

5SGraph: A Modeling Tool for Digital Libraries

by
Qinwei Zhu

Thesis submitted to the Faculty of the Virginia Polytechnic Institute and
State University in partial fulfillment of the requirements for the degree of

**MASTER OF SCIENCE
in
Computer Science**

Prof. Edward A. Fox, Chair
Prof. H. Rex Hartson
Prof. Chris North

November 24, 2002
Blacksburg, Virginia

Keywords: digital libraries, domain-specific modeling tool,
5S model, metamodel, 5SGraph

5SGraph: A Visual Modeling Tool for Digital Libraries

Qinwei Zhu

(Abstract)

The high demand for building digital libraries by non-experts requires a simplified modeling process and rapid generation of digital libraries. To enable rapid generation, digital libraries should be modeled with descriptive languages. A visual modeling tool would be helpful to non-experts so they may model a digital library without knowing the theoretical foundations and the syntactical details of the descriptive language.

In this thesis, we describe the design and implementation of a domain-specific visual modeling tool, 5SGraph, aimed at modeling digital libraries. 5SGraph is based on a metamodel that describes digital libraries using the 5S theory. The output from 5SGraph is a digital library model that is an instance of the metamodel, expressed in the 5S description language (5SL). 5SGraph presents the metamodel in a structured toolbox, and provides a top-down visual building environment for designers. The visual proximity of the metamodel and instance model facilitates requirements gathering and simplifies the modeling process. Furthermore, 5SGraph maintains semantic constraints specified by the 5S metamodel and enforces these constraints over the instance model to ensure semantic consistency and correctness. 5SGraph enables component reuse to reduce the time and efforts of designers. The results from a pilot usability test confirm the usefulness of 5SGraph.

Acknowledgements

I would like to express my deepest gratitude to my advisor, Professor Edward A. Fox, for his unceasing support, direction, help and patience throughout this work. I would also like to thank Professor H. Rex Hartson and Professor Chris North, for their guidance and valuable and constructive suggestions on this work.

I give special acknowledgement to Marcos André Gonçalves, a PhD student of Dr. Fox, for his help and assistance during the past year. Without his help, this research work would not have been possible.

Finally, I greatly appreciate the insightful and constructive comments from all those who participated in my experiment.

Table of Contents

List of Figures	VII
List of Tables	IX
Chapter 1. Introduction.....	1
1.1 Problem Statement	1
1.2 Our Approach	5
1.3 Organization	6
Chapter 2. Related Work	8
2.1 Concepts, Definitions, and Examples of Digital Libraries	8
2.1.1 History of Digital Library Concepts	8
2.1.2 Definitions of Digital Libraries.....	11
2.1.3 Digital Library Examples	12
2.1.3.1 MARIAN	12
2.1.3.2 Open Digital Libraries	13
2.2 Modeling of Digital Libraries.....	14
2.2.1 Some Early Attempts at Digital Library Modeling.....	14
2.2.2 Greenstone's Model	15
2.2.3 Castelli's Models for Digital Libraries.....	16
2.2.4 5S Model	17
2.3 Construction of Digital Libraries.....	17
2.3.1 Greenstone	18
2.4 Related Tools.....	19
2.4.1 XML Modeling Tools	20
2.4.1.1 Limitations of Those XML Tools.	20
2.4.2 Domain-Specific vs. General Purpose Visual Modeling Tools.....	21
Chapter 3. Design of 5SGraph	23
3.1 Process of Building a Digital Library	23
3.2 Architecture of the 5SGraph.....	26
3.3 Brief Description of the 5S Model.....	28
3.3.1 Stream Model.....	29
3.3.2 Structural Model	30
3.3.3 Spatial Model.....	33
3.3.4 Scenario Model.....	34
3.3.5 Societal Model	34
3.4 Design Details of 5SGraph.....	35
3.4.1 Toolbox	36
3.4.2 Icons	39
3.4.3 Tree Representation	41
3.4.3.1 Node and Link Approach.....	42
3.4.3.2 Nested Approach.....	43
3.4.3.3 Reasons for Choosing Node-link Representation	44
3.4.4 Truncated Visualization of Full Trees in 5SGraph	45

3.4.4.1 Problems Caused by Deep Nodes	45
3.4.4.2 Visible Path and Consistent Lists.....	45
3.4.4.3 Truncated Display	47
3.5 User Interactions	48
3.5.1 Basic Operations	48
3.5.2 Synchronization	52
3.5.3 Component Reusability	54
3.6.3.1 Scenario of Component Reuse.....	55
3.6.3.2 Non-Reusable Components	56
3.6.3.3 Top-down or Bottom-up Methodology	56
Chapter 4. Metamodel	58
4.1 Requirements for the Metamodel	58
4.2 Overview of a Metamodel Example	59
4.3 DataSet	60
4.4 DataType	61
4.4.1 SubNodes and Icon	62
4.4.2 Property	66
4.5 Constraint Management	70
4.5.1 Value Constraint	71
4.5.2 Association Constraint	74
4.6 Summary	77
Chapter 5. Preliminary Test of 5SGraph	78
5.1 Test Objectives	78
5.2 Test Methods and Procedures.....	78
5.2.1 Participants	78
5.2.2 Apparatus.....	79
5.2.3 Tasks.....	79
5.2.4 Procedures	80
5.2.5 Test Measures	80
5.3 Results	81
5.3.1 Performance Results.....	81
5.3.2 Satisfaction Results	83
5.4 Discussion	84
5.4.1 Effectiveness.....	84
5.4.2 Efficiency	85
5.4.3 Satisfaction	86
5.4.4 Other Observations	87
Chapter 6. An Example Modeling Process Using 5SGraph.....	89
6.1 The Description of NDLTD	89
6.2 The Modeling Process for NDLTD	91
Chapter 7. Conclusions.....	98
7.1 Contributions of 5SGraph	98
7.2 Future Work	99
7.2.1 Integration with Other Tools	99
7.2.2 Extensions to the Visualization Functionality	100
7.2.3 Extensions to the Interaction Functionality	100

7.2.4 Better Evaluation Test.....	100
References.....	102
Appendix A. Implementation of 5SGraph.....	106
A.1 UML Overview.....	106
A.2 Model-view separation.....	109
A.3 Constraint Management	111
Appendix B. Metamodel.....	112
Appendix C. The 5SL File for NDLTD	120
Appendix D. Documents for the Experiment	123
D.1 Informed Consent.....	123
D.2 Task Descriptions.....	125
D.2.1 Task One.....	125
D.2.2 Task Two	127
D.2.3 Task Three	129
D.3 Questionnaire.....	132

List of Figures

Figure 2.1 Architecture of MARIAN [adapted from [DLRL02]].....	13
Figure 2.2 ODL Network [adapted from [Suleman02]].....	14
Figure 3.1 Building Digital Libraries with the 5S Model	23
Figure 3.2 The Digital Library Generation Process with 5SGraph.....	25
Figure 3.3 Relationship Between Model and Metamodel	26
Figure 3.4 The Interaction of 5SGraph with Other Tools (Future Work)	28
Figure 3.5 Overview of the 5S model	29
Figure 3.6 The Hierarchy of the Stream Model.....	29
Figure 3.7 The Hierarchy of the Structural Model.....	30
Figure 3.8 A StructuredStream for an ETD [adapted from [GonçalvesTOIS]].....	32
Figure 3.9 The Hierarchy of the Spatial Model	33
Figure 3.10 The Hierarchy of the Scenario Model	34
Figure 3.11 The Hierarchy of the Societal Model.....	35
Figure 3.12 A Screen Shot of a Concept Map	37
Figure 3.13 The Toolbox of BeanBox.....	38
Figure 3.14 The Workspace and the Structured Toolbox in 5SGraph	39
Figure 3.15 Visual Components without Icons and Cardinality.....	40
Figure 3.16 Visual Components with Icons and Cardinality.....	41
Figure 3.17 A Node-link Representation by a JTree.....	43
Figure 3.18 Node-Link and Treemap Representations of a Tree.....	43
Figure 3.19 Node-link Representation has Clear Depth Information	44
Figure 3.20 A Tree with One Visible Branch at Each Level.....	45
Figure 3.21 Consistent Lists	47
Figure 3.22 The Operation of Exploring	49
Figure 3.23 The Operation of Exploring	50
Figure 3.24 The Scenario of Adding	51
Figure 3.25 The Operation of Changing Properties	52
Figure 3.26 The User Adds an Instance of <i>Stream_Model</i>	54
Figure 3.27 The User Adds an Instance of <i>Struct_Model</i>	54
Figure 3.28 The Scenario of Component Reuse	56
Figure 3.29 Create an Instance of the Stream Model Using 5SGraph	57
Figure 4.1 Overview of a Metamodel	60
Figure 4.2 MetaData_Content.....	61
Figure 4.3 Image_Content and Audio_Content	61
Figure 4.4 Digital_Library and its Visual Notation	62
Figure 4.5 Stream_Model and its Visual Notation.....	63
Figure 4.6 Screen Shot for Scenario 1.....	64
Figure 4.7 Screen Shot for Scenario 2.....	66
Figure 4.8 Digital_Library and its Property Sheet	67
Figure 4.9 Audio and its Property sheet	68
Figure 4.10 Catalog and its Property Sheet	69
Figure 4.11 Ontology and its Property Sheet.....	70
Figure 4.12 Actor and Services.....	71

Figure 4.13 Screen Shot 1 for Scenario 3	72
Figure 4.14 Screen Shot 2 for Scenario 3	73
Figure 4.15 Screen Shot 3 for Scenario 3	73
Figure 4.16 Screen Shot 4 for Scenario 3	74
Figure 4.17 The Declaration of the Catalog	74
Figure 4.18 Screen Shot 1 for Scenario 4	75
Figure 4.19 Screen Shot 2 for Scenario 4	76
Figure 5.1 The Comparison of Pre and Post Understanding of 5S Theory	87
Figure 6.1 Load a Metamodel	91
Figure 6.2 Add a Digital Library – NDLTD	92
Figure 6.3 Add the Stream Model to NDLTD	93
Figure 6.4 Add the ETDCollection to the Structural Model	94
Figure 6.5 Add the ETDCatalog to the Structural Model	94
Figure 6.6 Add the Scenario Model into NDTLD	95
Figure 6.7 Add the Societal Model into NDLTD	96
Figure 6.8 Associate the Services with Actors and Managers	97
Figure A.1 High Level of Structure of Core 5SGraph Components	106
Figure A.2 Illustration of N-ary Tree	107
Figure A.3 An Example of n-ary Tree	108
Figure A.4 DownList/UpList classes	109
Figure A.5 Model-view Separation	110
Figure A.6 NodePropEditor Class	110
Figure A.7 Constraint Management	111

List of Tables

Table 3.1 SSL Descriptions of Streams30

Table 5.1 Performance Results for Task 1.....82

Table 5.2 Performance Results for Task 2.....82

Table 5.3 Performance Results for Task 3.....83

Table 5.4 Overall Performance Results for Three Tasks83

Table 5.5 Rating Data.....84

Chapter 1. Introduction

1.1 Problem Statement

With the advent of the Internet and the World Wide Web (WWW), the digital library (DL) field has emerged as an important application area. Distinct from traditional libraries, digital libraries process large collections of digital objects and provide on-line information services. They are very important for archiving and utilizing human knowledge records in the new networked world.

The importance and challenges of digital libraries have attracted many researchers. Some of the well-known digital library related research areas include classification, interoperability between heterogeneous collections, communication protocols and standards, search engines, information visualization, usability, and human computer interaction issues [Fox95]. While tremendous attention has been paid to the study of how to make a better digital library, very little focus has been on simplifying the process of building a digital library.

A digital library is a complex information system. It is an integration of many application fields of computer science such as information retrieval, databases, and hypertext. To build a digital library, many questions need to be answered: what is the specification of the content to be stored; how is that content organized, structured, described, and

accessed; what kinds of services are offered (e.g., searching, browsing, personalizing, collaborating); how do patrons use those services and interact with each other in the DL environment [Gonçalves02]. Until now, none of those questions have been answered perfectly. Much research needs to be done. Accordingly, it is difficult and time-consuming to build a new system right now.

Meanwhile, the demand for new digital libraries is strong. Hundreds of digital libraries have been built around the world, and hundreds of digital library projects are ongoing. Different user communities need different digital libraries to satisfy their requirements. Many existing digital libraries are monolithic, tightly integrated, inflexible, and lack interoperability connections with each other. It usually takes a huge amount of effort and time to create a digital library that satisfies a specific need. Furthermore, designers of digital libraries often are not experts in digital libraries. They may be on the library technical staff, computer scientists, or high school teachers. They may lack knowledge in either software engineering or information science.

We need to be able to model a generic digital library properly before we are able to make the building process easy. The multidisciplinary nature of digital libraries makes the generic modeling of a digital library very difficult. Different perspectives have led to different concepts of digital libraries. That is the reason why we have so many definitions of digital libraries [Chapter 15, Baeza-Yates99].

Recognizing the difficulties in modeling digital libraries, Gonçalves, Fox, et al. have proposed the 5S theory [GonçalvesTOIS]. 5S represents Streams, Structures, Spaces, Scenarios, and Societies. The Streams Model specifies the communication content between digital libraries and users. The Structures Model specifies how to organize information in usable ways. The Spaces Model specifies how to present information in retrievable and usable ways. The Scenarios Model specifies available information services. Finally, the Societies Model specifies how the digital library satisfies users' demands for information.

5S provides a formal model to capture the complexities of digital libraries. The formality of that model makes it possible to unambiguously specify the characteristics and behaviors of digital libraries. The formality also enables automatic mapping from 5S models to actual implementations.

Gonçalves and Fox also proposed 5SL, a language for declarative specification and generation of digital libraries [Gonçalves02]. 5SL is an XML realization of the 5S model. It is a high-level, domain-specific language. It is specific to digital libraries and represents digital libraries at a very fine granularity. If a digital library generator is to be built with the 5SL language, a semi-automatic digital library generation process would include the following steps:

1. The designer of the digital library writes a 5SL specification that captures the requirements for a specific digital library.

2. The digital library generator is fed with the 5SL file and generates a digital library for the designer.

The designer does not need to be an expert in software engineering or information science. The designer only needs to have a clear conceptual picture of the needed digital library and be able to transform the conceptual picture to 5SL files. This greatly reduces the burden on designers, speeds up the building process, and increases the quality of the digital libraries built.

However, 5SL has its own problems and limitations.

1. The designer must understand 5SL well enough to be able to write a 5SL file and to correctly use it to express his/her ideal digital library.
2. The 5SL file, which represents a digital library, consists of five sub-models (Stream model, Structural model, Spatial model, Scenarios model, and Societal model). Although all of the five sub-models are expressed in XML, they use different sets of concepts and have different semantics. These differences make a 5S model compatible and extensible, because many existing standard formats can be used in the 5S model. However, it is frustrating in that to build one digital library, the designer needs to understand five or more different semantic specifications to express the system.

3. When large and complex digital libraries are to be built, it is very hard even for experts to manually write those XML files without any assistance from a tool.

4. It is very difficult to obtain the big picture of a digital library just from a huge set of XML files. This inconvenience may cause troubles for maintenance, upgrade, or even understanding of an existing system.

We need to overcome these disadvantages of 5SL to let people really appreciate the benefits of 5S/5SL. We discuss our approach in the next section.

1.2 Our Approach

Our approach to simplify the building process of digital libraries is to separate the designers of digital libraries from the technical details of the 5SL. We achieve this by introducing and developing a graphical modeling tool. This thesis presents such an innovative modeling tool, 5SGraph, developed for the purpose of building digital libraries based on the 5S model and 5SL.

Reflecting on the above analysis of the disadvantages of 5SL, we consider the following four functions of 5SGraph to be essential:

1. To help digital library designers understand the 5S model quickly and easily.
2. To help digital library designers build their own digital libraries without difficulty.
3. To help digital library designers transform their models into 5SL files automatically.

4. To help digital library designers understand, maintain, and upgrade existing digital library models conveniently.

We adopt the idea that visualization helps people understand complex models. 5SGraph is able to load and display digital library metamodels. The designer does not need to memorize all the details of digital library modeling theories. The visual model shows the structure and different concepts of a digital library and the relationship among these concepts. 5SGraph also provides a structure editor to let the designer build a digital library by manipulation and composition of visual components. Furthermore, 5SGraph is able to produce correct 5SL XML files according to the visual model built by the designer. Syntactical details of the 5S model and 5SL are hidden from the designer. As such, 5SGraph eliminates the disadvantages of 5SL.

1.3 Organization

We begin by discussing related concepts, technologies, and systems in Chapter 2. At the beginning of Chapter 3, we give reasons why this tool is necessary, and then we describe the architecture of the tool. Afterwards, the design details of the tool also are presented. In Chapter 4, the design of the metamodel and scenarios of modeling using this tool are presented. Chapter 5 describes the usability test of 5SGraph and discusses the experimental results. Chapter 6 gives an example of how to use 5SGraph to model a digital library. Conclusions are presented in Chapter 7.

The UML overview of the system is presented in Appendix A. The metamodel for digital libraries is included in Appendix B. The 5SL file generated from the example process in Chapter 6 is presented in Appendix C. Documents related to the usability test are included in Appendix D.

Chapter 2. Related Work

This chapter reviews several important concepts and tools that are related to the work presented in this thesis. The definition and history of digital libraries are presented along with some brief descriptions of two existing digital libraries. Following the definitions and history, the modeling aspect of digital libraries is discussed. At the end of this chapter, we give a concise summary of some related modeling tools.

2.1 Concepts, Definitions, and Examples of Digital Libraries

2.1.1 History of Digital Library Concepts

The concept of digital libraries was given a clear expression in Vannevar Bush's Memex, described in the 1940s [Bush45]. Memex is a futuristic digital library. Vannevar Bush portrays Memex as “a device in which an individual stores all his books, records, and communications, and which is mechanized so that it may be consulted with exceeding speed and flexibility.” One might imagine extending the idea, so this device could be operated from a distance, but in the time of Bush, telecommunications were relatively primitive. Since the idea of Memex was proposed before digital storage technologies, Bush had all the contents of Memex on microfilm.

With the commercial availability of computers and the development of digital storage and communications technologies, digital libraries have become computer-based. Computer-

based indexing was first studied in the 1950s and 1960s. Project Intrex was an information storage and retrieval system developed at MIT in the 1960s [Intrex]. Researchers believed then that computer-based digital libraries would be faced with operating on two basically different types of data—that which is digitally stored and that which is photographically stored in some microfilm form [Haring68].

Participants in Project Intrex at MIT described future digital libraries: “The library will be the central resource of an information transfer network that will extend throughout the academic community. Students and scholars will use this system not only to locate books and documents in the library, but also to gain access to the university's total information resources, through Touch-Tone telephones, teletypewriter keyboards, television-like displays, and quickly made copies. The users of the network will communicate with each other as well as with the library. The information traffic will be controlled by means of the university's time-shared computer utility in much the same way in which today's verbal communications are handled by the campus telephone exchange. Long-distance service will connect the university's information transfer network with sources and users elsewhere.” [Intrex]

The predictions before 1970 may look a little dated. The idea proposed by Lancaster in 1978, however, is very close to modern concepts [Lancaster78]. Lancaster predicted a paperless society and fully electronic system in his book “Toward Paperless Information Systems”. Due to technical difficulties, those dreams were not feasible until the 1990s.

In the 1990s, the prosperity of Internet and World Wide Web (WWW) provided impetus to develop modern digital libraries. More and more people utilize the web to obtain their information. At the same time, the WWW grows by almost a million electronic pages per day, which generates a tremendous amount of retrievable information. Clifford Lynch points out: information retrieval on the Internet is different from finding information in a traditional library, because the Internet information “lacks organization and structure” [Lynch00]. Digital libraries provide a method to organize the information on the web and facilitate information retrieval for different user communities.

Realizing the critical role of digital libraries, the Library of Congress launched a project to build a National Digital Library (NDL) in 1994, which is based on more than thirty years of groundwork. The Library of Congress plans to embrace the online world through its NDL program. A large number of collections are planned to be digitized and made available online [LOC95].

There was another prominent project effort announced in 1994 – the Digital Libraries Initiative (DLI). DLI was initially supported by the National Science Foundation (NSF), the National Aeronautics and Space Administration (NASA), and the Defense Advanced Research Projects Agency (DARPA). The Initiative's focus was to dramatically advance the means to collect, store, and organize information in digital forms, and make it available for searching, retrieving, and processing via communication networks -- all in user-friendly ways [DLI94]. Six large projects were funded in Phase I from 1994 to 1998. Those projects focused on developing the National Information Infrastructure and on

future technological problems [Chapter 15, Baeza-Yates99]. The follow-on program, DLI – Phase 2 continued from 1998. DLI – Phase 2 shifted the emphasis from on technology research to on interoperability and interactions between digital libraries and humans [Chapter 15, Baeza-Yates99].

The concept of digital libraries changed dramatically from Vannevar Bush's Memex to DLI – Phase 2's ideal digital libraries, which are human-centric systems and are able to serve people from all disciplines.

2.1.2 Definitions of Digital Libraries

Because a digital library is a very complicated information system and the research on digital libraries is relatively new, there exist many definitions for digital libraries. These definitions provide different perspectives to view digital libraries.

Here are some definitions from the digital library literature:

- Digital libraries are organizations that provide the resources, including the specialized staff, to select, structure, offer intellectual access to, interpret, distribute, preserve the integrity of, and ensure the persistence over time of collections of digital works so that they are readily and economically available for use by a defined community or set of communities [DLF00].

- A library that maintains all, or a substantial part, of its collection in computer-accessible form as an alternative, supplement, or complement to the conventional printed and microfilm materials that currently dominate library collections. Used in this context, the term "collection" denotes the documents that a library acquires or maintains [William95].

The Association of Research Libraries identified the following elements as common to most definitions [<http://sunsite.berkeley.edu/ARL/definition.html>]:

- The digital library is not a single entity;
- The digital library requires technology to link the resources of many;
- The linkages between the many digital libraries and information services are transparent to the end users;
- Universal access to digital libraries and information services is a goal;
- Digital library collections are not limited to document surrogates: they extend to digital artifacts that cannot be represented or distributed in printed formats.

2.1.3 Digital Library Examples

2.1.3.1 MARIAN

MARIAN is an indexing, search, and retrieval system optimized for digital libraries. It was developed initially at the Virginia Tech Computing Center, with development

continuing at the Digital Library Research Laboratory [DLRL02]. It is a monolithic system. The architecture of MARIAN system is described in Figure 2.1.

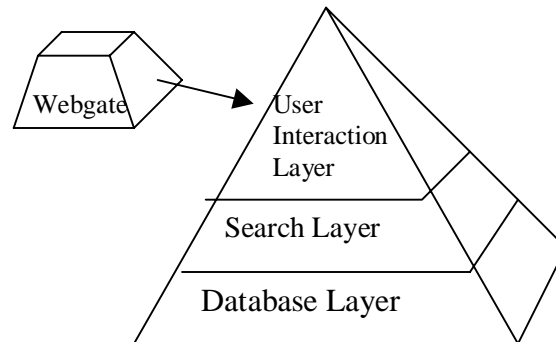


Figure 2.1 Architecture of MARIAN [adapted from [DLRL02]]

2.1.3.2 Open Digital Libraries

Open Digital Libraries (ODLs) are systems built as networks of extended Open Archives [Suleman02]. The basic philosophy of Open Digital Libraries adopts the notions of simplicity and reusability from the Open Archives Initiative [OAI], and adds extensibility and componentization into the mix [Suleman02].

ODLs are componentized systems. Protocols for inter-component communication within a single digital library are designed as extensions of the OAI Protocol for Metadata Harvesting [OAI], and then components that adhere to these protocols are composed to operate as the back-end of a digital library [Suleman02]. A typical ODL network is as in Figure 2.2.

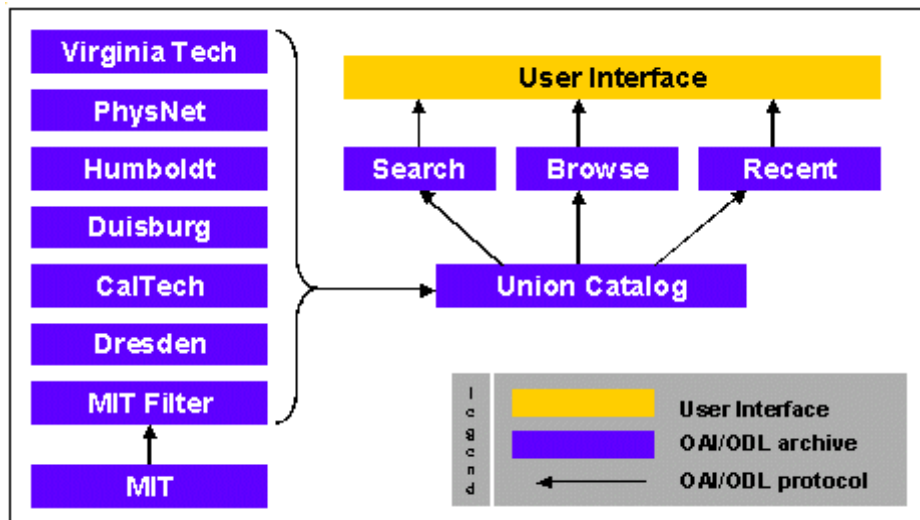


Figure 2.2 ODL Network [adapted from [Suleman02]]

2.2 Modeling of Digital Libraries

Digital libraries are usually huge and complex information systems. It is very important to have formal models and theories for such complex systems. With formal models and theories, people are able to describe, specify, and understand complex systems precisely and clearly. Most mature classes of information systems have established formal models and theories. Database systems have relational models and object-oriented models [Elmasri00]. Information retrieval has three classical models: Boolean, vector, and probabilistic [Baeza-Yates99]. Hypertext systems also have their own formal models, such as the Dexter Hypertext Reference Model [Halasz94].

2.2.1 Some Early Attempts at Digital Library Modeling

The work on formally modeling digital libraries is much less mature than for other information systems. One of the early attempts is Wang's "hybrid system approach for supporting digital libraries" [Wang99]. In Wang's work, the abstract structure of a digital library is defined as a combination of a specific purpose database and a user-friendly interface. He proposes a formal data structure for linking an object-oriented database with hypermedia to support digital libraries. One major problem with Wang's model is that it does not describe many other specific characteristics of digital libraries: interoperability, classification, organization tools, and many more. Another early attempt was made by Kalinichenko et al. [Kalinichenko00]. They present a canonical model that is able to represent heterogeneous digital library collections uniformly. The mapping from heterogeneous data and objects to canonical model is studied to preserve information and ensure semantic interoperability. The attempt is incomplete, however, in that it does not include any descriptions of information services, classification, and many other aspects of digital libraries.

2.2.2 Greenstone's Model

Greenstone incorporates a simple and practical though informal digital library model in its software. In Greenstone [Witten02], a digital library has organized information and the information is available over the Internet. A library includes many different collections, each organized differently. Those collections are composed of documents. A document is the fundamental information-bearing message in electronically recorded form. Documents may include text, audio, video, images, and so on. Each collection has a

uniform interface to make information in documents accessible. Every collection becomes maintainable, searchable, and browsable after an automatic building process that creates structures for each collection. New material can be integrated into the collection by rebuilding. The major disadvantage of the Greenstone model is that it is not a formal model.

2.2.3 Castelli's Models for Digital Libraries

Castelli et al. propose two models for digital libraries. One of the models is a mathematical model for the modeling of digital contents [Castelli02a]; the other model is an architectural model that specifies services provided by a digital library [Castelli02b].

The mathematical model [Castelli02a] consists of two main elements: the document model and the metadata model. The document model has four aspects: documents, document instances, views, and manifestations. Here views represent the content and the structure of a document. The metadata model is composed of formats and specifications, which are able to incorporate as many metadata formats as possible. The architectural model [Castelli02b], also known as OpenDLib, includes a federation of digital library services, which implement the functionalities with few assumptions about the documents stored and disseminated. The model is divided into a kernel, which models the basic elements of the architecture and its instances, and a service types specification, which models the service types that implement the functionality. The functions provided by

OpenDLib include acquisition, storage and preservation, search, browsing and retrieval, selection and dissemination of documents, authorization, and authentication of the users.

2.2.4 5S Model

The 5S model [GonçalvesTOIS] is proposed by Gonçalves, Fox, et al. to provide a formal model that completely describes digital library systems. 5S stands for the theory of Streams, Structures, Spaces, Scenarios, and Societies for digital libraries. The 5S model uses the Society Model to describe how to satisfy information needs of different user communities, the Scenarios Model to describe how to provide all kinds of information services to users, the Space Model to deal with the problem of presenting information in useful ways, the Structure Model to describe the organization of information, and the Stream Model to deal with the problem of communicating information to users.

5S represents a digital library at a very fine granularity and covers almost all primitives in a digital library. 5S is more complete than the early attempts. It is more formal and at a higher level than Greenstone's model. It is more integrated than Castelli's models. More details of 5S are presented in Chapter 3.

2.3 Construction of Digital Libraries

After years of practice, people have accumulated experience in building digital libraries. There are a few existing guidelines for the construction of digital libraries. Most of them

emphasize project planning [Kentuckiana]. Their common problem is that they do not lead to a working version of a digital library directly. They assume that the people who design and build the digital library will have enough technical background, which often is inaccurate.

Meanwhile, more and more tools and techniques have been built or are under construction with the purpose of helping build digital libraries. For the server side, there are Mini SQL, Eprints, Harvest, Sprite, Real Audio, Dienst, ISite, SiteSearch, and many others. However, most of the tools only focus on one or more components of the whole digital library system. It is still required that the digital library builders know where to use these tools and how to tie them together.

In the next section, Greenstone is discussed in more detail. Unlike other above-mentioned tools, Greenstone is a comprehensive and easy to use tool that is able to build a complete digital library.

2.3.1 Greenstone

Greenstone is a suite of software, which has the ability to serve digital library collections and to build new collections. It provides a new way of organizing information and publishing it on the Internet. To accommodate the broad requirements of digital libraries, Greenstone is open-source, distributed under the terms of the GNU General Public License [Greenstone02].

In Ian H. Witten and David Bainbridge's book "How to build a digital library" [Witten02], the authors show how many of the needs of digital libraries are satisfied by Greenstone software:

- Accessible via web browser
- Full-text and fielded search
- Flexible browsing facilities
- Creates access structures automatically
- Makes use of available metadata
- Plug-ins extend the system's capabilities.
- Documents can be in any language.

However, compared with the 5S model, the concept of digital library in Greenstone is simpler and coarser. The 5S model can describe a digital library from more perspectives and more completely than can Greenstone's model. Therefore, it is argued that a tool based on the 5S model is more descriptive, more integrated, and more formal than the Greenstone tool.

2.4 Related Tools

5SGraph is a domain-specific visual modeling tool. It also generates XML files as the final modeling result. As such, we need to introduce some of the related works on XML modeling tools and domain-specific visual modeling tools.

2.4.1 XML Modeling Tools

There are two categories of tools for building XML files. The first category includes any ordinary *XML editing tool*. The ordinary XML editing tools are pure text editors with some particular attributes for building XML files. Those XML editors, which are only a text editor with tree views, are included in the first category.

In the second category we have any *XML modeling tool*. Most XML modeling tools provide a visual interface for users. XML modeling tools are not modeling tools like UML modeling tools that enable users to build a model using UML. They are also different from XML schema design tools that provide means to build or edit an XML schema. The common work procedure of a XML modeling tool is that it loads a DTD or XML schema, and uses the structure information to help a user build an XML instance of that DTD or XML schema. But the actual text may not be available for users to edit directly [SpeedLegal02].

Xerlin [SpeedLegal02] is a representative open-source example of XML modeling tools. Xerlin arose from an open-source project. A DTD is required in order for Xerlin to work. Xerlin provides the interface for users to add, remove, rearrange, and edit XML elements and XML attributes. The DTD specifies what elements are valid, and what attributes are appropriate. Currently, Xerlin does not support XML schema.

2.4.1.1 Limitations of Those XML Tools.

XML modeling tools have semantic limitations because those tools base the modeling process on DTD/XML schema. A DTD formally defines the valid syntax of a class of XML files. After people realized the severe limitations of DTDs [Beech99], use of XML schema was proposed to describe the data content of XML files very formally and rigorously. However, a DTD or XML schema alone does not have enough semantic power to describe a complex system. Semantic tools are often required for web XML applications [Patel-Schneider02].

Due to the semantic limitations of DTD/XML schema, XML modeling tools are at a lower level than 5SGraph.

2.4.2 Domain-Specific vs. General Purpose Visual Modeling Tools

Domain-specific modeling involves constructing an explicit model, whether formal or informal, of domain-specific concepts, relationships, and knowledge concerning them, in a declarative way [Park92]. 5Sgraph is a domain specific visual modeling tool that is specifically made for the modeling of digital libraries based on the 5S model.

The difficulty of general visual modeling is often due to the fact that the concepts in large systems are very abstract and not easily mapped to visual notations. Domain-specific visual modeling overcomes this problem by limiting the problem to a well-identified domain. The varieties of building components and relationships are greatly reduced. The metamodel behind the modeling is very close to the models. The modeling process

becomes more intuitive and easier to grasp. A maximum number of methods can be applied to optimize the modeling process, including specific methods that are only applicable to this domain. It is commonly believed that domain-specific modeling tools are much more efficient than general modeling tools for a specific-domain task [Tolvanen01]. In the work described in this thesis, a domain-specific modeling approach is adopted.

Chapter 3. Design of 5SGraph

3.1 Process of Building a Digital Library

The current process of digital library generation with MARIAN based on the 5S model is described as follows (see the general approach in Figure 3.1):

1. First, a digital library designer conducts a 5S analysis of the digital library he wants to build.
2. Second, the digital library designer describes the digital library in 5SL.
3. Then, the 5SL file is fed to the digital library generator, along with a component pool. The component pool includes stock components that can form the new digital library. Examples of these stock components are SAX, DOM, routines in the MARIAN hierarchy of classes, OAI harvester, and many more. With the component pool, the digital library generator produces a new digital library according to the 5SL specification.

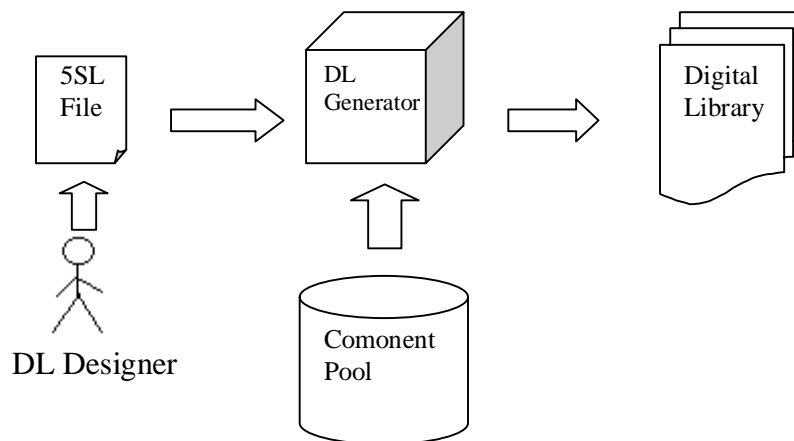


Figure 3.1 Building Digital Libraries with the 5S Model

As we point out in chapter one, it is not easy for digital library designers to formalize a description of digital libraries directly in 5SL, because the designer needs to have a firm and thorough understanding of the 5SL. Besides that, 5SL has five sub-models, where each sub-model has its own syntax and semantics, and it is hard for ordinary designers to write completely correct 5SL files for all parts of the model. There are also semantic constraints between sub-models. Designers need extra effort to keep the consistency of the whole model. Our solution is to insert an extra step (i.e., a helping tool, 5SGraph) between the digital library designer and the digital library generator, as shown in Figure 3.2. 5SGraph provides an easy-to-use graphical interface and automatically generates desired 5SL files for the designer. With the modeling tool, the process of automatic digital library generation becomes as follows (see Figure 3.2):

1. A digital library expert creates a metamodel for digital libraries and feeds the metamodel to the modeling tool. The metamodel is based on the 5S theory. It describes a generic digital library.
2. The modeling tool processes the metamodel, allowing a user to visualize the components of the metamodel.
3. A digital library designer uses the modeling tool to describe his own digital library. The visualization of the metamodel helps the designer understand the structure of a generic digital library and reduces the learning time. The designer then can use those visualized components of the metamodel to put together the final model of his own digital library.
4. After the designer describes his own digital library visually, the modeling tool can transform the visualized digital library into 5SL files that follow XML syntax.

5. The generated 5SL file is fed into the digital library generator with the component pool. Then, a working instance of the desired digital library is produced.

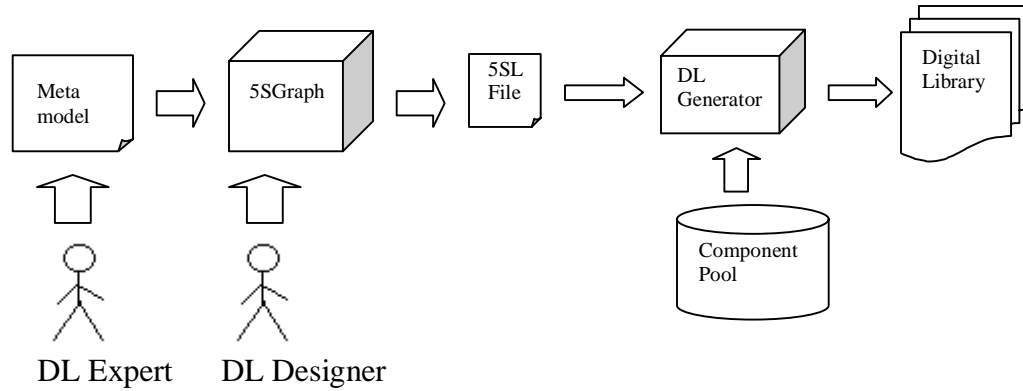


Figure 3.2 The Digital Library Generation Process with 5SGraph

With the help of a visual modeling tool (i.e., the 5Sgraph), the difficulty of a designer's work is minimized dramatically. The designer only needs to deal with a graphical interface and pull visual components together. It is not required for him to memorize the details of the syntax and semantics of 5SL. Cognitive load is reduced. Typing effort and typing errors are reduced. The visualization of the model also provides guidance while the designer is building his model. Moreover, correctness and consistence can be automatically guaranteed by 5SGraph.

The concept of metamodel is utilized here. This metamodel describes a generic digital library. The model for a specific digital library is an instance of the metamodel as shown in Figure 3.3. This relationship between metamodel and model is similar to the metamodel and model definitions of the OMG's Meta Object Facility (MOF) [OMG02]. However, the definition of metamodel by MOF is larger and more general than our

definition of metamodel here. The metamodel used in this thesis is a domain-specific metamodel (i.e., specific to the domain of building digital libraries).

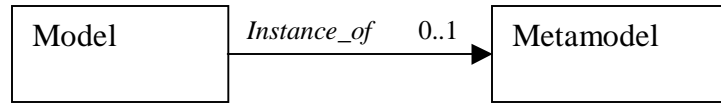


Figure 3.3 Relationship Between Model and Metamodel

It should be noted that a digital library expert who is familiar with the 5S model is required to build the initial metamodel. The digital library expert role brings in flexibility. The 5S model is still in its early stage of development. It is expected that more changes and additions will be made in the future. The digital library expert is in charge of creating and changing the metamodel that is to be changed with the 5S model. Therefore, being given a new metamodel, the tool can be used with future versions of the 5S model as well.

3.2 Architecture of the 5SGraph

5SGraph is a modeling tool that supports domain-specific modeling (digital library modeling). In domain-specific modeling, the model is made up of elements that are part of the domain world, not the whole entity world. 5SGraph is tailored to accommodate a certain domain metamodel, the 5S model. The methods that are appropriate only to this domain can be used to optimize the modeling process. Reuse in a specific domain is also more realistic and efficient, because the models in one domain have more characteristics in common.

The 5S model extensively uses existing standard description languages, for example, UIML. The reason is that the 5S model involves many sub-domains, and there have been already many standard specifications for each sub-domain. It is important for the 5S language to be able to interoperate with other standards. 5SL is designed to be able to combine and merge with other standard description languages.

There are also many well-developed tools for those sub-domains. For example, metadata is an important element in the 5S model. Many important metadata standards have been widely accepted, such as ETD-MS [Atkins01], Dublin Core, IMS, and MARC. Several existing metadata editors can be used to view and edit metadata. Another example is in the scenario part of the 5S model. A specific scenario can be modeled and described by UML sequence diagrams. Existing UML modeling tools can be used for this purpose.

The 5SGraph tool should not “re-invent the wheel” to do those tasks. Therefore, the tool is designed to be a super-tool, which means it provides an infrastructure based on the 5S model and calls existing tools when needed. The future architecture of 5SGraph is sketched in Figure 3.4. Since the work in this thesis is the first step in this direction, the current tool focuses only on the infrastructure part and leaves the part that calls other software for future work.

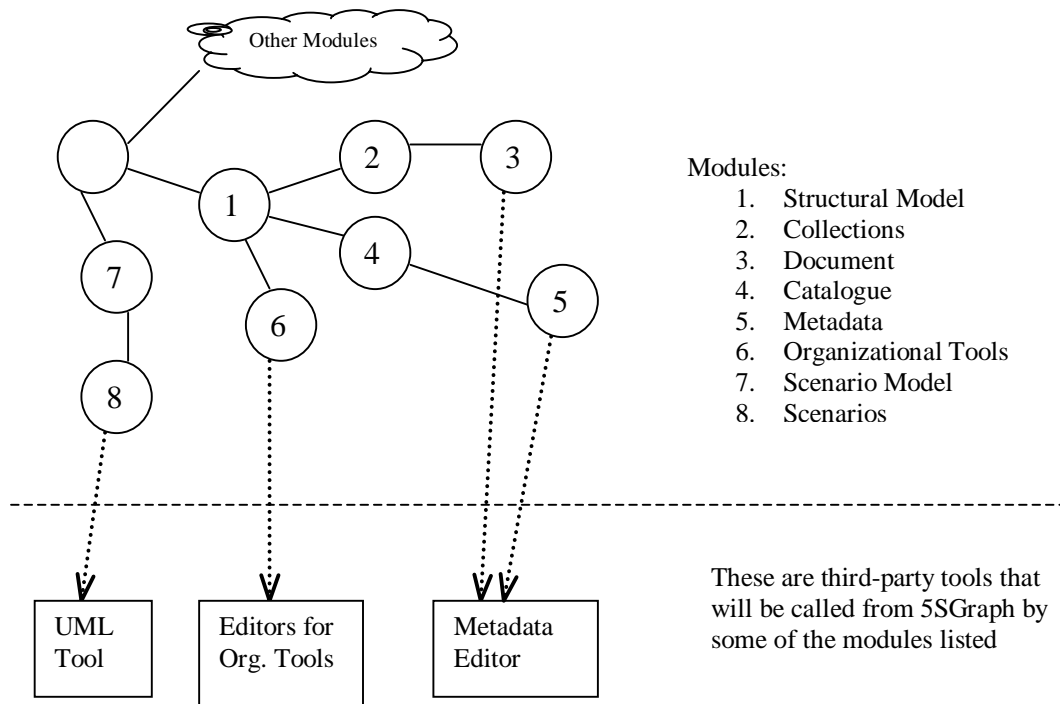


Figure 3.4 The Interaction of 5SGraph with Other Tools (Future Work)

3.3 Brief Description of the 5S Model

A description of 5S/5SL is given in this section. The description does not cover the entire 5S/5SL. 5S is a very complicated model [GonçalvesTOIS]. We talk about the major components and the important relationships between them. The metamodel used in this thesis work is based on this description.

The 5S model describes digital libraries from five different and complementary perspectives. Different S models have different primitives and serve for different objectives. Those S models are not necessarily independent from each other. A primitive

in an S model may be dependent on a primitive in another S model. Figure 3.5 gives an overview of the 5S model.

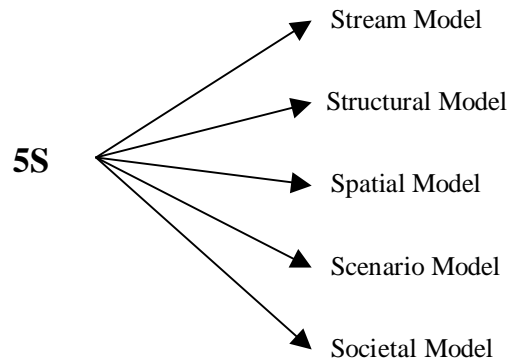


Figure 3.5 Overview of the 5S model

3.3.1 Stream Model

The Stream Model states the properties of the digital contents supported by the digital library. The digital contents are, for example, text, video, audio, images, and software programs. Their common characteristic is that they all can be represented as a sequence of more elementary items (e.g., characters, etc.). The current version of the hierarchy of the Stream Model is shown (see Figure 3.6) as:

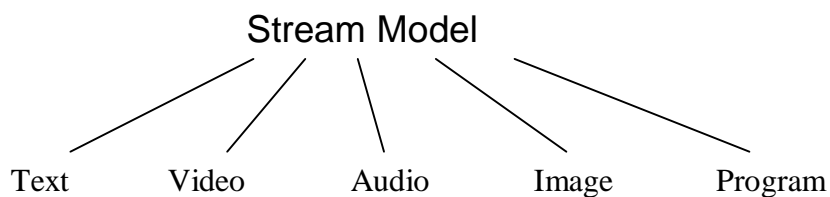


Figure 3.6 The Hierarchy of the Stream Model

The stream properties include the encoding and format of the sequence. For the sake of standardization and interoperability, the official Internet standard, MIME (Multipurpose Internet Mail Extensions), is used. MIME header fields are adapted to become the XML elements of 5SL. Examples of how 5SL describes the streams are shown in Table 3.1.

Streams	MIME header field	5SL
A text stream in an ETD (electronic thesis or dissertation)	Content-Type: text/plain; charset=ISO-8859-1	<text name='ETDText'> <content-type> text/plain </content-type> <charset> ISO-8869-1 </charset> </text>
A image stream in an ETD	Content-type: image/gif	<Image name='ETDImage'> <Content-type>image/gif </content-type> </Image>

Table 3.1 5SL Descriptions of Streams

3.3.2 Structural Model

The Structural Model specifies all kinds of structures in the 5S model, which include the internal structure of a digital object, metadata standards, properties of collections and catalogs, and knowledge organization tools. The current version of the hierarchy is shown in Figure 3.7.

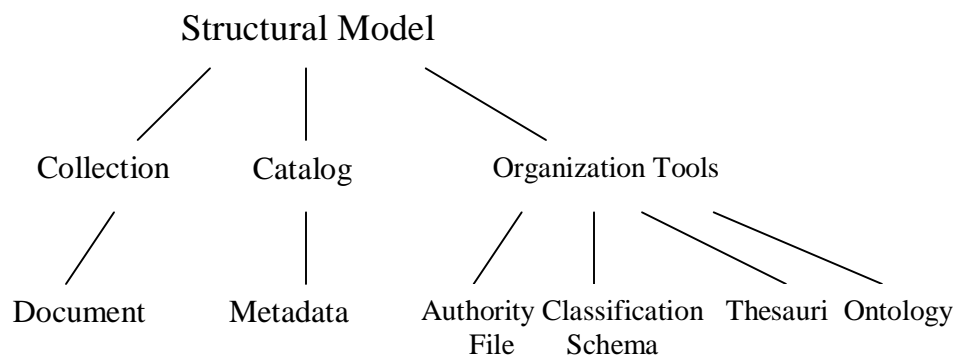


Figure 3.7 The Hierarchy of the Structural Model

Digital objects (documents) are important building blocks of digital libraries. The formal 5S definition of digital objects is to impose structures over stream(s), or use structures to organize streams together to have a digital object. Here is the formal definition from [GonçalvesTOIS]:

“A digital object is a tuple $(h, SM, ST, StructuredStreams)$, where

1. $h \in H$, where H is a set of universally unique handle (labels);
2. $SM = \{sm_1, sm_2, \dots, sm_n\}$ is a set of streams;
3. $ST = \{st_1, st_2, \dots, st_m\}$ is a set of structural metadata;
4. $StructuredStreams = \{stsm_1, stsm_2, \dots, stsm_p\}$ is a set of StructuredStream functions defined from the streams in the SM set (the second component) of the digital object and from the structures in the ST set (the third component).”

A StructuredStream specifies the combination of a structure and a stream. An example of a StructuredStream is illustrated in Figure 3.8 for an electronic thesis or dissertation. It is a textual stream with a structure.

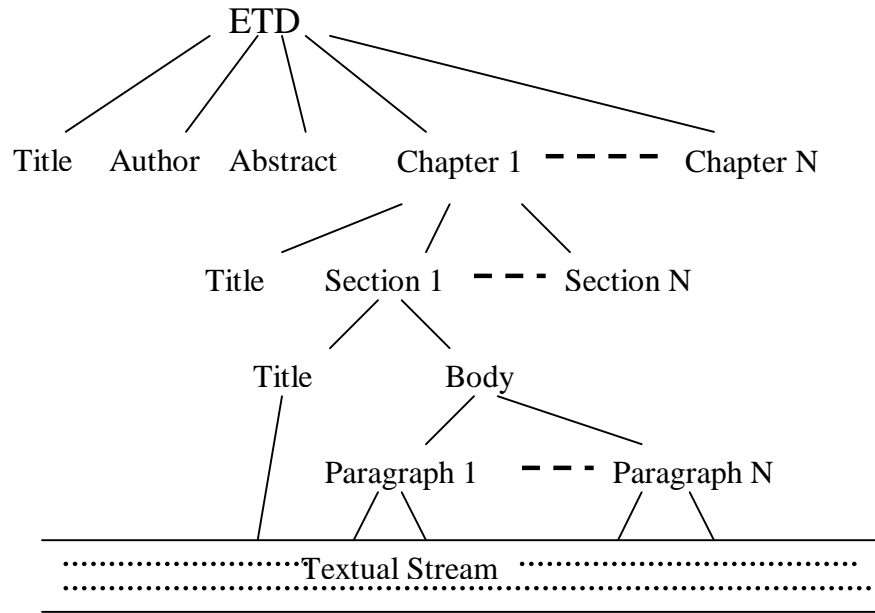


Figure 3.8 A StructuredStream for an ETD [adapted from [GonçalvesTOIS]]

An exemplar description of a document (electronic thesis or dissertation) in 5SL is as follows:

```

<document name='ETD'>
  <stream_enum>
    <stream> ETDText</stream>
    <stream> ETDAudio </stream>
    <stream> PDF </stream>
  </stream_enum>
  <Structured_stream> XMLSchema</structured_stream>
</document>

```

Semantic constraints include:

- The streams in the definition of a digital object (document) are predefined in the Stream Model.
- A collection consists of different kinds of documents. A catalog describes a collection, since a catalog collects all the administrative or descriptive metadata that apply to the collection. A catalog, therefore, is dependent on a collection.

3.3.3 Spatial Model

The Spatial Model describes those specific digital library components that have an underlying metric space and relational operations on the space. The primitives in the Spatial Model include digital library indexing for collections and catalogs, retrieval models, and the user interface. The current version of the hierarchy is shown in Figure 3.9.

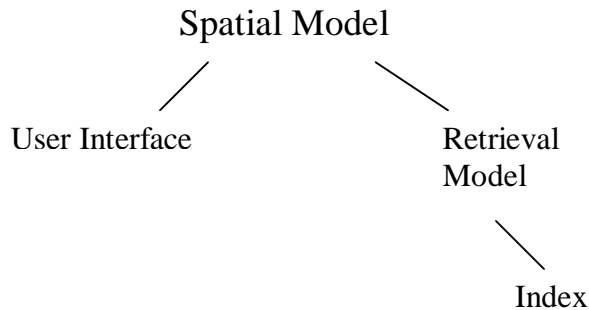


Figure 3.9 The Hierarchy of the Spatial Model

The user interface is a very complicated component. The existing UIML-based model [Harmonia02] is chosen to describe the user interfaces of digital libraries. UIML is an abbreviation for User Interface Markup Language [Harmonia02]. It is an XML language for defining user interfaces. The tags in UIML describe the structure of user interface elements (such as a graphical button, icon, or text input box), their properties (such as color or size), and the behavior that occurs when you interact with user interface elements. In the future, interfaces in UIML will be automatically generated from scenarios.

3.3.4 Scenario Model

The Scenario Model describes the behavior of digital library services. The primitives include service, event, condition, and action. A UML sequence diagram is a good visual tool to express different scenarios [Gonçalves02]. The current hierarchical structure of the Scenario Model is shown in Figure 3.10.

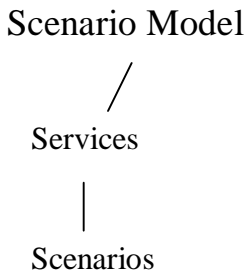


Figure 3.10 The Hierarchy of the Scenario Model

3.3.5 Societal Model

The Societal Model is based on the concept of ‘community’. A community refers to a set of people or computers that share the same characteristics and behavior. The Societal Model is composed of the types of communities, the functionalities of communities, and the relationships among them. Generally, there are two types of communities: managers and actors. Managers, which are normally electronic agents, take care of the interface, index, repository, searching, browsing, and other administrative services. Actors participate in and use any of the services defined in the Scenario Model. Figure 3.11 illustrates the current version of the hierarchy.

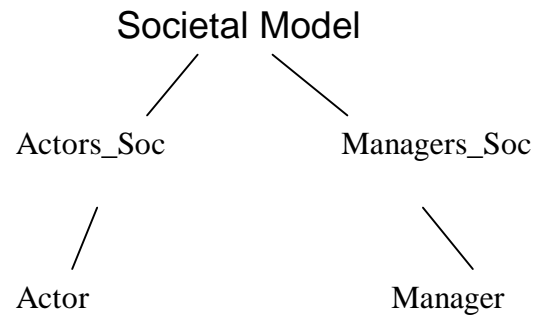


Figure 3.11 The Hierarchy of the Societal Model

Semantic constraints include:

- The services that the actor participates in are the services defined in the Scenario Model.
- The services that the manager manages are the services defined in the Scenario Model.

3.4 Design Details of 5SGraph

It is observed that a tree structure and a set of semantic constraints are enough to describe the relationships among major components of the 5S model. As such, we use trees to represent the metamodel and the instance model in 5SGraph. Graph representation is not chosen because it adds unnecessary layout complexity into the tool. Another reason for choosing a tree is that both the metamodel and model are specified in XML. Tree structures in the tool are harmonious with XML, as research shows tree structures are good at showing XML instances [Pietriga01, Erwig00].

3.4.1 Toolbox

The first design question to ask concerns how the tool helps the user build a model that is an instance of a metamodel. A component in the metamodel represents a type, and a component in the instance model represents an instance of a certain type. The requirement behind the question is that any component that appears in the instance model must be an instance of a certain type in the metamodel. This requirement has two implications. First, a component instance in the instance model should have all the properties of the corresponding type of the metamodel. Second, the component instances in the instance model should maintain the same structural relationships as their corresponding types in the metamodel.

Let us look at other systems before we explain our answer to this question.

Some visual systems, e.g., for Concept Maps [IHMC01], do not have predefined visual components, or predefined structural relationships. They utilize generic visual components and links. In a Concept Map, those generic visual components with different labels can be used to represent different concepts. Thus, links with different labels stand for different relationships.

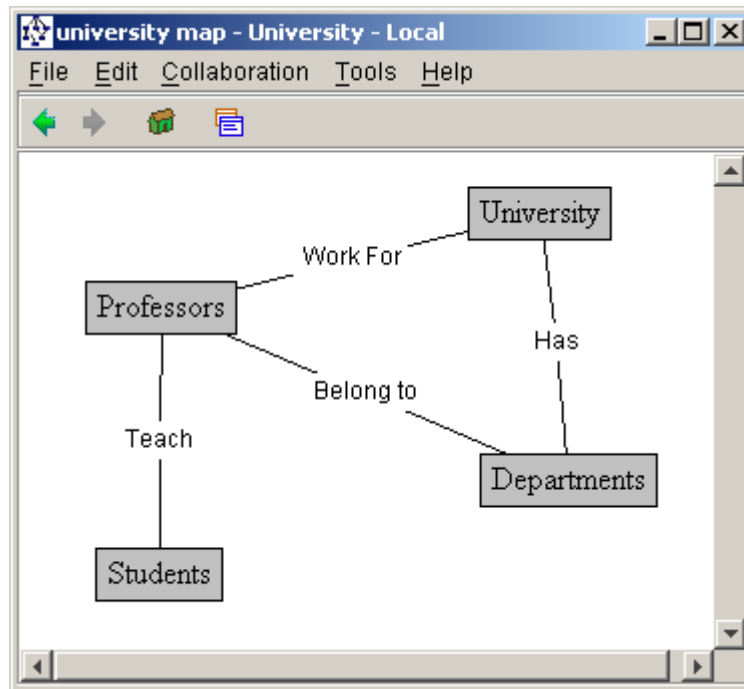


Figure 3.12 A Screen Shot of a Concept Map

The major disadvantage of applying this approach to digital library design is that, without predefined types and relationships, the user would have to build the components from scratch and maintain correct structural relationships among them. This process is usually very time-consuming and tedious. It is particularly true when the metamodel is large and complicated. Another disadvantage of this approach is that the system would need a very powerful type-checking mechanism to ensure that all the types, attributes, and relationships are included correctly. Therefore, the majority of modeling tools do not adopt this approach. Instead they prefer to use predefined components and relationships.

Most visual systems, which have predefined components, use a toolbox to show all available visual components. The visual components can be chosen and added to a user's model directly. Example systems are the BeanBox for JavaBeans [Sun02a], Khoros

[Khoral01], and many other commercial visual component software packages. Most toolboxes are in the form of either a list, as in the BeanBox, or a categorized list, as in Cantata [Khoral01]. A typical toolbox is shown in Figure 3.13 [Sun02b]. The relationships between those components obey the rules defined by a certain metamodel. Only valid connections are allowed.

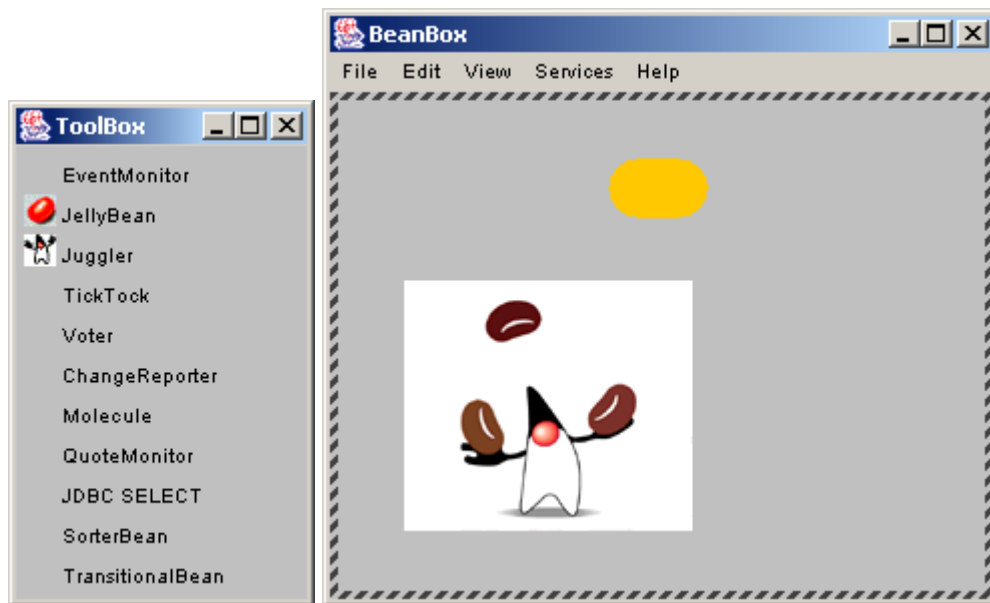


Figure 3.13 The Toolbox of BeanBox

The problem with these toolboxes is that the structural relationships among the visual components are not visible from the toolbox. The user needs to have sound knowledge of the metamodel in order to maintain the correct structural relationship among components. A trial-and-error approach is necessary to build a correct system.

How can we overcome the disadvantages of unstructured toolboxes? A structured toolbox is introduced into our tool, which not only provides all the visual components of the metamodel, but also shows the structural relationships among these components. See Figure 3.14.



Figure 3.14 The Workspace and the Structured Toolbox in 5SGraph

As illustrated by Figure 3.14, the tool is divided into two parts. The lower part is the structured toolbox that shows all available components of the metamodel and the relationships among them. It also can be considered as a visualization of the metamodel. The upper part is the workspace in which users can create their instance models.

3.4.2 Icons

Visual components in the toolbox and in the workspace have different meanings and different behaviors. However, they all have the same shape (a rectangle). The reason for using the same shape for all visual components is that the tool itself does not have any predefined shape/type association. The tool does not know a priori what types it will have before it gets the input of a certain metamodel. Using the same shape for all components, we simplify the layout strategy and get consistent visualization effects. In fact, many visualization systems use the same shape (usually a rectangle) for all components.

However, there is an issue related with this. A visual component in the toolbox represents a type. A visual component in the workspace represents an instance. For each type, there could be several instances. Each visual component has a name to distinguish itself from others. The names of visual components in the toolbox are not alterable, since they are type names. The names of visual components in the workspace are changeable, since they are instance names. As a result, the user may easily forget the association of an instance and its type after the instance name is changed. This is illustrated in Figure 3.15.

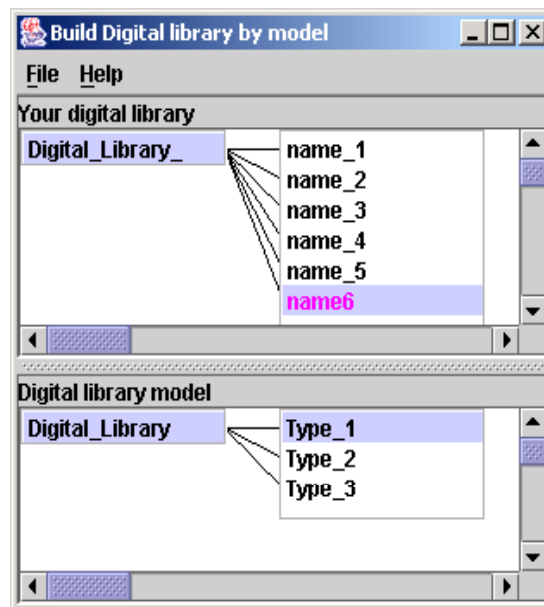


Figure 3.15 Visual Components without Icons and Cardinality.

To solve the problem, we associate types with different icons, which are inserted at the left side of the rectangle of every component. The names of the instances may change, but the icons do not change, which helps the user to find the correct type for their instances, as shown in Figure 3.16.

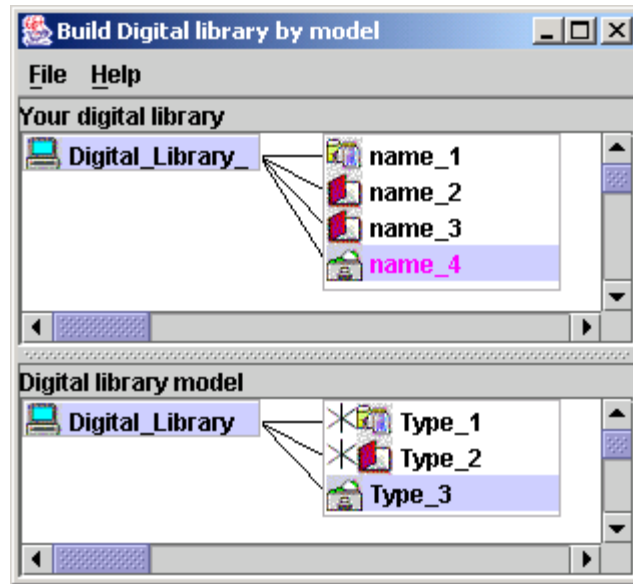


Figure 3.16 Visual Components with Icons and Cardinality

Another difference between the component in the toolbox and that in the workspace is: the component in the toolbox has cardinality associated with it, denoting how many instances of this component can be added to a parent node in the workspace. We currently distinguish two occurrence cases specifying whether an instance of the type should occur at most once, or any number of times. The indicator (*) means any number of times. Having no indicator means it can occur at most once. More cardinality indicators will be added as necessary in the future.

3.4.3 Tree Representation

The analysis of the 5S model shows that the structure of the 5S metamodel can be viewed as a tree structure, accompanied by a set of domain specific constraints. We next compare

two different approaches to representing tree structures and give reasons why the Node-Link approach is chosen. Constraint management is discussed in the next chapter.

3.4.3.1 Node and Link Approach

The traditional way to represent a tree structure is with a node and link representation, which is a rooted, directed graph, as in Figure 3.17. There are some existing Integrated Development Environments that provide graphic representation of tree structures, like Java's JTrees [Sun02b], illustrated in Figure 3.17. The traditional approach has its advantages and disadvantages.

The advantages of the node-link approach include:

- It is straightforward.
- The absolute depth can be easily observed, since the nodes at the same level have the same depth.
- It is easy to manipulate.

The disadvantages of the node-link approach include:

- It wastes most of the occupied screen space.
- It does not scale well. A scroll bar is needed for large structures. For example, in JTree, a deep node may cause too much vertical expansion.

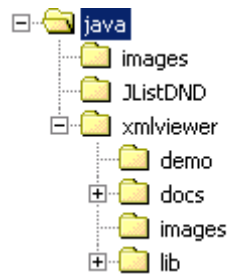


Figure 3.17 A Node-link Representation by a JTree

To overcome visualization problems associated with traditional trees, researchers invented several variants, e.g., the hyperbolic tree [Pirulli01] and the cone tree [Robertson93]. These variants improve visualization capabilities, with the cost of reducing the convenience of manipulation.

3.4.3.2 Nested Approach

Treemap [Shneiderman01], employing a nested display method, is a totally different approach from the node-link representation. Trees are shown with a 2-D space-filling representation, which consists of nodes as rectangles and nested child nodes in their parent (see Figure 3.18).

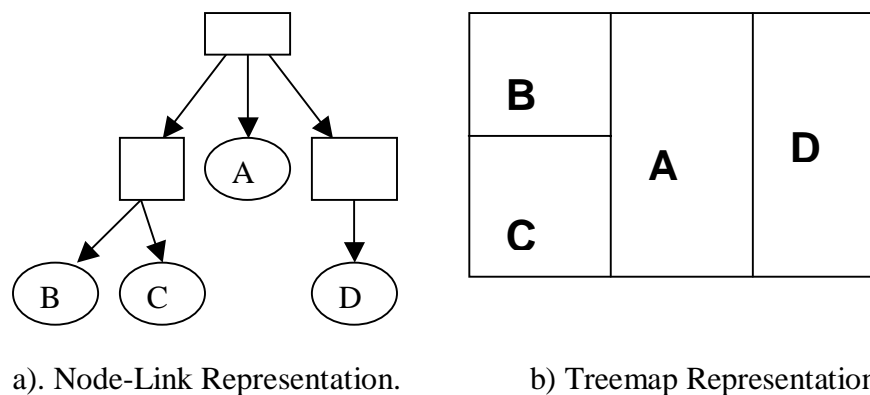


Figure 3.18 Node-Link and Treemap Representations of a Tree

The advantages of Treemap [Pietriga01] include:

- better use of the screen real estate, and the
- ability to visualize large hierarchies.

The disadvantages include:

- It requires zooming capability, because the deep nodes may be very small.
- No obvious information about depth is available.

3.4.3.3 Reasons for Choosing Node-link Representation

We choose the traditional node-link representation because the depth information is obvious, while a nested representation does not convey the depth information as clearly. The node-link representation shown in Figure 3.19 makes clear the correspondence between the levels of the metamodel and the user's instance model. It is required that the level of an instance component in its instance tree should be the same as the level of its corresponding type component in the metamodel tree. Since it is very important for the tool designer to have an explicit and clear overview of the depth information of both trees, we prefer the node-link representation.

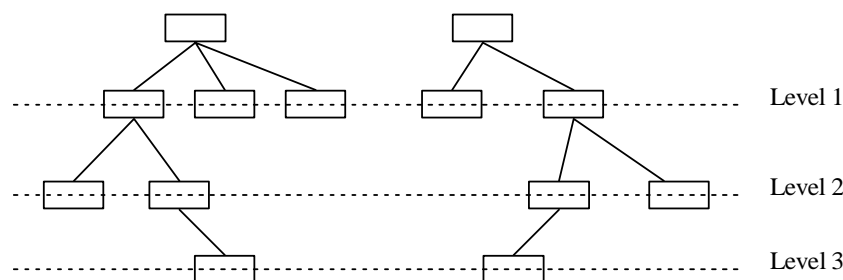


Figure 3.19 Node-link Representation has Clear Depth Information

3.4.4 Truncated Visualization of Full Trees in 5SGraph

3.4.4.1 Problems Caused by Deep Nodes

There often will be many nodes at deep levels of a large tree. Theoretically, for an n -ary tree, there are at most n^{h-1} nodes at level h . For example, there are at most 81 nodes at level 5 of a ternary tree. We need to reduce the number of nodes being displayed to avoid serious difficulties with layout and icon search speed [Byrne93]. A simple but very efficient solution is to show only one branch, which is in the form of a *list*, at each level. An example is shown in Figure 3.20.

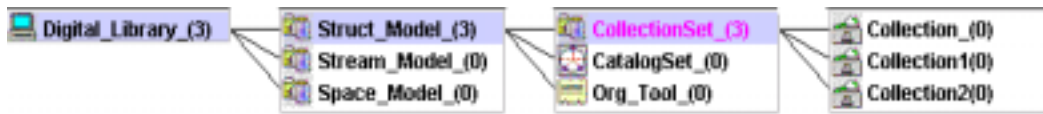


Figure 3.20 A Tree with One Visible Branch at Each Level

The parenthetical after the name of a node gives the number of direct child nodes.

3.4.4.2 Visible Path and Consistent Lists

For the convenience of explaining the truncated display and the user interactions, we introduce the concepts of *visible path* and *consistent lists*. The path from root to the deepest visible leaf is referred to as a *visible path*. The path from any visible node to a deepest visible leaf is referred to as a visible path starting from that visible node. A visible path in Figure 3.20 is:

Digital_Library → Struct_Model_ → CollectionSet_ → Collection_

A visible path starting from Struct_Model_ is:

Struct_Model_ → CollectionSet → Collection_

A node list, ListA, in the toolbox and a node list, ListB, in the instance model are consistent with each other if and only if the parent node of ListB has the type of the parent node of ListA. As a result, all the types of the components in ListB are in ListA.

An example is shown in Figure 3.21. The two-way arrows point to lists that are consistent with each other. The list of S models in the user model (“Your digital library”) is consistent with the list of S models in the metamodel (“Digital library model”). The list of streams is consistent with the list of stream types in the metamodel.

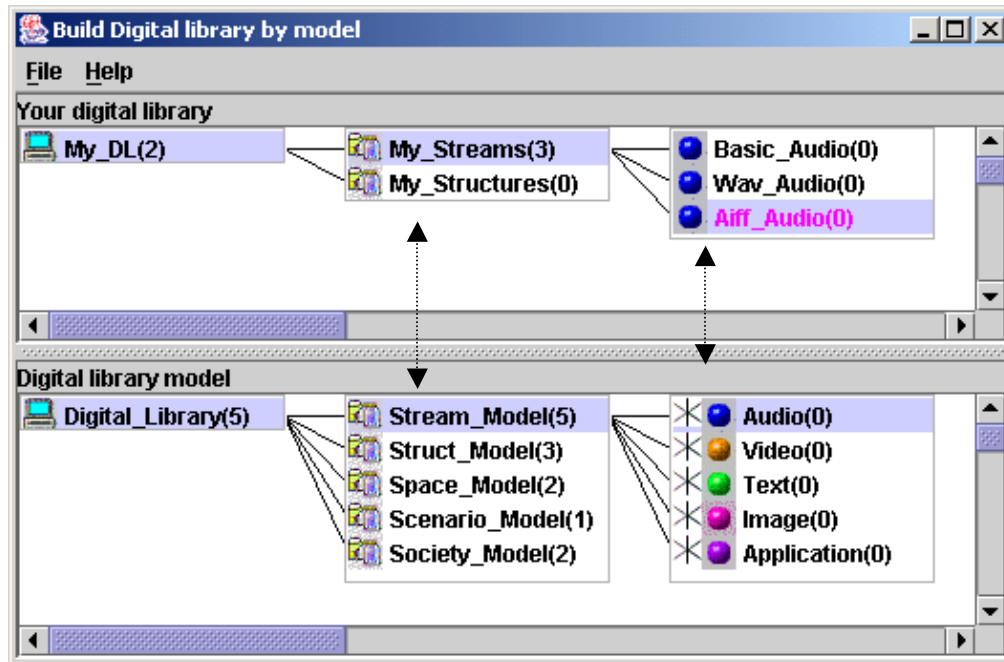


Figure 3.21 Consistent Lists

3.4.4.3 Truncated Display

Truncated display of trees in 5SGraph minimizes layout difficulties, reduces the size of the icon set that needs to be searched, and focuses a user's attention on the currently active branch. Although a truncated display shows less information than a full display, it shows the most relevant information in the tree. It manages to dynamically focus only on the active branch of the tree on which the user is working, while leaving out irrelevant branches.

The information presented in the truncated display includes the:

- parent node of a node,
- child nodes of a node,

- siblings of a node, and
- level of the node in the entire tree.

For example, in Figure 3.21, the information that is displayed for the chosen component *Stream_Model* includes:

- Its parent component: *Digital_Library*.
- Its child components: Audio, Video, Text, Image, and Application.
- Its siblings: *Struct_Model*, *Space_Model*, *Scenario_Model*, and *Society_Model*.

The level of *Stream_Model* in the metamodel tree is 2.

3.5 User Interactions

In this section, we first describe some basic operations supported by 5SGraph. Then, we explain an important feature of 5SGraph: the synchronization mechanism. Finally, we discuss component reuse in 5SGraph.

3.5.1 Basic Operations

5SGraph supports four basic operations: exploring, adding, deleting, and changing properties. They are introduced and described in detail in the following paragraphs.

Exploring: Exploring is invoked by single clicks. The operation is similar to the expanding operation of JTree in Java. It enables the subtree of a selected node to be visible.

Scenario of exploring: A user clicks on the node of *Stream_Model* and looks at the content of *Stream_Model* that is included in the subtree (see Figure 3.22). After that, the user clicks on the node of *Struct_Model* and checks the content of the *Struct_Model* that is now visible in the subtree of *Struct_Model*. *Struct_Model* has three children: *CollectionSet*, *CatalogSet*, and *Organization_tools*. The user is interested in *CollectionSet*, so he clicks on the *CollectionSet* and looks at the children of it. In this way, the user can see the entire model (see Figure 3.23).

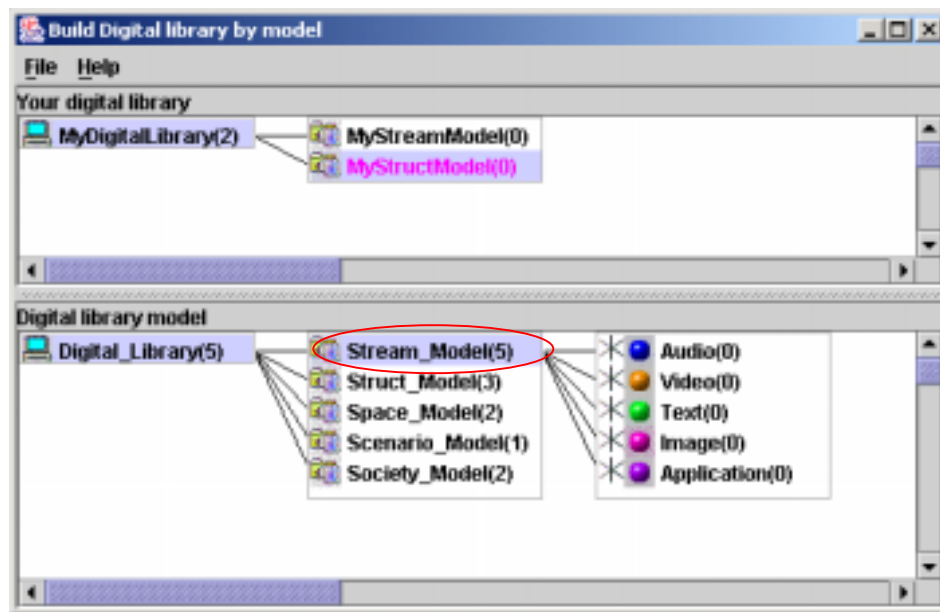


Figure 3.22 The Operation of Exploring

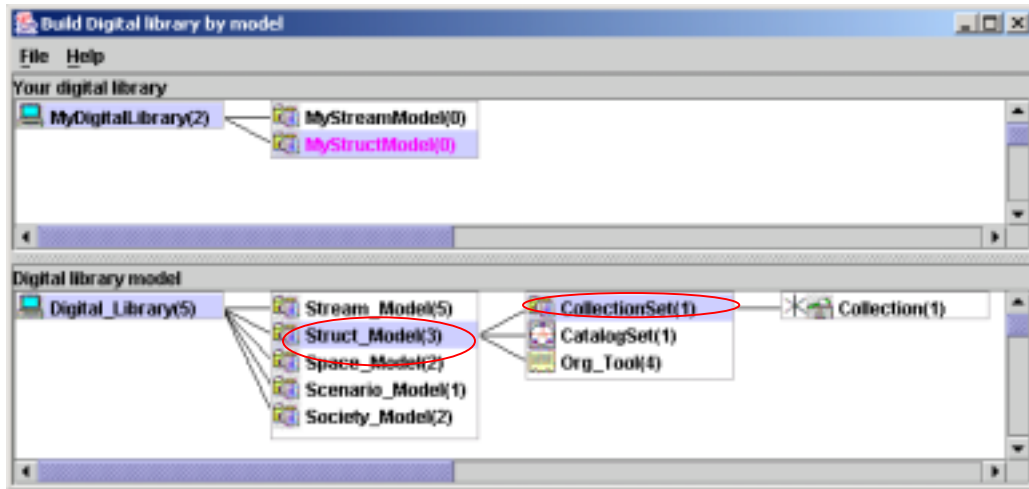


Figure 3.23 The Operation of Exploring

Usability tips of exploring: The tool keeps track of all visible paths starting from any node. As such, if a user clicks on any previously selected node, the tool can immediately show to the user the last visible path starting from that node. This design is based on the assumption that the path that a user has just visited is most likely to be visited again.

Adding: The user adds an instance into the instance model by double clicking on a component in the toolbox. Alternatively, the user may right click on a component in the toolbox and choose the adding operation from the pop-up menu.

5SGraph utilizes a top-down modeling strategy in creating a model. The user has to add the parent into the model before he/she can add any child components. If a new child component is to be added into the model, just like when using any similar tools, the user needs to specify a position for the new component. This is done by selecting a parent

node first and then appending the child component to the end of the children list of the parent node.

Scenario of adding: A user browses through his model. When he reaches the Stream_Model, his click on Stream_Model brings up all the child nodes. He finds out that he needs to add one more image type, image/gif, to his model. So he double-clicks on the image/gif component in the toolbox and adds it to his model. A top-down assumption: the user clicks on the parent node to check its content before he decides to add a component to the parent node (see Figure 3.24).

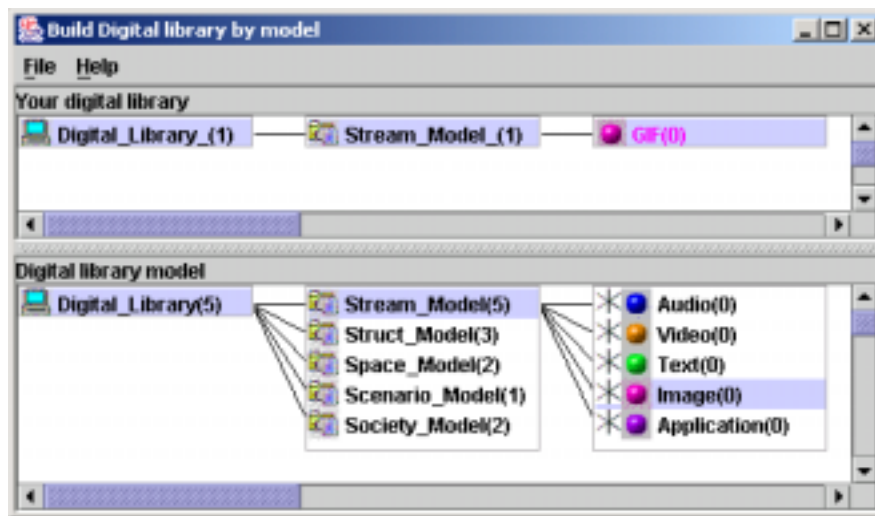


Figure 3.24 The Scenario of Adding

Usability tips of adding: When the user selects a component in his instance model, the tool synchronizes with it by showing a visible path in the toolbox that goes from the root to the type of the selected component. Therefore, the user always knows which components in the toolbox can be added as the sibling or children of the selected component.

Deleting. Pressing the “DEL” key triggers deletion. The user can only delete nodes in the user model.

Changing properties: Each instance component has its own property sheet that goes with its type. The property sheet is not visualized in the toolbox, or in the workspace, because it adds unnecessary complexity. The user can get the property sheet by double clicking on the component in the user model. The properties in the property sheet are changeable (see Figure 3.25).

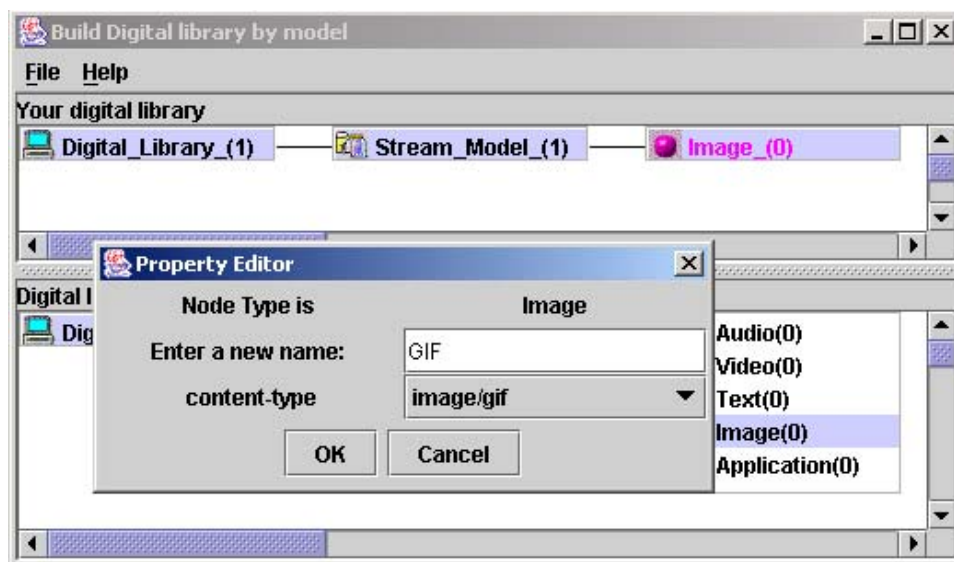


Figure 3.25 The Operation of Changing Properties

3.5.2 Synchronization

There are two views in the tool. One is for the toolbox (metamodel); the other one is for the user model. These two views are related through the type/instance relationships between components in the toolbox and components in the user model.

When a user selects an instance component in the workspace (user model), 5SGraph is able to synchronize the view of the toolbox by showing a visible path from the root to the corresponding type component of the selected instance component. The convenience of synchronization is that:

- The user does not need to manually search all the components in the toolbox to find the correct type component.
- The tool helps the user focus on the most important relationships of the type components.
- The child components that can be added to the current component are within immediate reach of the user.

Scenario of synchronization: The user clicks on “*MyStreamModel*”. The toolbox finds that the type of the component is *Stream_Model*, and expands the subtree of *Stream_Model* automatically. The user adds an instance of *Audio* to *MyStreamModel* (see Figure 3.26). The user then clicks on *MyStructModel*. The toolbox expands the subtree of *Struct_Model* automatically. The user looks at the child nodes of *Struct_Model* and adds an instance of *CollectionSet* and an instance of *CatalogSet* to his model (see Figure 3.27).

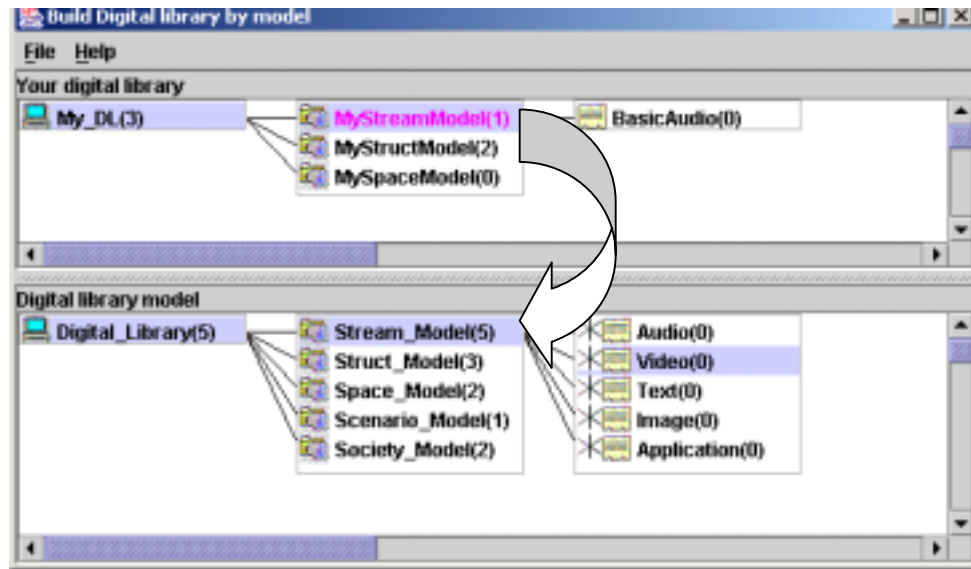


Figure 3.26 The User Adds an Instance of *Stream_Model*

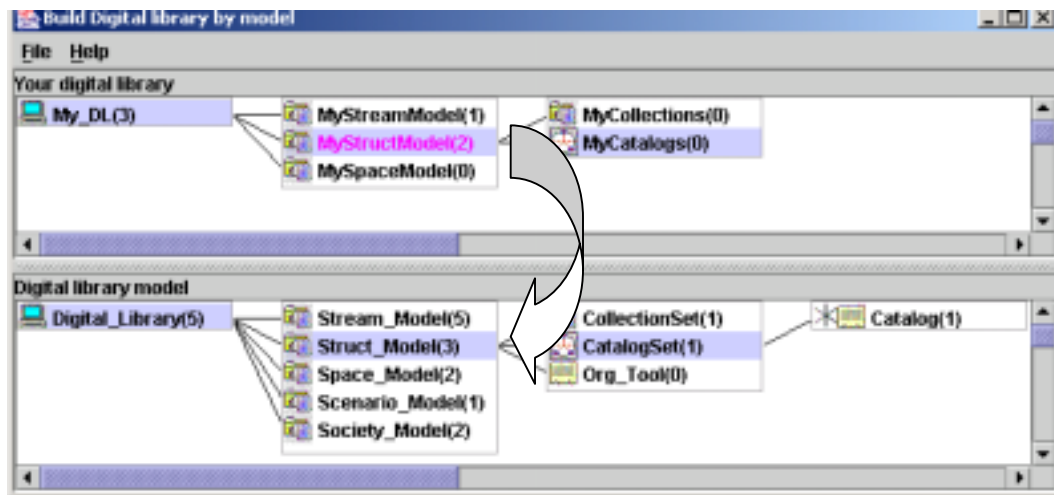


Figure 3.27 The User Adds an Instance of *Struct_Model*

3.5.3 Component Reusability

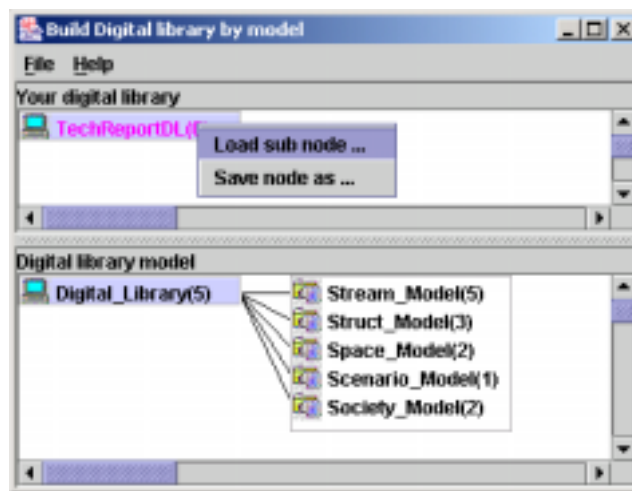
In our case, component reusability means the components that have been built in one user model can be saved and reused by other user models.

Reusability saves time and effort. There are components that are common for many different digital library systems. For example, many digital libraries share the same data formats, and the same descriptive metadata. The components representing the Stream Model or the metadata can be built once and reused in different digital libraries.

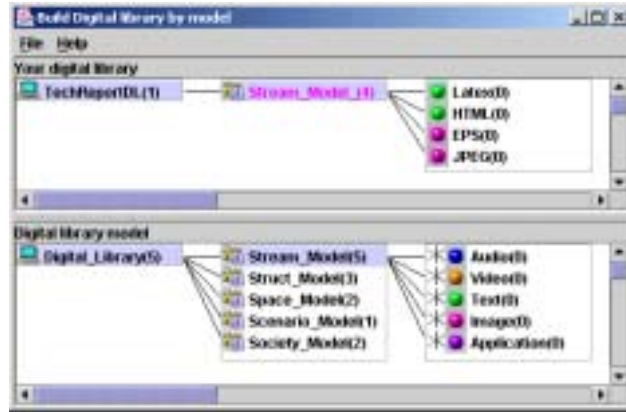
When a new component is needed, the user does not need to build a component from scratch. He loads a similar component and spends relatively less time by making minor changes to the loaded component.

3.6.3.1 Scenario of Component Reuse

A user builds the Stream Model once and saves it as a file on a local disk. Later, when the user starts to build a new digital library, he right-clicks on the new digital library name, chooses “load” in the pop-up menu (see Figure 3.28) and loads the previously saved Stream Model file. Then, a Stream Model appears in the new digital library.



(a) Before Loading



(b) After Loading

Figure 3.28 The Scenario of Component Reuse

3.6.3.2 Non-Reusable Components

Not all components are reusable. A component to be reused should not contain any parts that are constrained to a specific model. In other words, a reusable component should be self-contained and independent of any other specific models.

3.6.3.3 Top-down or Bottom-up Methodology

5SGraph is designed to support a top-down modeling methodology. With the aid of component reuse, the tool also can be used in a bottom-up way. The essential steps are as follows.

- 1) Create a metamodel for each type of component.
- 2) Use the tool to build basic components as building blocks.
- 3) Build higher-level components after all building blocks have been built.
- 4) Build top-level component after all underlying components have been built.

The bottom-up methodology relies on the fact that 5SGraph can create not only a complete digital library, but also the components of a digital library. For example, we can create an instance of the Stream Model using 5SGraph (see Figure 3.29). In this example, a metamodel for the Stream Model is loaded first. Then, a Stream Model instance is created which later can be reused to build a complete digital library. This illustrates the bottom-up methodology.

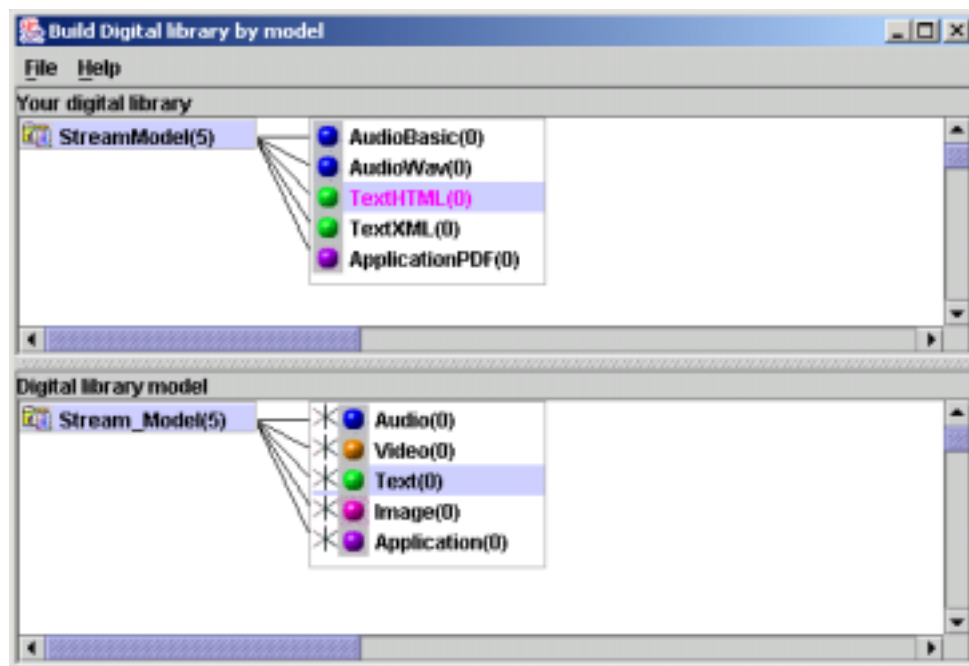


Figure 3.29 Create an Instance of the Stream Model Using 5SGraph

Chapter 4. Metamodel

This chapter describes the specification of a metamodel and the display of the metamodel in 5SGraph.

4.1 Requirements for the Metamodel

A metamodel is central to the functioning of the 5SGraph tool. Without a metamodel, the tool is just an empty facility for hierarchical frameworks with no substance. A metamodel describes a generic digital library, provides the building blocks, and sets up the relationships among these building blocks. A metamodel is based on the 5S theory and should be created by a digital library expert who has a good understanding of the 5S theory.

As we point out in previous chapters, input from digital library experts to the 5SGraph tool includes not only modeling information, but also configuration information for visual effects, such as icon/type associations. This configuration information also is included in the specification of our metamodel.

To summarize, a metamodel should provide the following information:

- Component types
- Properties with each type

- Icon/type associations
- Hierarchical structure information
- Cardinality constraints
- Semantic constraints

4.2 Overview of a Metamodel Example

The metamodel of 5SGraph specifies two types of elements: *DataType* and *DataSet*. The root element is always *DLMetaModel*. All *DataSet* elements must be defined before the elements of *DataType*, as is shown in Figure 4.1.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<DLMetaModel>
  + <Text_Content type="DataSet">
  + <lang_content type="DataSet">
  + <char_content type="DataSet">
  + <Audio_Content type="DataSet">
  + <Video_Content type="DataSet">
  + <Image_Content type="DataSet">
  + <Application_Content type="DataSet">
  + <MetaData_Content type="DataSet">
  + <RenderingContent type="DataSet">
  + <RetrievalSpaceContent type="DataSet">
  + <StemmingAlgContent type="DataSet">

  + <Digital_Library type="DataType">
  + <Stream_Model type="DataType">
  + <Struct_Model type="DataType">
  + <CollectionSet type="DataType">
  + <Space_Model type="DataType">
  + <Scenario_Model type="DataType">
  + <Society_Model type="DataType">
  + <ServiceSet type="DataType">
  + <Services type="DataType">
  + <Scenario type="DataType">
  + <Text type="DataType">
  + <Audio type="DataType">
```

```

+ <Video          type="DataType">
+ <Image          type="DataType">
+ <Application    type="DataType">
+ <Collection     type="DataType">
+ <Document       type="DataType">
+ <Stream         type="DataType">
+ <CatalogSet     type="DataType">
+ <Catalog        type="DataType">
+ <MetaDataFormat type="DataType">
+ <Actor          type="DataType">
+ <Actors_Soc     type="DataType">
+ <Managers_Soc   type="DataType">
+ <Manager        type="DataType">
+ <Repository_MGR type="DataType">
+ <Org_Tool       type="DataType">
+ <AuthorityFile  type="DataType">
+ <ClassificationSchema type="DataType">
+ <Thesaurus      type="DataType">
+ <Ontology       type="DataType">
+ <UI            type="DataType">
+ <IR            type="DataType">
+ <Rendering      type="DataType">
+ <Index         type="DataType">
</DLMetaModel>

```

Figure 4.1 Overview of a Metamodel

We explain in detail the structure, attributes, and visual notations for these two types of elements in the following sections. Scenario examples also are given to facilitate better understanding of these concepts.

4.3 DataSet

DataSet represents a set of data. It is a range of data from which a value can be chosen.

For example, a declaration for *MetaData* is illustrated in Figure 4.2, as follows:

```

<MetaData_Content type="DataSet">
  <item value="Dublin Core" />
  <item value="IMS" />
  <item value="MARC" />
  <item value="RFC1807" />

```

```

    <item value="IEEE LTSC-LOM" />
</MetaData_Content>

```

Figure 4.2 MetaData_Content

The attribute *type* demonstrates that *MetaData_Content* is a *DataSet*, which has a set of values. The above example means *MetaData_Content* has values of Dublin Core, IMS, MARC, RFC1807, and IEEE LTSC-LOM. *DataSet* has no independent visualization in the tool. It only acts as a data source that is used by a *DataType*. A *DataSet* may represent a set of image types, a set of audio types, and many other value sets (see Figure 4.3).

```

= <Image_Content type="DataSet">
    <item value="image/gif" />
    <item value="image/jpeg" />
    <item value="image/bmp" />
    <item value="image/eps" />
    <item value="image/tiff" />
</Image_Content>

= <Audio_Content type="DataSet">
    <item value="audio/basic" />
    <item value="audio/wav" />
    <item value="audio/x-aiff" />
</Audio_Content>

```

Figure 4.3 Image_Content and Audio_Content

4.4 DataType

A *DataType* is represented by a component of the metamodel, which specifies a type and can be instantiated into many instances in the user model. Examples of *DataTypes* are digital library, stream model, structure model, and document. An element that has been declared as a *DataType* is visualized in the toolbox as a named rectangle. Every

DataType element consists of three parts: *SubNodes*, *Icon*, and *Property*. In the following sections, the visualization of each part is introduced, along with examples and scenarios.

4.4.1 SubNodes and Icon

An example of a *DataType* declaration and its visual notation appears in Figure 4.4.

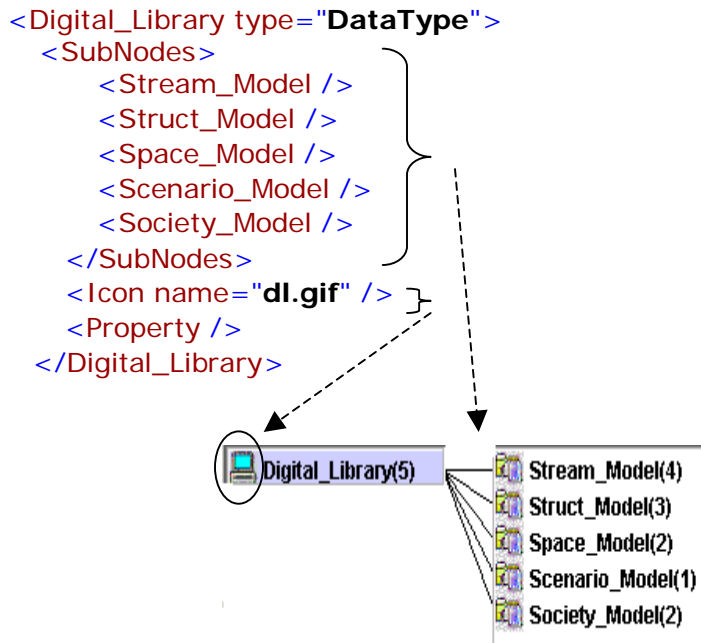



Figure 4.4 *Digital_Library* and its Visual Notation

The attribute *type* shows that the element *Digital_Library* is a *DataType*. The element *SubNodes* contains other *DataTypes* that are the child nodes of this element. In this example, the *SubNodes* of *Digital_Library* consist of *Stream_Model*, *Struct_Model*, *Space_Model*, *Scenario_Model*, and *Society_Model*.

In the visualization, the number of child nodes is specified in parenthesis after the name of *Digital_Library*. The *Icon* specifies that the image file “dl.gif” () is used as the icon for the element *Digital_Library*. No Property is defined for *Digital_Library*.

The *Digital_Library* example does not specify the cardinality of the child nodes, which means the default cardinality is used. The default value of cardinality is set to be “at most once”. Therefore, *Digital_Library* can have at most one *Stream_Model*, at most one *Struct_Model*, at most one *Space_Model*, at most one *Scenario_Model*, and at most one *Society_Model*. However in the *Stream_Model*, its cardinality is explicitly specified.

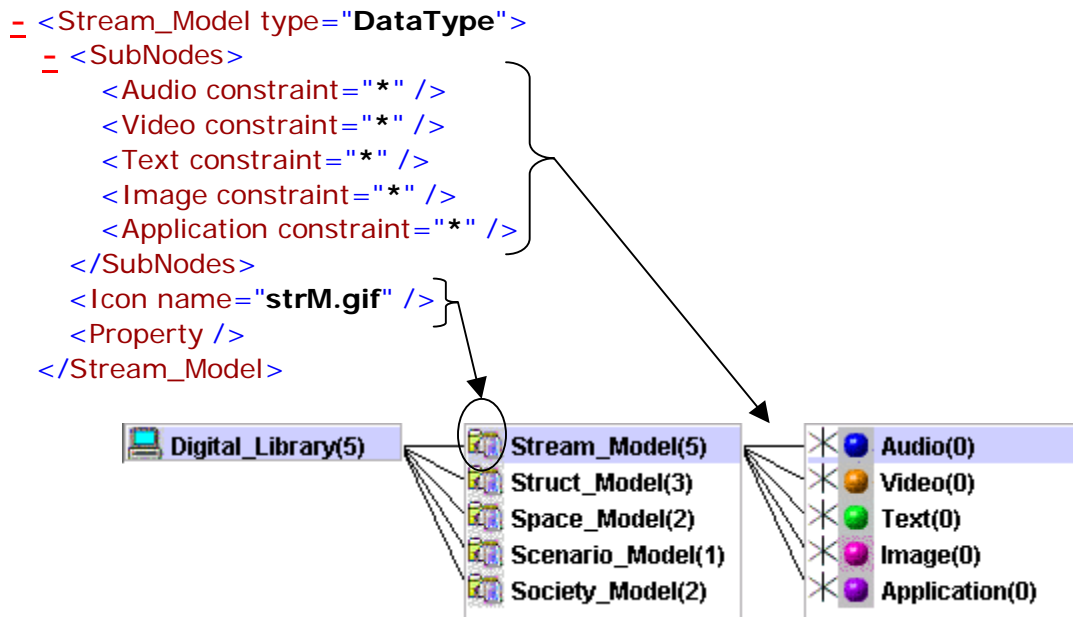


Figure 4.5 *Stream_Model* and its Visual Notation

The *SubNodes* part contains four other *DataTypes*: *Audio*, *Video*, *Text*, *Image* and *Application*. This means that *Stream_Model* has five types of elements as its child nodes. The “constraint” attribute specifies the cardinality. The constraint of “*” means ‘any’

which means an instance of the *Stream_Model* may have any number of instances from among the kinds: audio, video, text, etc.

Scenario 1

Purpose: This scenario shows how to use an element of *DataType* in the user model.

A user wants to create his own digital library (see Figure 4.6). First, he adds an instance of *Digital_Library* to his model. Then he clicks on the instance and changes the name to *My_DL*. His clicking on the instance triggers the synchronization mechanism. The tool expands the *Digital_Library* and shows the child nodes of *Digital_Library* in the toolbox. Now, the user can add the necessary child nodes to his model. He adds a *Stream_Model* and a *Struct_Model*, and changes the names to *My_Streams* and *My_Structures*.

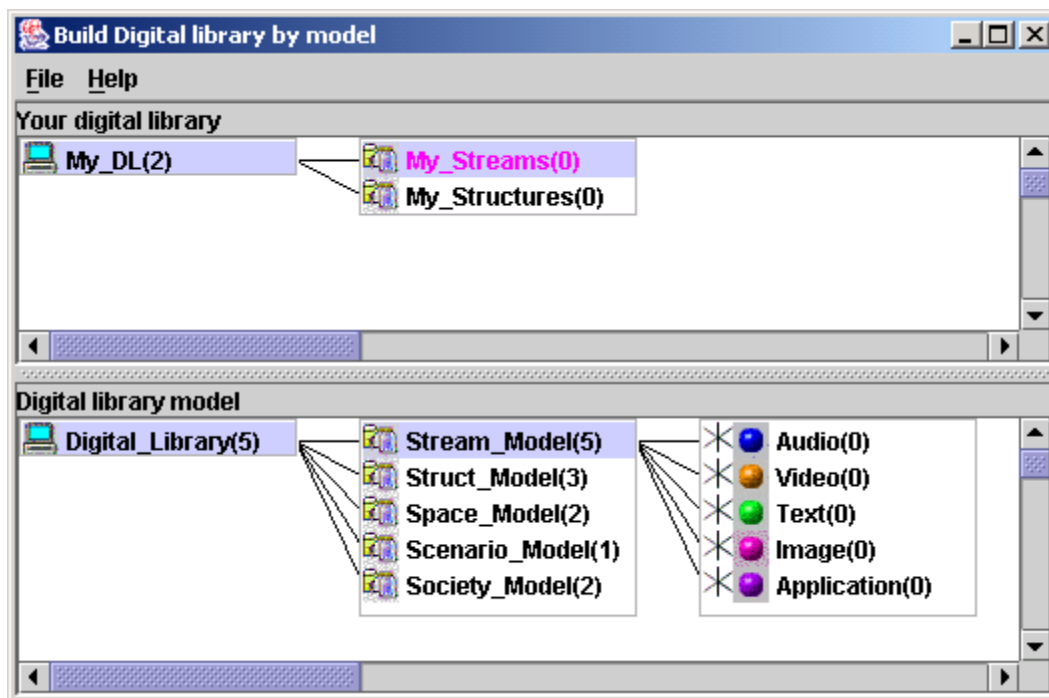


Figure 4.6 Screen Shot for Scenario 1

Scenario 2

Purpose: Explain how to use cardinality in *DataType*.

The user (see Figure 4.7) decides to specify streams in his *Stream_Model*: *My_Sstreams*. When he clicks on *My_Sstreams*, the tool synchronizes to show the child nodes of the *Stream_Model* in the toolbox. Therefore, he knows what kind of streams his Stream Model can have. He adds three kinds of audio: *Basic_Audio*, *Wav_Audio*, and *Aiff_Audio*.

There are three points we need to pay attention to:

1. The user can add any number of audio streams because the cardinality of audio in *Stream_Model* indicates the instance of Audio can appear any number of times with one parent.
2. Cardinality is only indicated in the toolbox (i.e., the metamodel), not in the user model.
3. The icons of all the instances of Audio are the same, because they are of the same type (audio type).

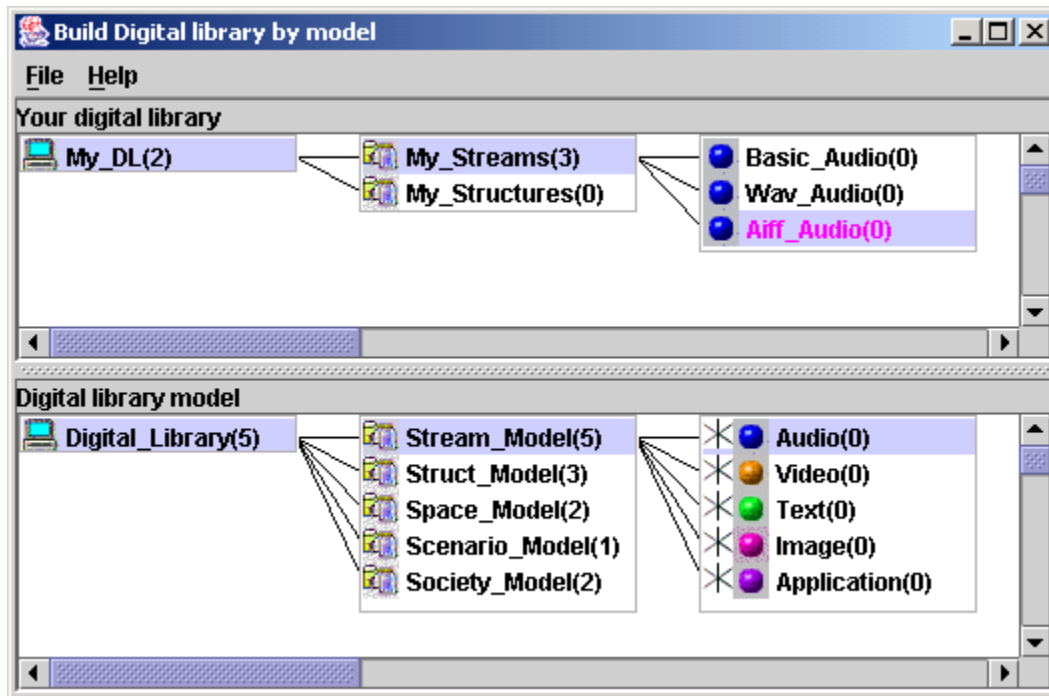


Figure 4.7 Screen Shot for Scenario 2

4.4.2 Property

The *Property* of a *DataType* specifies what properties the element should have. All the specified properties are shown on the property sheet that is associated with each *DataType* element. The metamodel may specify properties for a *DataType* element or may not specify any properties.

If no property is specified for an element, the element will only have the default properties on its property sheet. For example, no property is specified for the element *Digital_Library*. All properties that the *Digital_Library* can have are those default properties. The property sheet of *Digital_Library* is shown in Figure 4.8. The 'Node

Type' and 'Enter a new name' are default properties that every *DataType* element will have.

```
= <Digital_Library type="DataType">  
  = <SubNodes>  
    <Stream_Model />  
    <Struct_Model />  
    <Space_Model />  
    <Scenario_Model />  
    <Society_Model />  
  </SubNodes>  
  <Icon name="dl.gif" />  
  <Property />  
</Digital_Library>
```

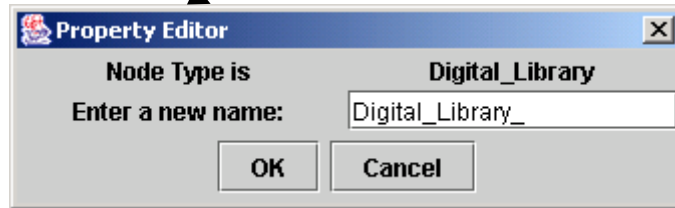


Figure 4.8 Digital_Library and its Property Sheet

The properties that can be specified in *Property* are in the format of interface elements: Combo Box and Text Field. The next example shows how to use Combo Box in the *Property* part. The declaration of *DataType* Audio (see Figure 4.9) is:

```

<Audio_Content type="DataSet">
  <item value="audio/basic" />
  <item value="audio/wav" />
  <item value="audio/x-aiff" />
</Audio_Content>

.....
<Audio type="DataType">
  <Icon />
  <Property>
    <ComboBox name="content-type" src="Audio_Content" />
  </Property>
</Audio>

```

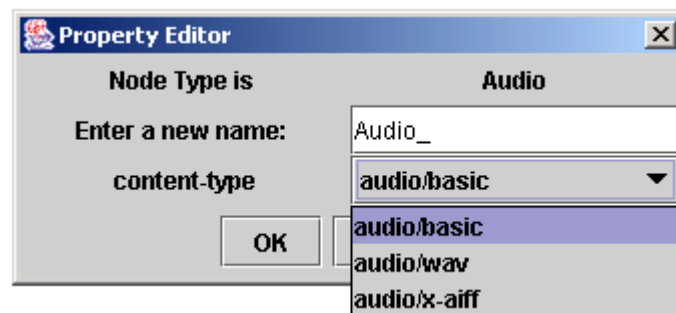
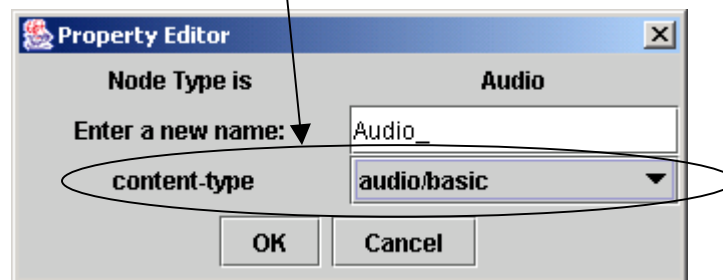


Figure 4.9 Audio and its Property sheet

The property sheet of *Audio* has three parts: Node Type, Enter a new name, and a ComboBox named “content-type”. The content of the ComboBox is obtained from the DataSet “*Audio_Content*”.

Text field is another interface component that can be added to the property part. For example (see Figure 4.10), *DataType Catalog* has four defined properties as shown below.

```

=<Catalog type="DataType">
=<SubNodes>
  <MetaDataFormat constraint="*" />
</SubNodes>
<Icon />
=<Property>
  <TextField name="Description" />
  <TextField name="Creator" />
  <TextField name="Maintainer" />
  <ComboBox name="Collection" src="CollectionSet" />
</Property>
</Catalog>

```

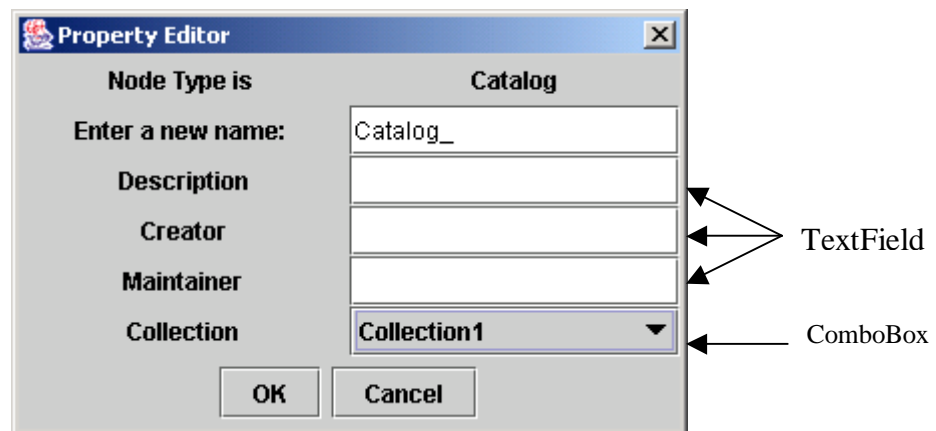


Figure 4.10 Catalog and its Property Sheet

Three text fields “Description”, “Creator”, and “Maintainer” are added to the property sheet of *Catalog*.

If the text field has an attribute ‘Load’ with the value of ‘true’, it means the value of the property can be loaded from some file on disk.

```

= <Ontology type="DataType">
  <SubNodes />
  <Icon />
  = <Property>
    <TextField name="Load_Ontology" load="true" />
  </Property>
</Ontology>

```

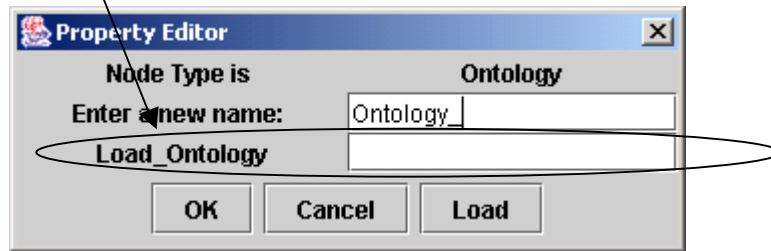


Figure 4.11 Ontology and its Property Sheet

We have described the specification and the visualization of our metamodel. In the following section, a very important feature of the 5SGraph tool is introduced – constraint management.

4.5 Constraint Management

The analysis of 5S theory in Chapter 3 shows that there are certain inherent semantic constraints in the hierarchical structure of the 5S model. The semantic constraints in 5S are divided into two categories: value constraint and association constraint. The *value constraint* specifies the range of possible values of an element, while the *association constraint* defines the relationships among different components. For example:

- An *Actor* can only participate in the services that have been defined in the *Scenario model*. This is a value constraint on the services.

- A *Catalog* must have a 1:1 relationship with a *Collection*. This is an association constraint between a *Catalog* and a *Collection*.

The 5SGraph tool is able to implement and manage these constraints. The following explanation uses two examples to show how the metamodel specifies constraints. Modeling scenarios also are given.

4.5.1 Value Constraint

Example. An *Actor* can only participate in the services that have been defined in the *Scenario Model*. The declaration of *Actor* is shown in Figure 4.12.

```
= <Services type="DataType">
  = <SubNodes>
    <Scenario constraint="*" />
  </SubNodes>
  <Icon name="strM.gif" />
  = <Property>
    <TextField name="Load_Services" load="true" />
  </Property>
</Services>

= <Actor type="DataType">
  <SubNodes src="Services" />
  <Icon />
  <Property />
</Actor>
```

Figure 4.12 Actor and Services

The *SubNodes* part of *Actor* specifies that the child nodes of *Actor* come from source ‘*Services*’, which means only the existing instances of *Services* can become the child nodes of *Actor*.

Scenario 3

Purpose: Shows how 5SGraph maintains value constraints.

The designer is building a model for the NDLTD digital library using 5SGraph (see Figure 4.13). He browses to the *Society Model* and adds anew an *Actor* named “Students” to his instance model. If he wants to add services to students now, he cannot do it because no service has been specified in the *Scenario Model* yet. (This maintains the value constraint.)

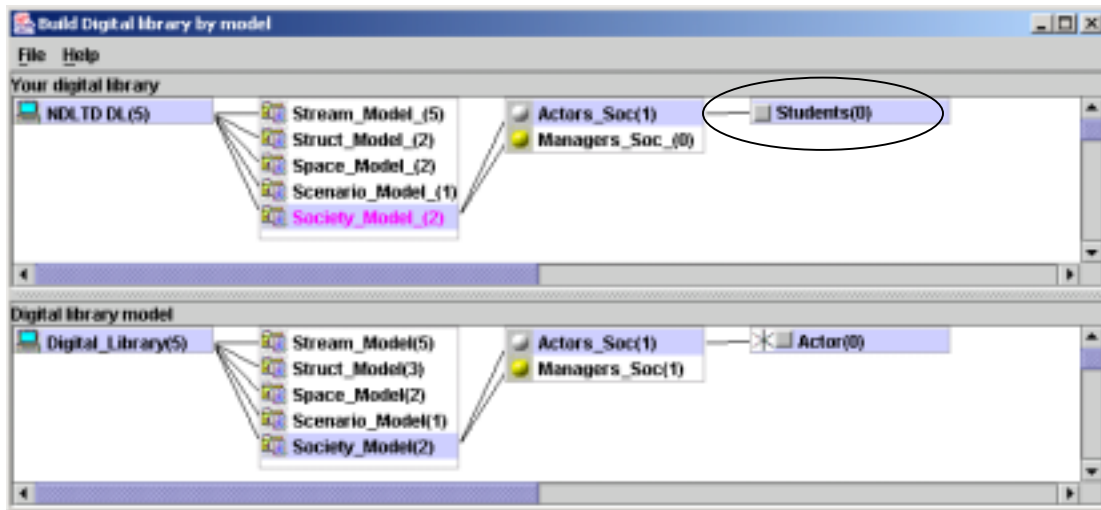


Figure 4.13 Screen Shot 1 for Scenario 3

Then, the designer browses to the *Scenario Model* part and creates five services: fulltext_search, metadata_search, browsing, submission, and training (see Figure 4.14).

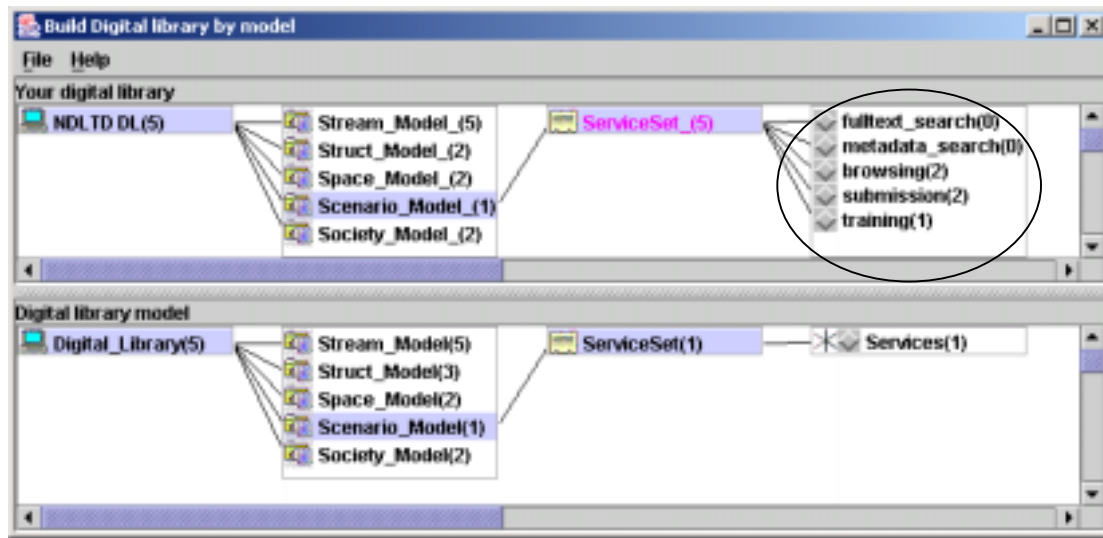


Figure 4.14 Screen Shot 2 for Scenario 3

When the designer browses back to *Students*, he finds out that five services are automatically added into the metamodel under the node “Actor” (see Figure 4.15). 5SGraph dynamically maintains this for the user.

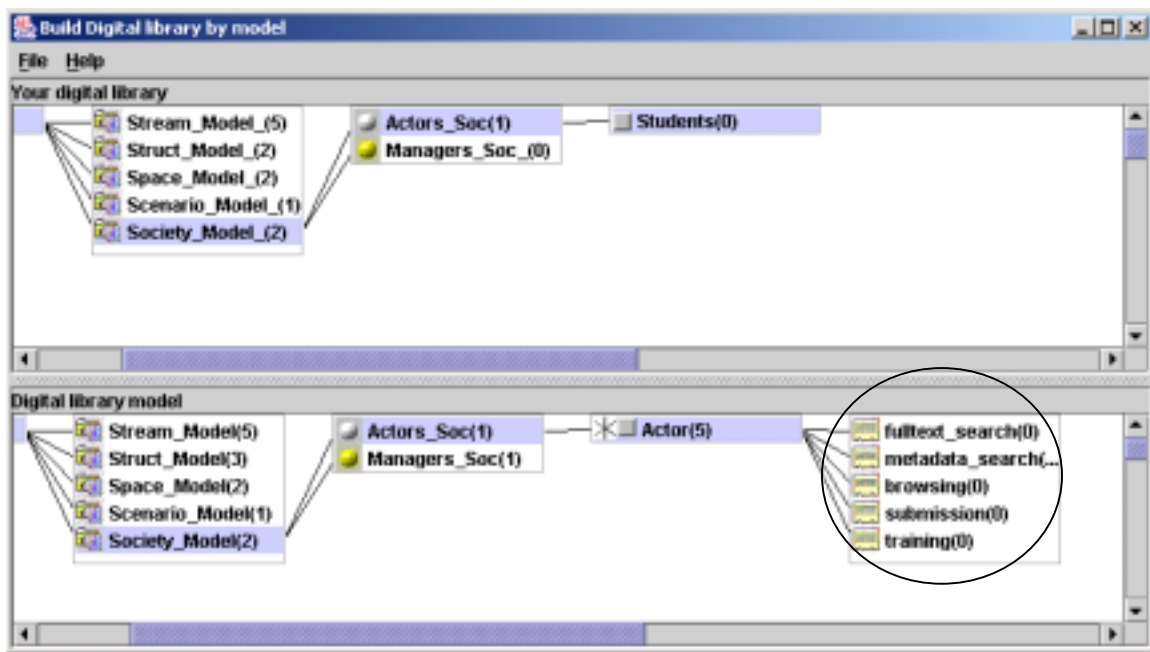


Figure 4.15 Screen Shot 3 for Scenario 3

The designer then adds all five services to *Students* (see Figure 4.16), which means *Students* can participate in all five services provided by the NDLTD digital library.

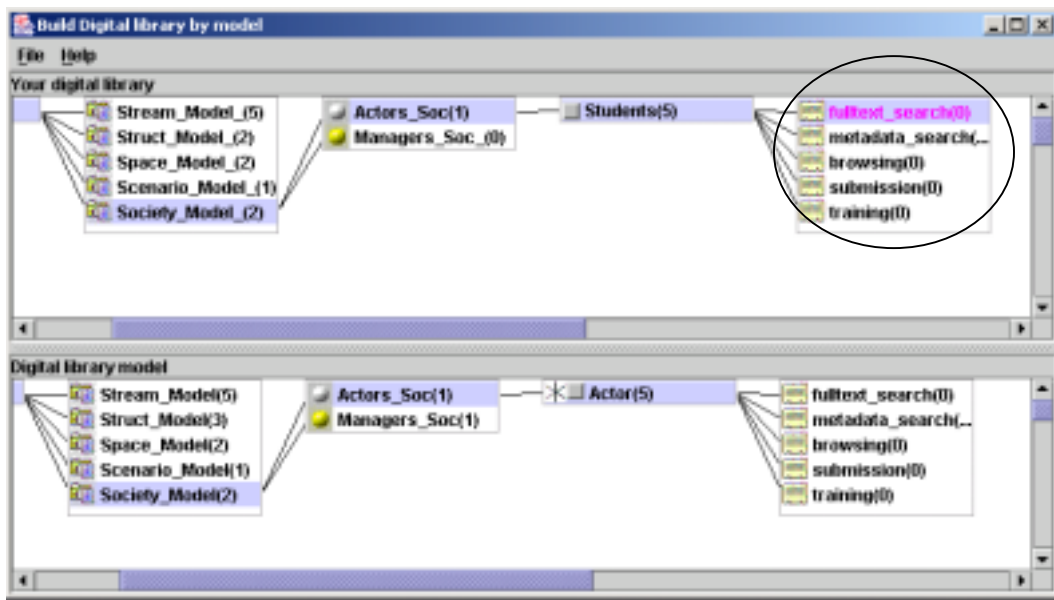


Figure 4.16 Screen Shot 4 for Scenario 3

4.5.2 Association Constraint

Example. A *Catalog* has descriptive metadata for digital objects in a specific *Collection*. Therefore, a *Catalog* must have a 1:1 relationship with a *Collection*, which means a *Catalog* is not independent. A *Catalog* must depend on one existing *Collection*.

The declaration of *Catalog* is described (see Figure 4.17) below:

```

=<Catalog type="DataType">
  <SubNodes>
    <MetaDataFormat constraint="*" />
  </SubNodes>
  <Icon />
  <Property>
    <TextField name="Description" />
    <TextField name="Creator" />
    <TextField name="Maintainer" />
    <ComboBox name="Collection" src="CollectionSet" />
  </Property>
</Catalog>

```

Figure 4.17 The Declaration of the Catalog

In the declaration of Combo box, the attribute *src* refers to *CollectionSet*, which is a *DataType*, not a *DataSet*. If the *src* refers to a *DataSet*, the content of a Combo box is fixed, because the data from the *DataSet* is fixed. If the *src* refers to a *DataType*, the content of the Combo box depends on the content of the instance of that *DataType*. The content of a Combo box dynamically changes with the user model. In this case, the content of the ComboBox depends on what the designer puts into the *CollectionSet*.

Scenario 4

Purpose: Illustrate how to utilize an association constraint.

A designer is building a model for the CITIDEL digital library. He first builds nine collections in his *Structural Model* (see Figure 4.18).

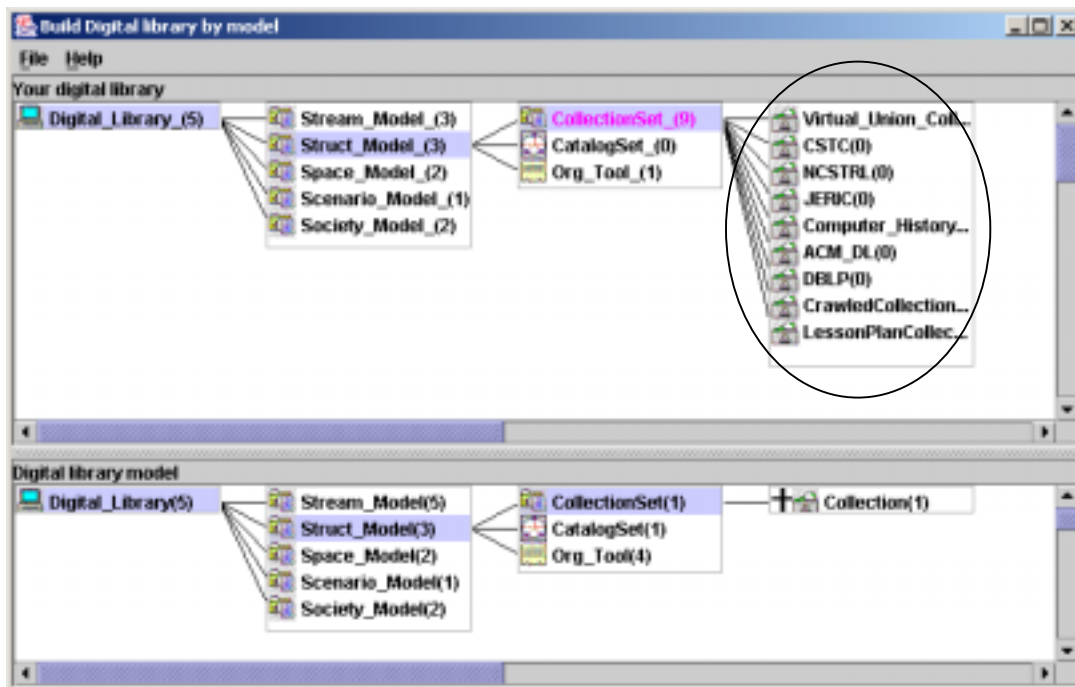


Figure 4.18 Screen Shot 1 for Scenario 4

Then he creates a catalog (see Figure 4.19).

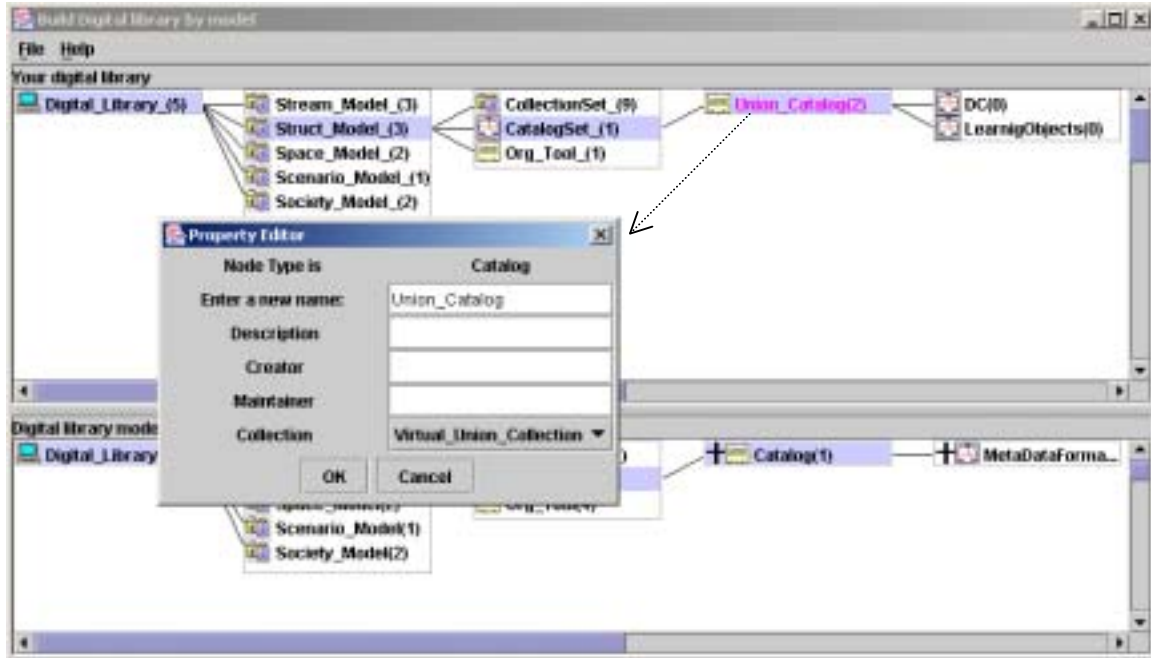


Figure 4.19 Screen Shot 2 for Scenario 4

In the property sheet of the *Catalog*, the *ComboBox* forces him to choose a collection from existing collections for this catalog. He chooses the *Virtual_Union_Collection* and then names the Catalog as *Union_Catalog* (see Figure 4.19).

From the above examples, we conclude that it is very easy and convenient to maintain and manage semantic constraints (e.g., a value constraint and association constraint) using 5SGraph. This feature of 5SGraph greatly improves the usability of the tool and frees the user from the burden of remembering and maintaining the constraints manually.

4.6 Summary

The specification of the metamodel and the related scenarios presented in this chapter address the following questions about the metamodel.

1. What are the types in the metamodel?
2. What are the properties of the types?
3. What icons should the types use?
4. What are the hierarchical structural relationships among the types?
5. What cardinality does a type have?
6. How can one represent the semantic constraints?
7. How can one create an instance model based on the metamodel using 5SGraph?

Based on the specification of the metamodel, 5SGraph not only renders the hierarchical structure of the metamodel and the user model, but also enforces inherent semantic constraints to ensure the consistency and correctness of the user model.

Chapter 5. Preliminary Test of 5SGraph

5.1 Test Objectives

The aim of the testing is to test the performance of the 5SGraph tool. The questions to be answered are:

- Is the tool effective in helping users build digital library models based on the 5S theory?
- Does the tool help users efficiently describe digital library models in the 5S language?
- Are users satisfied with the tool?

Users are asked to complete three tasks. Measures are taken of effectiveness, efficiency, and user satisfaction.

5.2 Test Methods and Procedures

5.2.1 Participants

Participants of this preliminary test include seventeen volunteers from a graduate level Information Storage and Retrieval class, and from the digital library research group of Virginia Tech. We choose participants who have basic knowledge of digital libraries and

have the motivation to create digital libraries. These types of people are also the target users of the tool.

5.2.2 Apparatus

5SGraph is running on a Dell Workstation PWS420. The operating system is Windows 2000. Sessions are recorded by Camtasia, a screen recorder.

5.2.3 Tasks

Three representative tasks with different levels of difficulty are selected:

Task 1: build a simple model of a Technical Report Digital Library using reusable components. The difficulty level of this task is low. Its purpose is to help the participants to get familiar with the 5S model and the 5SGraph tool.

Task 2: finish an existing partial model of CITIDEL (Computing and Information Technology Interactive Digital Educational Library). The difficulty level of this task is medium. In this task, participants create components with more details and complete an existing partial model.

Task 3: build a model of ND LTD (Networked Digital Library of Theses and Dissertations) from scratch. The difficulty level of this task is high. In this task, participants are required to create a complete model from scratch.

5.2.4 Procedures

The documents mentioned in the following procedures are included in Appendix D.

- i) The participant is asked to sign the informed consent form.
- ii) The participant is asked to read some background documents about the 5S model and the modeling methodology.
- iii) The participant is given an introductory presentation on 5S Graph.
- iv) We give the participant a description of task 1 and record how he/she completes it.
- v) After the participant finishes each task, he/she is given the next task description immediately.
- vi) After the participant finishes all the tasks, he/she is given a questionnaire form to fill out.

5.2.5 Test Measures

Effectiveness

- *Completion rate:* percentage of participants who complete each task correctly.
- *Goal achievement:* extent to which each task is achieved completely and correctly.

Efficiency

- *Task time*: time to complete each task.
- *Closeness to expertise*: minimum task time^{*} divided by task time.

* Minimum task time is the shortest period of time that is needed to finish the task, which is measured by the time that an expert with the 5SGraph tool spends on finishing the task.

Satisfaction

Satisfaction is measured using a subjective rating scale. After each participant finishes all three tasks, he/she is given a questionnaire in which the participant is asked to rate the overall learnability, effectiveness, and satisfaction based on his/her observation.

5.3 Results

5.3.1 Performance Results

For task 1:

Participant #	Task time (min)	Closeness to Expertise*	Goal Achievement (%)
2	12	0.417	100
3	10	0.500	90
4	10	0.500	100
5	6	0.833	100
6	11	0.455	80
7	7	0.714	100
8	15	0.333	100
9	8	0.625	100
10	11	0.455	100
11	12	0.417	100
13	9	0.556	100
14	17	0.294	95
15	10	0.500	100
16	16	0.313	95
17	8	0.625	100
18	16	0.313	100
20	14	0.357	95
Mean	11.3	0.483	97.353
Std Deviation	3.331	0.151	5.338

Table 5.1 Performance Results for Task 1

*: The minimum time used to calculate *Closeness to Expertise* is 5 minutes.

For task 2:

Participant #	Task time (min)	Closeness to Expertise*	Goal Achievement (%)
2	11	0.727	100
3	11	0.727	100
4	9	0.889	100
5	9	0.889	100
6	12	0.667	95
7	11	0.727	100
8	25	0.320	85
9	9	0.889	95
10	8	1.000	100
11	8	1.000	100
13	12	0.667	90
14	9	0.889	100
15	10	0.800	100
16	14	0.571	95
17	11	0.727	100
18	14	0.571	100
20	11	0.727	95
Mean	11.412	0.752	97.353
Std Deviation	3.938	0.171	4.372

Table 5.2 Performance Results for Task 2

*: The minimum time used to calculate *Closeness to Expertise* is 8 minutes.

For task 3:

Participant #	Task time (min)	Closeness to Expertise*	Goal Achievement (%)
2	16	0.625	100
3	16	0.625	95
4	10	1.000	100
5	11	0.909	100
6	20	0.500	100
7	12	0.833	95
8	29	0.345	95
9	10	1.000	100
10	11	0.909	100
11	15	0.667	100
13	19	0.526	100
14	14	0.714	100
15	14	0.714	95
16	13	0.769	95
17	14	0.714	100
18	16	0.625	100
20	16	0.625	95
Mean	15.059	0.712	98.235
Std Deviation	4.603	0.178	2.463

Table 5.3 Performance Results for Task 3

*: The minimum time used to calculate *Closeness to Expertise* is 10 minutes.

The summary of the results of all three tasks is given in Table 5.4.

	Task 1	Task 2	Task 3
Completion Rate (%)	100	100	100
Mean Task Time (min)	11.3	11.4	15.1
Mean Closeness to Expertise	0.483	0.752	0.712
Mean Goal Achievement (%)	97.4	97.4	98.2

Table 5.4 Overall Performance Results for Three Tasks

5.3.2 Satisfaction Results

The subjective rating data is based on 10-point bipolar scales, where 1 is the worst rating and 10 is the best rating.

Pre-Understanding refers to the participant's understanding of the 5S model before using the tool. Post-Understanding refers to the participant's understanding of the 5S model after using the tool.

Participant #	Pre-Understanding (Q1)	Post-Understanding (Q2)	Satisfaction (Q4)	Usefulness (Q5)
2	4	8	9	9
3	5	9	10	9
4	9	10	8	10
5	4	7	9	9
6	6	8	9	8
7	5	9	8	10
8	3	10	9	10
9	7	9	9	10
10	7	10	10	10
11	3	5	9	9
13	8	9	9	9
14	7	9	10	8
15	7	9	10	10
16	5	9	10	8
17	3	7	7	7
18	7	10	9	10
20	10	10	9	10
Mean	5.9	8.7	9.1	9.2

Table 5.5 Rating Data

5.4 Discussion

5.4.1 Effectiveness

The high completion rate and the high goal achievement rate prove the effectiveness of 5SGraph.

5.4.2 Efficiency

Most participants finish tasks in less than 20 minutes and the results, the generated 5SL files, are incredibly accurate, which is a strong evidence of efficiency.

Closeness to Expertise reflects the learnability of the tool. We propose four hypotheses about *Closeness to Expertise* and test them using statistics.

Hypothesis 1: the mean *Closeness to Expertise* in task 2 is significantly greater than that in task 1.

This hypothesis is *accepted* by statistical analysis (*t* test) with $\alpha = 0.05$.

Hypothesis 2: the mean *Closeness to Expertise* in task 3 is significantly greater than that in task 1.

This hypothesis is *accepted* by statistical analysis (*t* test) with $\alpha = 0.05$.

Hypothesis 3: the mean *Closeness to Expertise* in task 3 is significantly greater than that in task 2.

This hypothesis is *rejected* by statistical analysis (*t* test) with $\alpha = 0.05$.

Hypothesis 4: the mean *Closeness to Expertise* in task 3 is significantly less than that in task 2.

This hypothesis is *rejected* by statistical analysis (*t* test) with $\alpha = 0.05$.

The acceptance of hypothesis 1 and 2 suggests that the tool is very easy to learn and use. A short task such as task 1 is enough for users to become highly familiar with the tool. Users get much closer to expert performance level after they use the tool for the first time. In fact, there are some participants (participant #9 and participant #10) with good computer skills who achieved a completion speed very close to the expert's in task 2 and task 3.

Hypothesis 3 and 4 are rejected, which suggests that users have similar performance in task 2 and task 3. The reason may be that users have become highly familiar with the tool after task 1. The difference between the participants and the expert may be due to other factors, e.g., typing speed, reading speed, and skills of using computers.

5.4.3 Satisfaction

The average rating of user satisfaction is 9.1 and the average rating of usefulness of the tool is 9.2. Both numbers are rated on a scale of 1 to 10 in which 1 is the lowest score. From these numbers, it appears that our participants are highly satisfied with the tool and consider this tool highly useful for building digital libraries based on the 5S model.

In addition to satisfaction rating, participants rate their understanding of 5S theory before and after using the tool. Mean values of their rating are shown in Figure 5.1. 10 is the best rating, which means the participant understands 5S theory very well. 1 is the worst rating, which means the participant does not understand the theory at all. Statistical

analysis (t test with $\alpha = 0.05$) shows that the mean value of post-understanding is greater than that of pre-understanding. It is observed that the tool is helpful to increase the understanding of the 5S theory.

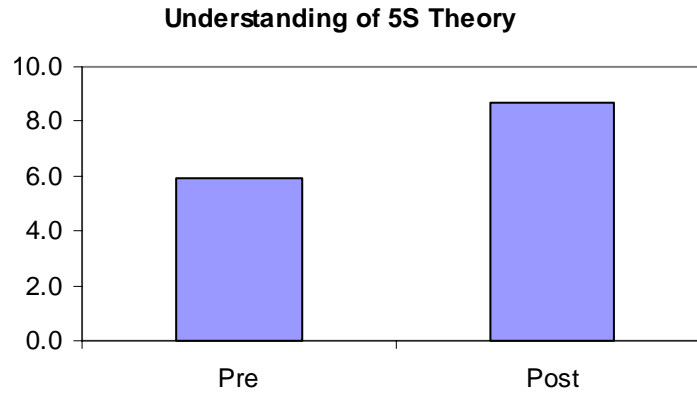


Figure 5.1 The Comparison of Pre and Post Understanding of 5S Theory

5.4.4 Other Observations

Most errors that happen in task 1 are related to reusable components. When some participants are asked to load several Actors or Managers into their Societal Model, they feel bored during the operations and make mistakes. Another example of error is that some of the participants load submodels into the wrong places.

We conclude:

- Reusable components should be large components. Small reusable components often frustrate users.
- Reusable components should come with enough documentation in order to facilitate better understanding and utilization of the components.

Our participants also gave some other suggestions that have been included in the future work section in the conclusion of this thesis.

Chapter 6. An Example Modeling Process Using 5SGraph

An example of modeling a digital library using 5SGraph is given in this chapter. We first give the description of the desired digital library – the Networked Digital Library of Theses and Dissertations (NDLTD). We then specify the detailed steps of using 5SGraph to model this digital library along with graphical illustrations. The final 5SL file is included in Appendix C.

6.1 The Description of NDLTD

The Networked Digital Library of Theses and Dissertations (NDLTD) seeks to change the future of scholarship by ensuring that the future leaders of research, in particular, those who complete a thesis or dissertation, have the requisite knowledge and skills to utilize and take advantage of electronic publishing and digital library (DL) technologies.

A NDLTD digital library involves a local collection of electronic theses and dissertations (ETDs) which students produce as a result of their graduate studies. One of the main objectives of NDLTD is to improve students' skills as effective communicators in the digital age. Therefore we have focused on promoting student's creativity through the use of diverse types of multimedia content in ETDs, while making students comfortable with the utilization of this technology to exploit richer modes of self-expression. Because of preservation and interoperability purposes NDLTD encourages students to use standard,

non-commercial multimedia formats such as XML for text and standards like PDF for texts/images, MPEG for video, and AIFF for audio.

In order to allow students to understand issues of electronic publishing, the NDLTD-DL requires them to submit their own work to the local repository of ETDs, along with corresponding descriptive metadata (e.g., author, abstract, department, etc.). NDLTD has developed and is promoting the Interoperability Metadata Standard for Electronic Theses and Dissertations (ETD-MS) as a standard descriptive metadata set for describing electronic theses and dissertations, which can be converted to MARC and Dublin Core for distribution purposes.

A submission service is controlled by an ETD workflow manager, and includes a cataloguing scenario and a review scenario. While cataloging, the ETD workflow manager helps students enter and edit the descriptive metadata about their ETDs. In the review phase, the university staff checks ETD files, the metadata submitted by the student, and payment of appropriate fees. If everything is OK, the ETD is approved and archived; if not a message about the corresponding problem is sent to the student.

The NDLTD local team should be focused on providing training services (through workshops, online materials, and help in media centers or library sites) to assist students with the authoring or creation of ETDs. One scenario includes giving training workshops to students.

Ideally, the DL also should offer information-seeking services for the local collection to the university patrons (e.g., students, faculty). Those may include fulltext and metadata-based searching as well as browsing by author and department.

6.2 The Modeling Process for NDLTD

The modeling process is as follows:

1. Load a metamodel, as shown in Figure 6.1.

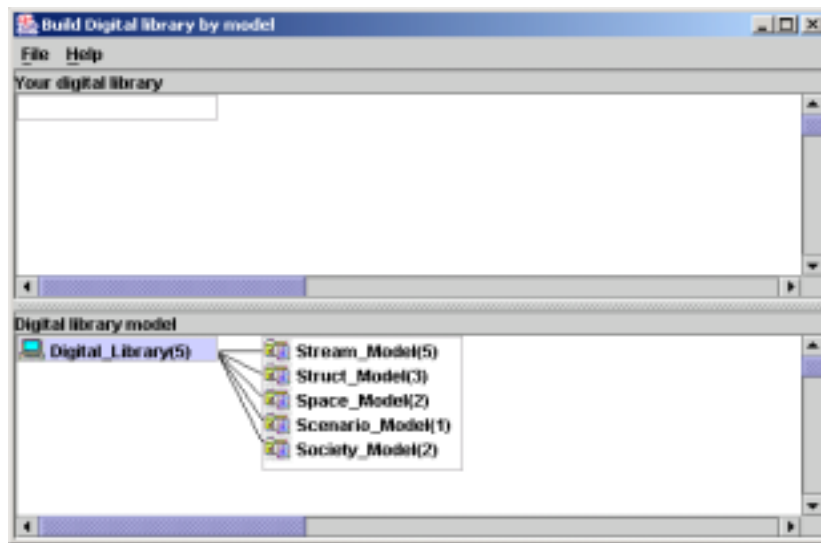


Figure 6.1 Load a Metamodel

2. Add a digital library, and name it as NDLTD. This is the root of the tree, as shown in Figure 6.2.

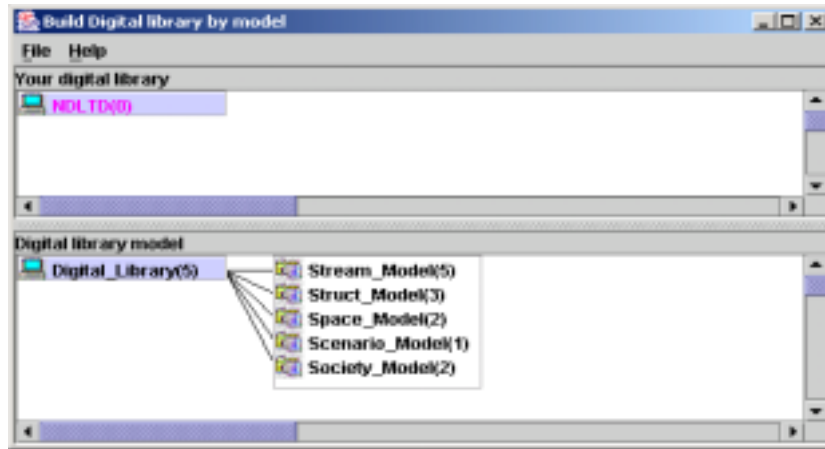


Figure 6.2 Add a Digital Library – NDLTD

3. Add a **Stream Model** with:
 - a. Text: name: XMLText; content-type: text/XML.
 - b. Application: name: PDF; content-type: application/pdf.
 - c. Image: name: JPEGImage; content-type: image/jpeg.
 - d. Video: name: MPEGVideo; content-type: video/mpeg .
 - e. Audio: name: ETDAudio; content-type: audio/x-aiff.

The result is shown in Figure 6.3.

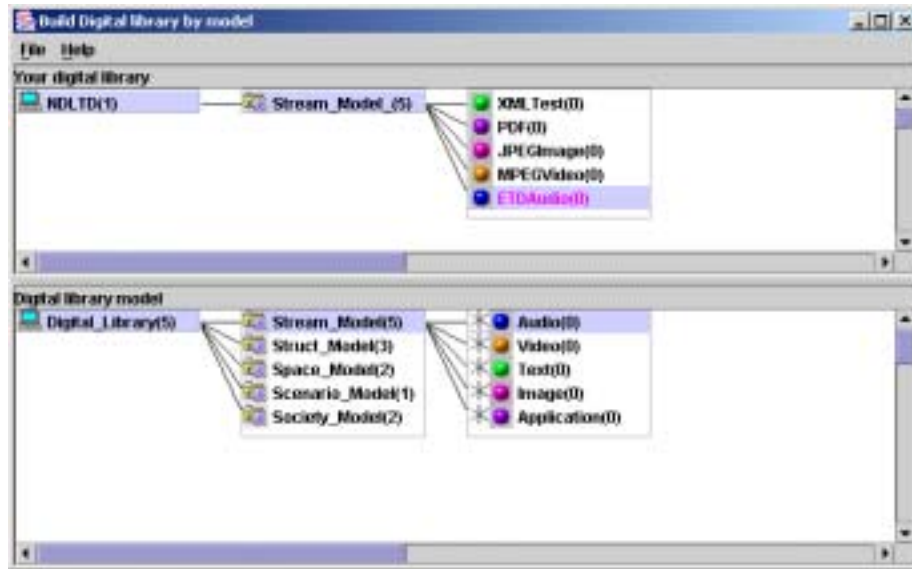


Figure 6.3 Add the Stream Model to NDLTD

4. Add a **Structural Model** with the following steps:
 - a. In CollectionSet: Create one collection (“ETDCollection”) with one type of document (“ETD”). The ETD document has a Structure_Stream property defined in an XML Schema file which should be loaded into the property editor of the document node from the file “etd.xsd”.
 - b. Add 6 stream nodes to the ETD document. Each node is supposed to represent a stream that can occur inside an ETD, and is defined by the stream model which was built in part 1. Specify different stream properties for all different nodes here. For example, the first node should have the name “XMLText” and the stream “XMLText” selected in the comboBox. The second node should be named as “PDF” and have the stream “PDF” selected in the comboBox, and so on. The result is shown in Figure 6.4.

- c. In CatalogSet: Create one catalog (“ETD_Catalog”) with three types of metadata format: Dublin Core, MARC, and ETDMS. Name them as “DC”, “MARC”, and “ETDMS”, respectively. The result is shown in Figure 6.5.

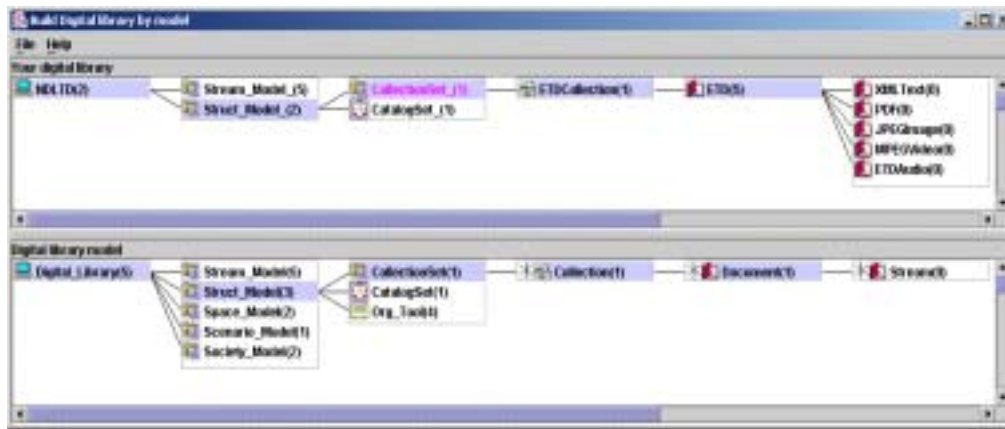


Figure 6.4 Add the ETDCollection to the Structural Model

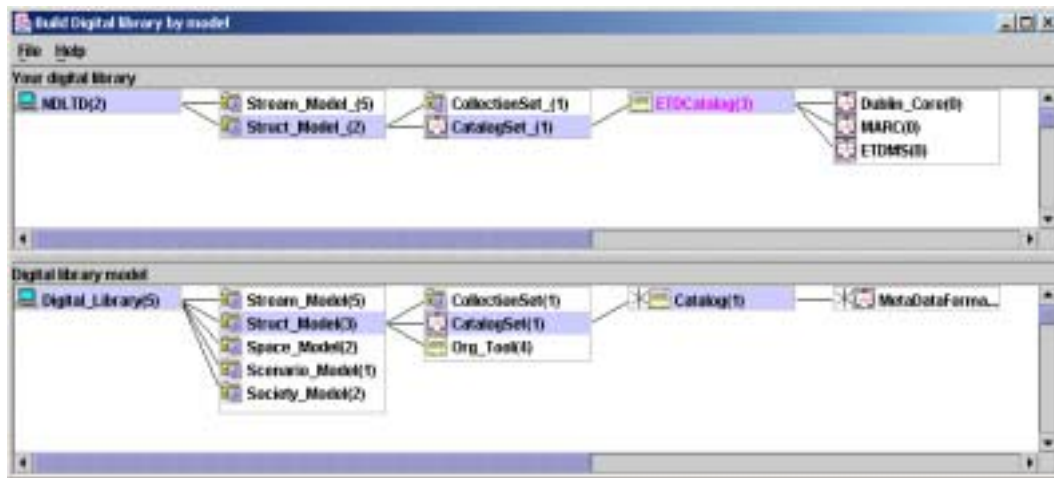


Figure 6.5 Add the ETDCatalog to the Structural Model

5. Add a **Scenario Model** and add the following NDLTD services:
 - a. Fulltext_search

- b. Metadata_search
- c. Browsing with two scenarios
 - i. By_author
 - ii. By_department
- d. Submission with two scenarios
 - i. Cataloguing
 - ii. Review
- e. Training with one scenario
 - i. Giving_workshop

The result is shown in Figure 6.6.

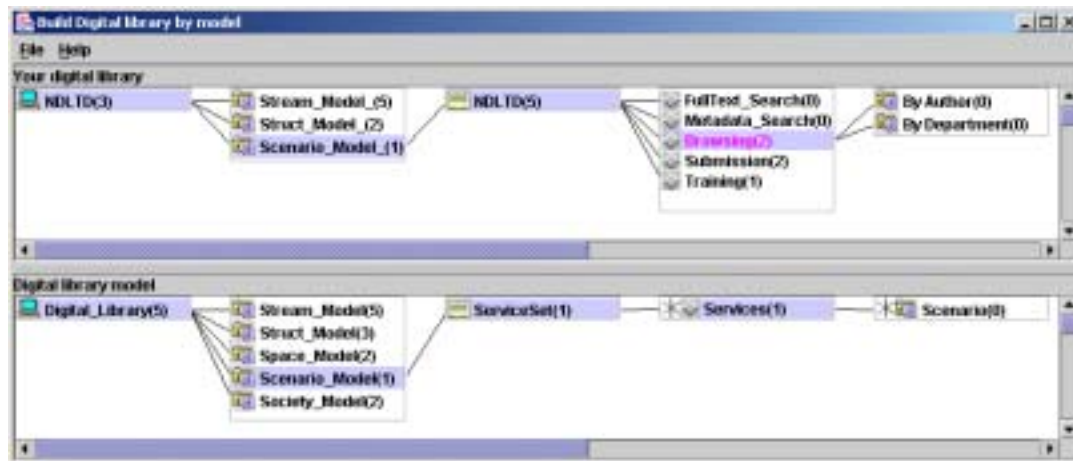


Figure 6.6 Add the Scenario Model into NDTLD

6. Add a **Society Model** with the following societies:
 - a. In the Actor Societies (Actors_Soc), four Actors:
 - i. Students

- ii. Researchers
- iii. NDLTD_Local_Team
- iv. University_staff

b. In the Managers Society, one manager:

- i. ETD_Workflow_Mgr

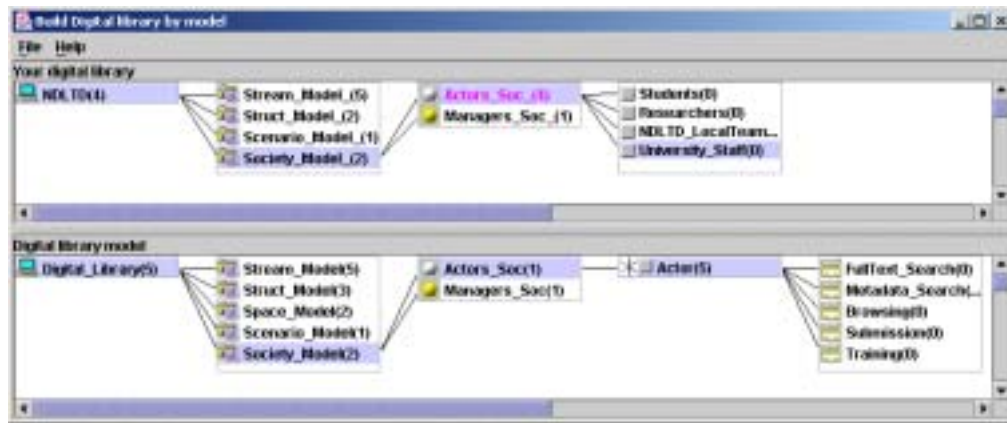


Figure 6.7 Add the Societal Model into NDLTD

7. Connect the societies created in the **Society Model** with the services that you created in the **Scenario Model**:

- a. For Actors:
 - i. Students use all the services.
 - ii. Researchers use information-seeking services (fulltext_search, metadata_search, and browsing).
 - iii. NDLTD_Local_Team is associated with training.
 - iv. University staff is associated with submission.

b. For managers:

i. ETD_Workflow_manager runs the submission service.

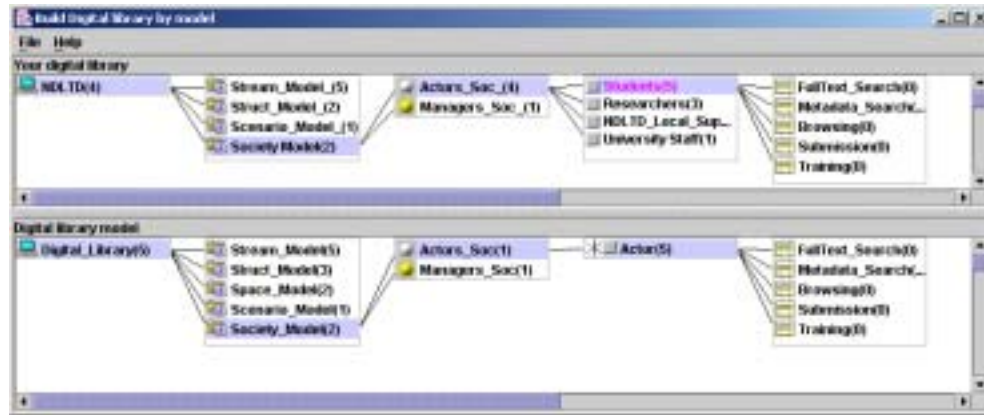


Figure 6.8 Associate the Services with Actors and Managers

7. Save the model.

The output is a 5SL file. It is included in Appendix C.

Chapter 7. Conclusions

A domain-specific visual modeling tool (5SGraph), built for the purpose of modeling and building digital libraries based on the 5S model, is presented in this thesis. We conclude by discussing the contributions of 5SGraph and suggesting directions for future work.

7.1 Contributions of 5SGraph

The work on 5SGraph is an important step in a large project that aims at rapid digital library generation. Rapid digital library generation requires the designers to describe their digital libraries using a specific description language as the first step. 5SGraph reduces the difficulties of this step for non-expert designers. To the best of our knowledge, there has been no other similar modeling tool in this area developed for this purpose.

Characteristics of 5SGraph include:

- 5SGraph is based on a metamodel that describes a generic digital library using the 5S theory.
- 5SGraph helps the designers describe their instance models in 5SL without the need to know the details of 5SL.
- 5SGraph displays the metamodel in a structured toolbox, visualizes both the components and the relationships among them, and provides domain-specific visual components for designers to use.

- 5SGraph provides a top-down hierarchical construction environment for designers to build their digital library models.
- The closeness of the metamodel and instance model helps the designers gather system requirements and reduces modeling difficulties.
- 5SGraph has a powerful capability for constraint management that enforces the underlying semantic constraints over visual components to ensure the consistency and correctness of the resulting instance model.
- Components and their sub-trees in 5SGraph can be saved and reused.
- 5SGraph is designed to be flexible and extensible. 5SGraph does not have any predefined components or semantics. All of its content is from a metamodel that is based on the 5S theory. 5SGraph can have very rich semantics if the semantics provided by the metamodel are rich.

7.2 Future Work

This section outlines possible opportunities for future development.

7.2.1 Integration with Other Tools

Many components in the 5S theory have existing models and editing tools. 5SGraph could associate components with other existing tools to get customized components. The result from other tools should be integrated into 5SGraph and included in the final 5SL file.

7.2.2 Extensions to the Visualization Functionality

5SGraph currently displays the model tree in a truncated way, which simplifies the layout problem and helps users focus on the present context. However, a complete view of the entire tree also may be helpful and useful because sometimes the user would like to see an overview of the entire tree. We believe that a capability for an overview of the entire tree can be added into 5SGraph in the future.

In addition, a print capability must be added to provide documentation of a completed model. Further, 5SGraph presently supports two cardinality indicators. More indicators can be added to enrich the syntax.

7.2.3 Extensions to the Interaction Functionality

The interface could be improved by showing more state information and more help information. Copy-paste-move capabilities are expected to be added as well. The property sheets in 5SGraph currently support two kinds of Java Swing components: JComboBox and JTextField. More could be added in the future to increase flexibility.

7.2.4 Better Evaluation Test

We need more librarians as our participants. We had only one who had the slowest speed in completing the experiment, but showed the best improvement in the understanding of the 5S theory.

Better evaluation tests should be added. These should only give participants some general descriptions of the requirements. Participants will be asked to create a model from scratch directly without step-by-step instructions.

Finally, 5SGraph should be developed for use in situ so as to allow digital library designers to benefit from its capabilities, and so that further insights and improvements can be prompted by real use experience.

References

- [Atkins01] A. Atkins, E. A. Fox, R. France, and H. Suleman. ETD-MS: an Interoperability Metadata Standard for Electronic Theses and Dissertations. Version 1.00. August 2001.
URL: <http://www.ndltd.org/standards/metadata/current.html>. Retrieved in August 2002.
- [Baeza-Yates99] R. Baeza-Yates, B. Ribeiro-Neto. Modern Information Retrieval. Addison-Wesley, Wokingham, UK, 1999.
- [Beech99] D. Beech, S. Lawrence, M. Maloney, N. Mendelsohn, and H. Thompson. XML schema part 1: Structures, May 1999. URL: <http://www.w3.org/TR/xmlschema-1/>
Retrieved in August 2002.
- [Bush45] Vannevar Bush. As We May Think. *Atlantic Monthly* 176(1):101-108, July 1945.
- [Byrne93] Michael D. Byrne. Using icons to find documents: simplicity is critical, *Proceedings of the conference on human factors in computing systems*, pages 446-453, January 1993, Amsterdam, The Netherlands.
- [Castelli02a] Donatella Castelli, Carlo Meghini, and Pasquale Pagano. Foundations of a Multidimensional Query Language for Digital Libraries. *ECDL 2002*: 251-265, Rome, Italy, September 16-18, 2002.
- [Castelli02b] D. Castelli, and P. Pagano. OpenDLib: A Digital Library Service System. *ECDL 2002*:292-308, Rome, Italy, September 16-18, 2002.
- [DLF00] Digital Library Federation. A Working Definition of Digital Library. April 21, 1999. URL: <http://www.clir.org/diglib/dldefinition.htm>. Retrieved in August 2002.
- [DLI94] Digital Libraries Initiative. 1998. URL: <http://dli.grainger.uiuc.edu/national.htm>. Retrieved in August 2002.
- [DLRL02] Digital Library Research Laboratory. Virginia Tech. MARIAN Digital Library Information System. URL: <http://www.dlib.vt.edu/projects/MarianJava/>. Retrieved in September 2002.
- [Elmasri00] R. Elmasri and S. B. Navathe. Fundamentals of Database Systems. Addison Wesley. New York. 2000.
- [Erwig00] M. Erwig. A visual language for XML. *16th IEEE Symposium on Visual Languages*, pages 47 – 54, 2000.

[Fox95] E. A. Fox, R. M. Akscyn, et al. Digital Libraries. *Communications of the ACM* 38(4): 22-28, 1995.

[Gonçalves02] Marcos André Gonçalves and Edward A. Fox. 5SL - A Language for Declarative Specification and Generation of Digital Libraries. *Second ACM/IEEE Joint Conference on Digital Libraries*, pages 263-272, July 2002. Portland, Oregon, USA.

[GonçalvesTOIS] Marcos André Gonçalves, Edward. A. Fox, Layne T. Watson, Neill A. Kipp. Streams, Structures, Spaces, Scenarios, Societies (5S): A Formal Model for Digital Libraries. Under review for ACM Transactions on Information Systems.

[Greenstone02] Greenstone digital library software.
URL: <http://www.greenstone.org/english/faq-general.html> Retrieved in August 2002.

[Halasz94] Frank Halasz, and Mayer Schwartz. The Dexter hypertext reference model. *Communications of the ACM*, 37(2):30-39, February 1994.

[Harmonia02] Harmonia Incorporation. The User Interface Markup Language (UIML).
URL: <http://www.harmonia.com/resources/tutorials/>. Retrieved in September 2002.

[Haring68] D. Haring. A Display Console for an Experimental Computer-based Augmented Library Catalog. *Proceedings of the 23rd ACM national conference*, pages 35-43, 1968.

[IHMC01] Institute for Human and Machine Cognition, The University of West Florida. Concept Map. May 2001. URL: <http://cmap.coginst.uwf.edu/>. Retrieved in August 2002.

[Intrex] Introduction of Intrex Project.
URL: <http://www.cas.usf.edu/lis/lis6260/lectures/diglib.htm>. Retrieved in August 2002.

[Kalinichenko00] L. A. Kalinichenko, D. O. Briukhov, N. A. Skvortsov, and V. N. Zakharov. Infrastructure of the subject mediating environment aiming at semantic interoperability of heterogeneous digital library collections. *In Proceedings of the 2nd Russian Scientific Conference on Digital Libraries: Advanced Methods and Technologies*, Protvino. Sep. 26-28, 2000.

[Kentuckiana] Kentuckiana Digital Library. Digital Library Production Guide version 1.0. URL: <http://www.kyvl.org/kentuckiana/bpguide/projectplans.shtml>. Retrieved in August 2002.

[Khoral01] Khoral Inc. Khoros Pro 2001. URL: <http://www.khoral.com/khoros>. Retrieved in August 2002.

[Lancaster78] F. W. Lancaster. Toward Paperless Information Systems. Academic Press, New York, USA, 1978.

[LOC95] The Library of Congress. The National Digital Library Program. 1995. URL: <http://lcweb.loc.gov/ndl/aug-95.html>. Retrieved in September 2002.

[Lynch00] Clifford Lynch. Searching the Internet: Special Report. *Scientific American*, 276:52-53, March 1997.

[OAI] Open Archives Initiative. URL: <http://www.openarchives.org/>. Retrieved in September 2002.

[OMG02] The Object Management Group. Meta-Object Facility. Version 1.4. 2002. URL: <http://www.omg.org/technology/documents/formal/mof.htm>. Retrieved in September 2002.

[Park92] Byung Joon Park. Domain Modeling for Knowledge-Based Systems: An Approach Based On Objects, Events, and Rules. Technical Report: UIUCDCS-R-97-1992. October 1997.

URL: http://www.cs.uiuc.edu/Dienst/UI/2.0/Describe/ncstrl.uiuc_cs/UIUCDCS-R-97-1992. Retrieved in September 2002.

[Patel-Schneider02] Peter Patel-Schneider, and Jerome Simeon. The Yin/Yang Web: XML Syntax and RDF Semantics. WWW 2002: 443-453. URL: <http://www2002.org/CDROM/refereed/231>. Retrieved in September 2002.

[Pietriga01] E. Pietriga, VXT: A Visual Approach to XML transformation. *Proceedings of the ACM Symposium on Document Engineering*, pages 1-10. November. 2001. Atlanta, Georgia, USA.

[Pirolli01] P. Pirolli, S. K. Card, and M. M. V. D. Wege. Visual information foraging in a Focus+Context visualization. In *Proceedings of CHI 2001*, pages 506-513. 2001. Seattle, Washington.

[Robertson93] G. Robertson, S. K. Card, and J. D. Manindy. Information Visualization Using 3D Interactive Animation. *Communications of the ACM*, 36(4):56-71, 1993.

[Shneiderman01] B. Shneiderman. Tree visualization with Tree-maps: A 2-d space-filling approach. *ACM Transactions on Graphics*, 11(1):92-99, 1992.

[Suleman02] Hussein Suleman. Open Digital Libraries. Ph.D. dissertation. Virginia Polytechnic Institute and State University, Department of Computer Science. Nov. 2002.

[SpeedLegal02] SpeedLegal Inc. Xerlin Project. Version 1.0. URL: <http://www.xerlin.org/>. Retrieved in September 2002.

[Sun02a] Sun Inc. JavaBeans. URL: <http://java.sun.com/>. Retrieved in August 2002.

- [Sun02b] Sun Inc. Java 2 Platform. Standard Version 1.3.1.
URL: <http://java.sun.com/j2se/1.3/docs/api/>. Retrieved in August 2002.
- [Tolvanen01] Juha-Pekka Tolvanen, and S. Kelly. Domain-Specific Modeling: 10 times faster than UML, *Proceedings of Embedded Systems Conference*, Stuttgart, Germany, 2001. URL: <http://www.metacase.com/papers/>. Retrieved in September 2002.
- [Wang99] Bing Wang. A hybrid system approach for supporting digital libraries. *International Journal on Digital Libraries*, 2(2-3):91-110, 1999.
- [William95] William Saffady. Digital Library Concepts and Technologies for the Management of Library Collections: An Analysis of Methods and Costs. *Library Technology Reports* 31.3: 221-380. 1995.
- [Witten02] Ian H. Witten and David Bainbridge, How to Build a Digital Library, *The Morgan Kaufmann Series in Multimedia Information and Systems*, Edward Fox, Series Editor, 552 pages, San Francisco, Calif., Morgan Kaufmann, 2002,

Appendix A. Implementation of 5SGraph

A.1 UML Overview

This part provides UML diagrams to give an overview of the implementation of 5SGraph. Figure A-1 shows the structure and relationships among the core set of classes in 5SGraph.

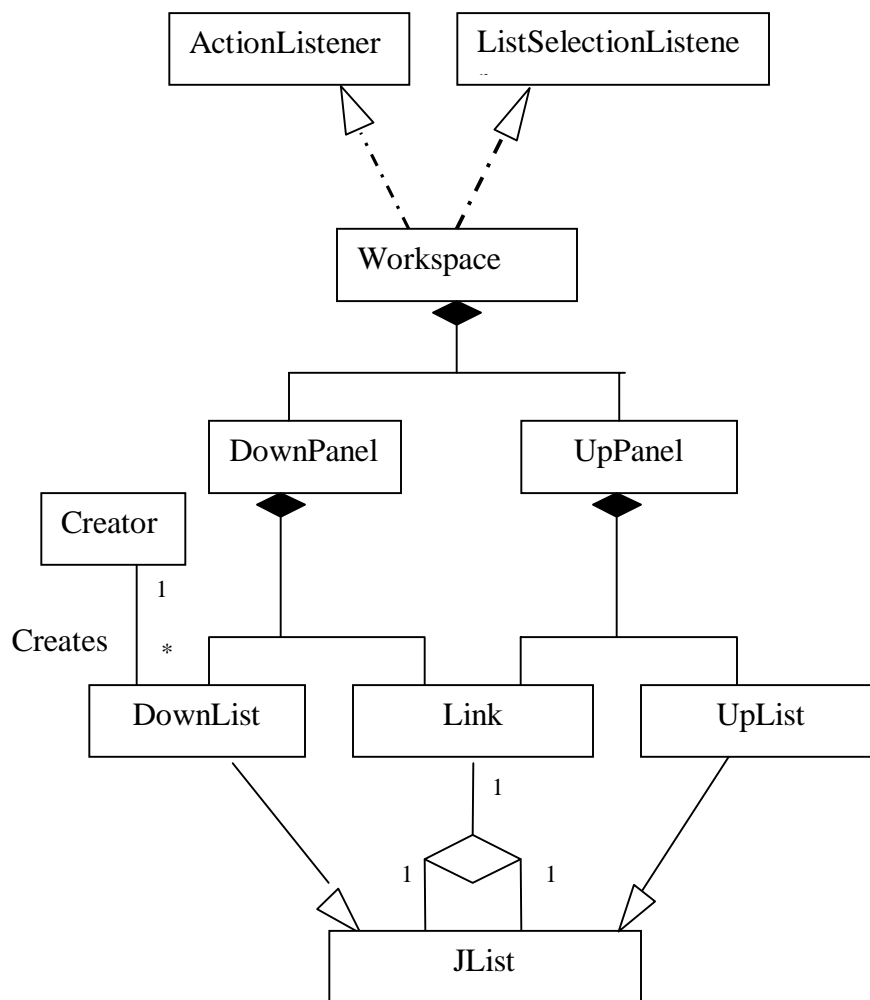


Figure A.1 High Level of Structure of Core 5SGraph Components

Workspace is the container that contains two panels. The upper one, named *UpPanel*, is for the user model. *UpPanel* holds a linked n-ary tree with nodes of *UpLists*. The lower one, *DownPanel*, is for the metamodel. *DownPanel* holds a linked n-ary tree with nodes of *DownLists*. Configured by a metamodel specification, the *Creator* creates all the specific components that are needed to build a model. All *DownLists* are created by the *Creator*, while all *UpList* entries are created based on user interaction. *UpList* and *DownList* are derived classes of *JList*. A *JList* is linked with another *JList* with a *Link*, which is represented as a ternary association.

The n-ary tree is illustrated in Figure A.2.

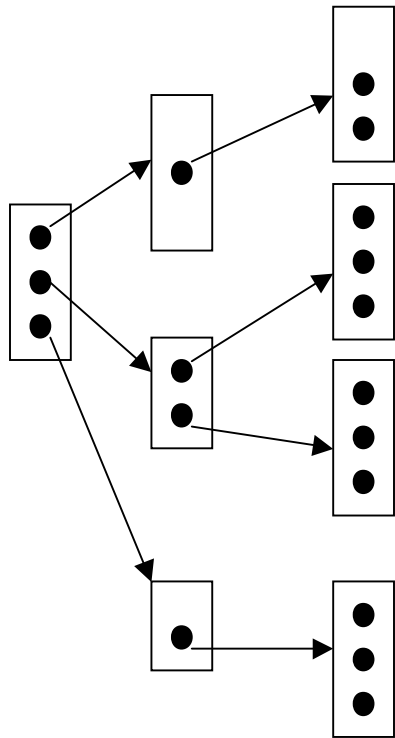


Figure A.2 Illustration of N-ary Tree

A part of the data for DL metamodel is shown (see Figure A.3) as follows.

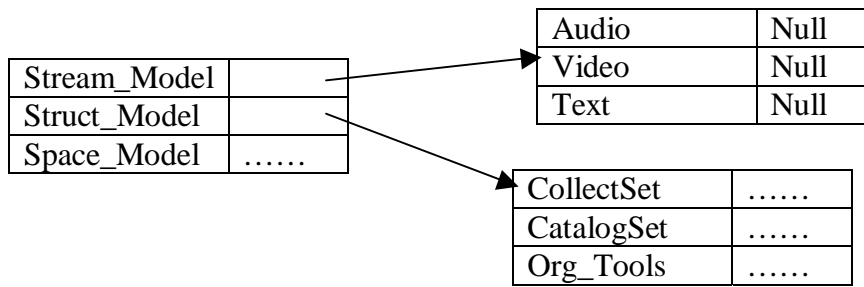


Figure A.3 An Example of n-ary Tree

A *DownList/UpList* implements a node of the tree. Since not all branches of a tree are visible at one time (truncated display), a *DownList/UpList* keeps the information of the currently visible path. The public interface parts of the *DownList* and *UpList* classes are shown in Figure A.4.

DownList	
Purpose	
Implement a node of an n-ary tree for a metamodel.	
Methods	
void	setNextList(DownList dl);
DownList	getNextList();
void	setPrevList(DownList dl);
DownList	getPrevList();
void	setPrevList();
void	setMirrorList(UpList ul);
UpList	getMirrorList();
void	setConsistent(boolean b);
boolean	isConsistent();

UpList	
Purpose	
Implement a node of an n-ary tree for a model.	
Methods	
void	setNextList(UpList dl);
UpList	getNextList();
void	setPrevList(UpList dl);
UpList	getPrevList();
void	setPrevList();
void	setMirrorList(UpList ul);
DownList	getMirrorList();

Figure A.4 DownList/UpList classes

The *getMirrorList* and *setMirrorList* functions are for setting up the consistency between the list in metamodel and the list in the user model. The state of consistency is reflected by the *isConsistent* function in Class *DownList*.

A.2 Model-view separation.

The implementation makes extensive use of a useful object-oriented technique, model-view separation, which means that we have one object to hold the data, and another object to display the view, as shown in Figure A.5. For each UpList, there is an UpListModel that contains data for the UpList. UpList provides the view. DownList and DownListModel have the same relationship as UpList and UpListModel. DownListModel and UpListModel keep the information of the full tree.

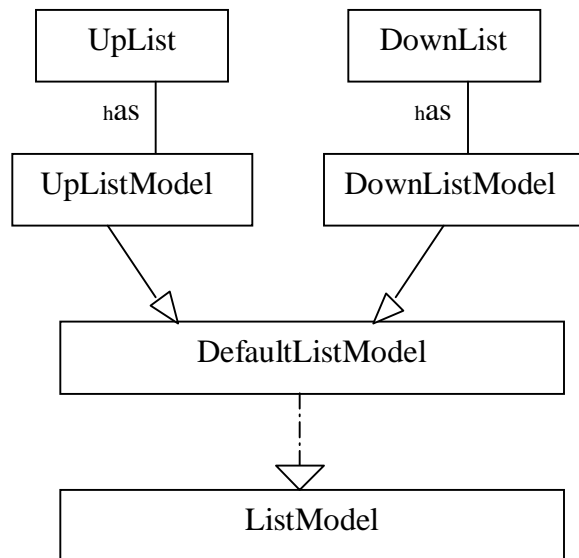


Figure A.5 Model-view Separation

For each data item in the *UpListModel* and *DownListModel*, there is a property editor that illustrates all the properties of the data item (see Figure A.6). For example, data items like *Stream_Model* and *CollectSet* have their specific property editors that are created by the *Creator* (see Figure A.6).

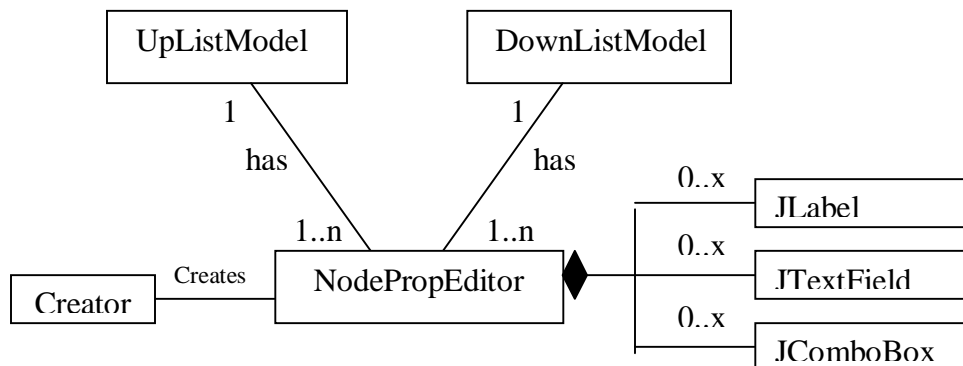


Figure A.6 NodePropEditor Class

A.3 Constraint Management

Constraint management is implemented by Java's event-listener model. If object A is a constraint of object B, object B becomes a listener on object A, i.e., B listens to A. Whenever A has any changes, B decides if it needs to change with it.

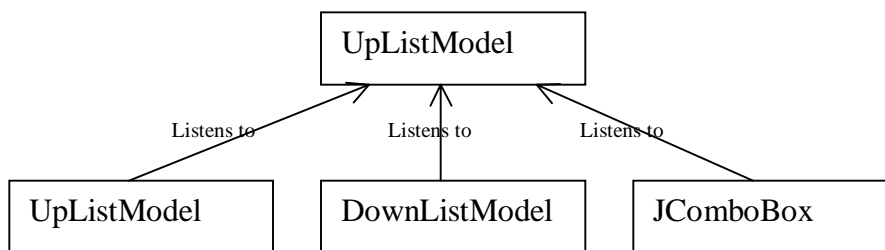


Figure A.7 Constraint Management

The implemented constraints are between *UpListModel* and *UpListModel*, or *UpListModel* and *DownListModel*, or *UpListModel* and *JComboBox*. One *UpListModel* can listen to another *UpListModel*. One *DownListModel* can listen to an *UpListModel*. One *JComboBox* can listen to an *UpListModel*. If an *UpListModel* is a constraint, it dispatches a *ListDataEvent* to all its listeners when its content is changed.

Appendix B. Metamodel

The following is the metamodel used throughout the thesis and the experiment.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <DLMetaModel>
-   - <Text_Content type="DataSet">
-       - <item value="text/plain" />
-       - <item value="text/richtext" />
-       - <item value="text/html" />
-       - <item value="text/latex" />
-       - <item value="text/xml" />
-   </Text_Content>
-   - <lang_content type="DataSet">
-       - <item value="English" />
-       - <item value="Chinese" />
-   </lang_content>
-   - <char_content type="DataSet">
-       - <item value="ISO-8869-1" />
-       - <item value="UTF-8" />
-       - <item value="Unicode" />
-   </char_content>
-   - <Audio_Content type="DataSet">
-       - <item value="audio/basic" />
-       - <item value="audio/wav" />
-       - <item value="audio/x-aiff" />
-   </Audio_Content>
-   - <Video_Content type="DataSet">
-       - <item value="video/mpeg" />
-       - <item value="video/avi" />
-   </Video_Content>
-   - <Image_Content type="DataSet">
-       - <item value="image/gif" />
-       - <item value="image/jpeg" />
-       - <item value="image/bmp" />
-       - <item value="image/eps" />
-       - <item value="image/tiff" />
-   </Image_Content>
-   - <Application_Content type="DataSet">
-       - <item value="application/base64" />
-       - <item value="application/postscript" />
-       - <item value="application/pdf" />
-       - <item value="application/msword" />
-       - <item value="application/x-zip-compressed" />
-       - <item value="application/x-compressed" />
-       - <item value="application/java" />
```

```

        <item value="application/x-msdownload" />
        <item value="application/macbinhex40" />
    </Application_Content>
- <MetaData_Content type="DataSet">
    <item value="Dublin Core" />
    <item value="IMS" />
    <item value="MARC" />
    <item value="RFC1807" />
    <item value="IEEE LTSC-LOM" />
</MetaData_Content>
- <RenderingContent type="DataSet">
    <item value="HTML" />
    <item value="Java" />
</RenderingContent>
- <RetrievalSpaceContent type="DataSet">
    <item value="Boolean" />
    <item value="Probability" />
    <item value="Vector" />
</RetrievalSpaceContent>
- <StemmingAlgContent type="DataSet">
    <item value="Porter" />
    <item value="LIMA" />
</StemmingAlgContent>
- <Digital_Library type="DataType">
    - <SubNodes>
        <Stream_Model />
        <Struct_Model />
        <Space_Model />
        <Scenario_Model />
        <Society_Model />
    </SubNodes>
    <Icon name="dl.gif" />
    <Property />
</Digital_Library>
- <Stream_Model type="DataType">
    - <SubNodes>
        <Audio constraint="*" />
        <Video constraint="*" />
        <Text constraint="*" />
        <Image constraint="*" />
        <Application constraint="*" />
    </SubNodes>
    <Icon name="strM.gif" />
    <Property />
</Stream_Model>
- <Struct_Model type="DataType">
    - <SubNodes>
        <CollectionSet />
        <CatalogSet />

```

```

        <Org_Tool />
    </SubNodes>
    <Icon name="strM.gif" />
    <Property />
</Struct_Model>
= <CollectionSet type="DataType">
= <SubNodes>
    <Collection constraint="*" />
</SubNodes>
    <Icon name="strM.gif" />
    <Property />
</CollectionSet>
= <Space_Model type="DataType">
= <SubNodes>
    <UI />
    <IR />
</SubNodes>
    <Icon name="strM.gif" />
    <Property />
</Space_Model>
= <Scenario_Model type="DataType">
= <SubNodes>
    <ServiceSet />
</SubNodes>
    <Icon name="strM.gif" />
    <Property />
</Scenario_Model>
= <Society_Model type="DataType">
= <SubNodes>
    <Actors_Soc />
    <Managers_Soc />
</SubNodes>
    <Icon name="strM.gif" />
    <Property />
</Society_Model>
= <ServiceSet type="DataType">
= <SubNodes>
    <Services constraint="*" />
</SubNodes>
    <Icon />
    <Property />
</ServiceSet>
= <Services type="DataType">
= <SubNodes>
    <Scenario constraint="*" />
</SubNodes>
    <Icon name="foot_motif.gif" />
= <Property>
    <TextField name="Load_Services" load="true" />

```

```

        </Property>
    </Services>
    = <Scenario type="DataType">
        <SubNodes />
        <Icon name="strM.gif" />
        = <Property>
            <TextField name="Description" />
            <TextField name="Load_Scenario" load="true" />
        </Property>
    </Scenario>
    = <Text type="DataType">
        <Icon name="green-ball.gif" />
        = <Property>
            <ComboBox name="content-type" src="Text_Content" />
            <ComboBox name="charset" src="char_content" />
        </Property>
    </Text>
    = <Audio type="DataType">
        <Icon name="blue-ball.gif" />
        = <Property>
            <ComboBox name="content-type" src="Audio_Content" />
        </Property>
    </Audio>
    = <Video type="DataType">
        <Icon name="orange-ball.gif" />
        = <Property>
            <ComboBox name="content-type" src="Video_Content" />
        </Property>
    </Video>
    = <Image type="DataType">
        <Icon name="pink-ball.gif" />
        = <Property>
            <ComboBox name="content-type" src="Image_Content"
                />
        </Property>
    </Image>
    = <Application type="DataType">
        <Icon name="purple-ball.gif" />
        = <Property>
            <ComboBox name="content-type"
                src="Application_Content" />
        </Property>
    </Application>
    = <Collection type="DataType">
        = <SubNodes>
            <Document constraint="*" />
        </SubNodes>
        <Icon name="coll.gif" />
        = <Property>

```

```

        <TextField name="Description" />
        <TextField name="Creator" />
        <TextField name="Maintainer" />
    </Property>
</Collection>
= <Document type="DataType">
    = <SubNodes>
        <Stream constraint="*" />
    </SubNodes>
    <Icon name="doc.gif" />
    = <Property>
        <TextField name="Structed_Stream" load="true" />
    </Property>
</Document>
= <Stream type="DataType">
    <SubNodes />
    <Icon name="doc.gif" />
    = <Property>
        <ComboBox name="Stream" src="Stream_Model" />
    </Property>
</Stream>
= <CatalogSet type="DataType">
    = <SubNodes>
        <Catalog constraint="*" />
    </SubNodes>
    <Icon name="meta.gif" />
    <Property />
</CatalogSet>
= <Catalog type="DataType">
    = <SubNodes>
        <MetaDataFormat constraint="*" />
    </SubNodes>
    <Icon />
    = <Property>
        <TextField name="Description" />
        <TextField name="Creator" />
        <TextField name="Maintainer" />
        <ComboBox name="Collection" src="CollectionSet" />
    </Property>
</Catalog>
= <MetaDataFormat type="DataType">
    <SubNodes />
    <Icon name="meta.gif" />
    = <Property>
        <ComboBox name="Structure" src="MetaData_Content"
        />
    </Property>
</MetaDataFormat>
= <Actor type="DataType">

```

```

        <SubNodes src="Services" />
        <Icon name="cross_ref_motif.gif" />
        <Property />
    </Actor>
= <Actors_Soc type="DataType">
    = <SubNodes>
        <Actor constraint="*" />
    </SubNodes>
    <Icon name="white-ball.gif" />
    <Property />
</Actors_Soc>
= <Managers_Soc type="DataType">
    = <SubNodes>
        <Manager constraint="*" />
    </SubNodes>
    <Icon name="yellow-ball.gif" />
    <Property />
</Managers_Soc>
= <Manager type="DataType">
    <SubNodes src="Services" />
    <Icon />
    <Property />
</Manager>
= <Repository_MGR type="DataType">
    <SubNodes src="Collection" />
    <Icon />
    <Property />
</Repository_MGR>
= <Org_Tool type="DataType">
    = <SubNodes>
        <AuthorityFile constraint="*" />
        <ClassificationSchema constraint="*" />
        <Thesaurus constraint="*" />
        <Ontology constraint="*" />
    </SubNodes>
    <Icon />
    <Property />
</Org_Tool>
= <AuthorityFile type="DataType">
    <SubNodes />
    <Icon />
    = <Property>
        <TextField name="LoadAuthorityFile" load="true" />
    </Property>
</AuthorityFile>
= <ClassificationSchema type="DataType">
    <SubNodes />
    <Icon />
    = <Property>

```

```

        <TextField name="Load_Classif_Schema" load="true" />
    </Property>
</ClassificationSchema>
= <Thesaurus type="DataType">
    <SubNodes />
    <Icon />
    = <Property>
        <TextField name="Load_Thesaurus" load="true" />
    </Property>
</Thesaurus>
= <Ontology type="DataType">
    <SubNodes />
    <Icon />
    = <Property>
        <TextField name="Load_Ontology" load="true" />
    </Property>
</Ontology>
= <UI type="DataType">
    = <SubNodes>
        <Rendering constraint="*" />
    </SubNodes>
    <Icon />
    <Property />
</UI>
= <IR type="DataType">
    = <SubNodes>
        <Index />
    </SubNodes>
    <Icon />
    = <Property>
        <ComboBox name="Retrieval_Space"
            src="RetrievalSpaceContent" />
    </Property>
</IR>
= <Rendering type="DataType">
    <SubNodes />
    <Icon />
    = <Property>
        <ComboBox name="Rendering" src="RenderingContent"
            />
    </Property>
</Rendering>
= <Index type="DataType">
    <SubNodes />
    <Icon />
    = <Property>
        <ComboBox name="Stemming"
            src="StemmingAlgContent" />
        <TextField name="Stopwords" load="true" />
    </Property>

```

```
</Property>  
</Index>  
</DLMetaModel>
```


Appendix C. The 5SL File for NDLTD

The following is the 5SL file for NDLTD. It is the result from the modeling process described in Chapter 6.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
: <DLModel>
:   : <Digital_Library name="NDLTD">
:     : <Stream_Model name="Stream_Model_">
:       : <Text name="XMLText">
:         <content-type>"text/xml"</content-type>
:         <charset>"ISO-8869-1"</charset>
:       </Text>
:       : <Application name="PDF">
:         <content-type>"application/pdf"</content-type>
:       </Application>
:       : <Image name="JPEGImage">
:         <content-type>"image/jpeg"</content-type>
:       </Image>
:       : <Video name="MPEGVideo">
:         <content-type>"video/mpeg"</content-type>
:       </Video>
:       : <Audio name="ETDAudio">
:         <content-type>"audio/x-aiff"</content-type>
:       </Audio>
:     </Stream_Model>
:     : <Struct_Model name="Struct_Model_">
:       : <CollectionSet name="CollectionSet_">
:         : <Collection name="ETDCollection">
:           <Description />
:           <Creator />
:           <Maintainer />
:         : <Document name="ETD">
:           <Structured_Stream />
:           : <Stream name="XMLText">
:             <Stream>"XMLText"</Stream>
:           </Stream>
:           : <Stream name="PDF">
:             <Stream>"PDF"</Stream>
:           </Stream>
:           : <Stream name="JPEGImage">
:             <Stream>"JPEGImage"</Stream>
:           </Stream>
:           : <Stream name="MPEGVideo">
:             <Stream>"MPEGVideo"</Stream>
:           </Stream>
:         </Collection>
:       </CollectionSet>
:     </Struct_Model>
  </Digital_Library>
</DLModel>
```

```

      : <Stream name="ETDAudio">
        <Stream>"ETDAudio"</Stream>
      </Stream>
    </Document>
  </Collection>
</CollectionSet>
: <CatalogSet name="CatalogSet_">
  : <Catalog name="ETDCatalog">
    <Description />
    <Creator />
    <Maintainer />
    <Collection>"ETDCollection"</Collection>
  : <MetaDataFormat name="Dublin_Core">
    <Structure>"Dublin Core"</Structure>
  </MetaDataFormat>
  : <MetaDataFormat name="MARC">
    <Structure>"MARC"</Structure>
  </MetaDataFormat>
  : <MetaDataFormat name="ETDMS">
    <Structure>"IMS"</Structure>
  </MetaDataFormat>
  </Catalog>
</CatalogSet>
</Struct_Model>
: <Scenario_Model name="Scenario_Model_">
  : <ServiceSet name="NDLTD">
    : <Services name="FullText_Search">
      <Load_Services />
    </Services>
    : <Services name="Metadata_Search">
      <Load_Services />
    </Services>
    : <Services name="Browsing">
      <Load_Services />
      : <Scenario name="By Author">
        <Description />
        <Load_Scenario />
      </Scenario>
      : <Scenario name="By Department">
        <Description />
        <Load_Scenario />
      </Scenario>
    </Services>
    : <Services name="Submission">
      <Load_Services />
      : <Scenario name="Cataloguing">
        <Description />
        <Load_Scenario />
      </Scenario>
      : <Scenario name="Review">

```

```

        <Description />
        <Load_Scenario />
    </Scenario>
</Services>
: <Services name="Training">
    <Load_Services />
    : <Scenario name="Giving Workshop">
        <Description />
        <Load_Scenario />
    </Scenario>
    </Services>
</ServiceSet>
</Scenario_Model>
: <Society_Model name="Society Model">
    : <Actors_Soc name="Actors_Soc_">
        : <Actor name="Students">
            <Services_src name="FullText_Search" />
            <Services_src name="Metadata_Search" />
            <Services_src name="Browsing" />
            <Services_src name="Submission" />
            <Services_src name="Training" />
        </Actor>
        : <Actor name="Researchers">
            <Services_src name="FullText_Search" />
            <Services_src name="Metadata_Search" />
            <Services_src name="Browsing" />
        </Actor>
        : <Actor name="NDLTD_Local_Support_Team">
            <Services_src name="Training" />
        </Actor>
        : <Actor name="University Staff">
            <Services_src name="Submission" />
        </Actor>
    </Actors_Soc>
    : <Managers_Soc name="Managers_Soc_">
        : <Manager name="ETD Workflow Mgr">
            <Services_src name="Submission" />
        </Manager>
    </Managers_Soc>
</Society_Model>
</Digital_Library>
</DLModel>

```

Appendix D. Documents for the Experiment

This appendix includes the documents that were used in the test of 5SGraph.

D.1 Informed Consent

VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY

Informed Consent for Participants

in Research Projects Involving Human Subjects

Title of Project_____5SGraph: A Modeling Tool for Digital Libraries _____

Investigator(s)_____Qinwei Zhu_____

I. Purpose of this Research/Project

To test the digital library modeling tool: 5SGraph.

II. Procedures

Introduction to 5S model, modeling methodology.

Tutorial about the software.

Three task:

- Create a simple model using existing components
- Complete a partial model
- Design a complete model from scratch

III. Risks

No known risks.

IV. Benefits

Subjects will have better understanding of the digital library modeling methodology.

Students in Course 5604 will get 2 points of bonus.

V. Extent of Anonymity and Confidentiality

All data will be kept anonymous.

VI. Freedom to Withdraw

Subjects are free to withdraw from a study at any time without penalty.

VIII. Approval of Research

This research project has been approved, as required, by the Institutional Review Board for Research Involving Human Subjects at Virginia Polytechnic Institute and State University, by the Department of Computer Science

IRB Approval Date Approval Expiration Date 2003

IX. Subject's Responsibilities

I voluntarily agree to participate in this study. I have the following responsibilities:

- Complete all three tasks and give the saved files to the experimenters.

X. Subject's Permission

I have read and understand the Informed Consent and conditions of this project. I have had all my questions answered. I hereby acknowledge the above and give my voluntary consent:

_____ Date _____

Subject signature

_____ Date _____

Witness (Optional except for certain classes of subjects)

Should I have any questions about this research or its conduct, I may contact:

Investigator(s) Telephone/e-mail

Qinwei Zhu qzhu@vt.edu

Faculty Advisor Telephone/e-mail

Dr. Fox fox@vt.edu

Departmental Reviewer/Department Head Telephone/e-mail

David M. Moore 231-4991/moored@vt.edu

Chair, IRB Telephone/e-mail

Office of Research Compliance

Research & Graduate Studies

D.2 Task Descriptions

We include the descriptions of the three tasks as follows.

D.2.1 Task One

Task one is to build a simple model for a digital library of departmental technical reports using pre-defined, fixed sub-models.

In this task, you will model a simple digital library to illustrate the concepts and the underlying 5S methodology of the 5SGraphic tool. The focus of this task is on showing how a digital library model can be built from previously defined, fixed sub-models.

You will model a departmental digital library based on a collection of *technical reports* (documents). There are only two types of technical reports: in LaTeX format for text with EPS figures, and in HTML for text with JPEG figures. There is no standard structural metadata for technical reports. The catalog has descriptive metadata only in Dublin Core format. There are only three services: submission of technical reports, searching, and browsing. This last one includes two scenarios, involving browsing by author and by year. Six societies participate in this DL. There are three managers: submission workflow, search manager, and browse manager. There are three actors: administrator, researcher, and professor.

Sub-tasks to create Technical Report DL with fixed sub-models:

- 1) Create a Digital Library Node from the Metamodel
 - a. Change name to “TechReportDL”
- 2) From the TechReportDL node, load SubNode Stream Model (TechRepStreamM.xml). Four streams will be loaded:
 - a. Text
 - i. Latex
 - ii. HTML
 - b. Picture
 - i. EPS
 - ii. JPEG
- 3) Add Structural Model

- 3.1. Add a CollectionSet
- 3.2 Add a Collection
 - 3.2.1 Change name of the Collection to “TechRepCollection”
- 3.3 From the “TechRepCollection” node, load sub-nodes for documents:
 - a. Type1 (TechRepDocType1.xml)
 - b. Type2 (TechRepDocType2.xml)
- 4) Add a CatalogSet
 - 4.1 Add a Catalog
 - 4.2 Change Name to “TechRepCatalog”
 - 4.3 From the “TechRepCatalog” node, load the sub-node for the Dublin Core metadata format (TechRepDublinMD.xml)
- 5) Add a Scenario Model
 - 5.1 Add a ServicesSet
 - 5.2 From the Services set node, load three nodes for Services:
 - a. Submission (TechRepServSubmission.xml)
 - b. Searching (TechRepServSearching.xml)
 - c. Browsing (TechRepServBrowsing.xml). This service has two scenarios
 - i. scenario1: by year
 - ii. scenario2: by author
- 6) Add a Societies Model

From the Societies Model node, load the Manager_Soc (Manager Societies) (TechRepManagerSoc.xml). Three managers will be loaded:

 - a. SubmissionWorkflowMGR
 - b. SearchMGR
 - c. BrowseMGR
- 7) In the Societies Model, add a Actor_Soc

From the Actor_Soc node, load sub-nodes for

 - a. Administrator (TechRepActorAdmin.xml)
 - b. Researcher (TechRepActorResarcher.xml)
 - c. Professor (TechRepActorProf.xml)
- 8) SAVE YOUR MODEL (Filename: “task1_” + your participant number + “.xml”; e.g., “task1_12.xml”)
- 9) Take a couple of minutes to re-read the textual description of the simple DL, browse the correspondent model with the tool and compare both, so that you get familiar with the underlying 5S methodology for the next tasks.
- 10) When you are done, call the instructor to prepare the tool for the next task.

D.2.2 Task Two

Task two is to Capture requirements and build a 5S model for the CITIDEL digital library using the 5Sgraphic Tool by completing a partial model.

The Computing and Information Technology Interactive Digital Educational Library (i.e., the CITIDEL Digital Library) will establish, operate, and maintain a part of the National Science Digital Library (NSDL) that will serve the computing education community in all its diversity and at all levels.

CITIDEL is part of the collections track of NSDL, therefore the main resource offered to its society of users is a union catalog that will include metadata for an aggregate virtual collection which includes documents (or resources) from a number of third-party collections, including CSTC, NCSTRL, JERIC, Computer History, ACM DL, and DBLP. There will be two major types of descriptive metadata formats supported by the union archive: basic Dublin Core, and the learning object metadata standard (IEEE LOMS, equivalent to the standard developed by IMS). CITIDEL also will support other organizational tools including citation networks and multiple classification schemes including the *ACM classification system*.

Basic information-seeking services provided by CITIDEL include: *metadata-based structured searching*, supported by the MARIAN system, *multi-classification browsing*, and *profile-based filtering*. The targeted societies for those services include teachers, learners, and researchers. Another type of service is *cataloging* in which registered users can provide metadata for external resources to be included in the CITIDEL union catalog.

CITIDEL also supports services for building collections of specific types of documents or resources (i.e., digital objects). One example involves resources collected from the web using a *focused crawler* service supported by an electronic agent called *crawlifier*.

Another example is a collection of *lesson plans*. A lesson plan is a type of document, which aggregates metadata about resources in CITIDEL and uses a specific structural metadata to organize and describe the set. A specific service manager called *VIADUCT* supports this service, which can be used only by teachers. In a typical scenario for building a lesson plan, the teacher uses the information-seeking services of CITIDEL to look for resources related to the specific lesson, assembles a number of them together using a *binder* manager, chosen using any subjective criteria (e.g., by relevance, by date), and associates descriptive metadata such as typical DC-based ones like author, identifier, language – as well as specific ones such as topic area, target audience, and time required for the whole lesson plan object. The newly created lesson plans are not publicly accessible. So, in another scenario of this service the teacher has to explicitly publish the lesson plan to allow everyone to view the lesson plan. To allow a select group of people to view the lesson plan, the teacher saves the plan, returns to the main VIADUCT user information page, re-opens the project, and gives the project URL to whomever she wishes. Finally, another service supported by VIADUCT is the *customized browsing of*

lesson plans. Four typical scenarios of use are supported: browsing ordered lists of resources; browsing unordered lists of resources; guided path browsing; and slide show browsing.

Specific tasks:

1. Load our partial model from file “CITIDELPartial.xml”.
2. In the Structural Model:
 - 2.1 Create a new collection named “CrawledCollection”, and create a document “CrawledDocument” in that collection. Then, add three stream nodes after the document node. Stream one is of type Webpage and named as Webpage as well. Stream two is of type postscript and named as PS. Stream three is of type PDF and named as PDF.
 - 2.2 In the Union Catalog, add to metadata formats: “DC” with structure Dublin Core, and “learningObjects” with structure “IEEE LTSC-LOM”.
 - 2.3 In the Organizational tool (node name “Org_Tool_”), add a classification scheme for the “ACM classification system”.
3. In the Space Model:
 - a. Change the rendering of the user interface (UI) from Java to HTML.
 - b. Change the properties of the node “Index” (which is under node “IR_”) by:
 - i. changing the stemming algorithm to LIMA
 - ii. in the stopwords property of the Index, loading a stopwords file called “CITIDELstopwords.txt”.
4. In the Scenario Model:
 - a. create a new Service named “lesson_plan_building” with two scenarios: “gathering_resources” and “publishing_lesson_plan”.
5. In the Societies Model connect the existent societies which the services either use or manage:
 - a. For actors:
 - i. Teachers use metadata search, multiSchemeBrowsing, profile_filtering, cataloguing, lesson_plan_building and LP_customized browsing.
 - ii. Learners use metadata search, multiSchemeBrowsing, profile_filtering, cataloguing, and LP_customized browsing.
 - iii. Researchers use metadata search, multiSchemeBrowsing, profile_filtering, and cataloguing.
 - b. For managers
 - i. VIADUCT manages lesson_plan_building and LP_customized browsing.
 - ii. MARIAN manages metadata search.
 - iii. Crawlifier manages focused_crawling.

5. SAVE YOUR MODEL (Filename: “task2_” + your participant number + “.xml”; e.g., “task2_12.xml”)
6. Before you perform the next task, take some time to browse the resulting model and compare it with the textual requirements so that you can better comprehend the tool and 5S methodology.
7. When you are done call the instructor, for preparing the tool for the next task.

D.2.3 Task Three

Task three is to Capture requirements and build a 5S model for an NDLTD digital library using the 5Sgraphic Tool from scratch.

The Networked Digital Library of Theses and Dissertations (NDLTD) seeks to change the future of scholarship by ensuring that the future leaders of research, in particular, those who complete a thesis or dissertation, have the requisite knowledge and skills to utilize and take advantage of electronic publishing and digital library (DL) technologies.

A NDLTD digital library involves a local collection of electronic theses and dissertations (ETDs) which students produce as a result of their graduate studies. One of the main objectives of NDLTD is to improve students’ skills as effective communicators in the digital age. Therefore we have focused on promoting student’s creativity through the use of diverse types of multimedia content in ETDs, while making students comfortable with the utilization of this technology to exploit richer modes of self-expression. Because of preservation and interoperability purposes NDLTD encourages students to use standard, non-commercial multimedia formats such as XML for text and standards like PDF for texts/images, MPEG for video, and AIFF for audio.

In order to allow students to understand issues of electronic publishing, the NDLTD-DL requires them to submit their own work to the local repository of ETDs, along with corresponding descriptive metadata (e.g., author, abstract, department, etc.). NDLTD has developed and is promoting the Interoperability Metadata Standard for Electronic Theses and Dissertations (ETD-MS) as a standard descriptive metadata set for describing electronic theses and dissertations, which can be converted to MARC and Dublin Core for distribution purposes.

A submission service is controlled by an ETD workflow manager, and includes a cataloging scenario and a review scenario. While cataloging, the ETD workflow manager helps students enter and edit the descriptive metadata about their ETDs. In the review phase, the university staff checks ETD files, the metadata submitted by the

student, and payment of appropriate fees. If everything is OK, the ETD is approved and archived; if not a message about the corresponding problem is sent to the student

The NDLTD local team should be focused on providing training services (through workshops, online materials, and help in media centers or library sites) to assist students with the authoring or creation of ETDs. One scenario includes giving training workshops to students.

Ideally, the DL also should offer information-seeking services for the local collection to the university patrons (students, faculty). Those may include fulltext and metadata-based searching as well as browsing by author and department.

Specific Tasks:

0. Add a digital library, name it as NDLTD. This is the root of your tree. From this root, do the following:
 1. Add a Stream Model with:
 - a. Text: name: "XMLText; content-type: text/XML
 - b. Application: name: PDF content-type: application/pdf.
 - c. Image: name: JPEGImage. content-type: image/jpeg
 - d. Video: name: MPEGVideo. content-type: video/mpeg
 - e. Audio: name "ETDAudio", content-type:audio/x-aiff.
 2. Add a Structural Model with:
 - a. In CollectionSet: Create one collection ("ETDCollection") with one type of document ("ETD"). The ETD document has a Structure_Stream property defined in an XML Schema file which should be loaded to the property editor of the document node from the file "etd.xsd".
 - b. Add 5 stream nodes to the ETD document. Each node is supposed to represent a stream that can occur inside an ETD, and is defined by the stream model which was built in part 1. You should specify different stream properties for all different nodes here. For example, the first node should have the name "XMLText" and the stream "XMLText" selected in the comboBox. The second node should be named as "PDF" and has the stream "PDF" selected in the comboBox, and so on.

- c. In CatalogSet: Create one catalog (“ETD_Catalog”) with three types of metadata format: Dublin Core, MARC, and ETDMS. Name them as “DC”, “MARC”, “ETDMS” respectively.
3. Add a Scenario Model and add the following NDLTD services:
 - a. Fulltext_search
 - b. Metadata_search
 - c. Browsing with two scenarios
 - i. By_author
 - ii. By_department
 - d. Submission with two scenarios
 - i. Cataloguing
 - ii. Review
 - e. Training with one scenario
 - i. Giving_workshop
4. Add a Society Model with the following societies:
 - a. In the Actor Societies (Actors_Soc) four Actors:
 - i. Students
 - ii. Researchers
 - iii. NDLTD_Local_Team
 - iv. University_staff
 - b. In the Managers Society, one manager:
 - i. ETD_Workflow_Mgr
5. Connect the societies created in the Society Model with the services that you created in the Scenario Model and in which each Society given below participates:
 - a. For Actors:
 - i. Student *use* all the services.
 - ii. Researchers use information-seeking services (fulltext_search, metadata_search, and browsing).
 - iii. NDLTD_Local_Team is associated with training.

- iv. University staff is associated with submission.
 - b. For managers:
 - i. ETD_Workflow_manager runs the submission service.
6. SAVE YOUR MODEL (Filename: “task3_” + your participant number + “.xml”; e.g., “task3_12.xml”)
7. Before you perform the next task, take some time to browse the resulting model and compare it with the textual requirements.
8. When you are done call the instructor. He or she will give a questionnaire for final evaluation of the tool.

D.3 Questionnaire

Participant # _____

Date: ____/____/2002

Closing Questionnaire.

Please answer the following questions with regard to this part of your experience.

1. How do you rate your understanding of the 5S model **before** starting this experiment?

Low 1 2 3 4 5 6 7 8 9 10 High

2. How do you rate your understanding of the 5S model **after** using the tool?

Low 1 2 3 4 5 6 7 8 9 10 High

3. How comfortable did you feel modeling a digital library using the 5S model?

Low 1 2 3 4 5 6 7 8 9 10 High

4. How comfortable did you feel using the software?

Low 1 2 3 4 5 6 7 8 9 10 High

5. Is the software helpful for you to create a 5SL file for your model?

Not helpful 1 2 3 4 5 6 7 8 9 10 Very helpful

6. You have finished three kinds of tasks. Which one do you think is the most convenient one? Rate them using (1, 2, 3) where 1 is the most convenient or choose No preference (with an X).

<input type="checkbox"/> Reuse sub-models	<input type="checkbox"/> Complete a model
<input type="checkbox"/> Build from Scratch	<input type="checkbox"/> No preference

5. At this point, please write any other comments (if you have any) in the space below or discuss with the experimenter your experience using the system. Thank you for your participation.

Vita

Qinwei Zhu was born in Jiangsu, China in September, 1975. She received her Bachelor of Engineering in Applied Chemistry and Bachelor of Engineering in Computer Science from Shanghai Jiao Tong University in July 1996. She received her Master of Engineering in Applied Chemistry from Shanghai Jiao Tong University in March 1999. She will be graduating with a Master of Science degree in Computer Science in December, 2002.