Rapid Modeling, Prototyping, and Generation of Digital Libraries – A Theory-Based Approach

Marcos André Gonçalves, Qinwei Zhu, Rohit Kelapure, Edward A. Fox
Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, VA, 24061, USA
{mgoncalv, qzhu, rkelapur, fox}@vt.edu

ABSTRACT

Despite some development in the area of DL architectures and systems, there is still little support for the complete life cycle of DL development, including requirements gathering, conceptual modeling, rapid prototyping, and code generation and reuse. Even when partially supported, those activities are uncorrelated within the current systems, which can lead to inconsistencies and incompleteness. Moreover, the current few existing approaches are not supported by comprehensive and formal foundations and theories, which brings problems of interoperability and makes it extremely difficult to adapt and tailor systems to specific societal preferences and needs of the target community. In this paper, having the 5S formal theoretical framework as support, we present an architecture and a family of tools that allow rapid modeling, prototyping, and generation of digital libraries. 5S stands for Streams, Structures, Spaces, Scenarios, and Societies and is our formal theory for DLs. 5SL is a domain-specific, declarative language for DL conceptual modeling. 5SGraph is a visual modeling tool that helps designers to model a digital library without knowing the theoretical foundations and the syntactical details of 5SL. Furthermore, 5SGraph maintains semantic constraints specified by a 5S metamodel and enforces these constraints over the instance model to ensure semantic consistency and correctness. 5SGraph also enables component reuse to reduce the time and efforts of designers. 5SLGen is a DL generation tool that takes specifications in 5SL and a set of component pools and generates portions of a running DL system. The outputs of 5SLGen include user interface prototypes, in a generic UI markup language, for validation of services behavior and workflow representations of the running system, generated to support the desired scenarios.

Categories and Subject Descriptors

H.3.7 [Information Systems]: Information Storage and Retrieval – Digital Libraries; D.2.2 [Software Engineering]: Design Tools and Techniques.

General Terms: theory, languages, modeling, user interfaces, and prototyping.

Keywords: 5S, 5SL, 5SLGen, 5SGraph, etc.

1. INTRODUCTION

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, most 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 modern networked world. However, while much attention has been paid to the study of how to make a better digital library, very little work has focused on simplifying the process of building DLs**.**

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 [9]. Until now, none of these questions has been answered perfectly. Much research needs to be done. Accordingly, it is difficult and time-consuming to build a new DL application right now.

Yet, 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. Nevertheless, many existing digital libraries are monolithic, tightly integrated internally, inflexible, and lack interoperability connections with each other. It usually takes a huge amount of effort and time to create or customize a digital library to satisfy specific needs and requirements. Furthermore, designers of digital libraries often are not experts in digital libraries; rather they may be new to that field. 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.

Despite some development in the area of DL architectures and systems, there is still little support for the complete life cycle of DL development, including requirements gathering, conceptual modeling, rapid prototyping, and code generation and reuse. Even when partially supported, those activities are uncorrelated within the current systems, which can lead to inconsistencies and incompleteness. Moreover, the current few existing systematic development approaches are not supported by comprehensive and formal foundations and theories, which brings problems of interoperability and make extremely difficult to adapt and tailor systems to specific societal preferences and needs of the target community.

In this paper, based on the 5S formal theoretical framework, we present an architecture and a family of tools that allow rapid modeling, prototyping, and generation of digital libraries. 5S stands for Streams, Structures, Spaces, Scenarios and Societies and is our formal theory for DLs. 5SL is a domain-specific, declarative language for DL conceptual modeling. 5SGraph is a visual modeling tool that helps designers to model a digital library without knowing the theoretical foundations and the syntactical details of 5SL. Furthermore, 5SGraph maintains semantic constraints specified by the 5S metamodel and enforces these constraints over the instance model (of the desired DL) to ensure semantic consistency and correctness. 5SGraph also enables component reuse to reduce the time and efforts of designers. 5SLGen is a DL generation tool that takes specifications in 5SL and a set of component pools and generates portions of a running DL system. The outputs of 5SLGen include user interface prototypes, in a generic UI markup language, for validation of services behavior and workflow representations of the running system, generated to support desired scenarios.

This paper is organized as follows. Section 2 presents an overview of the proposed architecture. Section 3 gives a brief introduction to our 5S theory and to the 5SL language. Section 4 focuses on our new 5SGraph modeling tool, its design, features, visualization properties and evaluation. Afterwards, the 5SGen tool, now including the aspects of scenario synthesis and user interface prototyping, is presented in Sections 5. Section 6 covers related work. Conclusions and future work are presented in Section 7.

2. OVERVIEW OF THE ARCHITECTURE

Our objective is to cover the whole process of DL development, from requirements to analysis, analysis to design, design to implementation -- in order to generate a final "tailored" DL software product which will satisfy the particular requirements of specific DL societies. The first step is to develop models, languages, and tools able to capture the rich set of DL requirements and properties of particular settings. Then we automatically convert these "patterns" into different representations by properly "compiling", transforming and mapping models in different levels and phases of the digital library development process. The assumption is that automatic transformations and mappings diminish the risk of inconsistencies and increase productivity. This view will be supported by: a) having 5S as a general and formal underlying framework; and b) incorporating and combining state-of-the-art research in Software Engineering into the process, including domain-specific languages [2], scenario synthesis [12] and componentized architectures [13]. Figure 1 illustrates the general architecture proposed in this work.

High-level DL conceptual abstractions and their properties are described in a *metamodel*, based on the 5S theory. A digital library expert creates a metamodel for digital libraries and feeds the metamodel to the 5SGraph modeling tool. The modeling tool processes the metamodel, allowing the digital librarian (or the DL designer) to visualize the components of the metamodel. The visualization of the metamodel helps the designer understand the structure of a generic digital library and reduces the learning time. The digital librarian interacts with the 5SGraph modeling tool to describe his own digital library model, based on the specific requirements and needs of the societies to which the DL is targeted. The designer uses suitable parts of the metamodel to put together the final model of his own digital library. The DL requirements acquired with the graphical tool are then formally captured using a domain-specific language - 5SL. Moreover, the graphical tool is able to enforce certain semantic constraints among the different sub-models of 5SL, therefore guaranteeing the correctness and consistency of the final model.

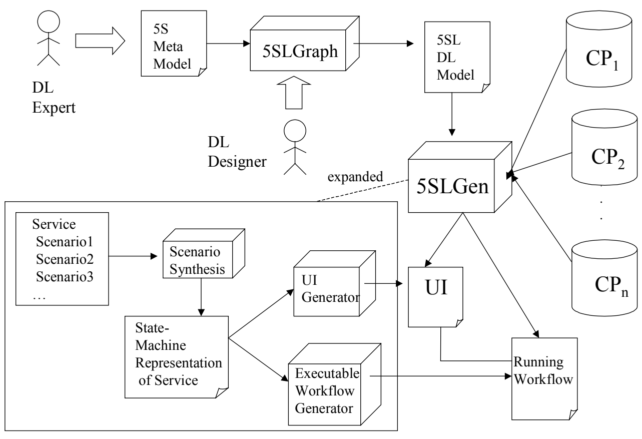


Figure 1. Overview of the architecture for DL modeling and generation with partial expansion of 5SLGen to show the services generation process (Keys: CP= component pools; UI= user interface).

The produced 5SL models are then fed to the 5SLGen digital library generator. The generator, armed with a powerful pool of DL components, generates a running version of the digital library, which can take the form of running workflows with customized components/classes that reuse the capabilities provided in the pool. Since in 5SL scenarios represent partial description of system behavior, our approach for scenarios composition or scenarios integration produces complete specifications of DL services. The generator also produces prototype user interfaces, which help the designer to validate the behavior of the generated DL services with prospective patrons. Ultimately, refined versions of those interfaces should be linked together with the running workflow of the system. In the following, we describe in detail each of the components of our solution architecture.

3. THE 5S THEORY AND THE 5SL DECLARATIVE LANGUAGE

Recognizing the difficulties in understanding, defining, describing, and modeling digital libraries, Gonçalves, Fox, et al. have proposed and formalized the 5S (Streams, Structures, Spaces, Scenarios, and Societies) theory of digital libraries [11]. 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. This enables automatic mapping from 5S models to actual implementations as well as the study of qualitative properties of these models (e.g., completeness, consistency, etc).

Gonçalves and Fox also proposed 5SL, a language for declarative specification and generation of digital libraries [Gonçalves02]. It is a high-level, domain-specific language, which: 1) raises the level of abstraction in digital library specification and modeling by offering specific abstractions for the domain at hand; and 2) shows how the several DL design issues may be combined in a coherent framework that enriches, extends, and customizes classical models for databases, information retrieval and hypertext. 5SL is an XML realization of the 5S model with specific considerations on interoperability and reuse in its design. Table 1 summarizes, for each of the 'S' models: its formal definition, the objective of the model within 5SL, and resources and sub-languages used in the 5SL implementation.

Model	Formal definition	Objective within 5SL	5SL Implementation
Streams	Sequences of arbitrary types	Describe properties of the DL content such as encoding and language for textual material or particular forms of multimedia data	MIME types
Structures	Labeled directed graphs	Specify organizational aspects of the DL (e.g., structural and descriptive metadata, hypertexts, taxonomies, classification schemes)	XML and RDF schemas; Topic maps ML (XTM)
Spaces	Sets of objects and operations on those objects that obey specific constraints[1]	Define logical and presentational views of several components.	MathML, UIML, XSL
Scenarios	Sequences of events that modify states of a computation in order to accomplish some functional requirement.	Detail the behavior of the DL services	UML sequence diagrams; XML serialization
Societies	Sets of communities and relationships (relations) among them	Define managers, responsible for running DL services, actors, that use those services, and relationships among them	UML adapted class diagrams; XML serialization

Table1. 5S/5SL overview.

4. THE 5SGRAPH MODELING TOOL

With 5SL, 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 describes 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. Thus, the 5SL specification is compatible and extensible, because many existing standard formats can be used within the 5SL language. However, it is frustrating 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 trouble for maintenance, upgrade, or even understanding of an existing system.

5. A number of semantic constraints exist between (inter-model constraints) and within (intra-model constraints) the sub-models. Designers need extra effort to ensure consistency in the whole model.

Reflecting on the above analysis of the disadvantages of 5SL, we consider the following four functions of a modeling tool based on the 5S/5SL framework to be essential:

- To help digital library designers understand the 5S model quickly and easily.
- To help digital library designers build their own digital libraries without difficulty.
- To help digital library designers transform their models into complete, correct, and consistent 5SL files automatically.
- To help digital library designers understand, maintain, and upgrade existing digital library models conveniently.

Accordingly, our 5SGraph modeling tool provides an easy-to-use graphical interface and automatically generates desired 5SL files for the designer. Here, we adopt the idea that *visualization* helps people understand complex models. 5SGraph is able to load and graphically display digital library metamodels. The visual model shows the structure and different concepts of a digital library and the relationship among these concepts. 5SGraph also provides a structured toolbox to let the designer build a digital library by manipulation and composition of visual components (see Figure

[1] The combination of operations on objects with the set of objects is what distinguishes spaces from streams and structures.

2). The structured toolbox introduced into our tool not only provides all the visual components of the metamodel, but also shows the structural relationships among these components. The visualization of the model thus provides guidance while the designer is building his model. 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. Furthermore, correctness and consistency can be automatically guaranteed by 5SGraph; thus producing correct and consistent 5SL XML files according to the visual model built by the designer. As such, 5SGraph eliminates the disadvantages of 5SL.

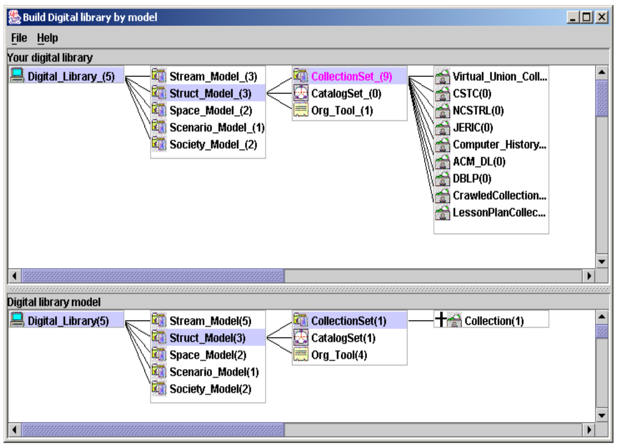


Figure 2. 5SGraph sample interface with structured toolbox (bottom part) and workspace upper part); figure shows modeling of collections for the CITIDEL digital library.

The concept of metamodel is very important here. The role of the digital library expert who builds the metamodel brings in flexibility. This metamodel describes a generic digital library. The model for a specific digital library is an instance of the metamodel, which in our case is a domain-specific metamodel, i.e., specific to the domain of building digital libraries.

The 5S framework is still under development. It is expected that more changes and additions will be made in the future, specially to 5SL. Therefore, being given a new metamodel, the tool can be used with future versions of the 5S model as well.

Some of the major features of the tool include:

1. Flexible and extensible architecture

5SGraph is domain-specific modeling tool. Thus, 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, for 5S. The methods that are appropriate only to 5S 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 5SL language extensively uses existing standards. The reason is that the specification of a digital library involves many sub-domains, and there are many standard specifications for each sub-domain. There also are many well-developed tools for those sub-domains. For example, metadata is an important element in 5S. Several existing metadata editors can be used to view and edit metadata. Another example is in the scenario part of 5S. A specific scenario can be modeled and described by UML sequence diagrams. Existing UML modeling tools can be used for this purpose [15].

The 5SGraph tool should not "re-invent the wheel". 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 as needed. In the interest of brevity, this paper focuses on how 5SGraph helps with modeling a digital library, not on how 5SGraph calls other tools to create customized components.

2. Reuse of components and sub-models

In 5SGraph, component reusability means that components designed in one user model can be saved and reused in 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 standards. The components representing the Stream Model or the metadata in the Structural Model 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 customize the loaded component (see Fig 3.).

Of course, not all components are designed to be reusable. A reusable component should be self-contained and independent of any other specific models.

3. Synchronization between the model and the metamodel

There are two views in the tool. One is for the toolbox (metamodel); the other 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 selected instance component. The convenience of synchronization is that: 1) The user does not need to manually search all the components in the toolbox to find the correct type component; and 2) 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.

4. Enforcing of semantic constraints

Certain inherent semantic constraints exist in the hierarchical structure of the 5S model. The semantic constraints in 5S are divided into two categories. The *value constraint* specifies the range of possible values of an element, while the *association constraint* defines the relationships among different components. Examples of such constraints include:

- The streams used 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 digital objects in the collection. A catalog, therefore, is dependent on a collection.
- The services that the actor (a member of the Society Model) uses or a manager (another member of the Society Model) runs can be only the services already defined in the Scenario Model.

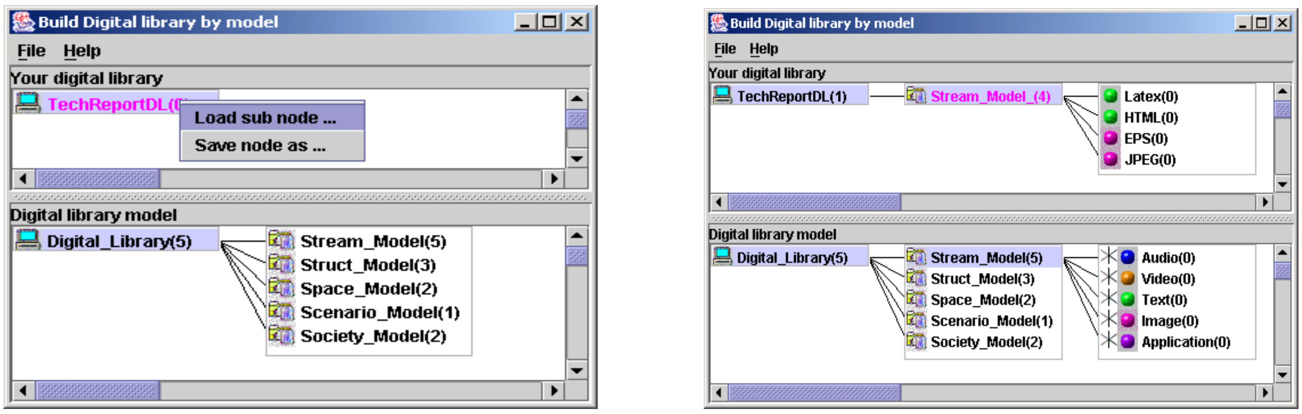
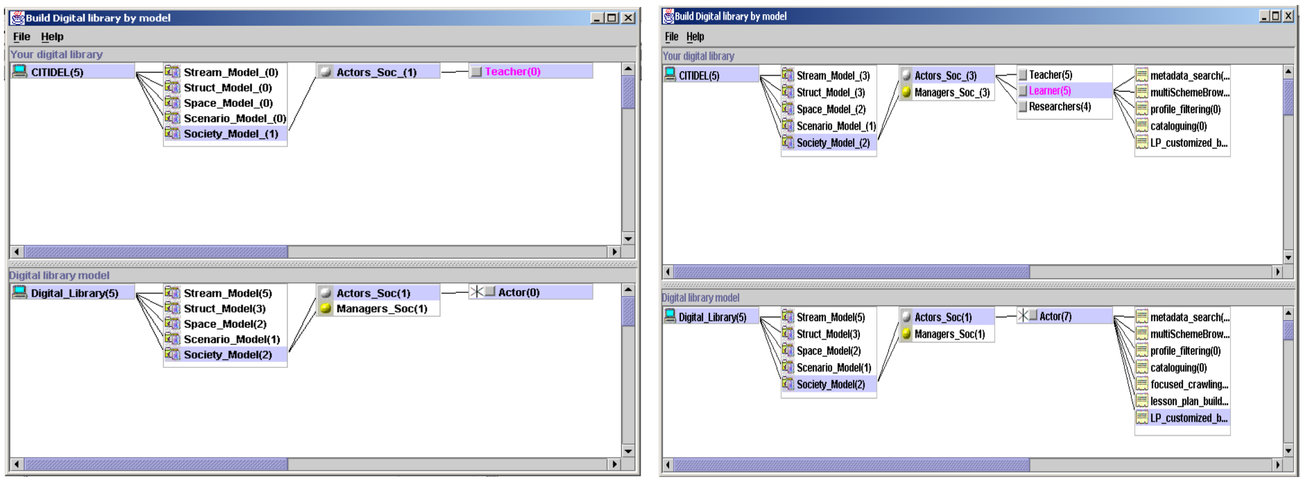


Figure 3. Reuse of models; before and after loading.

The 5SGraph tool is able to implement and manage these constraints. For example, an *actor* can only use services that have been defined in the *Scenario Model*. This is specified in the metamodel, where the *SubNodes* part of *actor* contains the *Datatype* '*Services'* (not shown in the figure), which means only existing instances of *Services* can become child nodes of an *actor*. The declaration of an actor, *Teacher*, is shown in Figure 4(a). In order to associate actors with the services they use, the designer browses back to the *Scenario Model* to create services: metadata search, multi-scheme browsing, profile filtering, browsing, cataloguing, focused crawling, lesson plan building, lesson plan customized browsing (this one with four scenarios: unordered and ordered browsing, guided path and slide show as supported by VIADUCT manager). When the designer browses back to *Actor* in the Scenario Model in the metamodel, she finds out that the created set of services are automatically added into the metamodel under the node "*Actor*" (Fig 4(b), structured toolbox), allowing the designer to connect the defined services with the actors that use them. In the example, *Learner* is connected to all but two services (focused crawling, run by the crawlifier manager, and lesson plan building, used only by teachers).



(a) Fig 4. Enforcing of semantic constraints in the CITIDEL digital library (b)

4.1 Evaluation

We conducted a pilot usability test to examine the performance of 5SGraph. The questions to be answered were: 1) Is the tool effective in helping users build digital library models based on the 5S theory? 2) Does the tool help users efficiently describe digital library models in the 5S language? 3) Are users satisfied with the tool? 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 the target users of the tool. Three representative tasks with different levels of difficulty were 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 5S 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.

Task 3: build a model of NDLTD (Networked Digital Library of Theses and Dissertations) from scratch. The difficulty level of this task is high.

The procedures were as follows: 1) the participant was asked to read some background documents about 5S and the modeling methodology; 2) the participant was given an introductory presentation on 5SGraph; 3) we gave the participant a description of task 1 and recorded how he/she completed it; 4) after the participant finished each task, he/she was given the next task description immediately; 5) after the participant finished all the tasks, he/she was given a questionnaire form to fill out.

We use the following 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[2] divided by task time.

- *Satisfaction*

Satisfaction is measured using a subjective rating scale. After each participant finishes all three tasks, he/she is given a questionnaire and asked to rate the overall learnability, effectiveness, and satisfaction based on his/her observation. 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 5S before using the tool. Post-Understanding refers to the participant's understanding of 5S after using the tool.

The summary of the results of all three tasks is given in Table 2.

	Task1	Task2	Task3
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 2. Overall Performance Results for Three Tasks

[2] 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.

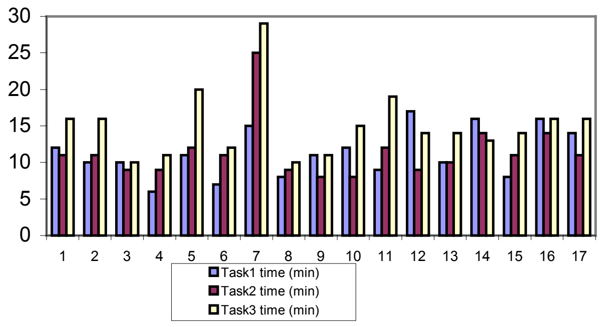


Figure 5. Task Time

- ***Effectiveness*** The high completion rate and the high goal achievement rate prove the effectiveness of 5SGraph (see Table 2).

- ***Efficiency*** Most participants finish tasks in less than 20 minutes (see Fig. 5) 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 (see Table 2, Fig.6). There are three observations, which have been confirmed by using statistics (t test with $\alpha = 0.05$).

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

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

Observation 3: the mean *Closeness to Expertise* in task 3 is not significantly different from that in task 2.

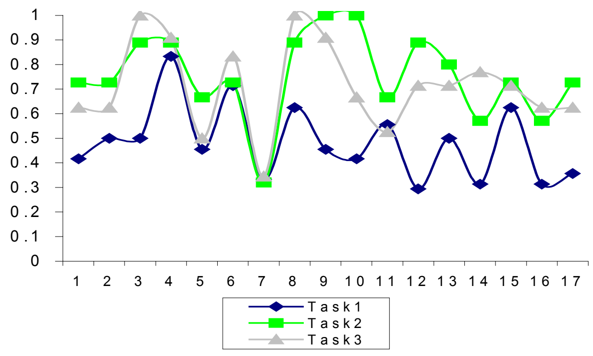


Figure 6. Closeness to Expertise

Observations 1 and 2 suggest 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 quite close 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. Observation 3 indicates 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., familiarity with the tasks, typing speed, reading speed, and skills in using computers.

Satisfaction The average rating of user satisfaction is 9.1 and the average rating of usefulness of the tool is 9.2. 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 5S. 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 7. 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.

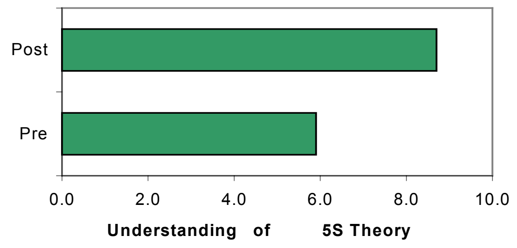


Figure 7. Pre and Post understand of 5S.

5. THE 5SLGEN DL GENERATOR

5.1 5SLGen for MARIAN

The first implementation of 5SLGen produces a running DL using the Java MARIAN digital library software as the main component pool. MARIAN is a digital library system designed and built to store, search, retrieve, and browse large numbers of diverse objects in a network of relationships [10]. MARIAN is built upon three basic principles: 1) unified representation based on *semantic networks*, which model internal structures of digital objects and metadata and different types of relationships among objects and concepts (e.g., as in thesauri, classification hierarchies or among word terms and structural portions of documents); 2) *weighting schemes* to support information retrieval services, including weighted nodes and links, and weighted objects sets wherein a set of objects whose relationship to some external proposition is encoded in their decreasing weight within the set; and 3) an *object-oriented class system*, which is used to organized nodes and links into hierarchies of object-oriented classes, with methods to store and maintain instance objects of their class, translate back and forth between object IDs and fully realized objects, and support matching and retrieval operations.

The MARIAN API is made of sets of reusable Java packages and hierarchies of classes. The main API implements basic functionality to create, manage, and match portions of semantic networks. Classes to support multi-lingual retrieval also are implemented. Besides the main API a set of supporting APIs also are implemented in MARIAN. There are APIs for database management, manipulation of weights and weighted sets used in the matching algorithms, and generalized document presentation. In particular, the latter include methods to present short versions of documents in ranked lists with links to different views of full document presentation through XSL stylesheets.

The 5SLGen for MARIAN generator is based on a DOM XML parser. 5SLGen parses 5SL specifications, extracts the required information and generates the corresponding customized DL infrastructure that completely supports a new, tailored digital library within the MARIAN system. This infrastructure includes the classes of objects and relationships that make up the DL, processing tools to create the actual library collection from raw documents, as well as services for searching, browsing, and collection maintenance. More specifically, the 5SLGen for MARIAN outputs include:

1. Class managers and indexing classes

Includes class managers to represent the MARIAN semantic network view of the documents/metadata as well as indexing classes, which represent sets of semantic bipartite weighted networks between document/metadata parts and their terms. Multilingual free text and controlled vocabulary types also can be specified.

2. ClassIDs and Tailored Database Tables.

Every object in Java MARIAN, whether an XML element, a MARC record attribute, a chunk of text, a unique term, etc., has a FullID, which is made of a class ID and an instance ID. ClassIDs for terms and controlled texts are normally pre-defined in some lexicon or authority file. The ClassIDs for collection dependent class managers and link managers are generated from the 5SL information. Also during generation of class managers, customized databases tables to actually store the semantic view of the library are created.

3. The Collection Loader and Document Handlers

The Loader class is responsible for taking a stream of documents, loading them into the corresponding databases, and creating all the indexing information. The loader receives a stream of incoming data and separates it into individual XML documents. Documents are checked against the Schema specifications and, if valid, the loading and inversion processes proceed. The inversion process is driven by structure, i.e., weighted links are created among terms and specific structural parts of documents/metadata and global statistics (e.g., inverse document frequency) are calculated in the context of those structural parts.

4. User interfaces

Three different types of user interfaces are generated by 5SLGen:

a. An HTML web query form where searchable fields in pull-down menus representing a flat view of all structural parts of documents, as defined by the XML Schema structure. The interface also allows the user to choose the collections to be searched and other personal preferences such as number of results to be returned and query time out. Comment buttons are linked to the email of the collection manager.

b. A customized Document Java class, which implements methods to present a short version of documents, as in the ranked list result set returned for a query, to create links to the full version presentation of documents, and to generate the full presentation itself.

c. Some specific XSL stylesheets, used by the Document Java class to actually transform XML documents to the kind of output that MARIAN expects. The transformation is guided based on the document structure itself and some of the MARIAN presentation styles, such as background colors, types of buttons, and table structures to show multiple documents.

5. Collection Configuration and Processing Classes

Since the configuration and business logic processing vary for different DLs and collections, this part of the implementation also has to be customized automatically. MARIAN is a general framework for different kinds of digital libraries; therefore it is necessary to customize it based on the underlying differences among specific collections. Thus, we use Java Reflection [14] to bind automatically generated collection configuration and business logic processor Java classes during runtime. The collection configuration class is loaded dynamically when the system is started by the resource manager, a class responsible for administering system resources. It instantiates the singleton collection processor and initializes all the specific, collection-dependent class and link managers. The collection processor acts as a facade to all the incoming queries, determines the language of the query data, selects and activates the correct class and link managers involved in the search, calls corresponding searcher modules to fuse and combine results based on document structure, and forwards results to the right document managers for rendering, possibly with different presentation options.

In Fall 2002 we applied 5SLGen for MARIAN to generate a digital library for CITIDEL (Computing and Information Technology Interactive Digital Library) within MARIAN encompassing thousands (~130K) of metadata records from the ACM Digital Library, NCSTRL Historic, CSTC, CS Virtual History Museum, and PlanetMath, dealing with millions of semantic links. Performance is comparable with similar tools.

5.2 Scenario Synthesis for Services and User Interface Prototype Generation

Despite the powerful capabilities of the current 5SLGen for MARIAN generator, the tool still has some drawbacks:

1. Lack of Generality: the current pilot systems and prototypes built using the 5SLGen are tied to the MARIAN API. Within the current 5SL framework, the generation of code prevents extensibility as components from other digital library component pools like ODL [13]; OpenDLib [5]; or Greenstone [19] cannot be plugged into the system (unless integrated with MARIAN).

2. Incomplete design and implementation of 5SLGen: 5SLGen for MARIAN took capabilities for basic services such as searching and browsing for granted due to the powerful MARIAN API. However, the MARIAN generator did not take into account a multiple scenario representation of the services. Since scenarios represent partial description of system behavior, an approach for scenarios composition or scenarios integration is needed to produce complete specifications of generic DL services.

3. Partial User Interface prototype generation: generation of a UI prototype for the services offered by a DL supports rapid prototyping. 5SLGen for MARIAN generates a partial UI prototype for basic search and browsing services. Nevertheless, this prototype is tied to the functionalities provided by the MARIAN API and does not mirror all the services that can be offered by a DL.

Accordingly, we have been extending the capabilities of our current generators to overcome many of those limitations. The first extensions deal with the question of scenario synthesis [3] and generation of multi-platform user interface prototypes for validation of services' behavior. In the area of scenario synthesis, one of the most difficult problems is the integration of hierarchical scenarios (i.e., scenarios that are composed of other scenarios) and the problem of scenario interleaving (i.e., when scenarios share the same state, the integrated scenario can have paths of execution that were not allowed by the design). Khriss et al. [12] have proposed a set of algorithms to deal with these problems. Their scenario synthesis algorithms are implemented in a tool called SUIP, which can generate prototype user interfaces [7] based on the integrated scenarios. Those user interfaces emulate the behavior of the scenarios from which they were generated, therefore allowing rapid validation of the proposed services with prospective users. However the SUIP tool has its own limitations. For scenario acquisition they adopt proprietary textual formats and grammars to represent UML collaboration and class diagrams. Moreover, the code generated for the UI prototype is in JAVA, is not extensible for other platforms and devices (e.g., HTML, WML), and is dependent on specific UI APIs of commercial tools, namely Java Café^{TM}.

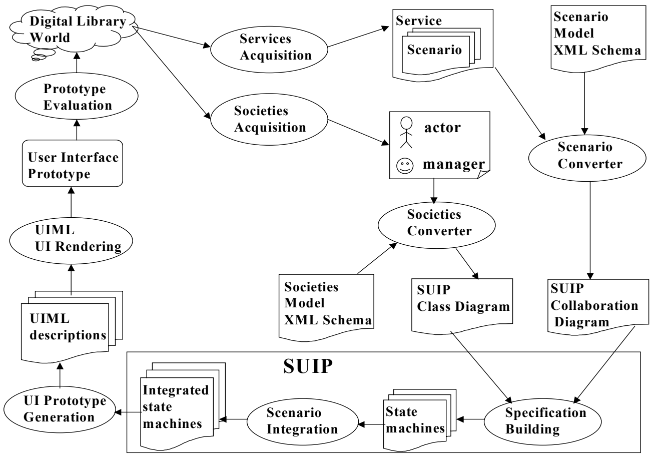
We have adapted, extended, and integrated the SUIP algorithms and tools within our generators to deal with the problem of rapid building, prototyping, and validation of DL services. The approach is illustrated in Fig. 8, which shows the sequence of activities in the proposed process. In the *Service Acquisition* activity, the digital librarian elaborates the services by describing, in 5SL, all the scenarios that compose the desired services. External tools can be integrated with 5SGraph to support drawing those scenarios; for example, based on UML sequence diagrams. Without such tools, 5SGraph still can be used to logically organize related small scenario models within services using the loading and reuse mechanisms of the tool. In the *Societies Acquisition* activity, the digital librarian uses the 5SGraph tool to associate actors and managers with already defined services/scenarios, which they either use or run, respectively.

With the help of specific converters, the 5SL-XML Scenario Model for every service is transformed in a specific textual representation for UML collaboration diagrams as expected by the SUIP tool. Similarly, the Societies models whose entities are involved in the service are mapped to textual representations of UML class diagrams. The converters are based on JDOM parsers and XML schemas for the two 5SL sub-models. The *Specification Building* activity consists of taking the two transformed models and generating state machine representations of the scenarios, which will permit the next activity of integration. Accordingly, during *Scenario Integration,* the state machines corresponding to each scenario of a same service are iteratively merged to obtain an integrated state machine of the complete service. Integrated state machines serve as input to the *UI Prototype Generation* activity. Our extended algorithm then produces a User Interface Markup Language (UIML) description of the interface [1]. UIML

[3] The term used in Software Engineering literature for integration of multiple, related scenario representations.

is an XML-based domain-specific markup language for describing user interfaces and their behavior in a highly device-independent way. The UIML descriptions are given to a *Rendering* tool, which can translate the UI specification to whatever environment, platform, or device the actual DL will run, therefore overcoming some of the deficiencies of the original SUIP tool. During *Prototype Evaluation*, the generated prototypes are executed and evaluated by the digital librarian or the end user.

Figure 8. Activities in scenario synthesis and user interface

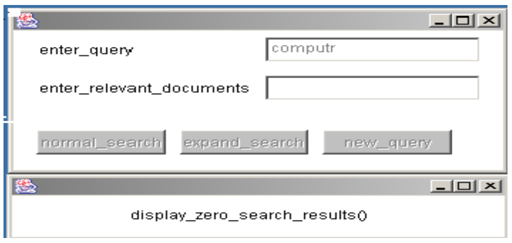


generation.

To support prototype execution, a *Simulation Window* is generated (Figure 10, bottom window), as well as a dialog box to Choose Scenarios (Figure 10). For example, after selecting the service *RelevanceFeedbackSearching*, a message is displayed in the simulation window that confirms the service selection and prompts the user to click the button *normalSearch*. At that point, the execution reaches a place from which several continuation paths are possible. The prototype then displays the dialog box for scenario selection. In the example, the upper selection corresponds to the scenario *regularSearch* and the lower one to the scenario *errorSearch*. Once a path has been selected the execution continues accordingly. If the user selects *errorSearch*, a message corresponds to "no results" is shown (Figure 9). Otherwise, the input field *EnterRelevantDocs*, and the buttons *ExpandSearch* and *NewQuery* are enabled. Clicking in *ExpandSearch* a new search is performed using some relevance feedback algorithm (e.g., Rocchio) which issues a transformed query with terms from the relevant documents, results are returned accordingly, and the process can be repeated until the user gets satisfied or choose *NewSearch,* when the system then returns to the initial state.

6. RELATED WORK

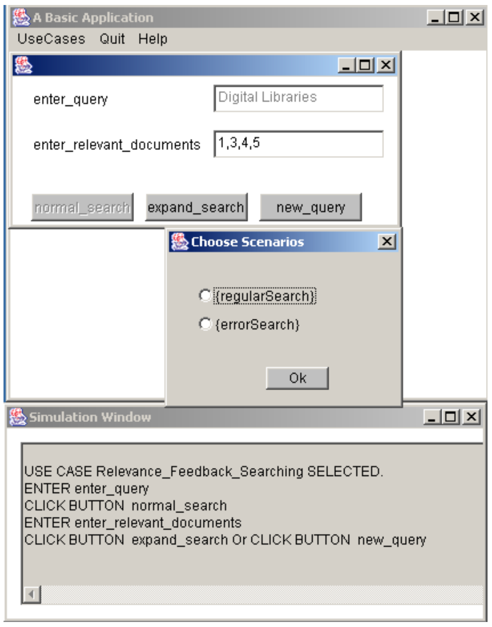
Theoretical, formal approaches for DLs are surprisingly almost completely missing in the DL literature. One could conjecture that this is due to the previously argued complexity of the field. The few existing attempts to give some formalization to portions of the field (e.g., [17], [4]) are incomplete. Our 5S formal approach has provided the most comprehensive formalization of the DL field we know so far.


enter_query computr

enter_relevant_documents

display_zero_search_results()

Figure 9. Frames generated for the Service Relevance Feedback Searching showing result of scenario errorSearch.



enter_query Digital Libraries

enter_relevant_documents 1,3,4,5

```
USE CASE Relevance_Feedback_Searching SELECTED.
ENTER enter_query
CLICK BUTTON  normal_search
ENTER enter_relevant_documents
CLICK BUTTON  expand_search Or CLICK BUTTON  new_query
```

Figure 10. Prototype Execution

In recent years, many DL systems with different architectures have been proposed including monolithic systems (e.g., Greenstone [19], MARIAN [10]), componentized architectures (e.g, ODL [13], OpenDlib[5]), agent-based architectures (e.g., UMDL [18]), and layered architectures (e.g., Alexandria [8]). There is no reason why those systems and their components can not be incorporated to our component pools, given that they export clear, reusable software interfaces with accessible entry points. Alternatively, we can think of versions of 5SLGen developed for specific target environments, like ODL (our next goal) or Greenstone.

Compared to our descriptive language, the WebML modeling language [6] provides powerful abstractions to describe and generate the hypertext and navigation structure of Web sites while the Digital Library Definition Language (DLDL) [16] is focused on describing external behavior of DLs for purposes of supporting interoperability in terms of federated searching. More recently, in the context of DL requirements gathering, Bolchini and Paolini have proposed a hypermedia-based methodology that uses scenarios for goal-oriented requirements specification for digital libraries [3]. All these approaches, however, are limited in scope and in their ability to cover all of the challenges in the modeling and construction of complex digital libraries. More importantly, none of those works are supported by a sound, formal theory for digital libraries as in our case.

The closest approach to our DL generators is the collection services and plug-in architecture of Greenstone [19]. However their architecture covers only portions of our Stream and Structural models with mainly no support for modeling and generation of customized DL services (other than searching and browsing).

Finally, we are unaware of a similar domain-specific graphical modeling tool for DL design like 5SGraph, which includes support for rich interaction styles as well as advanced capabilities such as extensibility, synchronization, reusability of components, and automatic enforcing of semantic constraints.

7. CONCLUSIONS AND FUTURE WORK

We have presented a comprehensive architecture along with theories, models, languages, and tools that support the complete life cycle of DL development, including requirements gathering, conceptual modeling, rapid prototyping, and code generation and reuse. All those activities are completely integrated within the architecture through automatic mappings and transformations.

Current work is focused on generalizing 5SLGen for multiple, diverse digital library component pools. The enhanced 5SLGen should be able to generate workflow interfaces that mirror the executable state machine of the representative services. These workflow interfaces would allow components from other digital library architectures to be plugged into the implementation of the interface. Adding this layer of abstraction between the design and implementation enables extensibility and interoperability by allowing digital library components from multiple, diverse component pools to implement the interface. On the theory side, we are exploring semantic relationships, properties, and constraints existent in the 5S formal model including issues of quality in digital library modeling and generation. The results of these investigations will be incorporated in our (meta) models, languages, and tools. Finally, all our software, including, MARIAN, ODL, 5SGraph, etc., is open source and all of them run in prototype mode. Therefore more extensive tests and development is required to guarantee robustness, scalability, etc.

Acknowledgments

Thanks are given for the support of NSF through its grants: IIS-9986089, IIS-0002935, IIS-0080748, IIS-0086227, DUE-0121679, DUE0121741, and DUE-0136690. The first author also is supported by CAPES, process 1702-98. Thanks also go to A. Prabhune, R. Mahajan, M. Singhal, A, Shah, L. Lobo, P. Shastri, and V. Colaso, for their work on SUIP.

5. REFERENCES

[1] M. Abrams, C. Phanouriou, A. L. Batongbacal, S. M. Williams, J. E. Shuster: UIML: An Appliance-Independent XML User Interface Language. Computer Networks (11-16): 1695-1708 (1999)

[2] D. S. Batory, C. Johnson, B. MacDonald, D. von Heeder: Achieving extensibility through product-lines and domain-specific languages: a case study. TOSEM 11(2):191-214, 2002

[3] D. Bolchini, P. Paolini: Goal-Oriented Requirements Specification for Digital Libraries. ECDL 2002: 107-117, Rome, Italy, September 16-18, 2002.

[4] D. Castelli, C. Meghini, P. Pagano: Foundations of a Multidimensional Query Language for Digital Libraries. ECDL 2002: 251-265

[5] D. Castelli, and P. Pagano. OpenDLib: A Digital Library Service System. *ECDL 2002*: 292-308, Rome, Italy, September 16-18, 2002.

[6] S. Ceri, P. Fraternali, A. Bongio: Web Modeling Language (WebML): a modeling language for designing Web sites. Computer Networks 33(1-6): 137-157, 2000.

[7] M. Elkoutbi, I. Khriss, and R. K. Keller, Generating User Interface Prototypes from Scenarios, in Proceeding of the 4th IEEE International Symposium on Requirements Engineering, pages 150-158, Limerick, Ireland, June 1999.

[8] J. Frew, M. Freeston, N. Freitas, L. L. Hill, G. Janee, K. Lovette, R. Nideffer, T. R. Smith, Q. Zheng: The Alexandria Digital Library Architecture. IJODL 2(4):259-268, 2000.

[9] M. A. Gonçalves and E. A. Fox. 5SL - A Language for Declarative Specification and Generation of Digital Libraries. 2*nd ACM/IEEE Joint Conference on Digital Libraries*, 263-272, July, 2002. Portland, Oregon.

[10] M. A. Gonçalves, P. Mather, J. Wang, Y. Zhou, M. Luo, R. Richardson, R. Shen, L. Xu, E. A. Fox: Java MARIAN: From an OPAC to a Modern Digital Library System. SPIRE 2002, 194-209, Lisbon, Sept., 2002.

[11] M. A. Gonçalves, E. A. Fox, L. T. Watson, N. A. Kipp. Streams, Structures, Spaces, Scenarios, Societies (5S): A Formal Model for Digital Libraries. Under review for ACM TOIS.

[12] I. Khriss, M. Elkoutbi, and R. K. Keller, Automating the Synthesis of UML Statechart Diagrams from Multiple Collaboration Diagrams, in Proc. of UML'98: Beyond the Notation, 115-126, Mulhouse, France, June 1998.

[13] H. Suleman. Open Digital Libraries. Ph.D. dissertation. Virginia Tech, Department of Computer Science. Nov. 2002. http://scholar.lib.vt.edu/theses/available/etd-11222002-55624/

[14] Sun, Java Reflection API, ftp://ftp.javasoft.com/docs/jdk1.1/java-reflection.ps

[15] Tigris.com, ArgoUML: A UML design tool with cognitive support, http://argouml.tigris.org/

[16] M. Zubair, K. Maly, I. Ameerally, and M. Nelson. Dynamic construction of federated digital libraries. Proceedings of the Ninth International World Wide Web, Conference, Amsterdam, May, 2000, poster.

[17] B. Wang: A Hybrid System Approach for Supporting Digital Libraries. IJODL2(2-3):91-110 , 1999.

[18] P. Weinstein, W. P. Birmingham: Creating Ontological Metadata for Digital Library Content and Services. IJODL 2(1): 20-37, 1998.

[19] I. H. Witten and D. Bainbridge, How to Build a Digital Library, San Francisco, Calif., Morgan Kaufmann, 2002.