType>
            <xs:sequence>
            <xs:element name="EnableRSS" type="xs:boolean"
            maxOccurs="1" />
            </xs:sequence>
      </xs:complexType>
</xs:element>
<!— Space sub-model, stores UI and Information Retrieval details -->
<xs:element name="Space_Model">
      <xs:complexType>
            <xs:sequence>
            <xs:element name="UI">
                  <xs:complexType>
                        <xs:sequence>
                        <xs:element name="LogoImgPath" type="xs:string"
                        />
                        <xs:element name="HeaderPath" type="xs:string"
                        />
                        <xs:element name="SidebarPath" type="xs:string"
                        />
                        <xs:element name="NewsTopPath"
                        type="xs:string"/>
                        <xs:element name="NewsSidePath"
                        type="xs:string"/>
                        <xs:element name="CSSPath" type="xs:string"/>
                        <xs:element name="Manakin" minOccurs="0">
                              <xs:complexType>
                                    <xs:sequence>
                                    <xs:element name="DLDescription"
                                    type="xs:string" />
                                    <xs:element name="DLPublisher"
                                    type="xs:string" />

      <xs:element name="Aspects">
```

```
      <xs:complexType>
            <xs:sequence>
            <xs:element name="Aspect" maxOccurs="unbounded">
                  <xs:complexType>
                        <xs:attribute name="name"
                        type="xs:string" use="required"/>
                        <xs:attribute name="path"
                        type="xs:string" use="required"/>
                  </xs:complexType>
            </xs:element>
            </xs:sequence>
      </xs:complexType>
</xs:element>
<xs:element name="Themes">
      <xs:complexType>
            <xs:sequence>
            <xs:element name="Theme" maxOccurs="unbounded">
                  <xs:complexType>
                        <xs:attribute name="name"
                        type="xs:string" use="required"/>
                        <xs:attribute name="regex"
                        type="xs:string" use="required"/>
                        <xs:attribute name="path"
                        type="xs:string" use="required"/>
                  </xs:complexType>
            </xs:element>
            </xs:sequence>
      </xs:complexType>
</xs:element>

                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="BrowseItemsToShow">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element name="Item"
                                    type="xs:string"
                                    maxOccurs="unbounded" />
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>

      <xs:element name="IR">
            <xs:complexType>
                  <xs:sequence>
                  <xs:element name="IndexFields">
                        <xs:complexType>
                              <xs:sequence>

<xs:element name="Field" maxOccurs="unbounded">
      <xs:complexType>
      <xs:simpleContent>
            <xs:extension base="xs:string">
```

```
                  <xs:attribute name="name" use="required">
                        <xs:simpleType>
                              <xs:restriction base="xs:string" />
                        </xs:simpleType>
                  </xs:attribute>
                  </xs:extension>
            </xs:simpleContent>
            </xs:complexType>
      </xs:element>

                                    </xs:sequence>
                              </xs:complexType>
                        </xs:element>
                        </xs:sequence>
                  </xs:complexType>
            </xs:element>
</xs:element>
                        </xs:sequence>
                        <xs:attribute name="name" type="xs:string"
use="required"/>
                  </xs:complexType>
            </xs:element>
            </xs:sequence>
      </xs:complexType>
</xs:element>
</xs:schema>
```

Appendix B. Schema from CITIDEL CSTC Collection Case Study

In Chapter 5 we provided a case study of our method being applied to an existing collection from the CITIDEL digital library [47], [48]. In this section we provide the full XML specification we created that describes the nature of the CITIDEL CSTC collection and adheres to the model for DSpace DLs provided in Appendix A. (Again, we add indentation in some places in the following XML in order to make the specification more readable.)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<DLModel xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="dspace_gen.xsd">
<Digital_Library name="CITIDEL">
      <InstalledDirectory>/dspace</InstalledDirectory>
      <SourceDirectory>/dspace-src/dspace-1.4.1-
      source</SourceDirectory>
      <JDBCPath>/home/dgorton/thesis/lib/postgresql-8.1-
      407.jdbc2.jar</JDBCPath>
      <URL>http://inti.gotdns.org:8095/dspace</URL>
      <Hostname>inti.gotdns.org</Hostname>
      <Port>8080</Port>
      <DBType>postgres</DBType>
      <DBDriver>org.postgresql.Driver</DBDriver>
      <DBHostname>localhost</DBHostname>
      <DBPort>5432</DBPort>
      <DBUser>dspace</DBUser>
      <DBPass>dspace</DBPass>
      <SMTPServer>mail.irealm.org</SMTPServer>
      <SMTPAuthUser>dgorton@domain.org</SMTPAuthUser>
      <SMTPAuthPass>password</SMTPAuthPass>
      <FromEmailAddr>dgorton@domain.org</FromEmailAddr>
      <FeedbackEmailAddr>dgorton@domain.org</FeedbackEmailAddr>
      <AdminEmailAddr>dgorton@domain.org</AdminEmailAddr>
      <HandlePrefix>10117</HandlePrefix>
      <PatchPath></PatchPath>
      <Stream_Model>
            <Stream name="Test">
                  <Content-Type>text/test</Content-Type>
                  <Extensions>
                        <Extension>tst</Extension>
                  </Extensions>
                  <Description>test format</Description>
            </Stream>

      </Stream_Model>
      <Structure_Model>
            <CollectionSet name="CSTC Resources">
                  <Description></Description>
                  <IntroText></IntroText>
```

```
            <CopyrightText>All copyrights are retained by the
            photographers.</CopyrightText>
            <SidebarText><![CDATA[]]></SidebarText>
            <LogoPath></LogoPath>
            <Collection name="CSTC Resources">
                  <LogoPath></LogoPath>
                  <Description></Description>
                  <IntroText></IntroText>
                  <CopyrightText></CopyrightText>
                  <SidebarText></SidebarText>
                  <Source>/home/dgorton/
                  GortonCSTCCollection</Source>
                  <Authorization>
                        <Policy>
                              <Action>READ</Action>
                              <Group>Anonymous</Group>
                        </Policy>
                        <Policy>
                              <Action>DEFAULT_ITEM_READ</Action>
                              <Group>Anonymous</Group>
                        </Policy>
                        <Policy>
                              <Action>DEFAULT_BITSTREAM_READ
                              </Action>
                              <Group>Anonymous</Group>
                        </Policy>
                        <Policy>
                              <Action>ADD</Action>
                              <Group>studentEditors</Group>
                        </Policy>
                  </Authorization>
            </Collection>
      </CollectionSet>
</Structure_Model>
<Society_Model>
      <Managers_Society>
            <Manager>
                  <Email>dgorton@domain.org</Email>
                  <Password>password</Password>
                  <PhoneNumber>540-230-6766</PhoneNumber>
                  <FirstName>Doug</FirstName>
                  <LastName>Gorton</LastName>
            </Manager>
      </Managers_Society>
      <Actors_Society>
            <Actor>
                  <Email>student1@domain.org</Email>
                  <Password>password</Password>
                  <PhoneNumber>540-230-6766</PhoneNumber>
                  <FirstName>Student</FirstName>
                  <LastName>One</LastName>
            </Actor>
            <Actor>
                   <Email>student2@domain.org</Email>
                   <Password>password</Password>
                   <PhoneNumber>540-230-6766</PhoneNumber>
                   <FirstName>Student</FirstName>
```

```
                <LastName>Two</LastName>
            </Actor>
            <Actor>
                <Email>student3@domain.org</Email>
                <Password>password</Password>
                <PhoneNumber>540-230-6766</PhoneNumber>
                <FirstName>Student</FirstName>
                <LastName>Three</LastName>
            </Actor>
            <Actor>
                <Email>student4@domain.org</Email>
                <Password>password</Password>
                <PhoneNumber>540-230-6766</PhoneNumber>
                <FirstName>Student</FirstName>
                <LastName>Four</LastName>
            </Actor>
            <Actor>
                <Email>student5@domain.org</Email>
                <Password>password</Password>
                <PhoneNumber>540-230-6766</PhoneNumber>
                <FirstName>Student</FirstName>
                <LastName>Five</LastName>
            </Actor>
      </Actors_Society>

      <Groups>
            <Group name="studentEditors">
                  <GroupMember
                  type="User">student1@domain.org</GroupMember>
                  <GroupMember
                  type="User">student2@domain.org</GroupMember>
                  <GroupMember
                  type="User">student3@domain.org</GroupMember>
                  <GroupMember
                  type="User">student4@domain.org</GroupMember>
                  <GroupMember
                  type="User">student5@domain.org</GroupMember>
            </Group>
      </Groups>
</Society_Model>
<Scenario_Model>
      <EnableRSS>true</EnableRSS>
</Scenario_Model>
<Space_Model>
      <UI>
            <LogoImgPath>citidel-logo.png</LogoImgPath>
            <HeaderPath>header-default.jsp</HeaderPath>
            <SidebarPath>sidebar.htm</SidebarPath>
            <NewsTopPath>newstop.htm</NewsTopPath>
            <NewsSidePath>newsside.htm</NewsSidePath>
            <CSSPath>styles.css.jsp</CSSPath>
            <BrowseItemsToShow>
                  <Item>dc.title</Item>
                  <Item>dc.title.alternative</Item>
                  <Item>dc.contributor.*</Item>
                  <Item>dc.subject</Item>
                  <Item>dc.date.issued(date)</Item>
```

```
                        <Item>dc.publisher</Item>
                        <Item>dc.identifier.citation</Item>
                        <Item>dc.relation.ispartofseries</Item>
                        <Item>dc.description.abstract</Item>
                        <Item>dc.description</Item>
                        <Item>dc.identifier.govdoc</Item>
                        <Item>dc.identifier.uri(link)</Item>
                        <Item>dc.identifier.isbn</Item>
                        <Item>dc.identifier.issn</Item>
                        <Item>dc.identifier.ismn</Item>
                        <Item>dc.identifier</Item>
                  </BrowseItemsToShow>
            </UI>
            <IR>
                  <IndexFields>
                        <Field name="author">dc.contributor.*</Field>
                        <Field name="author">dc.creator.*</Field>
                        <Field name="title">dc.title.*</Field>
                        <Field name="keyword">dc.subject.*</Field>
                        <Field name="abstract">
                        dc.description.abstract</Field>
                        <Field name="author">
                        dc.description.statementofresponsibility
                        </Field>
                        <Field name="series">
                        dc.relation.ispartofseries</Field>
                        <Field name="abstract">
                        dc.description.tableofcontents</Field>
                        <Field name="mime">dc.format.mimetype</Field>
                        <Field name="sponsor">
                        dc.description.sponsorship</Field>
                        <Field
                        name="identifier">dc.identifier.*</Field>
                        <Field name="language">dc.language.iso</Field>
                  </IndexFields>
            </IR>
      </Space_Model>
</Digital_Library>
</DLModel>
```

References

[1] L. Atkinson, "The Rejection of D-Space: Selecting Theses Database Software at the University of Calgary Archives," University of Calgary, Calgary, Canada, Technical Report 11/28/2006, 2006.

[2] L. Sheble, "Greenstone User Survey," School of Information and Library Science, University of North Carolina, Technical Report. 2006.

[3] J. M. Ockerbloom, "Toward the next generation: Recommendations for the next DSpace Architecture," DSpace Architecture Review Group, San Antonio, TX, Report 1/24/2007, 2007.

[4] R. Reddy and I. Wladawsky-Berger, "President's Information Technology Advisory Committee. Panel on Digital Libraries Digital Libraries: Universal Access to Human Knowledge," National Coordination Office for Information Technology Research and Development, 2001.

[5] D. Waters, "What Are Digital Libraries?," *CLIR Issues (Council on Library and Information Resources),* vol. 4, 1998.

[6] G. Marchionini and E. A. Fox, "Progress toward digital libraries: augmentation through integration" *Inf. Process. Manage.,* vol. 35, pp. 219-225, 1999.

[7] E. A. Fox. "DL Definitions - http://ei.cs.vt.edu/~dlib/def.htm." Retrieved 1/16/2007, from http://ei.cs.vt.edu/~dlib/def.htm. 2001.

[8] M. A. Gonçalves, E. A. Fox, L. T. Watson, and N. A. Kipp, "Streams, structures, spaces, scenarios, societies (5s): A formal model for digital libraries," *ACM Trans. Inf. Syst.,* vol. 22, pp. 270-312, 2004.

[9] L. Candela, D. Castelli, Y. Ioannidis, G. Koutrika, P. Pagano, S. Ross, H.-J. Schek, and H. Schuldt, "The Digital Library Manifesto," DELOS: A Network of Excellence on Digital Libraries, Pisa, Italy ISBN: 2-912335-24-8, 2006.

[10] "Digital Imaging Tutorial - Metadata." Retrieved 03/08/2007, from http://www.library.cornell.edu/preservation/tutorial/metadata/metadata-01.html. 2003.

[11] "The Dublin Core Metadata Initiative." Retrieved 1/15/2007, from http://dublincore.org/. 2007. Last Updated 01/15/2007.

[12] "Metadata Encoding and Transmission Standard (METS) Official Web Site." Retrieved 03/08/2007, from http://www.loc.gov/standards/mets. 2007. Last Updated 03/01/2007.

[13] H. Suleman and E. A. Fox, "Designing protocols in support of digital library componentization," in *Research and Advanced Technology for Digital Libraries. 6th European Conference, ECDL 2002. Proceedings, 16-18 Sept. 2002*, Rome, Italy, 2002, pp. 568-82 ISBN - 3-540-44178-6.

[14] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. "Extensible Markup Language (XML) 1.0 (Fourth Edition)." Retrieved 3/10/2007, from http://www.w3.org/TR/2006/REC-xml-20060816/. 2007. Last Updated 9/29/2006.

[15] C. M. Sperberg-McQueen and H. Thompson. "W3C XML Schema." Retrieved 3/23/2007, from http://www.w3.org/XML/Schema. 2000. Last Updated 1/23/2007.

[16] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, "Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI," *IEEE Internet Computing,* vol. 6, pp. 86-93, 2002.

[17] M. Brambilla, S. Ceri, S. Comai, M. Dario, P. Fraternali, and I. Manolescu, "Declarative specification of Web applications exploiting Web services and workflows http://doi.acm.org/10.1145/1007568.1007688 " in *Proceedings of the 2004 ACM SIGMOD international conference on Management of data* Paris, France ACM Press, 2004, pp. 909-910

[18] M. Smith, M. Barton, M. Bass, M. Branschofsky, G. McClellan, D. Stuve, R. Tansley, and J. H. Walker, "DSpace — An Open Source Dynamic Digital Repository," in *D-Lib Magazine*. vol. 9, 2003.

[19] "DSpace Federation." Retrieved 1/15/2007, from http://www.dspace.org/. 2007. Last Updated Jan-03-2007.

[20] R. Tansley, M. Bass, M. Branschofsky, G. Carpenter, G. McClellan, and D. Stuve. "DSpace System Documentation." Retrieved 4/1/2007, from http://dspace.org/technology/system-docs/. 2005. Last Updated 10/5/2005.

[21] T. Donohue and D. Salo, "DSpace How-To Guide," JCDL 2006, Chapel Hill, NC 6/11/2007.

[22] "Digital Initiatives: Manakin." Retrieved 1/15/2007, from http://di.tamu.edu/projects/xmlui/manakin/. 2005.

[23] I. H. Witten, R. J. McNab, S. J. Boddie, and D. Bainbridge, "Greenstone: a comprehensive open-source digital library software system," in *Proceedings of the Fifth ACM Conference on Digital Libraries, 2-7 June 2000*, San Antonio, TX, USA, 2000, pp. 113-21 ISBN - 1-58113-231-X.

[24] D. Bainbridge, K. J. Don, G. R. Buchanan, I. H. Witten, S. Jones, M. Jones, and M. I. Barr, "Dynamic digital library construction and configuration," in *Proceedings of Research and Advanced Technology for Digital Libraries, 8th European Conference, ECDL 2004*, Bath, UK, 2004, pp. 1-13.

[25] C. Lagoze, S. Payette, E. Shin, and C. Wilper, "Fedora: an architecture for complex objects and their relationships," *International Journal on Digital Libraries,* vol. 6, pp. 124-138, http://www.dlib.org/dlib/december01/suleman/12suleman.html. 2006.

[26] H. Suleman and E. A. Fox, "A framework for building open digital libraries," *D-Lib Magazine,* vol. 7, 2001.

[27] L. Eyambe and H. Suleman, "A digital library component assembly environment " in *Proceedings of the 2004 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries* Stellenbosch, Western Cape, South Africa South African Institute for Computer Scientists and Information Technologists, 2004, pp. 15-22

[28] P. Roberto, M. A. Gonçalves, and H. Alberto, "Design, Implementation, and Evaluation of a Pool of Web Services-Based Components for Building Digital Libraries," in *ECDL 2006*, Alicante, Spain, 2006.

[29] R. Santos, P. Roberto, M. A. Goncalves, and H. Alberto, "Design, Implementation, and Evaluation of a Wizard Tool for Setting Up Component-Based Digital Libraries," in *ECDL 2006*, Alicante, Spain, 2006.

[30] J. Y. L. Thong, W. Hong, and K. Y. Tam, "What leads to user acceptance of digital libraries?," *Communications of the ACM,* vol. 47, pp. 78-83, 2004/11/ 2004.
[31] D. M. Levy and C. C. Marshall, "Going digital: a look at assumptions underlying digital libraries," *Communications of the ACM,* vol. 38, pp. 77-84, 1995/04/ 1995.
[32] "DSpace Hardware Requirements." Retrieved 3/10/2007, from http://dspace.org/what/dspace-hp-hw.html.
[33] R. Wyles, et al., "Technical Evaluation of selected Open Source Repository Solutions v1.3," Open Access Repositories in New Zealand (OARINZ) project, Christchurch Polytechnic Institute of Technology 13 September 2006.
[34] "EduTools" Retrieved 1/15/07, from http://www.edutools.info/static.jsp?pj=4&page=LOR. 2007.
[35] I. H. Witten and D. Bainbridge, *How to Build a Digital Library*: Elsevier Science Inc., New York, NY, 2002.
[36] I. H. Witten, "Examples of practical digital libraries collections built internationally using Greenstone," *D-Lib Magazine,* vol. 9, 03/2003, 2003.
[37] M. A. Gonçalves and E. A. Fox, "5SL-a language for declarative specification and generation of digital libraries," *JCDL'02: Joint Conference on Digital Libraries, 14-18 July 2002*, Portland, OR, USA, 2002, pp. 263-72 ISBN - 1-58113-513-0.
[38] R. Shen, M. A. Goncalves, W. Fan, and E. Fox, "Requirements gathering and modeling of domain specific digital libraries with the 55 framework: an archaeological case study with ETANA," in *Research and Advanced Technology for Digital Libraries. 9th European Conference, ECDL 2005. Proceedings, 18-23 Sept. 2005*, Vienna, Austria, 2005, pp. 1-12 ISBN - 3-540-28767-1.
[39] U. Ravindranathan, R. Shen, M. A. Goncalves, W. Fan, E. A. Fox, and J. W. Flanagan, "ETANA-DL: managing complex information applications - an archaeology digital library," in *Proceedings of the Fourth ACM/IEEE Joint Conference on Digital Libraries, 7-11 June 2004*, Tucson, AZ, USA, 2004, pp. 414 BN - 1 58113 832 6.
[40] M. Abrams, C. Phanouriou, A. L. Batongbacal, S. M. Williams, and J. E. Shuster, "UIML: an appliance-independent XML user interface language," in *Proceeding of the eighth international conference on World Wide Web* Toronto, Canada: Elsevier North-Holland, Inc., 1999.
[41] J. Suzuki and Y. Yamamoto, "Making UML Models Interoperables with UXF," *Selected papers from the First International Workshop on The Unified Modeling Language «UML»'98: Beyond the Notation*: Springer-Verlag, 1999.
[42] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language user guide*: Addison Wesley Longman Publishing Co., Inc., 1999.
[43] Q. Zhu, "5SGraph: A Modeling Tool for Digital Libraries," *Computer Science*. Master of Science Degree, Blacksburg, VA: Virginia Tech, 2002.
[44] R. Kelapure, "Scenario-Based Generation of Digital Library Services," *Computer Science*. Master of Science Degree, Blacksburg, VA: Virginia Tech, 2003.
[45] M. A. Gonçalves, A. A. Zafer, N. Ramakrishnan, and E. A. Fox, "Modeling and Building Personalized Digital Libraries with PIPE and 5SL," in *43rd Joint DELOS-NSF Workshop on Personalization and Recommender Systems in Digital Libraries*, Dublin, Ireland, 2001, pp. 67-72.

[46] E. A. Fox, "From the WWW and minimal digital libraries, to powerful digital libraries: why and how," in *Digital Libraries: Implementing Strategies and Sharing Experiences. 8th International Conference on Asian Digital Libraries, ICADL 2005. Proceedings, 12-15 Dec. 2005*, Bangkok, Thailand, 2005, pp. 525 ISBN - 3-540-30850-4.

[47] "CITIDEL Homepage." Retrieved 3/25/2007, from http://www.citidel.org/. 2003. Last Updated 10/13/2003.

[48] "Computer and Information Technology Interactive Digital Education Library." Retrieved 3/25/2007, from http://citidel.villanova.edu/. 2007.

[49] P. Kruchten, "The 4+1 View Model of Architecture," *IEEE Software, v*ol. 12, pp. 42-50, 1995.

[50] P. Jiang, Q. Mair, and J. Newman, "Using UML to Design Distributed Collaborative Workflows: from UML to XPDL," in *Proceedings of the Twelfth International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*: IEEE Computer Society, 2003.