# SCENARIO-BASED GENERATION OF DIGITAL LIBRARY SERVICES

ROHIT KELAPURE

**A Thesis Presented 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**

**June 12, 2003**
**Blacksburg, Virginia**

**Keywords: Digital Libraries, 5S, 5SL, 5SLGen, Modeling, Code Generation**

# SCENARIO-BASED GENERATION OF DIGITAL LIBRARY SERVICES

**ROHIT KELAPURE**

**Committee Chairman: Dr. Edward A. Fox**
**Computer Science**

## (Abstract)

With the enormous amount of information being created digitally or converted to digital formats and made available through Digital Libraries (DLs), there is a strong demand for building tailored DL services to attend the preferences and needs of diverse targeted communities. However, construction and adaptation of such services takes significant effort when not assisted by methodologies, tools, and environments that support the complete life cycle of DL development, including requirements gathering, conceptual modeling, rapid prototyping, and code generation/reuse. With current systems, these activities are only partially supported, generally in an uncorrelated way that may lead to inconsistencies and incompleteness. Moreover, such existing approaches are not buttressed by comprehensive and formal foundations and theories. To address these issues we describe the development, implementation, and deployment of a new generic digital library generator yielding implementations of digital library services from models of DL "societies" and "scenarios". The distinct aspects of our solution are: 1) an approach based on a formal, theoretical framework; 2) use of state-of-the-art database and software engineering techniques such as domain-specific declarative languages, scenario-synthesis, and componentized and model-driven architectures; 3) analysis centered on scenario-based design and DL societal relationships; 4) automatic transformations and mappings from scenarios to workflow designs and from these to Java implementations; and 5) special attention paid to issues of simplicity of implementation, modularity, reusability, and extensibility. We demonstrate the feasibility of the approach through a number of examples.

Thanks is given to the AmericanSouth.org project funded by the Mellon Foundation.

## Acknowledgements

First, I would like to thank my advisor Dr. Fox for providing guidance, support and encouragement. The opportunities and the resources provided by Dr. Fox have made me what I am today.

I would like to thank my thesis committee members Dr. Carroll and Dr. Ehrich for their support and willing ears.

I thank Marcos André Gonçalves for being an elder brother and making me see the light at the end of the tunnel. The idea for this thesis was born out of a discussion with Marcos in August 2002. His suggestions, ideas, support, help, and constructive criticism have helped me immeasurably.

I thank my parents whose unconditional love has made this effort possible. I thank my mother for making sense of it all. I thank my father for providing guidance and support.

I thank Rao Shen, Aaron Krowne, and Ming Luo for providing help, clearing my doubts, and being good friends. I will miss you.

I thank all the members of the Digital Library Research Laboratory. These members are my extended family and have helped create a home away from home.

## Table of Contents

## List of Figures

## List of Tables

## List of Examples

# 1. Introduction

## 1.1. Context

Before launching into a detailed analysis of the problem space, the context for the problem and the solution is provided in sections 1.1.1, 1.1.2, 1.1.3 and 1.1.4.

### 1.1.1. What is a Digital Library?

The term "digital library" is the most recent in a long series of names for a concept that has been discussed since the development of the first computer. Vannevar Bush, head of the US Office of Scientific Research and Development during World War II, wrote about the "memex", which stimulated most of the early application of computers to information retrieval and anticipated the idea of hypertext [1]. Licklider in his seminal work referred to his vision of a fully computer-based library as a "library of the future" [2].

The D-Lib Working Group on Digital Library Metrics gave the definition of digital libraries as, "The collection of services and the collection of information objects and their organization, structure, and presentation that support users in dealing with information objects available directly or indirectly via electronic/digital means" [3].

The Digital Library Federation defines digital libraries as "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" [4].

Each of these definitions attempts to model different facets of existing systems. Some definitions attempt to place emphasis on the human aspects whereas other definitions try to fit DLs into formal frameworks. For the purpose of this study we adopt the definition of digital libraries provided by Fox who defines digital libraries as "complex data/information/knowledge systems that help: satisfy the information needs of users (societies), provide information services (scenarios), organize information in usable ways (structures), manage the location of information (spaces), and communicate information with users and their agents (streams)" [5].

### 1.1.2. What are Digital Library (DL) Services?

A DL service is the fusion of computing, storage, and communication machinery coupled with the software needed to reprise, emulate, and extend the services provided by conventional libraries based on paper and other material means of collecting, storing, cataloging, finding, and disseminating information [6]. The DL service exposes a specific functionality to its users to fulfill the users' information needs. The services exposed by a DL include: services to support management of collections, services to provide replicated and reliable storage, services to aid in query formulation and execution, services to assist in name resolution and location, and services for access to the library items and the processing of the information contained in the items and communication of information about the items. Basic DL services include services for indexing, searching, browsing and cataloging digital resources.

### 1.1.3. What are Scenarios?

Scenarios provide the means for capturing requirements specifications as well as a means of communication between users and software developers. In the context of requirements and software engineering, a scenario is a sequence of events that occur during one particular execution of a system [7]. Scenarios describe an aspect of the external behavior of a system from a users' viewpoint. Scenarios are used in the analysis phases of software development to elicit, capture, and document requirements. Scenarios take a user-centered view by default and keep the design discussion focused on user activities [8, 9]. For these and other reasons, scenarios have gathered widespread acceptance in the software and requirements engineering community.

### 1.1.4. What is Meant By Scenario-Based Generation of DL Services?

Scenario-based generation of DL services can be defined as modeling of DL services with a scenario-based requirements analysis and design methodology so as to generate an implementation for the services. We assume that each service can be modeled as a set of overlapping scenarios. This assumption is motivated by the role played by scenarios in the requirements elicitation process. The implementation of the DL services is generated semi-automatically and is comprised of all the software components needed for a functional DL to expose the modeled services. Scenario-based generation of DL services requires an approach for generating code based on the requirements elicited in the form of user-level scenarios. The implementation of DL services contains code for not only implementing the functionality of the service but also coupling the services and exposing them to the users.

The title of this thesis is scenario-based generation of DL services rather than scenario-based generation of DLs as we do not model and generate all components of a DL. Digital libraries consist of a set of electronic resources and associated technical capabilities for creating, searching, and using information. The content of digital libraries includes data, metadata that describe various aspects of the data (e.g., representation, creator, owner, reproduction rights). We do not model the data and the metadata stored in a DL. We are only concerned with services that operate on the data rather than with storage, creation, archiving, and preservation. In the context of this work, the generation of DL services means generation of a DL that exposes the services without generating the content (data and metadata) stored in the DL.

## 1.2. Motivation

Digital Libraries are complex information systems, which integrate research and findings from disciplines such as hypertext, information retrieval, multimedia services, database management, and human computer interaction. Some of the well-known digital library related research areas include classification, interoperability between heterogeneous collections, communication protocols, search engines, crawlers, information visualization, usability, and human computer interaction issues [10]. The broad and deep requirements of DLs demands models in order to understand better the interaction among its components. Models are crucial to specify and understand clearly and unambiguously the structure and behavior of complex information systems.

While much attention has been paid to the study of making better digital libraries, little focus has been put on simplifying the process of modeling and building DLs. The process of building a digital library involves specification of the content to be stored; how that content is organized, structured, described, and accessed; which services are offered by the library (e.g., searching, browsing, personalizing, collaborating); and how patrons (and automated agents) ultimately use those services and interact with each other in the DL environment [7]. Thus, by their own nature, DLs are complex and inter-disciplinary information systems making it difficult and expensive to construct new DLs. This complexity of DLs coupled with the ad-hoc approaches taken by DL designers, many of whom are either library technical staff with little or no formal training in software engineering, or computer scientists who have little background in information science, has resulted in the construction of DLs of arbitrary complexity that are difficult to maintain and extend.

In domain-specific modeling, a design engineer describes a system by constructing a model using the terminology and concepts from a specific domain. Analysis then can be performed on the model, or the model can be synthesized into an implementation. To get to a situation of domain modeling followed by fully automatic code generation, three things are required: a modeling tool with support for the domain-specific modeling language, a code generator, and a domain-specific component library. There is a lack of domain-specific languages, domain-specific prototyping tools, and CASE environments for DLs. Current prototyping tools in this inter-disciplinary domain are limited to database and web applications. They do not capture the specific abstractions and notations for the domain at hand. They do not support specific DL patterns, models, and methodologies.

There are few software toolkits available to build DLs. As a result, most DLs are custom-built using homegrown architectures without making use of conceptual modeling, requirements analysis, and methodological approaches. The general trend has been to develop tools to solve small parts of the problem, whereas the root of the problem – the lack of specific DL patterns, models, methodologies, formalisms, and languages – is almost completely ignored.

DLs are either built as monolithic, tightly integrated, and generally inflexible systems or by assembling components. The lack of models and processes for constructing these systems results in problems in interoperability and adaptability. For full-service, large-scale digital libraries to develop and interoperate with others, a fair degree of standardization is required. Such standards must rest on some agreed-upon framework and reference model, building upon careful definition of requirements [10]. The process of requirements analysis helps the DL designer capture the functional requirements of a software system through the creation of structural and behavioral models represented by DL "societies" and "scenarios". Scenarios are key artifacts in systems engineering, but their management is poorly understood. Scenarios are a good communication base with naive users and other non-technical people and support early validation of requirements at a low abstraction level A scenario-based design approach to DL development is lacking in current DL toolkits and prototyping tools. This also hampers the ability to tailor DL components and behaviors to particular user communities.

Recent work on DLs has shown that they can be built by connecting small components that communicate through a family of lightweight protocols, using XML as the data interchange mechanism [11]. This work successfully attempts to provide a solution to the interoperability problems facing DLs, however it still leaves certain questions unanswered such as:

- How are DLs built when components do not follow protocols?
- How are DLs built when services cannot be modeled as components?

## 1.3. Approach

To address the issues raised above, we present the development, implementation, and deployment of a new generic DL generator (5SLGen) yielding implementations of DL services from models of DL "societies" and "scenarios".

5SLGen is a DL generator that combines theory, language, and tools in a coherent and cohesive way to allow automatic generation of tailored DL services. 5SLGen provides a framework for building DL services that provides programmers with an infrastructure that supports a coherent architectural model, allowing developers to concentrate on applying their expertise to the problem domain. In the case of 5SLGen, the framework takes care of the common functionality that every DL needs by reusing components, so that you can implement the services that are important and specific to your DL. Focusing on the societies and the scenarios models of a DL, 5SLGen builds on the Open Digital Libraries framework and the State and Model-View-Controller design patterns to create an implementation for DL services [11-14].

Our objective is to cover the process of DL development, from requirements to analysis, analysis to design, and design to implementation. We aim to generate "tailored" DL software satisfying the particular requirements of specific DL societies. The basic idea is to develop models, languages, and tools able to capture the rich set of DL requirements and properties of particular settings and to automatically convert these "patterns" into different representations by properly "compiling", transforming, and mapping models in different levels and phases of the DL development process. The assumption is that automatic transformations and mappings diminish the risk of inconsistency and increase productivity. This view is supported by:

1. Having a model based approach that allows the DL designer to describe: 1) the kinds of multimedia information the DL supports (Stream Model); 2) how that information is structured and organized (Structural Model); 3) different logical and presentational properties and operations of DL components (Spatial Model); 4) the behavior of the DL (Scenarios Model); and 5) the different societies of actors and managers of services that act together to carry out the DL behavior (Societal Model) [7]. These and other DL notions have been organized and formalized into the 5S (Streams, Structures, Spaces, Societies, Scenarios) formal framework [15]. This formal framework provides a foundation for the DL generator.

2. Using a domain-specific language based on 5S, 5SL, for declarative specification and automatic generation of DLs [7]. Domain-specific languages enable applications to be programmed with domain abstractions, thereby allowing compact, clear, and machine-processable specifications to replace detailed and abstruse code [16].

3. Using scenario-based design for defining the behavior of a system. Scenarios keep design discussion focused on the level of task organization that actors experience in their tasks [17]. In 5S, we envision scenarios as sequences of events that modify states of a computation in order to accomplish some functional requirement. We use scenarios to describe the behavior of DL services and societal interactions.

4. Implementing a code generator that allows a DL designer to provide a modeling specification in terms of scenarios and societies. This generates implementations using precise transformations/mappings. The generated DL makes use of well-defined components that each carry out key DL functions interacting with one another using lightweight protocols. We draw heavily upon work with the Open Archives Initiative Protocol for Metadata Harvesting

(OAI-PMH [18]) and Open Digital Libraries [11].
We pay special attention to the issues of flexibility, extensibility, and reusability in the implementation of DL services. The role of 5SLGen in providing these features is explained below.

5SLGen exports models of 5SLSocieties and 5SLScenarios to other modeling languages such as the Unified Modeling Language (UML). UML is the de facto industry standard object-oriented modeling language [19]. UML consists of several sublanguages, which are suited to model structural and behavioral aspects of a software system. It has immense support in the form of graphic modeling and CASE tools, which provide for forward, reverse, and roundtrip engineering. In order to leverage the advantages of both established modeling languages and associated CASE tools, along with domain-specific component modeling, we map the domain solution (5SLSocieties and 5SLScenarios models) to core UML models. This open interchange of 5SL DL models with other CASE tools provides flexibility in modeling DLs.

The services modeled and generated using 5SLGen expose a clean interface that can be easily redeployed, while implementing other DLs. The set of 5SL models (5SLSocieties and 5SLScenarios models) generated during the course of this study can be reused as-is while implementing services for new DLs. This ability to redeploy services and reuse 5SL models supports our claim of reusability.

5SL models for DL services can be extended according to specific community needs. These services can be implemented with specific components customized for particular societies. Extensibility of 5SL models provides for extensibility of the implemented services.

Design of a domain-specific modeling language and the implementation of a model-driven code generator based on a scenario-based design methodology constitute the main work in this thesis. Activities two, three and four mentioned above are the focus of this thesis.

## 1.4. Research Contributions

This research makes the following contributions to the fields of DL modeling and architecture.

1. Designed an approach for semi-automatic generation of DL services using a component-oriented, scenario-based software development design methodology.

2. Implemented a DL generator that generates DL services based on an analysis centered on scenario-based design and DL societal relationships.

3. Used a formal theory to help design and generate high quality DLs.

4. Implemented a model-driven architectural approach to DL design and development.

5. Developed a set of 5SLSocieties and 5SLScenarios models for common DL services that can be reused by other DLs.

6. Facilitated the exchange of DL models among different CASE tools.

## 1.5. Outline of the Thesis

- Chapter 1 outlines the motivation, problem space, and scope of the research.
- Chapter 2 presents a survey of the relevant literature in the field of DL interoperability, DL modeling, DL architectures, DL generation, scenario-based requirements analysis and design, and CASE tools.
- Chapter 3 outlines the 5S methodology for generation of DLs.
- Chapter 4 elaborates on the design and architecture of 5SLGen.
- Chapter 5 covers the implementation of 5SLGen and provides a detailed explanation of the steps involved in transforming 5SL scenarios and societies models to code.
- Chapter 6 presents an analysis of the services and the DLs generated using 5SLGen followed by observations on the entire modeling and generation process.
- Chapter 7 presents conclusions and recommendations for future work.

# 2. Review of Literature

This chapter provides information on the important concepts that relate to the work done. We initially start with a discussion of DL concepts such as characteristics of DLs or categorizations of services exposed by a DL, and provide examples of DLs running in a production environment. This is followed by a comprehensive listing of DL architecture efforts. Thereafter the process of modeling and construction of DLs is elaborated. The later half of the chapter focuses on software engineering processes and methodologies, particularly scenario-based requirements analysis and design. We conclude by elaborating briefly on the underlying technology and standards involved.

## 2.1. DL: Concepts

### 2.1.1. DL Definitions

We presented a number of definitions for DLs in section 1.1.1. Common to all those definitions are certain fundamental elements. The Association of Research Libraries [20] sums up the commonalties of these different definitions as follows:

- 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.

### 2.1.2. Defining the Digital Library – Where Does Service Fit In?

Most of the definitions of a DL have an emphasis on technology and information resources and a noticeable lack of discussion of the service aspects of a DL. We attempt to bridge this information gap by elaborating on the type of services offered by DLs and classifying the services into categories. Such classification helps us build a vocabulary and reduce the inherent complexity of the field by grouping similar concepts. The discussion on the types of services offered has been drawn from Gary Marchionini's treatise on the research and development of DLs [21].

#### 2.1.2.1. DL Services

A service is an entity that has a well-defined interface and behavior that can be referenced by users. The range and depth of services that a library provides to its users is driven by its service mission and policies. Policies determine usage, availability, quality, resource allocation and types of services offered to the patrons of the DL. Libraries offer different types of reference and referral services, instructional services, added value services, and promotional services. Reference and referral services encompass services for ready reference, exhaustive search, and selective dissemination of information. Instructional services include services for bibliographic instruction and database searching. Added value services incorporate services for bibliography preparation and language translation. Promotional services include services that support freedom of expression and literacy (each one teach one).

#### 2.1.2.2. Categorization of DL Services

The services exposed by a DL are of two types: composite (information satisfaction services) and elementary (infrastructure services).

Infrastructure services are elementary services. These services provide the basic infrastructure for the DL. Examples include searching, collecting, indexing, rating, linking, and browsing services. Infrastructure services do not rely on other services to fulfill their responsibilities while information satisfaction services act like umbrella structures that bring together other services, which collaborate to implement certain functionality.

Information satisfaction services are composite services. They are composed of other services (elementary or composed) by reusing or extending them. Examples include multi-classification browsing, relevance feedback search, reference and question answering, filtering, and selective dissemination services. Infrastructure services may be used for creation, preservation, transformation, or aggregation of data and metadata. Information satisfaction services then take a representation of a user interest/need (e.g., a query, a browsing action by means of selecting some navigation anchor, or a profile) and produce a number of different outputs. For example, a searching service generates weighted sets of pairs (handle, weight) where the weight corresponds with how well the digital object with that handle matches the query.

### 2.1.3. DL: Examples

We now consider examples of two popular DLs, viz., Computing and Information Technology Interactive Digital Educational Library (CITIDEL) and NDLTD, that expose a wide range of both elementary and composite services. The purpose of focusing on these DLs is twofold; first it provides context for the work done as the services exposed by these DLs are later modeled using 5SLGen, and second it provides insight into some of the services offered by modern DLs.

#### 2.1.3.1. CITIDEL

The CITIDEL is a comprehensive, pedagogically focused digital library in Computer Science (CS) and Information Technology. It provides access to a number of collections from publishers, corporate research efforts, volunteer initiatives, CS departments, educational initiatives, and universities [22]. It is a composite collection of CS resources with services layered over the aggregated collections. CITIDEL acquires content either through harvesting resources using the OAI-PMH, manual discovery and input, and focused crawling of the web. The services offered by CITIDEL include Searching, Filtering/Browsing, Review/Rankings, Annotation, Discussion, Instruction, and Classification. The implementation of CITIDEL draws and builds on work done with the MARIAN system for the DL core and the ODL framework for federation and componentization [23]. The philosophies of ODL and CITIDEL mesh well. The communication between CITIDEL and other harvesting DLs and among CITIDEL components mirrors the communication that occurs among ODL components that constitute a DL. Figure 2.1 illustrates both the collections and services that constitute the CITIDEL.

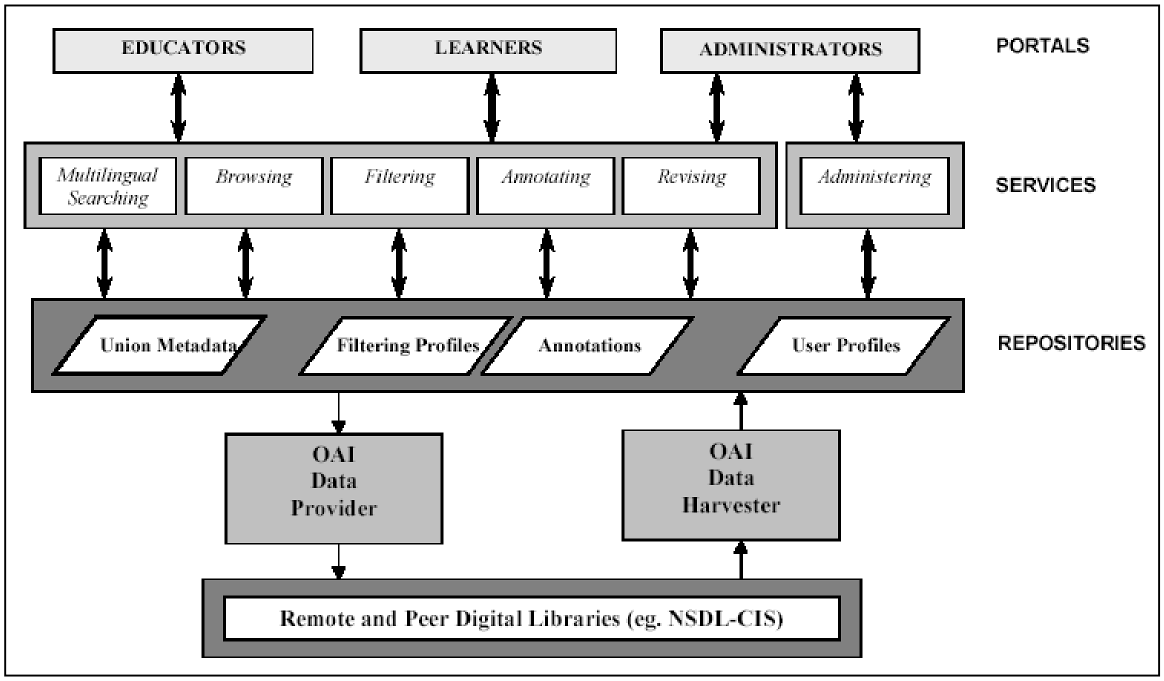


**Figure 2.1 CITIDEL architecture from original proposal**

### 2.1.3.2. NDLTD OAI Based Union Catalog

The Networked Digital Library of Theses and Dissertations (NDLTD) OAI Based Union Catalog (hereafter, the Union Catalog) is an experimental DL constituted from ODL components that seeks to illustrate the benefits of the componentized approach to DLs [24]. The Union Catalog harvests data from other source archives through OAI-PMH interfaces. The data is aggregated into a central archive for use by local services. Search, Browse, and Recent high-level services are provided using this data. Search indexes the data and exposes an OAI-like interface for specifying keyword queries. Browse sorts the data and exposes a slightly different OAI-like interface for accessing items by controlled vocabulary elements. Recent stores recent items and upon request returns a random sample of those [12]. Figure 2.2 shows the architecture of the Union Catalog system taken from [11]. All arrows represent data being accessed or transferred through OAI or extended OAI interfaces, and all nodes labeled “OA” are Open Archives.

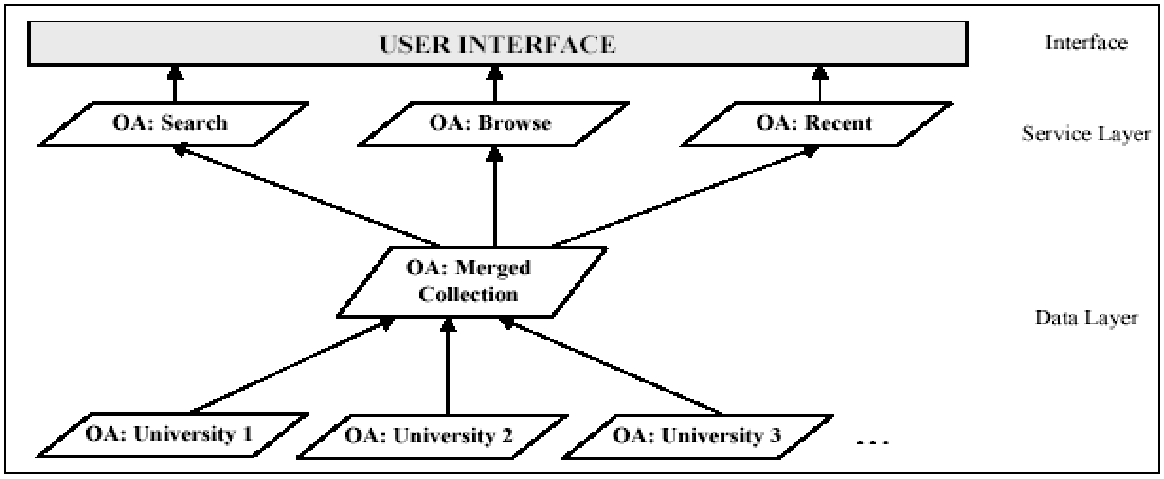


Figure 2.2 Architecture of the NDLTD Union Catalog

## 2.2. DL Architecture Efforts

We categorize DL architectures into two types, viz., componentized and monolithic.

### 2.2.1. Componentized Architectures

Componentized architectures represent DLs as an open federation of distributed services/components. Each component/service carries out a specific functionality and communicates with other components using either open or proprietary protocols. A listing of such efforts is provided below

#### 2.2.1.1. Dienst System

The Dienst architecture specifies four core digital library services. User-interface services provide a human-friendly gateway to the information obtained from other services. Repository services store and provide access to documents, according to the Dienst document model, which provides transparent access to documents stored in a distributed environment. Index services provide search capabilities, accepting a query and returning a list of document identifiers that match the query. Collection services define the components, services and documents of the digital library collection, making it possible for user-interface services to interact with them [25]. The Dienst architecture served as the technical foundation of the Networked Computer Science Technical Research Library and was a leading influence on the development of the OAI protocol.

#### 2.2.1.2. Fedora

The Fedora project implemented a multi-layered service structure that evolved from concepts implemented in the Dienst repository architecture. Fedora was envisioned to be a part of a larger open-architecture framework in which the functionality of a digital library was partitioned into a set of services with well-defined interfaces. These core services included: repository services, index services, collection services, naming services, and user-interface services. The well-defined interfaces of these core services allow them to be combined with each other and other value-added services to create usable instantiations of digital libraries [26].

#### 2.2.1.3. OpenDLib

OpenDLib is a software toolkit that can be used to create a digital library by assembling a federation of services that implement the DL functionality [27]. The set of services can be extended to provide additional functionality. The federation of different services is governed by a formal model. Dependent services communicate among one another via the OpenDLib Protocol (OLP). The federation is managed by a manager service that is completely parametric with respect to the type and number of managed services.

#### 2.2.1.4. Open Digital Libraries

In developing the metadata harvesting protocol, the OAI provided a mechanism to separate data providers from service providers. In this process, the OAI established best practices to support their protocol, which are relevant to digital library design. These best practices implement some of the ideas that were a part of the Kahn and Wilensky's Repository Access Protocol. These ideas have now been realized in OAI's broadly-supported DL interoperability protocol [28].

The Open Digital Library (ODL) project has exploited the conceptual framework provided by the OAI protocol in order to form the base for a general-purpose inter-component interaction protocol for digital libraries [12]. ODL defines popular services as self-contained components and defines interfaces for these components to interact with upstream open archive (OA) data providers and peer components, as well as downstream components. ODL assumes that all components are OA data providers to exploit the simplicity of the OAI-PMH protocol. ODL components communicate among themselves through the overlaid semantics of the OAI-PMH. A DL is made up of a network of ODL components. Figure 2.3 taken from [11] from illustrates the architecture of a simple DL built from a network of ODL components.

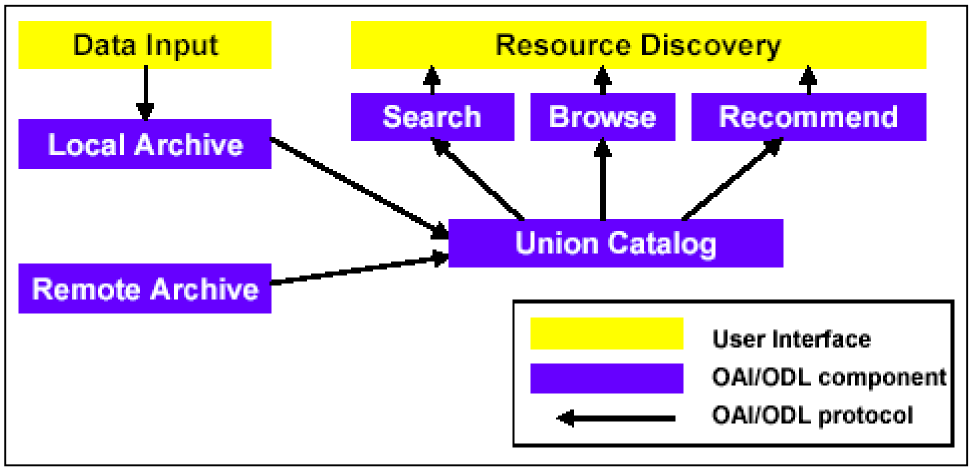


**Figure 2.3 Example networked architecture of an ODL**

### 2.2.1.5. PhysNet

The PhysNet portal is a one-stop portal providing a number of distributed services to physicists and graduate students. The PhysNet portal uses the uPortal toolkit to build distributed, componentized, and personalized DLs, using OAI and web services to enhance existing PhysNet services. It builds on the work done in the ODL project, particularly modeling a DL as a network of interconnected components with web services enabled.

## 2.2.2. Monolithic Architectures

Monolithic architectures represent DLs with tightly integrated components that lack the flexibility to add or remove new functionalities at short notice. However, monolithic DLs are highly customized and optimized for specific community needs and consequently offer better performance as compared to similar componentized systems.

### 2.2.2.1. MARIAN

Multiple Access and Retrieval of Information with Annotations (MARIAN) is a DL system with an extensible set of services including browsing, searching, retrieving, automatic collection building, and uniform preservation over networked collections. MARIAN is a monolithic DL with a layered architecture [29].

#### 2.2.2.2. Greenstone

The Greenstone DL software is an open-source system for the construction and presentation of information collections. Greenstone allows the construction of complex DLs and tailoring of many parts of DLs to specific domains and needs. However to achieve these goals Greenstone utilizes heterogeneous machinery including Perl modules, proprietary markup languages and macros, CORBA, Standard Template Library (STL) in C++, etc. [30] The customized DL generated is a monolithic entity.

## 2.3. Modeling of DLs

### 2.3.1. What is a Model? What is a Metamodel? What is a Meta-Metamodel?

There are always three or more levels of abstraction for any modeling context. There is the model itself (layer M1), there are instances of the model (layer M0), and there is a set of constructs/rules for constructing the model (layer M2). In the context of DLs, the 5S theory serves as the metamodel for defining models of DLs. Table 2.1 puts the 5S metamodel in context using OMG's four-layer metamodel architecture.

**Table 2.1 OMG metamodel architecture**

| OMG Terms | Description | Example |
|---|---|---|
| meta-metamodel (layer M3) | The infrastructure for metamodeling architecture. Defines the language for specifying metamodels. | Specifies meta-metaclasses for the UML metamodel |
| metamodel (layer M2) | An instance of a meta-metamodel. Defines the language for specifying a model. | Specifies metaclasses for the UML and 5S metamodel |
| model (layer M1) | An instance of a metamodel. Defines a language to describe an information domain. | Specifies classes for the UML and 5SL (societies, structures, scenarios, spaces, streams) models |
| user objects (layer M0) | An instance of a model. Defines a specific information domain | User objects that are instances of UML, 5SL user model |

### 2.3.2. Formal Modeling of DLs

Most of the disciplines that constitute the field of DLs have a foundation based on formal models. However, modeling in the domain of DLs is still in its infancy. For instance, the hypertext/hypermedia community has rich abstraction models and decompositions for hypermedia systems. Examples include OOHDM and Web2000 [31, 32], or WebML [33]. The database community also has the relational and object-oriented models. The field of IR has the vector, Boolean, and probabilistic models. Table 2.2 summarizes the formal modeling efforts in the DL domain and provides context for our work with the 5S formal framework.

**Table 2.2 Characterization of formal models in the DL domain**

| Model | Characteristics | Drawbacks |
|---|---|---|
| Wang's, "Hybrid system approach for supporting digital libraries" [34] | Abstract structure of a DL is defined as a combination of a specific purpose database and a user-friendly interface.<br><br>Formal data structure for linking an object-oriented database with hypermedia to support digital libraries.<br><br>Formalizes DLs in terms of the formal language Z. | Does not describe many characteristics of DLs such as interoperability, classification, organization tools, etc. |
| Kalinichenko's canonical model for information systems [35] | Compositional design of information systems applied to semi structured data on the Web.<br><br>The digital library is designed as a composition of fragments of web sites. | Provides a partial solution for interoperability in DLs |
| Castelli's mathematical and architectural model for the modeling of digital contents [36] and specifying the services provided by a digital library [27] respectively | Formalized the concepts of documents, based on the notion of views and versions, metadata formats and specifications, and a first-order logic based language | Incomplete in its coverage of all the concepts of the DL domain |
| 5S model for DLs [15] | Formal, theory-based approach to the problems of defining, understanding, modeling, building, personalizing, and evaluating DLs.<br><br>Use of mathematically based formal methods to develop a theoretical framework for DLs.<br><br>A more complete description of the 5S model can be found in section 3.1. | The abstractions of 5S provide a formal foundation to define, relate, and unify concepts -- among others, of digital objects, metadata, collections, and services required to formalize and elucidate DLs. |

### 2.3.3. 5SL: A Domain-Specific Declarative Language for DLs

5SL is a domain-specific, declarative language with a formal semantics used for conceptual modeling of digital libraries [7]. The formal semantics is understood in terms of a translation of language constructs into the 5S formal theory. 5SL models are instantiations of the 5S metamodel for DLs. Hereafter the 5SL models for the societies and scenarios abstractions of the 5S metamodel are referred to as 5SLSocieties model and 5SLScenarios model, respectively. 5SL has been expanded upon in detail in section 3.1**.** We introduce 5SL here to put it in context with other popular and standard modeling languages such as the UML.

### 2.3.4. The Unified Modeling Language

The Unified Modeling Language has been accepted as an industrial standard for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system [37]. It consists of several sublanguages, which are suited to model structural and behavioral aspects of a software system. The discussion below on UML has been adapted from [38].

UML provides class and object diagrams, to model all structural aspects of a system on a type and instance level, respectively. Objects are described by their attributes as well as by the signatures of operations that may change the state of an object. Structural relationships can be described as general associations or as weak or strong aggregation relationships between objects. Objects may be specified as passive or active objects, the latter ones having their own, permanently active thread of control.

UML offers five diagram types (use case diagrams, activity diagrams, Statechart diagrams, sequence diagrams, and collaboration diagrams) to model the behavioral aspect of a system. Each of them focuses on a certain view on a system.

Use case diagrams model the external view of the system. They help determine the main functionality or processes of a system (called use cases) and the actors participating in these use cases. Activity, sequence and collaboration diagrams provide the inter-object view of a system, i.e., the communication between and collaboration of different objects. A collaboration diagram shows an interaction, consisting of a set of objects and their relationships, including messages that may be dispatched among them. A sequence diagram is a collaboration diagram that emphasizes the temporal ordering of messages. Sequence diagrams present a scenario-oriented description of the interaction between objects in the system. Activity diagrams are essentially flowcharts showing flow of control from activity to activity.

In addition to these diagrams, UML provides component diagrams and the deployment diagrams to describe the transition from a model to the corresponding implementation. The component diagram describes the software architecture of the system in terms of components. The deployment diagram describes the hardware architecture of the system.

#### 2.3.4.1. What is the Relationship between UML and 5SL?

UML is a general-purpose language. This is the main advantage and at the same time the main drawback of UML. UML is a language which is not capable of providing features that are appropriate to express problem domain-specific situations [38]. This drawback of being a general-purpose language with lack of a precise semantics has been identified by the UML standardization groups and has led to establishing corresponding task groups and related Request For Proposals by the OMG in specific domains to overcome these shortcomings. In order to provide intuitive and expressive modeling features for a certain domain, UML has to be extended and adapted by appropriate profiles. 5SL represents such an extended simplified UML profile in the domain of DLs.

## 2.4. Construction of DLs

As the focus of this work is 5SLGen, a DL generator, it would be pertinent to look at other tools and techniques used for modeling, construction, and/or generation of DLs.

### 2.4.1. MARIAN DL Generator

The MARIAN DL generator, based on a DOM XML parser, automatically generates a set of class managers, indexing classes, an analyzer, and a user-interface from the 5SLSocieities and 5SLScenarios models of the modeled DL. These four elements help constitute the generated DL. One important feature of this generator is its use of XML namespaces and the MARIAN API. The 5SLSocieties and 5SLScenarios models use namespaces to import MARIAN types to specify properties of the many different parts of documents and metadata records. These properties include specific matching methods as well as methods for management of indexes, databases, and sets of instances of the particular class/type.

The MARIAN generator took for granted the capabilities of the MARIAN API for services such as searching and browsing. This powerful MARIAN API is a double-edged sword. It tremendously facilitates the process of DL construction, however, at the same time it enforces a tight coupling with the DL generator, making it is difficult if not impossible to incorporate other components into the generated DL.

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 scenario composition or scenario integration is needed to produce complete specifications of generic DL services.

The dependence on the MARIAN API and lack of a scenario-based approach are the principal drawbacks of the MARIAN DL generator.

### 2.4.2. Greenstone Suite of Software

Greenstone is a comprehensive software system for creating DL collections. It builds data-structures for searching and browsing from the material provided [30]. Once a collection already exists, new material can be added automatically. It has extensive support for new collections. Extensibility is achieved through software plug-ins written to accommodate documents, and metadata, in different formats. Greenstone is not based on any formal models. Greenstone expects the DL designer to provide input only in the form of changes to the configuration file rather than handcrafted models. As compared with the 5S model, the concept of digital library in Greenstone is simpler and coarser. The 5S model provides a more holistic view of the digital library as compared to Greenstone's model.

### 2.4.3. 5SGraph: A Domain-Specific Visual Modeling Tool

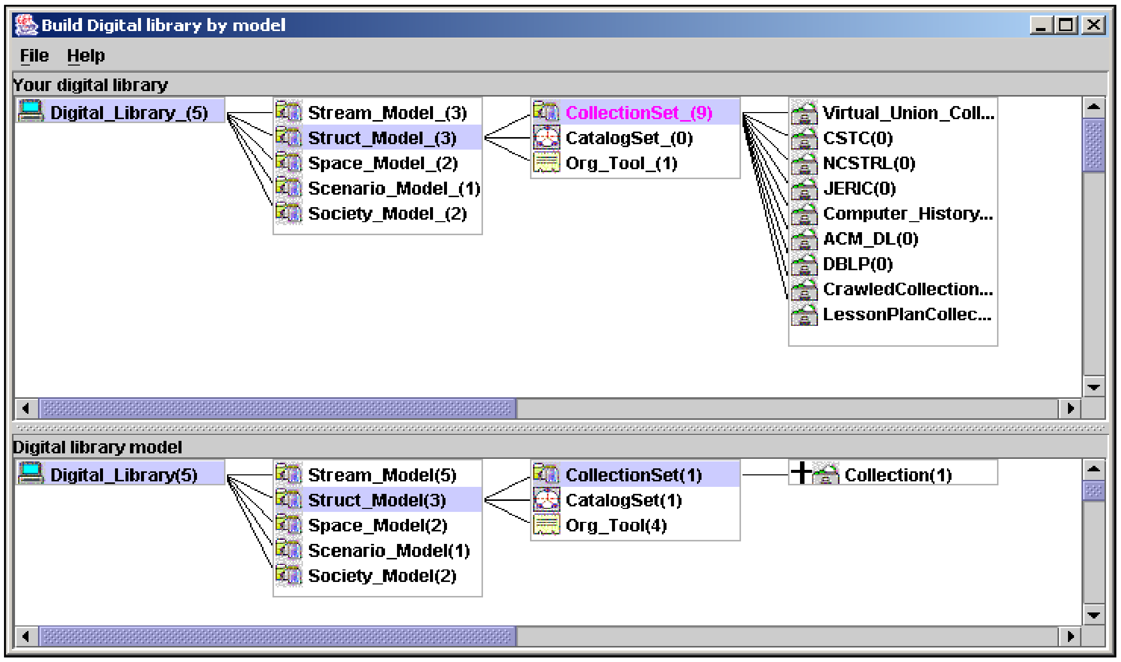


**Figure 2.4 User-Interface of 5SGraph**

5SGraph is a DL-specific visual modeling tool that helps DL designers model DLs using the 5S metamodel. 5SGraph presents the metamodel in a structured toolbox, which shows all visual DL components and the relationships among them, and provides a top-down visual building environment for designers. As illustrated by Figure 2.4 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. The upper part is the workspace in which users can create their instance models.

The visual proximity of the metamodel and instance model facilitates requirements entry and simplifies the model development process. 5SGraph outputs 5SL DL models for societies, scenarios, structures, spaces, and streams. 5SGraph maintains semantic constraints specified by the 5S metamodel and enforces these constraints over 5SL models to ensure semantic consistency and correctness. 5SGraph also is designed to accommodate and integrate several other complementary tools reflecting the interdisciplinary nature of DLs.

Ideally the 5SL models generated by 5SGraph could serve as the input to 5SLGen to generate instantiations of DLs, however 5SGraph in its current state does not support the modeling of scenarios and societies. 5SGraph represents an excellent starting point in the modeling of DLs based on the 5S formal theory. Once seamlessly integrated with 5SLGen, the family of 5S tools (5SGraph + 5SLGen) would provide a complete CASE environment for DLs. This topic is further elaborated upon in the section on future work.

### 2.4.4. CASE Tools

A Computer Aided Software Engineering (CASE) set of tools can provide automated assistance for software development. A typical CASE tool offers graphical editors to help developers model all the requirements of a software system. Most CASE tools are based on a metamodel that guides the creation of different software artifacts, as part of the software development process. The foundation provided by the metamodel allows CASE tools to validate the models created by the designers and to automate some of the transitions between software development phases, including the generation of code. As UML is the de facto industrial modeling language, nowadays most CASE tools are based on the UML metamodel. CASE tools provide support for forward and reverse engineering, and support for varying levels of abstraction (for example, from requirements to analysis to design to code). Popular commercial CASE tools include TogetherSoft's Together/J [39], Rational Rose [40], and Computer Associate's ERWin [40]. ArgoUML and Umbrello are instances of open-source CASE tools [41].

CASE tool interoperability is achieved by serializing models according to the XMI standard [42]. XMI, the OMG's XML Metadata Interchange format, is a vendor-independent format for saving and loading UML models. 5SLGen serializes 5SL models to XMI, which can be read by other CASE tools. Thus, 5SLGen interoperates with other CASE tools, harnessing the benefits offered by both domain-specific languages and CASE tools

### 2.4.5. Other Related Tools

Many tools 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. XML modeling tools help users to create XML files and allow users to load a DTD or a XML schema and create an instance that conforms to that DTD or schema. However, most of the tools only focus on one or more components of the digital library system. These systems place the onus on the digital library builders to use these tools and tie them together.

## 2.5. Scenario-Based Requirements Analysis and Design

We now focus attention on the task of DL generation starting with the scenario-based requirements analysis and design approach to software development. An introduction to scenarios was provided in 1.1.3. In this section, we focus on the role of scenarios in software design and the challenges faced in incorporating them into the software development process.

### 2.5.1. Importance of Scenarios in Software Design

In [43] Carroll argues for scenario-based design providing 5 reasons. This section will elaborate each reason in brief. Figure 2.5 taken from [43] summarizes the five issues raised in the discussion below.

1. Constructing scenarios evokes reflection in the context of design work, helping developers to reflect on the work they have already done and to coordinate further design action.

2. Scenarios are both concrete and flexible, to help developers manage changing requirements. They are concrete in the sense that they simultaneously fix an interpretation of the design situation and offer a specific solution. At the same time, scenarios are flexible, deliberately incomplete, and easily revised or elaborated.

3. Scenarios afford multiple views of an interaction, diverse kinds and amounts of detailing, helping developers manage the many consequences entailed by different given design moves. For instance, some scenarios can be developed in detail, for example the ones that

describe the core application functionality, whereas other less problematic scenarios can be merely sketched.

4. The roughness of scenarios coupled with their ability to be classified in terms of causal relations enables them to be abstracted and categorized, helping designers to recognize, capture, and reuse generalizations, and to address the challenge that technical knowledge often lags the needs of technical design.

5. Scenarios are work-oriented design objects. They describe systems in terms of the work that users will try to do when they use those systems. Scenarios promote work-oriented communication among stakeholders, keeping the designer focused on the real design situation experienced by the user.

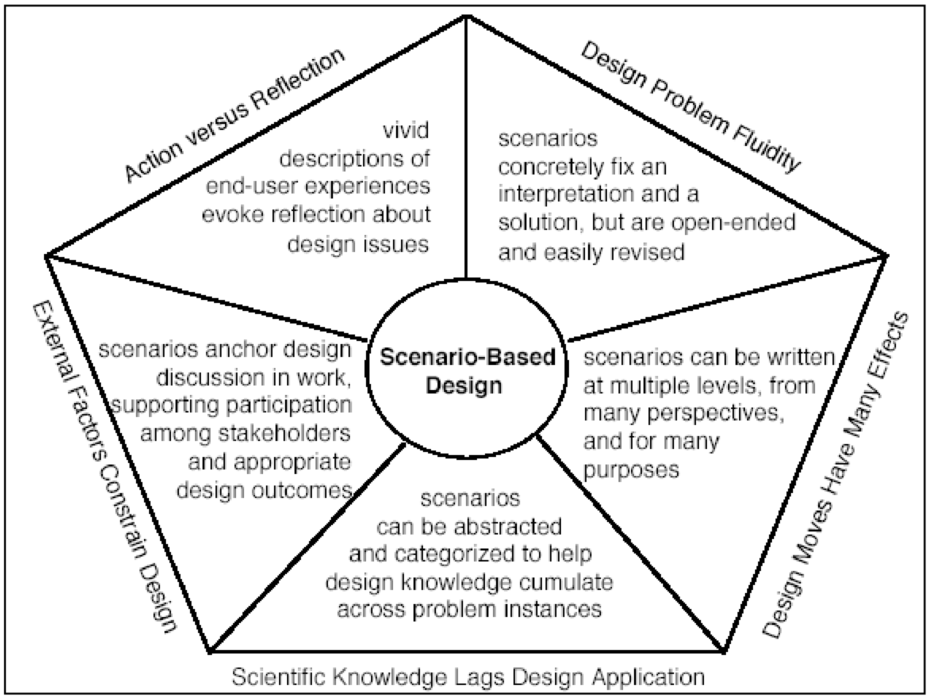

Action versus Reflection
Design Problem Fluidity
vivid descriptions of end-user experiences evoke reflection about design issues
scenarios concretely fix an interpretation and a solution, but are open-ended and easily revised
External Factors Constrain Design
Design Moves Have Many Effects
scenarios anchor design discussion in work, supporting participation among stakeholders and appropriate design outcomes
Scenario-Based Design
scenarios can be written at multiple levels, from many perspectives, and for many purposes
scenarios can be abstracted and categorized to help design knowledge cumulate across problem instances
Scientific Knowledge Lags Design Application

**Figure 2.5 Challenges and approaches in scenario-based design**

### 2.5.2. Scenario Research

Research in scenario-based requirements analysis and design has identified issues in scenario generation, in the management of scenario descriptions, in analysis and synthesis of scenarios, and in bridging the model-system gap from scenario descriptions to software designs and implementations [8]. Our focus in this study has been on the analysis and synthesis of scenario descriptions as well as bridging the gap between scenario description and software implementation. For the purpose of this study, we assume that the scenarios modeled by the DL designer accurately represent the functional requirements of the system and we focus our energies on applying different approaches for scenario-synthesis and automatic generation of code from system requirements into the 5SLGen DL generator.

#### 2.5.2.1. Need for Scenario-Synthesis

The full potential of scenarios is not always fully exploited because of the following problems: [44, 45]

1. **Large number of scenarios**. Typical object-oriented software systems are composed of very large sets of scenarios, and each component is usually involved in the execution of many different scenarios. Scenarios are highly redundant; some scenario parts are part of many scenarios, sometimes with small variations.

2. **Incompleteness of scenario models.** Scenario models are most often meant to describe typical system behavior paths at an abstract level. From this viewpoint, scenario models are unavoidably incomplete with respect to the overall set of system scenarios, with respect to the overall set of requirements, and finally they are incomplete as regards to the level of detail they give.

3. **Concurrency, interactions and dependency between scenarios.** Scenarios can be concurrent and may interact with each other. This is an aspect of object-oriented systems that makes them particularly complex and difficult to design and may lead to missing traceability between scenarios and other software artifacts.

4. **Lack of change management.** The set of scenarios that can be executed by a system or a component can change over time. The temporal nature of scenarios greatly complicates component behavior specification.

5. **Lack of Models.** Models used for capturing scenarios lack adequate expressiveness and semantic foundation.

6. **Weak visualization techniques** and **Insufficient tool support.** There is a lack of techniques and tools to visualize scenarios at varying levels of abstraction.

The first three problems can be addressed by designing an approach for synthesizing all the scenarios to create a state machine representation/Statechart of the entire system. The last three problems can be addressed by creating a toolkit for modeling of scenarios based on a formal framework.

#### 2.5.2.2. Approaches to Scenario-Synthesis

The different approaches to scenario-synthesis are summarized in this section. A comprehensive survey of all methods for scenario-synthesis is out of scope for this work. The relevant ones are listed below.

Koskimies [46] presents an algorithm, SMS (state machine synthesis), for synthesizing a Statechart from a set of scenarios. A scenario is modeled as a set of event traces. Koskimies addresses scenario-synthesis as a language inference problem and bases his algorithm on the Biermann-Krishnaswamy algorithm for learning a program from its sample traces. The main idea behind SMS is to infer a Statechart diagram able to execute all the given input traces. The SMS algorithm cannot be used within concurrent systems, and the resultant state machines present no structural information such as hierarchies. As the SMS algorithm is a backtracking algorithm, it has an exponential complexity in the worst case.

Desharnais [47] give a formal relation-based definition of scenarios and shows how different scenarios can be integrated. The view of scenarios is state-based, rather than event-based, like most of the other approaches. This approach uses Z notation to represent scenario relations and has the same weaknesses as the previous one. It supports neither hierarchy nor concurrency.

Glinz [48] introduces a Statechart-based model that allows the formal composition of all scenarios. He defines composition rules for creating an integrated, consistent model of external system behavior. He does not consider the problem of scenario-overlap. In his approach, hierarchy and concurrency are well supported. Glinz makes the simplistic assumption of representing each use case of the system by only one scenario.

Khriss [49] suggest a four-step process for synthesizing behavioral specifications from scenarios represented as UML collaboration diagrams. From a given set of collaboration diagrams, they generate the Statechart diagrams of all the objects involved. They address the issue of scenario-interleaving, concurrency, and hierarchy. Their approach is incremental and enables an iterative process for dynamic modeling. Their scenario-synthesis algorithms are implemented in a tool called SUIP, which can generate prototype user-interfaces [7] based on the synthesized scenarios.

We have explored the use of the SUIP tool as a component of 5SLGen in [50-52]. In the first project [52], we implemented a transformation from 5SLScenarios and 5SLSocieties to the textual representation of scenarios and classes expected by the SUIP tool. The SUIP tool synthesized the scenarios and output Statecharts for all the objects in the system in addition to a user-interface prototype that served to validate the scenarios modeled. In the second project [51], the SUIP tool was extended to generate a device independent user-interface prototype for services offered by a digital library from the synthesized scenarios.

The SUIP tool has its limitations. The tool takes UML class and collaboration diagrams as input in a proprietary textual format. The grammars to represent these UML collaboration and class diagrams are poorly explained. The synthesized Statecharts generated are not human readable. It is difficult to understand what needs to be modeled to generate the desired output. Even though this tool seemingly did everything we needed in terms of scenario-synthesis, it was not possible to use it to accomplish our objectives.

Whittle [53] presents an algorithm for generating synthesized Statecharts from scenarios. The approach taken identifies and resolves conflicts in the scenarios. The issues of scenario-overlap and hierarchy also are addressed. The algorithm expects intervention by the designers once the Statecharts are generated.

Simona [54] proposes a method of synthesizing Statecharts from multiple scenarios and describes a set of rules for performing the synthesis. Her method addresses the issues of scenario-interleaving, scenario-dependency, scenario-overlap, and scenario-concurrency. Simona's method bridges the gap between requirements and specification. We have implemented Simona's method in this study as part of the implementation of 5SLGen. Details of the implementation are provided in section 5.2.3.1.

## 2.6. Underlying Technology and Standards

In this section, we provide a brief overview of the important underlying standards that impact this study.

**HTTP** is a stateless message-response protocol used in the World Wide Web. It defines what actions web servers and browsers should take in response to various commands.

**XML** (eXtensible Markup Language) is a markup language for documents containing structured information. It differs from HTML (HyperText Markup Language), as it does not define a tag set and semantics. XML is a pared-down version of SGML, designed especially for Web documents. It allows designers to create their own customized tags, enabling the definition, transmission, validation, and interpretation of data between applications.

**XSD** (XML Schema Definition) provides a way to describe and validate data in an XML environment. A schema is a model for describing the structure of information. XSD is a recommendation of the W3C [55-58].

**XSL** (Extensible Style Language) and **XSLT** (Extensible Style Language Transformation). XSL is a specification for separating style from content while creating HTML or XML pages, and XSLT is the language used in XSL stylesheets to transform XML documents [59].

**JDOM** is a Java-based "document object model" for XML files. JDOM serves the same purpose as DOM, but is easier to use [60].

**SAX** is an application-programming interface for processing XML. SAX is an event-driven XML parser [61].

**OAI** (Open Archives Initiative). The Open Archives Initiative develops and promotes interoperability standards that aim to facilitate the efficient dissemination of content. The current OAI technical infrastructure is specified by the Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH). OAI-PMH is simply an interface that a networked web server employs to make metadata-describing objects housed at that server available to external applications that wish to collect this metadata. Such a web server that exposes metadata through the OAI-PMH is referred to as a Data Provider. A service provider uses the OAI-PMH protocol to gather metadata from data providers and then uses the metadata for building value-added services.

**OAI-PMH** (Open Archives Initiative Protocol for Metadata Harvesting). The OAI-PMH is a client-server protocol layered over HTTP, using CGI-encoded parameters in requests and XML-encoded data in responses. The aim of the protocol is to support the batch transfer of metadata from a server (data provider) to a client (service provider) using incremental updates whenever a transfer is initiated. This process of obtaining all the (new) metadata from a server, instead of only that which satisfies a search query, is commonly known as harvesting [28]. The OAI-PMH is made up of six requests and associated responses, three of which are administrative while the other 3 are for data transfer. These requests, and the semantics of their responses, are as follows:

1. Identify – general information about the archive, administrator, and policies.

2. ListMetadataFormats – a list of all the metadata formats supported by the archive as well their XML namespaces and schema locations.

3. ListSets – a list of all the subsections of the archive for selective harvesting.

4. ListIdentifiers – a list of identifiers for all records, corresponding to the required metadata format parameter and optional date range and/or set parameters.

5. GetRecord – a single record, specified by its unique identifier and metadata format.

6. ListRecords – a list of records in the specified metadata format, corresponding to optional date range and/or set parameters.

**OA** (Open Archive). An Open Archive (OA) is any archive that implements the OAI-PMH, thus allowing remote archives to access its metadata using an "open" standard.

**XMI** (XML format for Metadata Interchange). This specification is based on the W3C's Extensible Markup Language (XML). XML Metadata Interchange (XMI) is a standard for representing object-oriented information using XML. XMI provides a basis for the development of XML documents and DTDs. The XML Metadata Interchange format (XMI) uses XML to address the need for a textual representation of UML specifications. The main purpose of XMI is to enable easy interchange of metadata between different modeling tools, based on UML and other metadata repositories. One can use XMI to both serialize objects in documents, and to generate schemas from models. Most modeling tools export models to XMI documents following the UML-DTD. XMI, together with UML helps form the core of the OMG repository architecture that integrates object-oriented modeling and design tools [42].

## 3. The 5S Approach to the Generation of DLs

5SLGen is a DL generator that helps generate implementations of DL services from the 5SL model instances of DL societies and scenarios. Before delving into a detailed treatment of 5SLGen, we briefly elaborate on the 5S theory and present all the 5SL models with emphasis on the scenarios and societies models. This is followed by an overview of the modeling and generation process of DLs based on the 5S approach.

### 3.1. 5S Metamodel for DLs

The 5S theory explains why the presented set of 5S modeling concepts is necessary to represent the targeted modeling scope. The 5S theory provides a justification for the modeling concepts of the 5S metamodel. The precise relationship, and the steps involved in a transformation between a theory and its corresponding metamodel, is a research problem tackled in [62]. The 5S theory helps define a vocabulary for all the elements in the domain of DLs and helps others understand the problem by using the same language. It helps us to manage complexity by raising the level of abstraction at which we think and design. A complete description of all the formal constructs of 5S has been presented in [15].

The 5S metamodel provides streams, structures, spaces, scenarios and societies as the fundamental entities for modeling DLs. Streams are sequences of arbitrary items used to describe static and dynamic content. Structures can be viewed as labeled directed graphs, which impose organization. Spaces are sets with operations on those sets that obey certain constraints. Scenarios consist of sequences of events or actions that modify states of a computation in order to accomplish a functional requirement. Societies are sets of entities and activities and the relationships between and among them. Figure 3.1 adapted from [15] provides a diagrammatic overview of all the formalisms that constitute the 5S theory.

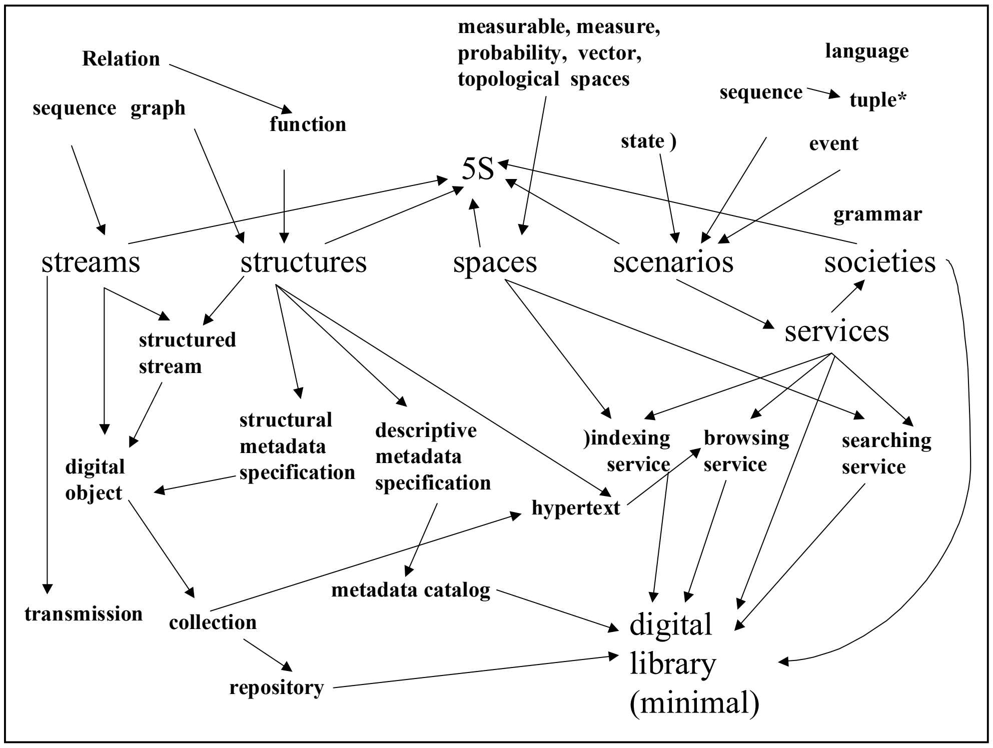


**Figure 3.1 Overview of 5S and DL formal definitions and compositions**

The formalisms of 5S help define higher-level DL concepts such as digital object, structural and descriptive metadata, collection, catalog, repository, hypertext, searching service, browsing service, etc. As illustrated in Figure 3.1, a structured stream consists of a number of digital objects. Structures provide an organization to the DL by way of structural and descriptive metadata specifications. A number of digital objects that adhere to the structural metadata specification constitute a collection. A collection also is described by a descriptive metadata specification, which serves as the metadata catalog for the collection. Other higher-level DL concepts are defined in a similar manner.

## 3.2. 5SL Models for DLs

5SL models are represented in XML with each model having an XML schema that defines the structure of the model. In the following discussion the terms 5SLSocieties and 5SLSocieties model as well as 5SLScenarios and 5SLScenarios model are synonyms.

### 3.2.1. Stream Model

The stream model specifies the kinds and formats of multimedia content supported by the DL. The model encodes information about the type, encoding, and format of the stream. This stream information is based on the WWW MIME types to ensure compatibility with current standards. Example 3.1 shows a part of the instance of the 5SL streams model for the NDLTD Union Catalog DL.

```
<streams>
  <text name='ETDText'>
    <content-type>text/xml</content-type>
    <charset>UTF-8</charset>
    <content-type>application/pdf</content-type>
    <lang>ENG</lang>
  </text>
. . .
</streams>
```

**Example 3.1 Partial 5SL stream model instance for the NDLTD Union Catalog DL**

### 3.2.2. Structures Model

The structural model is composed of the structures that impose organization on the DL. The structural model describes the internal structure of digital objects (documents), metadata standards, properties of collections and catalogs, knowledge organization tools, databases, data-structures, and all IR constructs. The structural models specify ways in which parts of a whole are arranged and organized. For instance, documents are defined by imposing structures over sets of streams or by using the structure to provide some organization among them. In other words, a document is seen as a structural composition of streams. Example 3.2 shows a part of the instance of the 5SL structures model for a document in the Union Catalog DL that uses the Dublin Core XML schema for defining the structure of a Dublin Core metadata record [63].

```
<structures>
    <document>
        <metadata>
        <oai_dc>
            <title>History of Virginia</title>
            <author>Rohit Kelapure</author>
        ……….
        </oai_dc>
    </metadata>
    </document>
</structures>
```

**Example 3.2 Partial 5SL structures model instance for a document**

### 3.2.3. Spaces Model

The spatial model captures the logical representations and operations of the DL IR constructs and metric spaces. This model gives details of the underlying DL retrieval models. It also describes indexes for collections and catalogs (each of which defines a sample space or a multidimensional vector space, depending on the retrieval model), and describes the user-interface appearance (i.e., sets of metric spaces) and behavior.

### 3.2.4. Societies Model

A society describes a set of entities (human and computer) and relationships among them. Different entities can be aggregated into a set of communities that share common characteristics and behavior. Societies are used to specify and construct the static aspects of the system. They can be viewed as representing the skeleton and scaffolding of the DL. In the societal model, we identify the different entities that interact within the DL environment and model their characteristics, functionalities, and semantic relationships. In 5SL, each entity is referred to as a Service Manager (SM). An SM serves as the binding point for societal relationships, scenario interactions, spatial visualizations, and the streams of a DL. Each Service Manager is modeled based on the classical object-oriented paradigm.

Classes are the most important building block of any object-oriented system. Classes can represent software things, hardware things, and even things that are purely at a conceptual level. An instance of a class is referred to as an object. Classes describe sets of objects that share the same attributes, operations, relationships, and semantics. A class is an abstraction of the entities in a problem domain and thus provides the ideal model for an SM. An SM can be thought of as a class in the classical OOP paradigm. The 5SLSocieties model captures all the classes of a DL and their relationships. The subsequent paragraphs explain the 5SLSocieties model in detail.

#### 3.2.4.1. 5SLSocieties Model: Terms and Concepts

Large and complex systems involve a large number of SMs collaborating in different ways. When modeling 5SLSocieities we model both the SMs and the relationships in which they participate. Each SM is described by its name, attributes operations, type, visibility, and relationships. We model three kinds of relationships among SMs: associations, dependencies, and generalizations. Figure 3.2 shows the 5 fundamental modeling constructs of the 5SLSocieties model, viz., attributes, operations, associations, generalizations, and dependencies – in a tree structure. Some of the modeling elements are modeled as attributes and some as elements in XML. To ensure clarity of view the attributes of the XML elements have not been shown. For the complete 5SLSocieties schema please refer to appendix A.1. In this section, a brief description of each modeling element is followed by the corresponding 5SLSocieties instance for the CITIDEL relevance feedback search service. Elaboration of the 5SLSocieties model will provide the context for the auto-generation of code from the 5SLSocieties model which is detailed in section 5.2.

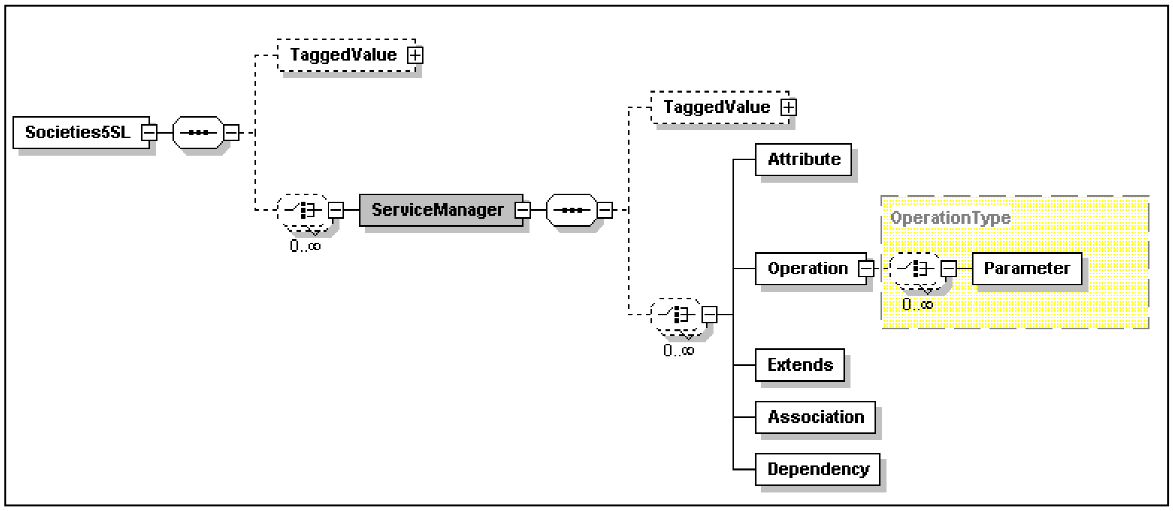


**Figure 3.2 The 5SLSocieties schema tree view**

#### 3.2.4.1.1 Name

The name of the SM serves to distinguish it from other SMs. Example 3.3 shows the 5SLSocieties instance for the name, type, and visibility of the SM.

```
<ServiceManager NAME="ODLSearchImpl" VISIBILITY="public" TYPE="class">
```

**Example 3.3 5SLSocieties instance showing name, visibility, and type of SM**

#### 3.2.4.1.2 Attributes

An attribute represents a property of the SM that is shared by all instances of the SM. An attribute is an abstraction of the kind of data or state an instance of the SM might encompass. Each attribute has a name and a type. The type could range from a primitive data type such as integer, character, string, and enumeration to other types such as SM instances. In addition to these, the DL designer also can specify the visibility, class-scope, multiplicity, an initial value, and constraints for each attribute.

The visibility of an attribute specifies whether it can be used by other SMs. The DL designer can specify three levels of visibility.

- public: An outside SM with visibility to the given SM can use the attribute.
- protected: Any descendant of the SM can use the attribute.
- private: Only the SM can use the attribute.

When an attribute is provided with a class-scope, all the instances of the SM will have the same attribute value, whereas if an attribute does not have a class-scope a new copy of the attribute is created for every instance of the SM.

Multiplicity is a specification of the range of allowable cardinalities an entity may assume. [0..1](zero or one), [*](zero or many), [1..n](one or many), [1..4](one to four) are examples of the multiplicities that the attribute may assume.

A constraint places certain restrictions on the value assumed by the attribute. Three possible constraints can be placed on an attribute:

- changeable: There are no restrictions on modifying the attribute's value.
- addOnly: For attributes with a multiplicity greater than one, additional values might be added, but once created, a value may not be changed.
- frozen: The attributes value may not be changed after initialization.

Example 3.4 shows the 5SLSocieties instance for the DEBUG attribute of an SM.

```
<Attribute   VISIBILITY="private"   TYPE="boolean"   NAME="DEBUG"   CLASS-
SCOPE="true"  CONSTRAINT="frozen" INITVAL="true"/>
```

**Example 3.4 5SLSocieties instance for an attribute**

#### 3.2.4.1.3 Operations

An operation is an abstraction of something that you can do to an instance of the SM. Each operation has a name, and a return-type. The range of values assumed by the return-type could vary from primitive datatypes to other SM instances. The DL designer also can specify the visibility, scope, parameter-list, concurrency semantics, and exceptions thrown for the operation. The visibility and the class-scope of an operation are specified in the same manner as those for attributes as explained above.

The parameter-list consists of a list of parameters, with each parameter having a name, type, and an optional default value. The concurrency semantics of an operation are of four types.

- isQuery: Execution of the operation leaves the state of the system unchanged.
- sequential: The SM instance can have only one thread of execution.
- guarded: The semantics and the integrity of the instance is guaranteed in the presence of multiple flows by sequentializing all calls to all of the instance's operations.concurrent: The semantics and the integrity of the object are guaranteed in the presence of multiple flows of control by treating the operation as atomic.

Example 3.5 shows the 5SLSocieties instance for the searchQuery operation of the SM which takes an encoded user query as an input parameter and returns a void data type.

```
<Operation VISIBILITY="public" NAME="searchQuery" RETURN="void">
   <Parameter TYPE="String" NAME="encUserQueryStr"/>
</Operation>
```

**Example 3.5 5SLSocieties instance for an operation**

#### 3.2.4.1.4 Type

An SM can be of three types viz., a normal, abstract or an interface.

- An SM of type class has both attributes (properties) and behavior. This is the default interpretation of an SM unless specified otherwise.
- An SM of type interface defines a collection of operations that specify some aspect of its behavior. An interface serves as a functional specification or a contract that can be implemented by other SMs.

An SM of type abstract cannot have any direct instances. It is usually used to model SMs at the top of a relationship hierarchy. Example 3.3 provides a sample encoding of SM type

### 3.2.4.1.5 Visibility

The visibility of an SM specifies whether it can be used by or related with other SM in the 5SLSocieties model. There are two levels of visibility:

- public: Any outside SM can use and associate with a public SM.
- private: Only the SM defined within the same model instance can use and associate with a private SM.

Example 3.3 provides a sample encoding of SM visibility.

### 3.2.4.1.6 Dependencies

A dependency is a relationship that states that a change in specification of one thing may affect another thing that is dependent on it. It is also referred to as a "using" relationship. While modeling societies, we use dependencies to model our dependence on other components and code libraries. Each dependency has a name, a peer, and a dependency kind. The peer signifies the SM dependent on, and the kind specifies the type of dependency, which can be of two types

- use: Signifies dependence between an SM and code library and other SMs.
- implements: Signifies dependence between an SM and an interface SM.

Example 3.6 presents the 5SLSocieties instance representing a dependency of the UIMFSM SM on the ODLSearch SM.

```
<Dependency PEER="ODLSearchImpl.*" DEPKIND="use"/>
```

**Example 3.6 5SLSocieties instance for a dependency**

### 3.2.4.1.7 Associations

An association is a structural relationship that specifies how SMs are connected to one another. Given an association connecting two SMs, one can navigate from an instance of one SM to an instance of the other SM and vice versa. When an SM participates in an association, it has a specific role that it plays in the relationship. The presence of a rolename implies navigability of the association. Similarly an absence indicates that the association is non-navigable. Roles are always mentioned with respect to the SM at the other end of the association. An association also has a multiplicity that defines the number of instances of one SM that can relate to one instance of the associated SM. When an association's multiplicity is one-to-many, there arises a need for a distinct identifier to qualify the instances on the many side of the relationship. Such an identifier is referred to as a qualifier for the association. A DL designer also can specify a class-scope, constraint, and the ordering of instances in an association. The modeling constructs for class-scope, multiplicity, visibility, and constraints are similar to those for attributes as explained above. Example 3.7 provides the 5SLSocieties instance representing the association between the uimFsmClone SM and the ODLUtilties SM being modeled. The UIMFSM SM, which plays the role of a uimFsmClone in the association, is modeled with a class-level scope and singular multiplicity.

```
<Association MULTIPLICITY="1" ROLENAME="uimFsmClone" PEER="UIMFSM"
CLASS-SCOPE="true"/>
```

**Example 3.7 5SLSocieties instance for an association**

#### 3.2.4.1.8 Generalizations

A generalization represents a relationship between a general thing (parent SM) and a more specific kind of thing (child SM). It is also referred to as an "is-a" relationship. In a generalization the child SM (subclass) inherits attributes and operations from the parent SM (superclass). The parent SM is more general than the child SM. When modeling a generalization the DL designer needs to specify the parent and child SMs involved. Example 3.8 shows the 5SLSocieties instance for a generalization relationship between the RelevenanceFeedbackSearchImpl SM (child) and the ODLSearchImpl SM (parent).

```
<EXTENDS FROM "ODLSearchImpl">
```

**Example 3.8 5SLSocieties instance for a generalization**

### 3.2.5. Scenarios Model

The importance and the role of scenarios in software design has already been reflected upon in section 2.5. Scenarios are used to specify and construct the dynamic aspects of the system. The purpose of the scenarios model is to describe the behavior of the DL services. In a sense, they represent the body of the DL, specifying the interaction between various parts and making sense of the whole. A scenario is modeled as a sequence of interactions among the SMs. An interaction comprises of a set of messages being exchanged among a set of objects within a context to accomplish a specific purpose [37].

A UML sequence diagram can be used to graphically represent the temporal sequence of events of a scenario. The time is represented by the vertical axis of the sequence diagram (normally time proceeds downward) and the participating objects by the horizontal dimension. The lifeline shows the existence of an object and is represented by a dashed line [64]. If an object is destroyed or terminated during this period, the lifeline ends with the object termination symbol (X). The interaction is realized using messages sent from the objects playing the sender role to the objects playing the receiver role. The communication between the objects is denoted by an arrow from the sender's to the receiver's lifeline that represents a message. The message arrow is labeled by the operation to be invoked. The message label is return-value: = operation-name (argument-list). A message that initiates the creation of a new object is represented by an arrow pointing with the arrowhead to the object symbol. For a detailed treatment of sequence diagrams the interested reader is referred to [19, 37].

The 5SLScenarios model provides an XML-based syntax to serialize the temporal ordering of messages in a sequence diagram. The messages sent between the sender and the receiver in the sequence diagram map to the operations defined by the SM. Figure 3.3 shows the hierarchy of the 5SLScenarios XML schema used to define the scenarios. For the complete 5SLScenarios schema please see appendix A.2.

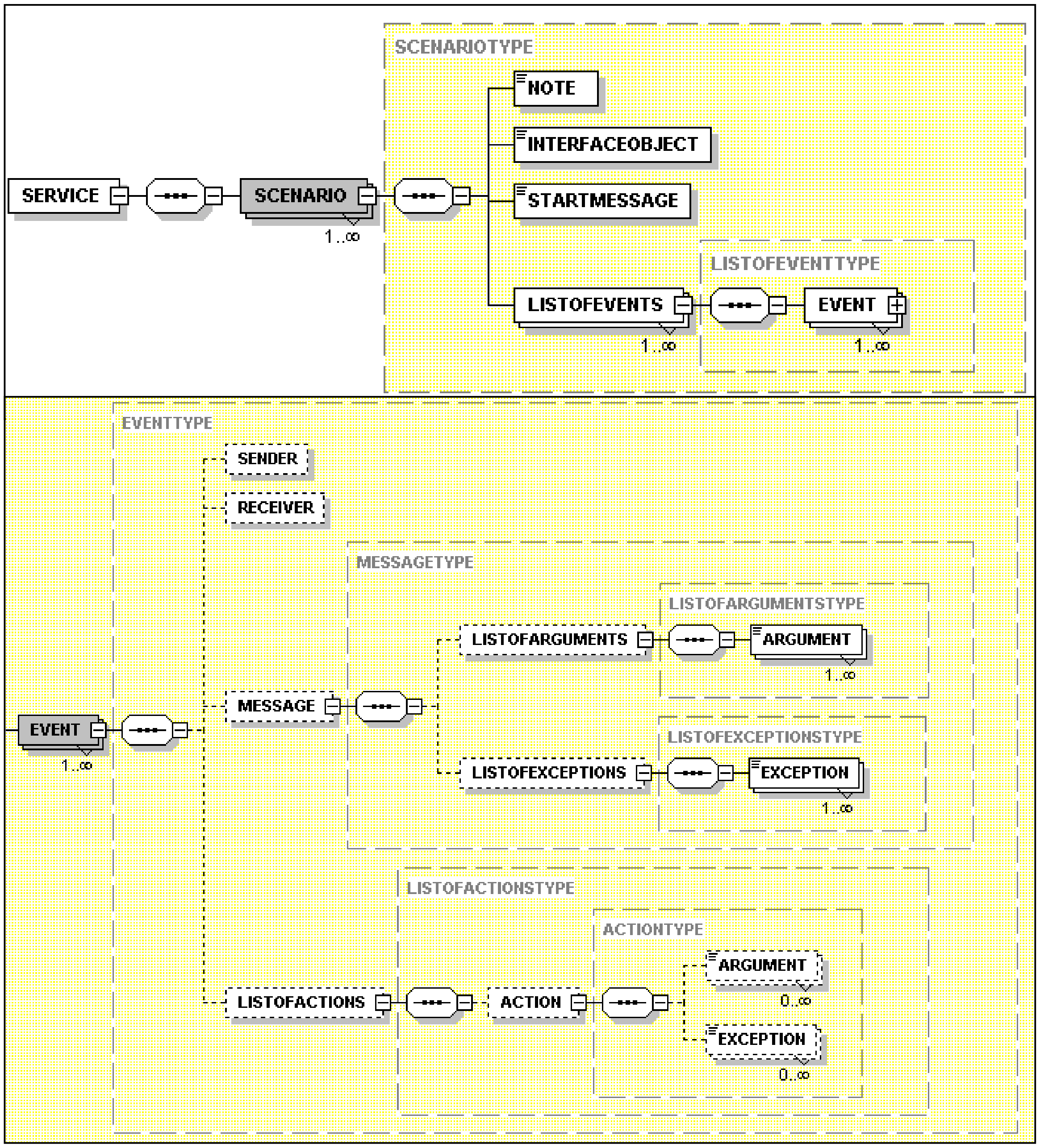


**Figure 3.3 The 5SLScenarios schema tree view**

### 3.2.5.1. 5SLScenarios Model: Terms and Concepts

In 5S, a DL service is represented by multiple correlated scenarios. Therefore, the 5SLScenarios model contains an XML serialization of all the sequence diagrams related to the service. As mentioned before a multitude of scenarios is required to model a service completely. We therefore require that for a complete specification of the DL, the DL designer must model a minimum of one primary scenario for each service. In the sections below we detail each modeling element of the 5SLScenarios model and provide an instance for the same. Elaboration of the 5SLScenarios model will provide adequate background for understanding the scenario-synthesis algorithm and auto-generation of code from scenarios, detailed in section 5.2.3.

#### 3.2.5.1.1 Service

As illustrated in Figure 3.3, each service has a name and is composed of one-to-many scenarios. Only one service can be modeled in a 5SLSocieties instance at a time. Example 3.9 shows a part of the 5SLScenarios instance for the name of a service.

```
<SERVICE  NAME="RelevanceFeedbackSearchService">
```

**Example 3.9 5SLScenarios instance for a service**

#### 3.2.5.1.2 Scenario

Each scenario consists of a note, an interface object, a start message, and a list of events (see Figure 3.3). A note serves the purpose of documenting the scenario. The interface object specifies the SM responsible for receiving user input events and presenting output to the user. The start message specifies the state of the interface SM before the scenario begins. The significance of this element will be elaborated upon later while explaining the scenario-synthesis algorithm. Example 3.10 shows the 5SLScenarios instance for the primary scenario of the CITIDEL relevance feedback search service.

```
<SCENARIO SC_NUMBER="1">
        <NOTE> This is the simple scenario for the relevance feedback search service wherein
                1 User issues a basic search for certain documents.
                 2 After seeing the results of the search the user does a relevance feedback
                search with the selected documents.
                3 After seeing the results of the relevance feedback search the user then
                retrieves a document of his choice.
                Any sequence of the above 3 steps can be repeated after the first step

        </NOTE>
        <INTERFACEOBJECT>UIMFSM</INTERFACEOBJECT>
        <STARTMESSAGE>MainMenu</STARTMESSAGE>
        <LISTOFEVENTS>
            ……..
</SCENARIO >
```

**Example 3.10 5SLScenarios instance for a scenario**

#### 3.2.5.1.3 Event

Each event (see Figure 3.3) is described by a sender, a receiver, the message between them, and the list of actions taken on receipt of the message. The sender sends the message to the receiver, which on the receipt of the message the receiver takes the appropriate actions. For instance, in Example 3.11 the user sends a GET HTTP event to the interface SM. On receiving the doGet event the SM in this case, the UIMFSM, calls on the parseRequest action to ascertain the nature of the request.

```
<EVENT SEQNO="1">
    <SENDER>User</SENDER>
    <RECEIVER>UIMFSM</RECEIVER>
    <MESSAGE NAME="doGet">
        <LISTOFARGUMENTS>
            <ARGUMENT>HttpServletRequest(request)</ARGUMENT>
            <ARGUMENT>HttpServletResponse(response)</ARGUMENT>
        </LISTOFARGUMENTS>
        <LISTOFEXCEPTIONS>
            <EXCEPTION>ServletException</EXCEPTION>
            <EXCEPTION>IOException</EXCEPTION>
        </LISTOFEXCEPTIONS>
    </MESSAGE>
    <LISTOFACTIONS>
        <ACTION NAME="parseRequest">
            <ARGUMENT>HttpServletRequest(request)</ARGUMENT>
            <ARGUMENT>HttpServletResponse(response)</ARGUMENT>
        </ACTION>
    </LISTOFACTIONS>
</EVENT>
```

**Example 3.11 5SLScenarios instance for an event**

#### 3.2.5.1.4 Message

As illustrated in Figure 3.3, a message has a name, a method, and a list of arguments. The method corresponds to the name of the operation defined in the 5SLSocieties model for the SM. The name signifies the state of the SM after sending the message. Each argument in the list of arguments has a name and type. Consider the message searchQuery sent from the UIMFSM SM to the ODLSearch SM in Example 3.12. Execution of the searchQuery method, defined in the 5SLSocieties instance for the ODLSearch SM (see Example 3.5), results in the sender, UIMFSM SM, going to the search state.

```
<EVENT SEQNO="3">
    <SENDER>UIMFSM</SENDER>
    <RECEIVER>ODLSearch</RECEIVER>
    <MESSAGE NAME="search" METHOD="searchQuery">
        <LISTOFARGUMENTS>
            <ARGUMENT>String(searchStr)</ARGUMENT>
        </LISTOFARGUMENTS>
    </MESSAGE>
</EVENT>
```

**Example 3.12 5SLScenarios instance for a message**

#### 3.2.5.1.5 Action

As illustrated in Figure 3.3, an action is comprised of a name, a list of arguments, and any exceptions thrown. An action is taken by a sender on receiving an event. For the 5SLScenarios instance encoding of an action please refer to Example 3.11. An action is always taken passively in response to an event whereas a message is sent actively by an SM.

## 3.3. Modeling and Generation of DLs

In this section, we present an overview of the entire modeling and generation process and provide the big picture (please look at Figure 3.4) of the complete process of building DLs using the 5S family of tools. This process of building DLs is then compared with different software development paradigms. All the previous sections have laid the groundwork for this section. Please note the role played by 5SLGen, the focus of this work.

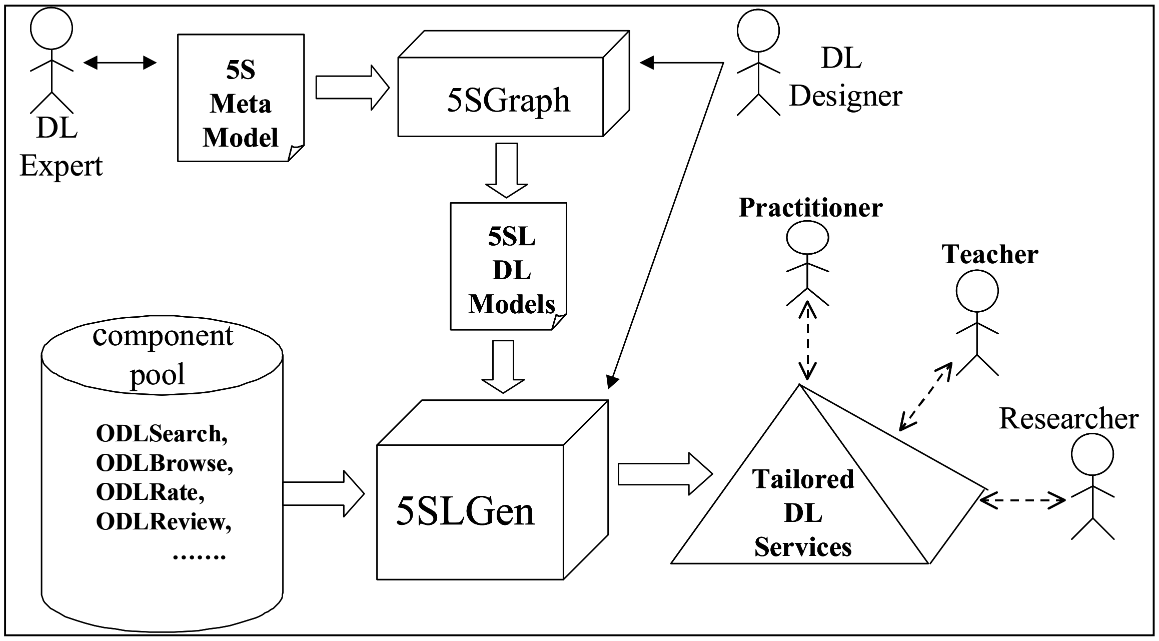


**Figure 3.4 Overview of the architecture for DL modeling and generation**

### 3.3.1. Process of Building a DL

We adopt an approach shown to be highly effective in other areas of computing: develop powerful theories and metamodels (i.e., 5S); use them to develop formal specifications (i.e., 5SL), and generate tailored systems from those specifications (using 5SLGen).
The process of building a DL with 5S is a 4-step process.

**Specifying the Metamodel:** First the DL expert captures all "societal" conditions and capabilities to which the DL must conform. 5S provides a common ground of terminology and a domain model that is close to the DL world and furnishes precisely defined concepts so that the resulting description is understandable by end users. The role of the DL expert is to design a metamodel for DLs, which will be used for modeling the DL. We already have defined the 5S metamodel for DLs. The DL expert may either create a new metamodel, use the 5S metamodel, or extend the 5S metamodel for digital libraries.

**Capture Requirements as 5SL Models**: The 5SGraph modeling tool processes the metamodel, allowing the digital librarian (or the DL designer) to visualize the components of the metamodel. The DL designer must be aware of the functional requirements — what services a community needs and what form of interaction these services should have with the users of the DL – say practitioners, teachers, and researchers. The designer uses those visualized graphical components of the metamodel to put together the final model of his own digital library. The DL requirements acquired with 5SGraph are captured with user-level 5SL models.

**Generation and Modification of 5SFramework classes:** The 5SL DL models are input to the 5SLGen DL generator, along with a pool of reusable DL components (a component implements a specific service such as search, browse, rate, etc.), to generate classes for the 5SFramework. The 5SFramework is a reusable design, expressed as a set of classes, which implements the modeled DL and supports reuse at a larger granularity than classes. This transformation from 5SL models to object-oriented classes involves scenario-analysis, scenario-synthesis, component pool utilization, and mapping of 5SL constructs to programming language concepts. The 5SLSocieties and 5SLScenarios models are instances of the 5SLSocieties and 5SLScenarios schema defined in sections 0 and 0.

**Deployment of the DL services:** The DL designer finishes the modeling and generation process by modifying the generated 5SFramework classes and coupling them with the user-interface provided by the web developer. Once a DL is generated, it is tested to complete the process of building a DL. Further customization entails refinement of 5SL models, regeneration, and modification of 5SFramework classes, followed by user-interface coupling.

### 3.3.2. 5S Approach VS. Software Development Paradigms

In this section, we will compare the 5S approach to the standard software development paradigms for systems development.

A Software Development Paradigm outlines a development strategy typically consisting of five activities: Requirements Analysis, Design, Implementation, Integration & Testing, and Maintenance [65]. Different paradigms decompose these activities in different ways based on the timing of an individual activity and its outcome. The first of these models, the Waterfall model, places the five development processes in a linear sequence. The disadvantages of this approach are well documented [66]. Subsequent approaches followed a model of prototyping and successively refining the implementation based on user feedback. This approach is referred to as the evolutionary-approach to software development. Examples of this approach include the Spiral model [67], Rapid Application Development [68], and the more recent Extreme Programming [69].

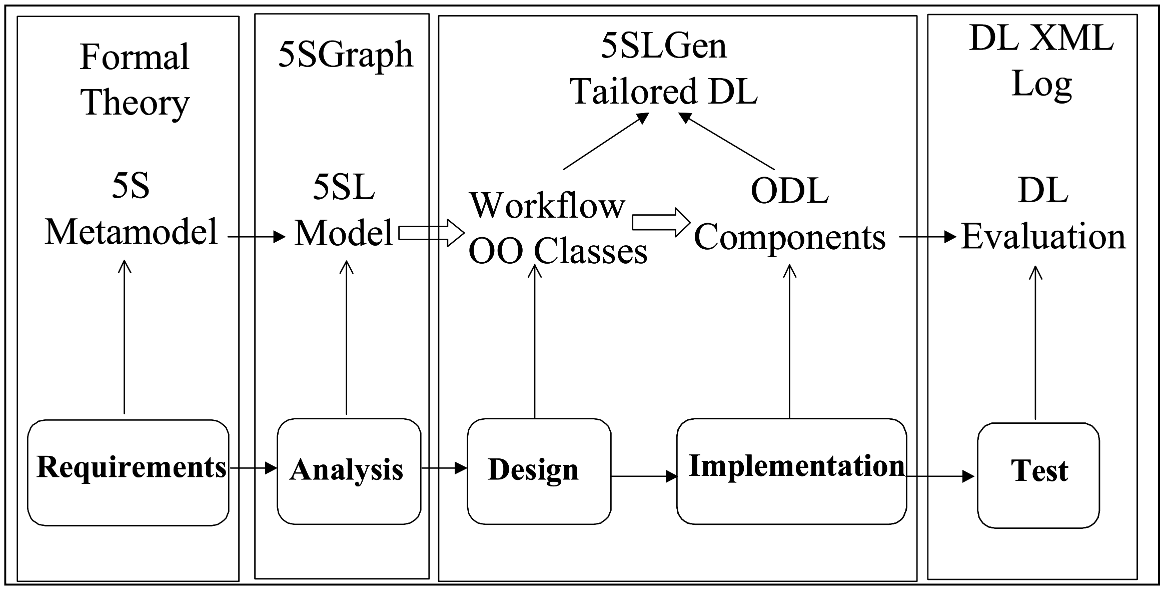


**Figure 3.5 5S approach vs. software development paradigms**

All the six activities of a software development paradigm can be mapped onto the 5S approach for building DLs. (Please look at Figure 3.5.) The 5S approach, particularly the 5SLGen tool, supports the evolutionary-prototyping approach by automatically generating object-oriented classes from the 5SL models that can be successively refined after user feedback.

During the Requirements Analysis phase, the DL designer captures the functional requirements of the DL in the form of 5SL models of societies, scenarios, spaces, structures, and streams.

The focus of the Design phase is to produce models that are closer to the implementation and the target architecture, but still preserve the structure of the system as captured during requirements analysis. 5SLGen produces design models from 5SL models by transforming higher-level 5SL concepts into object-oriented classes and workflows.

Finally, in the Implementation phase, 5SLGen uses the produced design models to generate running DL services by integrating components from pools, mapping models to specific target platforms and languages (e.g., Java, Perl), and compiling and producing new components and subsystems.

In the Integration and Testing phase, the generated DL services are integrated with the user-interface and tested for functionality. One possible way to evaluate and test a live system in a production mode is through log analysis. Accordingly, our team [70] has proposed an XML-based log standard for digital library log analysis based on 5S.

The reuse of components from the component pool as well as the 5SFramework architecture of the generated DL facilitates all activities in the Maintenance phase.

# 4. The 5SLGen Digital Library Generator: Design and Architecture

In this chapter, we describe the design and architecture of the 5SLGen DL generator. The emphasis here is on "who does what" rather than "how is it done". We first start with the design of 5SLGen. Thereafter, we elaborate on the input, output, and the architecture of 5SLGen. At the end of this section, the reader will have a grasp of the functionality and architecture of the 5SLGen DL generator.

The 5SLSocieties and 5SLScenarios models mentioned hereafter are instances of the 5SLSocieties and 5SLScenarios schemas (see appendix A). These models constitute the layer M0 of the OMG metamodel architecture and represent instantiations of the 5SLSocieties and 5SLScenarios layer M1 models.

## 4.1. 5SLGen: Design

We envision the services exposed by a DL to be either of the composite or elementary type. We have already explained the categorization of DL services in section 2.1.2.2. To summarize the discussion, elementary services provide the basic infrastructure for the DL whereas composite services fulfill the information seeking needs of DL users. Composite services are composed of other services (elementary or composed) by reusing or extending them.

The problem of composability of services has been studied recently, mainly in the web community [71-73]. However, DL services are restricted to certain specific types with constrained inputs and outputs, therefore making the problem more manageable and possible to be treated with domain-specific techniques. Figure 4.1 shows a model for the services exposed by the tailored DL produced by 5SLGen. The model defines composite services recursively as an aggregation of other services, composite or elementary.

The application logic of a composite service is described by a workflow, i.e., a combination of control and data flows that mirror the behavior defined in the scenarios for the service, including invocations of other services. Statecharts [48] and Petri-nets [74] are possible notations for formally representing workflows. In our implementations, we chose Statecharts to represent the workflow of a service. Statecharts, introduced by Harel [75], represent a compact way of describing the dynamic aspects of the system. Statecharts are an extension of finite state automata to include hierarchical decomposition, concurrency, and structured transitions. At the bottom of the decomposition, all Statecharts are ordinary finite state automata/state-event diagrams. Statecharts connect events and states. When an event is received, the system leaves its current state, initiates the actions specified for the transition, and enters a new state. The next state depends on the current state as well as the event.

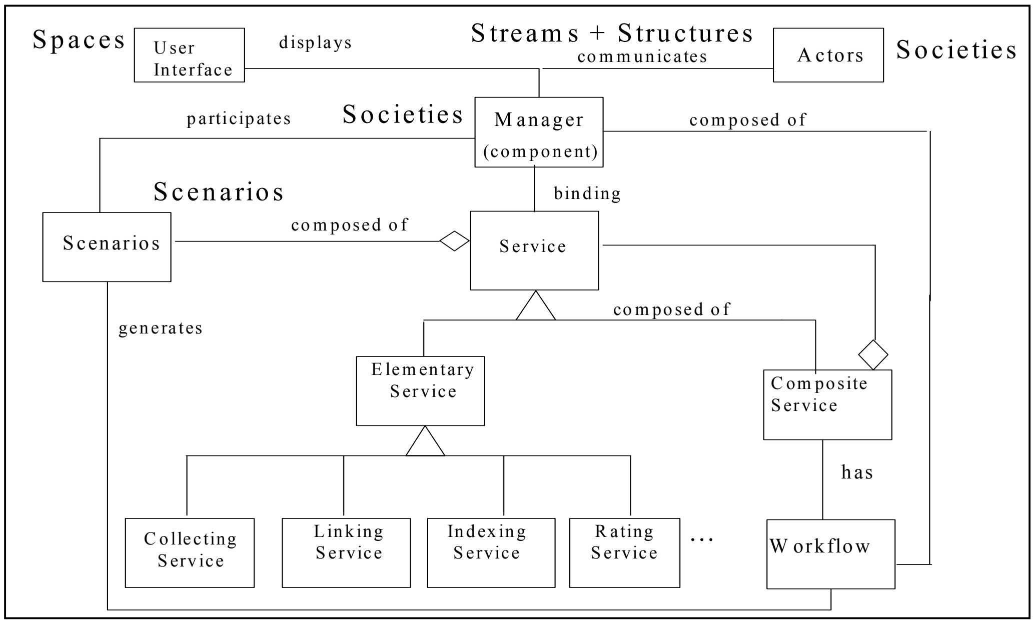


**Figure 4.1 DL service composition model**

The distinct aspects of this model are

1. The combination of an explicit workflow and service aggregation to support composite services.

2. The emphasis on scenario-based modeling of services and the automatic synthesis of Statecharts from them.

3. The role of the SM (a societal member) as the binding point for societal relationships, scenario interactions, and spatial visualizations.

From an architectural and implementation point of view, point 1 becomes significant, since combining a small set of basic DL services (like searching and browsing) from a pool of DL components allows a designer to model and generate most digital libraries (at least from the behavioral point of view) with a minimum amount of coding. The only situations when coding is unavoidable are, for example, when a specific behavior of a composite service (e.g., Multi-classification browsing) is not defined by any component in the core pool or cannot be reused (e.g., due to incompatibility of interfaces).

It is interesting to notice the connections between the roles of the SM and the classical Model-View-Controller (MVC) architecture of user-interfaces [14], which explicitly separates functionality, behavior, and presentation and has helped facilitate the development of user-interfaces that are modular and extensible. This relationship is further explored in section 4.3.

This model in Figure 4.1 shows how the five 'Ss' help when defining all components of a real, implemented DL. Services are implemented as components taken from the pool or automatically generated from the societies and their interactions/relationships. SMs define the context or functionality of the service in terms of its operations and the data it expects, and are associated with a spatial (presentational) model of a user-interface. This design provides the basic architectural underpinnings for the 5SLGen, as described in the next section.

## 4.2. 5SLGen: Input

The 5SLGen DL generator takes three things as input.

1. **5SLSocieties model of the services being modeled:**
   The DL designer models each society as a collection of cooperating SMs. The model specifies their attributes, operations, and relationships.

2. **5SLScenarios model of the services being modeled:**
   The DL designer models each service with multiple scenarios. A scenario involves a sequence of interactions among multiple collaborating SMs. A set of messages is exchanged in an interaction. Each message corresponds to an operation invocation on the receiver SM with certain exceptions.

3. **Reusable set of classes that compose the component pool:**
   The functionality of each elementary service such as searching, browsing, etc., is provided by a component. In terms of software, the component is a package/collection of classes that exposed a clean interface and implements a specific functionality. We consider components as "black boxes" that can be incorporated into a generated DL without knowing their implementations [76]. A collection of such components is referred to as the component pool. The ODL project has helped constitute such a pool of components [11].

   The implementation of the component can be in any language. Therefore, in order to use them on a particular platform or with a specific language we need to define translators or wrappers that translate any foreign operations on a component to its native operation. For instance if the DL being generated is a Java web application and the implementation of the search component is in Perl, then we need to define a wrapper that translates the Java method calls on the search component to Perl method invocations and returns the appropriate results from the Perl component to the Java web application. The job of a wrapper is to act as a bridge between two applications implemented in different programming languages or platforms.

   In the context of 5SLGen, the ODL components, originally implemented in Perl, have been encapsulated through a Java interface, allowing them to be imported by the Java classes for the SMs. Any component that follows the ODL protocols can be included into the component pool, by defining a wrapper that exposes its functionality in Java. The standard semantics of the OAI-PMH protocol thus enables the compositions of heterogeneous components with the wrappers acting as a bridge between the protocol implementation and the target platform.

## 4.3. 5SLGen: Output

The 5SLGen takes the above input and generates object-oriented classes. These classes along with the wrapper classes from the component pool constitute the 5SFramework. The service implementations reuse the implementation of the elementary services from the component pool. If a composite service is completely defined by a collection of elementary services, then a complete implementation is generated, however if the composite service contains certain functionalities that are not present in the component pool, 5SLGen generates a partial implementation. The DL designer completes this partial implementation, by adding code at certain extension points. The DL designer only writes code for the functionalities that are not present in the component pool. In this process, he is aided by 5SLGen, which provides a framework of classes to modify. The amount of code written depends largely on the overlap between the functionalities desired and the functionalities exposed by the components in the component pool.

The 5SFramework classes generated are differentiated into model, view, and controller based on the MVC design pattern. The MVC design pattern helps better separate the three essential views of a DL listed below:

a. The logic of the application (the model). The model consists of the SMs from the component pool and other SMs that are implemented/extended by the DL designer.

b. The control of the interaction triggered by the user's actions (the controller). The controller is responsible for interpreting the user's request, producing the appropriate request for action, examining the result of each action, and deciding what to do next. The controller class is generated by synthesizing all the scenarios of the composite service.

c. The interface presented to the user (the view). The view embodies the presentation logic for assembling the user-interface.

The model and the controller in the 5SFramework are coupled by 5SLGen. 5SLGen does not generate any classes for the view of the DL. We envision that the view for the DL will be constructed by web designer rather than the DL designer. The model and the controller are organized in the 5SFramework, in such a way that the view can be coupled easily by a web designer at a later stage. The interaction among the 5SFramework classes generated by 5SLGen is illustrated by Figure 4.2. Details of the interaction are provided in the next chapter.

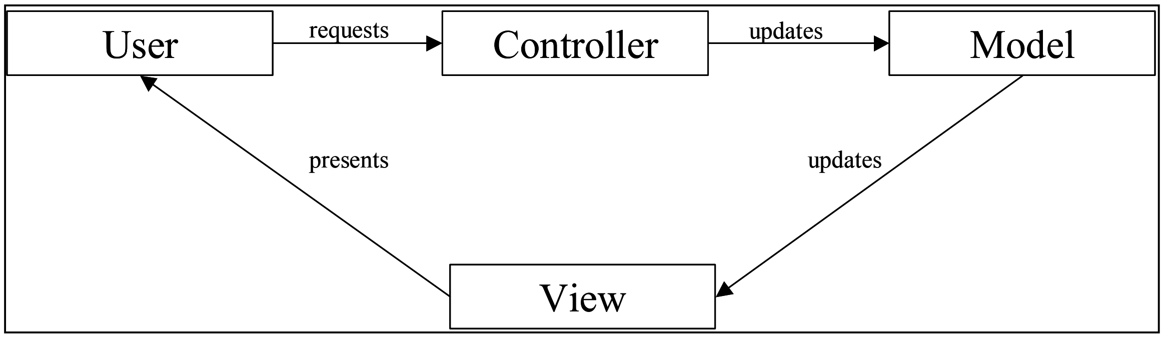


**Figure 4.2 Classes of the 5SFramework in context to the MVC design pattern**

### 4.4. 5SLGen: Architecture

5SLGen is primarily composed of two architectural components, viz., the scenarios-converter and the societies-converter. They are responsible for transforming the 5SLSocieties and 5SLScenarios models to the 5SFramework classes that implement the DL services. This section presents the component architecture of 5SLGen. We will dive into the implementation of each 5SLGen component in the next chapter. Figure 4.3 illustrates the component architecture of 5SLGen and the 5SFramework. Table 4.1 summarizes the transformations carried out by each 5SLGen architectural component.

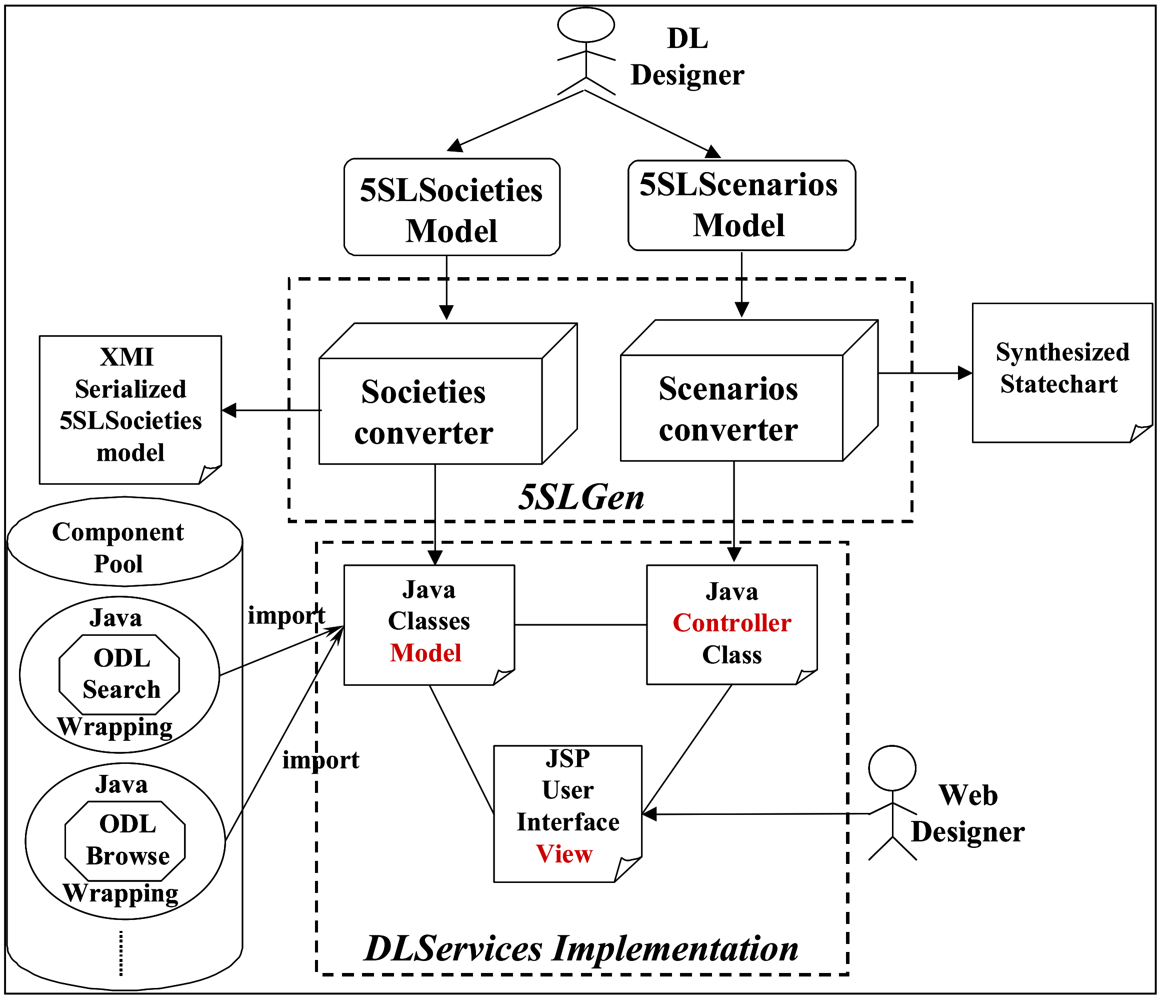


**Figure 4.3 Architecture of 5SLGen and the generated 5SFramework classes (expanding part of Figure 3.4)**

### 4.4.1. Societies-Converter

As the name suggests the societies-converter (please look at Figure 4.3) operates on the 5SLSocieties model. The societies-converter is responsible for transforming the 5SLSocieties model to a programming language specific code skeleton. The generated Java classes along with the classes from the component pool constitute the application logic/model of the DL.

We have chosen to transform the modeling constructs in the 5SLSocieties model to Java. This transformation can be applied because most of our 5SL modeling entities lend quite well to object-oriented programming concepts. The exact mapping from the 5SLSocieities model to Java language constructs is explained in the next chapter.

The societies-converter is also responsible for serializing the 5SLSocieities model to XMI. This serialization of the 5SLSocieties model to XMI entails deriving a mapping of the UML metamodel to the 5S metamodel. The societies-converter implements this mapping to serialize 5SLSocieties model to XMI according to the UML DTD for XMI released by the OMG. Exploring the XMI standard or the UML DTD for XMI is out of the scope of this report; instead, the emphasis here is on the transformation from 5SL to XMI. For a thorough coverage of the XMI standard, refer to [42, 77].

### 4.4.2. Scenarios-Converter

The scenarios-converter (please look at Figure 4.3) implements the scenario-synthesis algorithm on the 5SLScenarios models and generates a Statechart for the controller of the DL services. This transformation from multiple service scenarios to a single Statechart is explained in the next chapter. The Statechart generated can be given to any valid state machine compiler to generate a controller for the DL services. Generation of the Statechart as a separate artifact decouples our scenario-synthesis algorithm from the rest of the scenarios-converter implementation making the 5SLGen extensible and modular.

In addition to generating a Statechart, the scenarios-converter generates a controller Java class for the DL services based on the State design pattern [13]. The State pattern localizes state-specific behavior in an individual class for each state, and puts all the behavior for that state in a single state object eliminating the necessity for a set of long, look-alike conditional statements. In the context of 5SFramework, when the controller class receives an event, it delegates the request to its state object, which provides the appropriate state specific behavior.

**Table 4.1 5SLGen component functionality**

| Input | 5SLGen Component | Output |
|---|---|---|
| 5SLSocieties model | societies-converter | XMI:Class serialization of 5SL Models |
| 5SLSocieties model | societies-converter | Java class skeleton for the Societies model |
| 5SLScenarios model | scenarios-converter | Synthesized Statechart for the controller of the composite DL service |
| 5SLScenarios model | scenarios-converter | Java class for the controller of the DL composite service |

# 5. 5SLGen: Implementation

In this chapter, we describe the architecture and implementation 5SLGen. We start with a description of the platform and environment used for implementing 5SLGen. In the body of the chapter, we elaborate on the implementation of 5SLGen components and their workflow, placing emphasis on the transformation process from 5SLSocieties and 5SLScenarios models to code. Workflows of the components are illustrated with diagrams.

## 5.1. 5SLGen: Platform and Environment

5SLGen is implemented in Java. It can be deployed on any platform. The DLs created using 5SLGen can be deployed as web applications on any standard servlet container that implements the servlet-2.3 and JSP-1.2 specification. The servlet container of our choice was Tomcat. Each DL service is implemented as a servlet. 5SL XML documents were parsed using the Xerces open-source XML parser. The implementation of some of the modeled services uses the open-source mySQL database for persistence. We have chosen open-source tools and technologies whenever a valid and viable open-source choice was available.

Care was taken during the implementation of 5SLGen to avoid hard coding any variables in the source code. All configuration and customization is done from the command-line, via a properties file, or through properties classes. For instance, all the XML tag names used for serializing the 5SLSocieties model to XMI have been placed in a particular Java class. This facilitates future maintenance of the code, as any XMI schema change would require all changes to be made to only one file rather than the entire code base.

## 5.2. 5SLGen: Implementation

5SLGen consists of two components, viz., the scenarios-converter and the societies-converter. The implementation of each converter is explained below in brief.

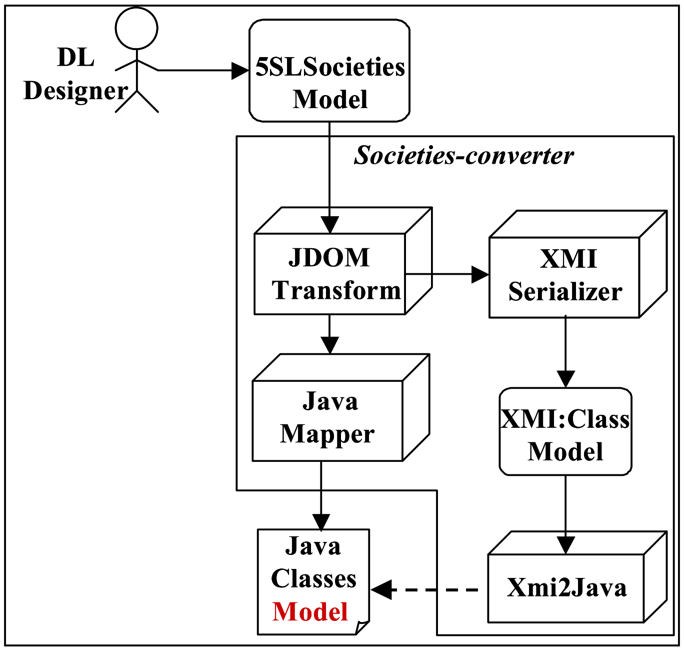


**Figure 5.1 Workflow of the societies-converter (expanded part of Figure 4.3)**

### 5.2.1. Societies-Converter: Implementation

The workflow of the societies-converter is illustrated in Figure 5.1. The societies-converter has two workflows corresponding to its functionalities listed in Table 4.1. Both the workflows have the same first step, i.e., the construction of the input language syntax tree. This is carried out by the JDOM transform package.

**JDOM Transform:** The 5SLSocieties model is parsed by an XML parser, in our case the Xerces Java parser, to construct a Java representation of the 5SLSocieties XML. The parse tree holds in-memory representation of the input 5SLSocieties model. Construction of the parse tree is absolutely essential since it captures the document in a data structure on which all further processing algorithms operate [78].

After construction of the parse tree, two kind of processing occur, namely, mapping to Java code and serialization to XMI. This corresponds to the two workflows of the societies-converter component.

#### 5.2.1.1. 5SLSocieties to Java classes

**Java Mapper:** The mapping of 5SLSocieties model constructs to Java [79] is carried out by the Java Mapper package (see Figure 5.1). The 5SLSocieties model captures the details of the SMs and their relationships. Each SM is described by its attributes, operations, associations, dependencies, and its generalizations. Each of these modeling entities maps on to a programming language construct in Java. Table 5.1 summarizes this mapping from the 5SLSocieties model to Java. The mapping provides insight into how we met certain challenges such as lack of multiple inheritance in Java. The 5SLSocieties modeling elements in the mapping are referred to as SM.name, SM.operation and so on. The Java language constructs in the mapping are represented in object.member notation.

**Table 5.1 Mapping of 5SLSocieties model elements to Java language constructs.**

| 5SLSocieties model Element | Java language construct | Notes |
|---|---|---|
| **Model element = SM**<br>An SM maps to a class in Java. | | |
| SM.name | class name | SM.name is the name of the Java class. |
| ServiceManager | class<br>interface<br>abstract class | SM.type =class→ Java construct = class<br>SM.type=interface → Java construct = interface<br>SM.type=abstract→Java construct=abstract class |
| SM.visibility | access modifiers | SM.visibility maps one on one with class visibility. SM.visibility= private → class.visibility = private<br>SM.visibility= public → class.visibility = public |
| **Model element = Attribute (SM.attribute)**<br>An SM attribute maps to an attribute in Java. | | |
| SM.attribute.name | attribute name | SM.attribute.name = attribute name |
| SM.attribute.class-scope | special modifier (instance variable or class variable) | SM.attribute.class-scope=false →<br>attribute is an instance variable<br>SM.attribute.class-scope=true →<br>attribute is a class variable |
| SM.attribute.visibility | access modifiers | 3 visibility modes are defined private, public and protected. A one-to-one mapping exists between |

| | | |
|---|---|---|
| | | attribute visibility=SM.visibility. |
| SM.attribute.multiplicity | arrays of instance/class variable | SM.attribute.multiplicity[1..n] → public Integer items[ ]; |
| SM.attribute.type | primitive datatypes,classes | SM.attribute.type=primitive datatype → Java datatype is a primitive datatype When, SM.attribute.type=SM → Java datatype is the corresponding class, |
| SM.attribute.constraint | special modifier | When, SM.constraint = "frozen" → Java modifier = "final" |
| **Model element = Operation (SM.operation)** An SM operation maps to a method in Java. | | |
| SM.operation.name | method name | SM.operation.name = method name |
| SM.operation.visibility | access modifier | See note above for attributes. |
| SM.operation.constraint | special modifier | See note above for attributes. |
| SM. operation class-scope | special modifier | SM.operation.class-scope="true" → special modifier="static" SM.operation.class-scope="false"→ no special modifier |
| SM.operation. parameter. name SM.operation.parameter.type | parameter list | SM parameter.names and SM.parameter.types are aggregated to generate a comma-separated list of parameter declarations. |
| SM.operation.concurrency | - | The SM.concurrency modeling element has not been mapped yet to Java. By default all operations as of now are sequential in nature. |
| **Model element = Dependency (SM.dependency)** An SM dependency either leads to a package/class import or implementation of an interface. | | |
| SM.dependency.peer | class/package name | SM.dependency.peer = class/package name. The SM on which the dependency exists maps to a class or a package depending on the kind of dependency. |
| SM.dependency.kind | OOP specific keywords | SM. dependency.kind ="import" → OOP specific keyword ="import" SM. dependency.kind ="implements"→ OOP specific keyword ="implements" |
| **Model element = Association (SM.dependency)** An association is modeled as an attribute in Java [80]. For instance, if an SM A is in association with an SM B, and the SM B plays the role of a teacher in the association, then the class A will have a data attribute of name teacher and type B. Java code: private B teacher; | | |
| SM.association.name | - | The SM association name serves to uniquely identify a bidirectional association. |
| SM.association.peer | class | This is the class that is in association with the modeled SM. |
| SM.association.role | name of the instance/class variable | If the rolename is not defined then the association is non-navigable. |
| SM.association.qualifier | HashTable class | A HashTable class takes care of uniquely identifying classes in a one-to-many association. |
| SM.association.class-scope | special modifier | See note above for attributes. |
| SM.association.constraint | special modifier | See note above for attributes. |

| **Modeling element = Generalization (SM.dependency)** | | |
|---|---|---|
| Since Java does not permit multiple inheritance, we implement multiple inheritance in Java using interfaces. Say class son inherits from classes mother and father. In that case, we create three implementation classes (sonImpl, motherImpl, fathterImpl) and three interfaces (sonIntf, motherIntf, fatherIntf). As multiple inheritance is permitted with interfaces in Java, we make the sonIntf extend the motherIntf and fatherIntf. At the same time, we make the sonImpl implement the sonIntf, motherImpl implement the motherIntf, and the fatherImpl implement the fatherIntf. As interfaces are contracts that are honored by their classes, the sonImpl class inherits from the motherImpl and fatherImpl classes. | | |
| SM.generalization.from | Class | The name of the parent class |

A 5SLSocieties model fragment for the RelevanceFeedback SM and the Java source code generated according to the mapping rules described above is shown in Example 5.1 and Example 5.2 respectively. The Java code generated from the 5SLSocieties model is a skeleton for the actual implementation of the SM. It is the responsibility of the DL designer to flesh out the implementation from the skeleton generated by 5SLGen.

```
<ServiceManager NAME="RelevanceFeedbackSearchImpl" VISIBILITY="public" TYPE="class">

        <Attribute VISIBILITY="private" TYPE="boolean" NAME="DEBUG" CLASS-
        SCOPE="true" CONSTRAINT="frozen" INITVAL="true"/>

        <Operation VISIBILITY="public" NAME="relevanceFeedbackSearch" RETURN="void">

            <Parameter TYPE="ArrayList" NAME="oaiIds"/>
        </Operation>

        <Association MULTIPLICITY="1" ROLENAME="uimFsmClone" PEER="UIMFSM"
        CLASS-SCOPE="true"/>
        <Association MULTIPLICITY="1" ROLENAME="handler" PEER="SAXPrintHandler" />

        <Association MULTIPLICITY="1" ROLENAME="ODLSearch" PEER="ODLSearchImpl" />


        <Dependency PEER="Java.util.*" DEPKIND="use"/>
        <Dependency PEER="org.xml.sax.XMLReader" DEPKIND="use"/>
        <Dependency PEER="org.xml.sax.InputSource" DEPKIND="use"/>
        <Dependency PEER="org.apache.xerces.parsers.SAXParser" DEPKIND="use"/>
    </ServiceManager>
```

**Example 5.1 5SLSocieties model for the relevance feedback search SM**

```
import org.apache.xerces.parsers.SAXParser;
import org.xml.sax.InputSource;
import org.xml.sax.XMLReader;
import Java.util.*;
import Java.util.*;

public class RelevanceFeedbackSearchImpl {

    //Attribute
    private static final boolean DEBUG  =true;

    //Operation
    public void relevanceFeedbackSearch ( ArrayList oaiIds  ){  }
```

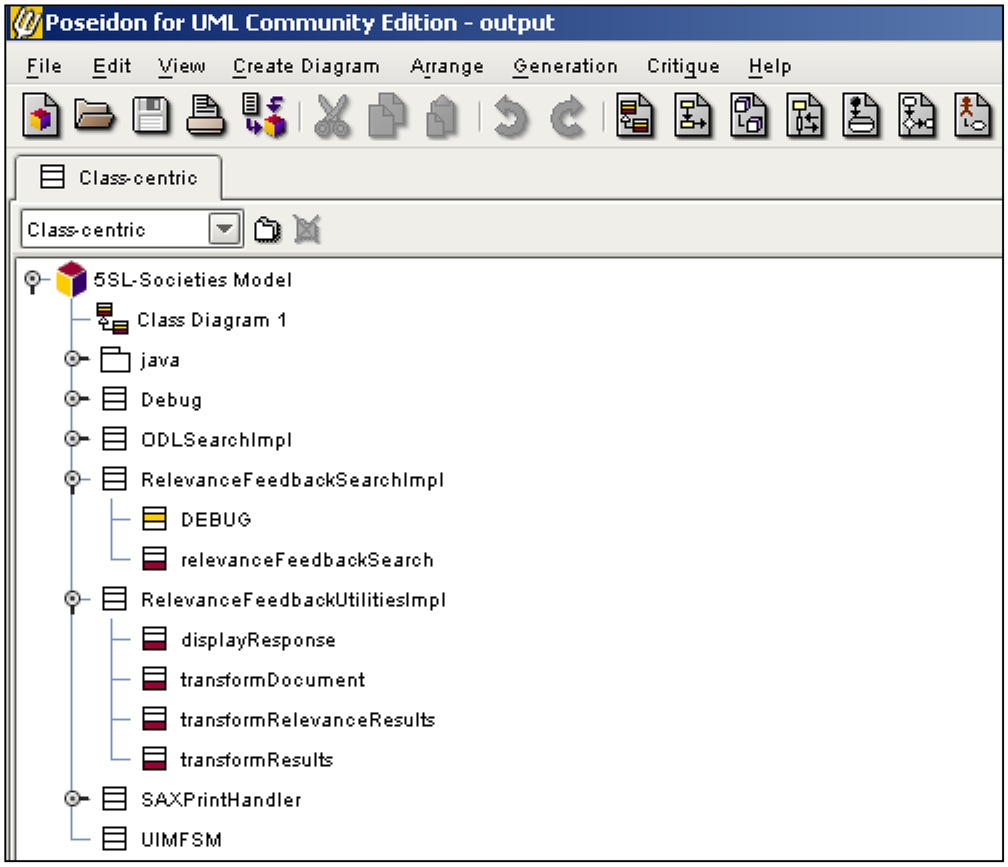
```
        //Associations
        private static UIMFSM uimFsmClone ;
        private SAXPrintHandler handler ;
        private ODLSearchImpl ODLSearch ;

        //Getters and Setters
         public UIMFSM getUimFsmClone (){   return(uimFsmClone );   }
         public void setUimFsmClone (UIMFSM  setObject ){      this.uimFsmClone = setObject;   }

         public SAXPrintHandler getHandler (){      return(handler );   }
         public void setHandler (SAXPrintHandler  setObject ){    this.handler = setObject;   }

         public ODLSearchImpl getODLSearch (){   return(ODLSearch );   }
         public void setODLSearch (ODLSearchImpl  setObject ){       this.ODLSearch = setObject;   }
}
```

**Example 5.2 Generated Java code for the relevance feedback search SM**

#### 5.2.1.2. 5SLSocieties to XMI

**XMISerializer:** The parse tree created by the JDOM Transform package serves as input to the XMISerializer package (see Figure 5.1). The XMISerializer then applies a set of transformation rules to create an XMI representation of the 5SLSocieties model. The XMISerializer has an operation similar to that of the Java Mapper, however instead of mapping to Java classes, we map from the 5SLSocieties model to the UML DTD for XMI (explained in section 4.4.2). This mapping cannot be elaborated here due to limitations on the size of the document and the complexity of the UML DTD for XMI specification [42]. Indeed, implementing the XMISerializer was one of the challenging tasks in this project. Several tools such as EMF, MDR, and XMI-Toolkit [77, 81, 82] were looked at, however none of them was able to map between the metamodels of 5S and UML. Failing to find a third party tool, we have implemented our own XMISerializer tool. For a 5SLSocieties model containing 139 lines of XML, the corresponding XMI document is 1080 lines long. This gives an idea of the verbosity of the XMI standard and the complexity of the task.

**XMI:Class:** The XMI generated from the XMISerializer is referred to as XMI:Class (see Figure 5.1) because we are mapping only the 5SLSocieties model to XMI. For a complete mapping to XMI, we also need to map the 5SLScenarios model to XMI. This XMI:Class can be imported into other CASE tools. Figure 5.2 presents snapshot of the 5SLSocieties model loaded into the Poseidon CASE tool [83].



5SL-Societies Model
Class Diagram 1
java
Debug
ODLSearchImpl
RelevanceFeedbackSearchImpl
DEBUG
relevanceFeedbackSearch
RelevanceFeedbackUtilitiesImpl
displayResponse
transformDocument
transformRelevanceResults
transformResults
SAXPrintHandler
UIMFSM

**Figure 5.2 5SLSocieties model imported into the Poseidon CASE tool using XMI**

**XMI2Java:** Once the XMI:Class model has been generated by the XMISerializer, one can use open-source XMI2Java or CASE tools for XMI2Java conversion (see Figure 5.1). The dotted arrow in Figure 5.1 from the XMI2Java package to the Java classes indicates that the workflow involving the XMISerializer, XMI:Class, and XMI2Java is optional. In most cases, we envision the DL designer will not use the XMI serialization of the 5SLSocieites model. Having said that, we have provided the flexibility to the DL designer to move away from 5SLGen's Java Mapper to other code generation tools and CASE tools. CASE tools possess the ability to generate class code from XMI. In addition to forward and reverse engineering, they also assist in modeling with UML.

### 5.2.2. Component Pool Utilization

Before moving on to the implementation of the scenarios-converter, we explain how the components from the component pool are incorporated into the model (MVC) of the DL. The functionality of each component from the component pool is provided by an SM that acts as the wrapper for the component. This existing pool of implemented SMs is used while implementing the basic services of the DL. The 5SLSocieties models for these SMs are fixed and can be copied as-is while modeling other DL services.

Once the societies-converter generates the skeleton Java classes for the 5SLSocieties model, the component SM implementations are placed along with the generated code in the web application. The generated SMs communicate with the component SMs according to the interaction specified by the 5SLScenarios model. The component SMs expose the functionalities from the component pool whereas the skeleton SMs implement the new functionalities modeled by the DL designer. The designer needs to complete the partial implementation of these skeleton SMs, before the DL can be deployed.

### 5.2.3. Scenarios-Converter: Implementation

Figure 5.3 illustrates the workflow of the scenarios-converter. The first step as before is parsing the 5SLScenarios model to create a parse tree. This is done by the JDOM Transform package. As this step has been elaborated before, we move to the next step of the transformation process, i.e., the scenario-synthesis algorithm. The parse tree created by the JDOM transformation yields all the data-structures, i.e., the list of scenarios of the service and their respective events, required by the scenario-synthesis algorithm.

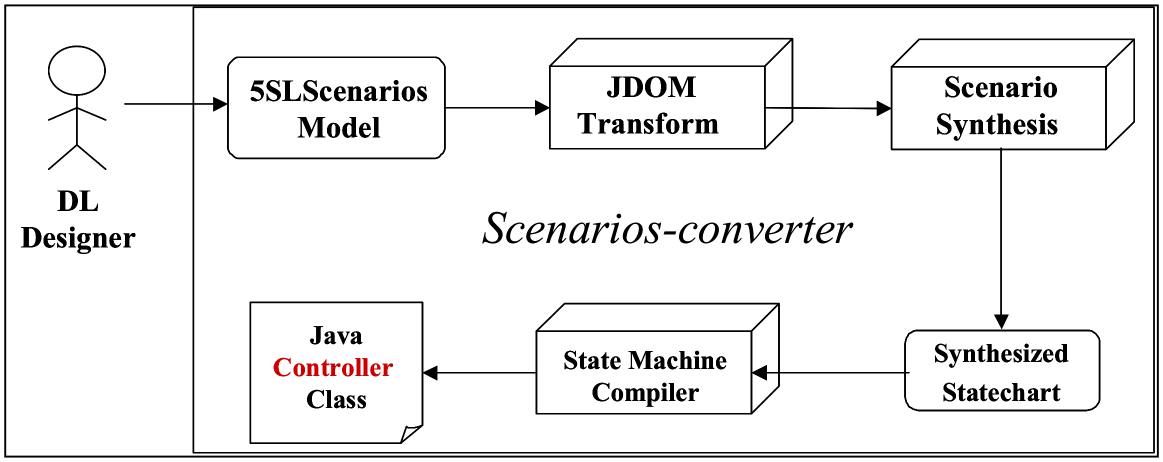


**Figure 5.3 Workflow of the scenarios-converter (expanded part of Figure 4.3)**

#### 5.2.3.1. Scenario-Synthesis

The scenario-synthesizer (see Figure 5.3) package implements the scenario-synthesis algorithm. We will illustrate the implementation of the scenario-synthesis algorithm with respect to the scenarios for the NDLTD Union Catalog DL. The Union Catalog DL exposes a composite service that is comprised of three elementary services, viz., search, browse, and search-similar (search all items similar to a particular item). To maintain clarity we explain the scenario-synthesis with respect to the primary scenarios for each service (search, browse, and search-similar). The scenario-synthesis will be carried out for the controller SM (UIMFSM) of the composite service.

A brief description of the scenario-synthesis algorithm implemented follows [54]. The scenario-synthesis algorithm consists of two transformations that are applied to the scenarios of a service. The scenario-synthesis is always carried out with respect to a particular object. The transformations required for scenario-synthesis are:

1. Generate separate Statecharts from all the single scenarios according to the following rule:

    For any object in a sequence diagram incoming events trigger transitions, outgoing events become actions of the transitions leading to the states. On receiving an event the list of actions associated with the event are carried out by the receiver.

    Consider the sequence diagram (see Figure 5.4) and the corresponding Statechart (see Figure 5.5) for the search scenario of the Union Catalog DL.

    The doGet, search_query and return_search_results are incoming events in the sequence diagram. These are mapped to the triggers of the Statechart.

    On the reception of doGet, search_query and return_search_results events, the controller carries out the UIMFSM.parseRequest, ODLSearch.odlsearch, and ODLUtilties.transformResults actions, respectively. These actions are mapped to their corresponding triggers. This mapping of trigger to actions completely specifies all the transitions of the Statechart.

    The outgoing event names, search and display, along with the initial state MainMenu, constitute the states in the Statechart. The initial state of the controller is specified in the 5SLScenarios model for the service. The other state names are obtained from the controller's outgoing events.

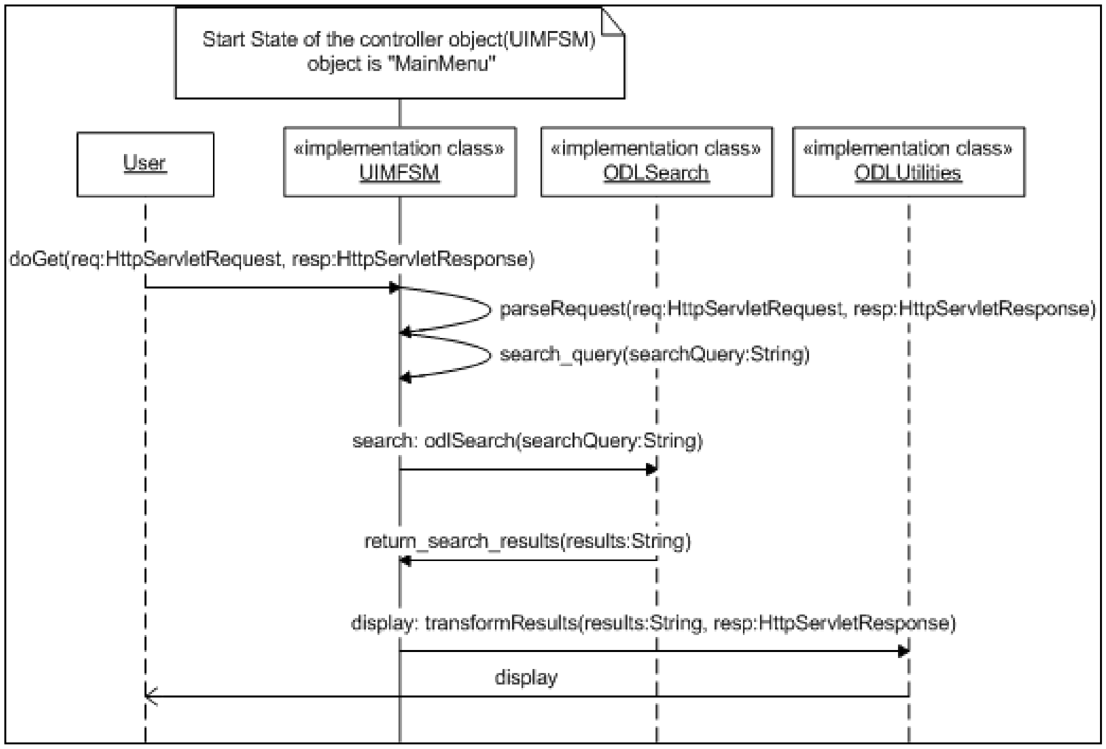

Start State of the controller object(UIMFSM) object is "MainMenu"
User
«implementation class» UIMFSM
«implementation class» ODLSearch
«implementation class» ODLUtilities
doGet(req:HttpServletRequest, resp:HttpServletResponse)
parseRequest(req:HttpServletRequest, resp:HttpServletResponse)
search_query(searchQuery:String)
search: odlSearch(searchQuery:String)
return_search_results(results:String)
display: transformResults(results:String, resp:HttpServletResponse)
display

**Figure 5.4 Sequence diagram for the search scenario of the Union Catalog DL**

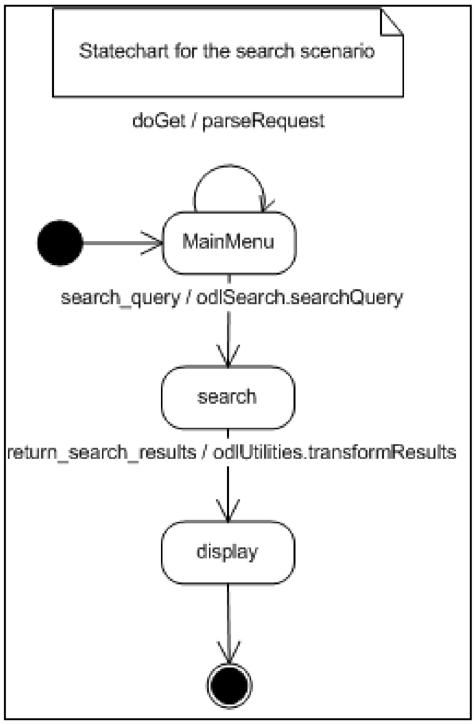

Statechart for the search scenario
doGet / parseRequest
MainMenu
search_query / odlSearch.searchQuery
search
return_search_results / odlUtilities.transformResults
display

**Figure 5.5 Statechart for the search scenario of the Union Catalog DL**

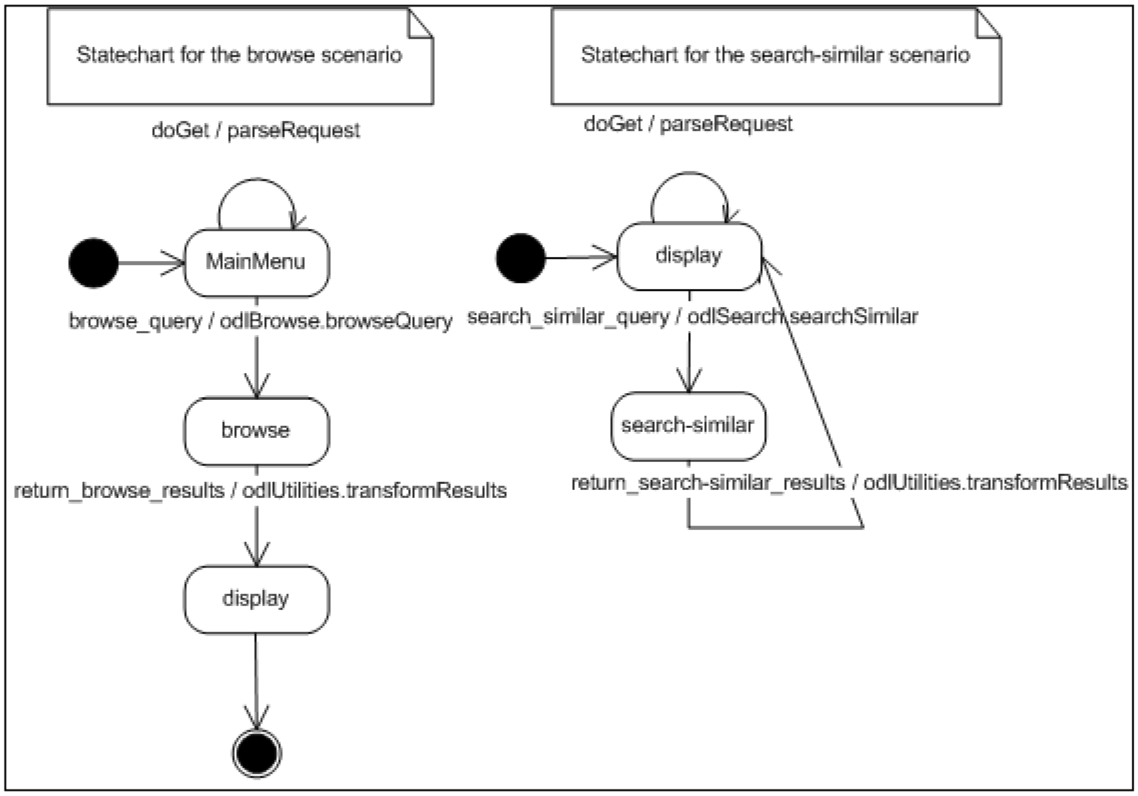

Statechart for the browse scenario
doGet / parseRequest
MainMenu
browse_query / odlBrowse.browseQuery
browse
return_browse_results / odlUtilities.transformResults
display
Statechart for the search-similar scenario
doGet / parseRequest
display
search_similar_query / odlSearch.searchSimilar
search-similar
return_search-similar_results / odlUtilities.transformResults

**Figure 5.6 Statecharts for browse & search-similar scenarios of the Union Catalog DL**

2. Merge all the Statecharts generated in step 1, according to the following rules:

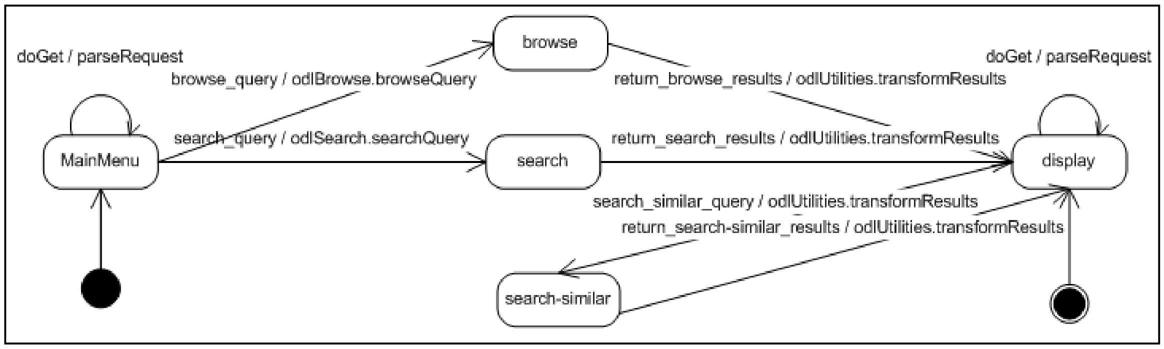
Rule 2.1: In a succession of two scenarios, the resulting Statechart merges the two basic corresponding Statecharts in temporal order.

Rule 2.2: If a transition is common to the two Statecharts, it will be taken only once in the final Statechart.

Rule 2.3: If at a certain moment in time either one or another scenario is executed, the Statecharts are combined with sequential (object can be in only one state) substates within a composite state.

Rule 2.4: If two scenarios are executed at the same time they are combined with concurrent substates (object can be in more than one state) within a composite state.

Figure 5.5 and Figure 5.6 show the Statecharts generated by applying rule 1 to the search, browse, and search-similar scenarios. Figure 5.7 shows the synthesized Statechart that results from application of rules 2.1, 2.2, 2.3, and 2.4. The search-similar scenario follows the search and browse scenarios in temporal order (application of rule 2.1). The doGet transition in the MainMenu state is common to both the search and the browse scenarios and appears only once in the synthesized Statechart (application of rule 2.2). The search and the browse states are joined in an OR relationship (application of rule 2.3). Rule 2.4 cannot be applied while merging the generated Statecharts, as two scenarios are never executed simultaneously.



doGet / parseRequest
browse_query / odlBrowse.browseQuery
browse
return_browse_results / odlUtilities.transformResults
doGet / parseRequest
MainMenu
search_query / odlSearch.searchQuery
search
return_search_results / odlUtilities.transformResults
display
search_similar_query / odlUtilities.transformResults
return_search-similar_results / odlUtilities.transformResults
search-similar

**Figure 5.7 Synthesized Statechart for the Union Catalog DL**

### 5.2.3.2. State Machine Compiler

While modeling 5SLScenarios, we always synthesize the Statechart with respect to the controller of the composite service. This controller is responsible for receiving user requests and calling the appropriate methods on the components of the DL. The synthesized Statechart serves as input to the state machine compiler (SMC) (see
Figure 5.3). The SMC reuses the data-structures created by the scenario-synthesizer and generates a Java class that serves as the controller. The controller Java class is based on the State design pattern [84]. The synthesized Statechart generated by the scenario-synthesizer gives the DL designer the flexibility of using other SMC tools.

The UML class diagram for the controller SM in Figure 5.8 illustrates the implementation of the State design pattern in the context of the Union Catalog service modeled above. The states of the synthesized Statechart map (MainMenu, search, browse, search-similar) map to the corresponding state classes in the State design pattern. Each state in the synthesized Statechart becomes a Java class. All the state-specific behavior for a state is placed in the class for that state. For instance, the method return_browse_results, which is specific to the browse state, is placed only in the browse state class. Please note that this is not the UML class diagram of the entire 5SFramework. This class diagram only covers the structure of the controller and its relationship with its state classes. It does not model the relationship of the controller with the component classes that constitute the model of the DL.

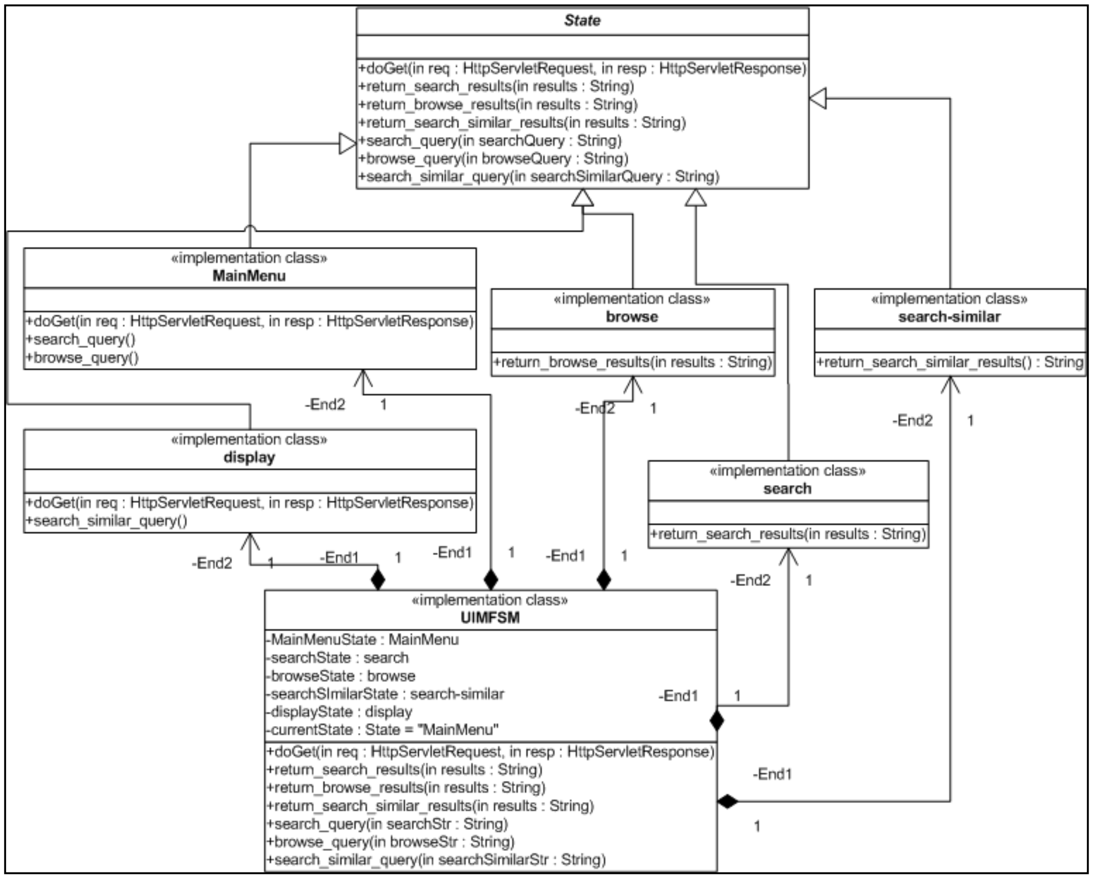

State
+doGet(in req : HttpServletRequest, in resp : HttpServletResponse)
+return_search_results(in results : String)
+return_browse_results(in results : String)
+return_search_similar_results(in results : String)
+search_query(in searchQuery : String)
+browse_query(in browseQuery : String)
+search_similar_query(in searchSimilarQuery : String)
«implementation class» MainMenu
+doGet(in req : HttpServletRequest, in resp : HttpServletResponse)
+search_query()
+browse_query()
«implementation class» browse
+return_browse_results(in results : String)
«implementation class» search-similar
+return_search_similar_results() : String
«implementation class» display
+doGet(in req : HttpServletRequest, in resp : HttpServletResponse)
+search_similar_query()
«implementation class» search
+return_search_results(in results : String)
«implementation class» UIMFSM
-MainMenuState : MainMenu
-searchState : search
-browseState : browse
-searchSimilarState : search-similar
-displayState : display
-currentState : State = "MainMenu"
+doGet(in req : HttpServletRequest, in resp : HttpServletResponse)
+return_search_results(in results : String)
+return_browse_results(in results : String)
+return_search_similar_results(in results : String)
+search_query(in searchStr : String)
+browse_query(in browseStr : String)
+search_similar_query(in searchSimilarStr : String)
-End1
-End2
1

**Figure 5.8 UML class diagram for the controller of the Union Catalog DL**

We now explain the implementation of the scenario shown in Figure 5.4 with respect to the controller Java class (UIMFSM). Please refer to the synthesized Statechart shown in Figure 5.7 and the controller UML class diagram shown in Figure 5.8 to understand the state transitions and actions involved.

The scenario starts with the user sending a doGet event to the UIMFSM through the browser. When the UIMFSM receives an event, it calls the relevant method of the state class involved, and changes its state according to the synthesized Statechart. On receiving the doGet event, the UIMFSM calls the doGet method of the MainMenu state class and transitions to the same state. The doGet method invokes the parseRequest method of UIMFSM, which in turn parses the request and generates the search_query internal event.

When the UIMFSM receives the search_query event, in the MainMenu state, it calls the search_query method of the MainMenu state class. The MainMenu state class then implements search state-specific behavior, which involves making method calls on the components of the generated DL services. In this case, the MainMenu state class calls the odlSearch method of the ODLSearch component and transitions to the search state.

On receiving results, through the return_results event from the ODLSearch component, the UIMFSM calls the return_results method of the search state class and transitions to the display state. The return_results method of the display state class in turn invokes the transformResults method of the ODLUtiltitiesImpl SM. This event-action-transition cycle terminates when the user receives the results from the ODLUtiltitiesImpl SM.

# 6. Case Studies

In order to test the feasibility of 5SLGen for generating DL services using real world scenarios, various composite services were implemented. Once the services exposed by a DL were implemented with the 5SFramework, we modeled and implemented the DLs that exposed these services. Thus, the approach we followed was that of modeling and generation of the services, followed by modeling and generation of the DL. CITIDEL, Virginia Instructional Architect for Digital Undergraduate Computing Teaching (VIADUCT) [85], and the Union Catalog – all real world DLs or DL services – were re-implemented in this manner. We deliberately chose to model real world DLs to ensure that our model implementation could be validated against production systems. The real world DLs provide ideal benchmarks for comparison between the old and new systems.

In addition to exposing the composite services, a DL also may expose many elementary services such as searching, browsing, etc.; these elementary services were integrated with the composite services during the modeling and generation of the DL. All the generated services and DLs are enumerated below.

Composite DL services generated with 5SLGen:
1. CITIDEL: multi-classification browsing service
2. CITIDEL: profile based filtering service
3. CITIDEL: relevance feedback search service
4. CITIDEL: binder service

DLs implemented using 5SLGen:
1. Union Catalog
2. CITIDEL
3. VIADUCT

The name of each service is prefixed with the name of the DL that exposes the service. The services exposed follow the model explained in section 4.1, and illustrate reusability and extensibility. More formally, a service Y reuses a service X if the behavior of Y incorporates the behavior of X. A service Y extends a service X if it subsumes the behavior of X and potentially includes conditional sub-flows of events.

In this chapter, we start with the description of each of the generated services. We describe the functionality of the service, the primary scenarios of usage, the composition of the service, and the synthesized Statechart for the service. The composition of the service is explained with the help of a UML class diagram that illustrates the 5SLSocieities model for the service. The primary scenario of usage is depicted by a UML sequence diagram. It is not possible to comment on all the scenarios of a particular service or a DL for the sake of brevity; instead we focus only on the primary scenarios. The synthesized Statechart provides a complete overview of the controller of the DL service. The primary scenario, along with the synthesized Statechart, seek to provide insight into the dynamic behavior of the services captured by the 5SLScenarios model.

As regards to the generated DLs, we comment on the most interesting aspects of the modeling and implementation process, explaining with diagrams and providing screenshots where necessary. We place special emphasis on the aggregation of the composite services exposed by the DL. We conclude by summarizing our observations regarding the entire modeling and generation process.

## 6.1. Composite DL Services

### 6.1.1. CITIDEL: Multi-Classification Browsing Service

The purpose of modeling this service is to show that complex services that do not reuse any elementary services can be implemented using the 5SFramework classes generated by 5SLGen. This service does not reuse any elementary services/components from the component pool. All the methods in the Java skeleton classes generated by 5SLGen have to be implemented by the DL designer.

#### 6.1.1.1. Functionality

Classification allows partitioning the DLs' contents into separately browseable and filterable areas. These classification schemes can become hierarchical, that is, they can take on the form of trees, in order to represent the natural levels of specificity and interrelatedness involved in different subjects. However, there is no standard classification scheme suitable for all DLs. When dealing with different classification schemes among libraries that communicate with one-another using OAI-PMH, there needs to exist a mechanism by which the records harvested from the host repository can be appropriately mapped to the correct classification categories in the destination repository. We need a system that will map between classification schemes, and to act accordingly.

The CITIDEL multi-classification browsing service provides a subject-hierarchical browsing interface, built upon categorization metadata of the DL resources that is invariant with respect to which supported scheme is being used. This means that users can browse the same set of content organized by schemes like ACM's Computing Classification System [86], Computing Curricula 2001 [87] and others, according to their preferences. Resources need only be classified under one scheme, with the invariant behavior provided by inter-scheme mappings. As a side effect, this system removes almost all of the work in migrating between subsequent revisions of classification schemes. This robust and useful functionality can be abstracted into a component and deployed in many other settings where multiple alternate classification schemes exist or multiple revisions of schemes are used [28].

### 6.1.1.2. Societies

The UML class diagram depicting the 5SLSocieties model of the service is shown in Figure 6.1. Note that all the attributes and operations of the SMs are not shown, to maintain clarity.

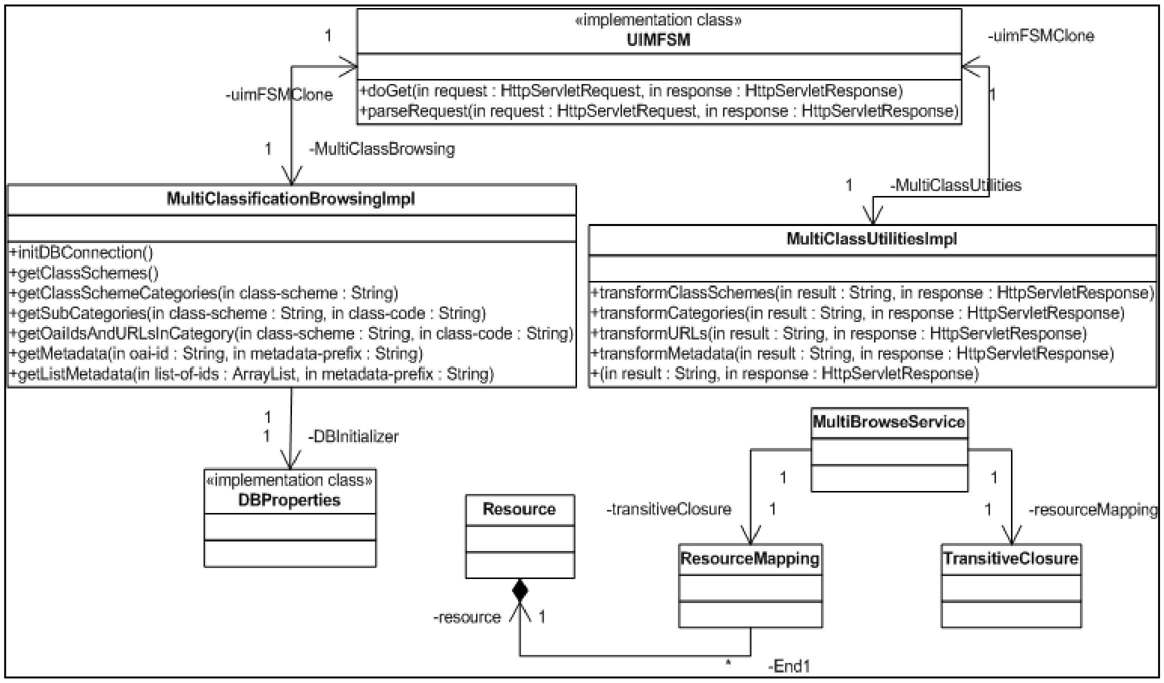

«implementation class»
UIMFSM
+doGet(in request : HttpServletRequest, in response : HttpServletResponse)
+parseRequest(in request : HttpServletRequest, in response : HttpServletResponse)
-uimFSMClone
1
-MultiClassBrowsing
-MultiClassUtilities
MultiClassificationBrowsingImpl
+initDBConnection()
+getClassSchemes()
+getClassSchemeCategories(in class-scheme : String)
+getSubCategories(in class-scheme : String, in class-code : String)
+getOaiIdsAndURLsInCategory(in class-scheme : String, in class-code : String)
+getMetadata(in oai-id : String, in metadata-prefix : String)
+getListMetadata(in list-of-ids : ArrayList, in metadata-prefix : String)
MultiClassUtilitiesImpl
+transformClassSchemes(in result : String, in response : HttpServletResponse)
+transformCategories(in result : String, in response : HttpServletResponse)
+transformURLs(in result : String, in response : HttpServletResponse)
+transformMetadata(in result : String, in response : HttpServletResponse)
+(in result : String, in response : HttpServletResponse)
-DBInitializer
«implementation class»
DBProperties
Resource
MultiBrowseService
-transitiveClosure
-resourceMapping
ResourceMapping
TransitiveClosure
-resource
*
-End1

**Figure 6.1 5SLSocieties model for the multi-classification browsing service**

The multi-classification browsing service is implemented by the MultiClassBrowsingImpl SM. The MultiClassBrowsingImpl provides the logic for the DL application. The application logic consists of all operations that help the user browse through the classified resources. For instance, given a particular category-code from a classification scheme, the DL application returns all the records within it.

The MultiBrowseService, Resource Mapping, Resource, and Transitive Closure SMs are responsible for creation of the mapping across the classification schemes and classifying the resources in the CITIDEL database. Mapping of classification schemes involves computing of a transitive closure on a matrix of all the relationships among the classification schemes. This is done by the Transitive Closure SM. The ResourceMapping and Resource SMs are responsible for classifying the resources in the database. This service assumes the existence of a database of records with categorization metadata.

The Multi-ClassUtilitiesImpl SM is responsible for formatting the output and represents the view of the DL service.

The UIMFSM SM serves as the controller of the service.

The SMs described above can be clearly categorized into the model, controller, and view abstractions of the MVC design pattern, which illustrates the architecture of the 5SFramework.

#### 6.1.1.3. Scenarios

Three scenarios were modeled for the multi-classification browsing service. The sequence diagram in Figure 6.2 shows the interaction among the various SMs in the first primary scenario. In this scenario the user first retrieves all the classification schemes supported by the system. After selecting a particular classification scheme the user retrieves all the categories in that classification scheme. Thereafter the user identifies a category of interest and retrieves all the record-identifiers belonging to that category. Finally, the user retrieves the metadata associated with all the record-identifiers retrieved in the previous step.

In the second primary scenario, the user first retrieves the classification schemes supported by the system. Having selected a particular scheme, he retrieves the categories from that scheme. Once all the categories of the scheme are presented, he retrieves the sub-categories of a particular category in the scheme. After narrowing down a category, in the penultimate step the user retrieves the record-identifiers in that category. In the last step, the user retrieves the metadata associated with an individual record-identifier.

The third scenario captures the navigation of the user across different states of the controller. The controller follows the synthesized Statechart of the service. This Statechart allows user access to services depending on its current state. Sometimes, it becomes necessary to return the controller to a previous state in order to repeat a user activity.

For instance, let us assume that the controller of the DL service is in the SchemesDisplay state (see the controller-Statechart in Table 6.1). When the controller is in the SchemesDisplay state, the user is unable to get all the classification schemes supported by the DL because the event get_class_schemes is valid only in the MainMenu state. In order to transition the controller back to the MainMenu state a back event needs to be generated on the controller. Such a back event takes the controller to the MainMenu state. Once in the MainMenu state the user can again issue a valid query to get all the classification schemes supported. These navigation scenarios help the user in navigation of the various services exposed.

All the DLs and DL services modeled using 5SLGen incorporate scenarios for user navigation across different states of the controller.

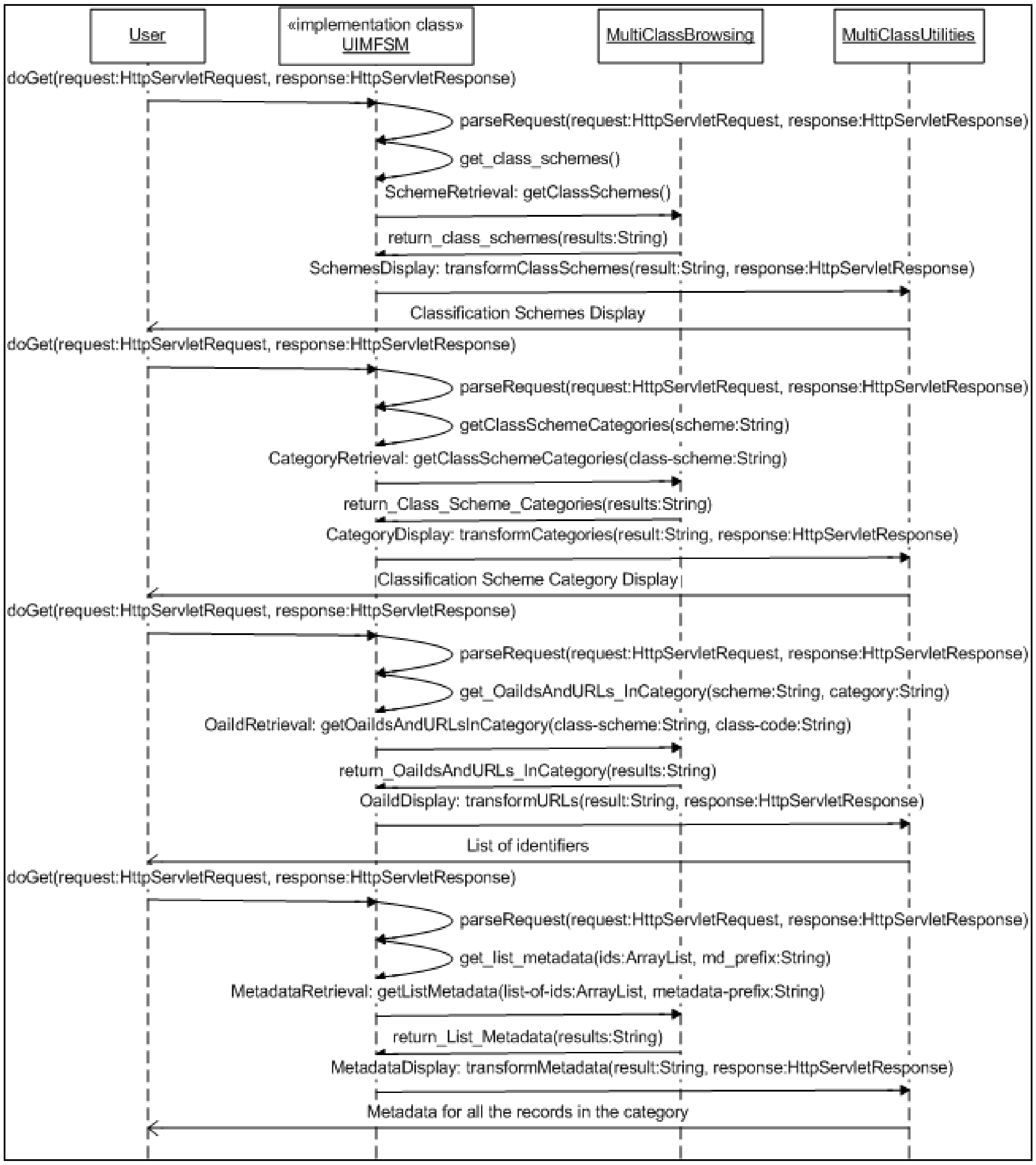

User
«implementation class» UIMFSM
MultiClassBrowsing
MultiClassUtilities
doGet(request:HttpServletRequest, response:HttpServletResponse)
parseRequest(request:HttpServletRequest, response:HttpServletResponse)
get_class_schemes()
SchemeRetrieval: getClassSchemes()
return_class_schemes(results:String)
SchemesDisplay: transformClassSchemes(result:String, response:HttpServletResponse)
Classification Schemes Display
doGet(request:HttpServletRequest, response:HttpServletResponse)
parseRequest(request:HttpServletRequest, response:HttpServletResponse)
getClassSchemeCategories(scheme:String)
CategoryRetrieval: getClassSchemeCategories(class-scheme:String)
return_Class_Scheme_Categories(results:String)
CategoryDisplay: transformCategories(result:String, response:HttpServletResponse)
Classification Scheme Category Display
doGet(request:HttpServletRequest, response:HttpServletResponse)
parseRequest(request:HttpServletRequest, response:HttpServletResponse)
get_OaiIdsAndURLs_InCategory(scheme:String, category:String)
OaiIdRetrieval: getOaiIdsAndURLsInCategory(class-scheme:String, class-code:String)
return_OaiIdsAndURLs_InCategory(results:String)
OaiIdDisplay: transformURLs(result:String, response:HttpServletResponse)
List of identifiers
doGet(request:HttpServletRequest, response:HttpServletResponse)
parseRequest(request:HttpServletRequest, response:HttpServletResponse)
get_list_metadata(ids:ArrayList, md_prefix:String)
MetadataRetrieval: getListMetadata(list-of-ids:ArrayList, metadata-prefix:String)
return_List_Metadata(results:String)
MetadataDisplay: transformMetadata(result:String, response:HttpServletResponse)
Metadata for all the records in the category

**Figure 6.2 Primary scenario of the multi-classification browsing service**

### 6.1.1.4. Controller-Statechart

Table 6.1 details the Statechart of the controller (the UIMFSM SM) of the service. The Statechart is synthesized after taking into account all three of the modeled scenarios. This Statechart governs the behavior of the multi-classification browsing service. We do not use the diagrammatic notation for the Statechart because of the large number of states and transitions involved. We have not shown the parameters passed to the events and the actions of the Statechart for the sake of brevity. All the events in the second column of Table 6.1 are invoked on the controller SM. The actions in the last column of Table 6.1 are denoted in object-oriented notation (object.member).

Each row of Table 6.1 signifies a transition in the Statechart. One can track the sequence of interactions in the primary scenario represented in Figure 6.2 through the Statechart. The first interaction between the user and the controller through the doGet event can be mapped to the first transition of the Statechart (first row of Table 6.1). The subsequent get_class_schemes internal event in the sequence diagram maps to the second transition of the Statechart (second row of Table 6.1). Other events in the sequence diagram can be tracked in a similar manner.

**Table 6.1 Statechart for the controller of the multi-classification browsing service**

**Initial state**: MainMenu, **SMs:**
MultiClasssificationBrowsingImpl mcb, MultClassUtilitiesImpl mui, UIMFSM uimfsm

| Current state | Event | Next State | Action |
|---|---|---|---|
| MainMenu | doget<br>get_class_schemes | MainMenu<br>SchemeRetrieval | Uimfsm.parseRequest<br>mcb.getClassSchemes |
| SchemeRetrieval | return_class_schemes | SchemesDisplay | mcb. TransformClassSchemes |
| SchemesDisplay | doget<br>get_class_scheme<br>_categories<br>back | SchemesDisplay<br>CategoryRetrieval<br><br>MainMenu | Uimfsm.parseRequest<br>mcb.getClassSchemeCategories<br><br>uimfsm.getBackToMainMenu |
| CategoryRetrieval | return_Class_Scheme<br>_Categories<br>return_Sub_Categories | CategoryDisplay<br><br>CategoryDisplay | mui.transformCategories<br><br>mui.transformCategories |
| CategoryDisplay | doGet<br>get_OaiIdsAndURLs_<br>InCategory<br>get_Sub_Categories<br>back | CategoryDisplay<br>OaiIdRetrieval<br><br>CategoryRetrieval<br>SchemesDisplay | uimfsm.parserequest<br>mcb.<br>GetOaiIdsAndUrlsInCategory<br>mcb.getsubcategories<br>uimfsm.getbacktoschemes<br>display |
| OaiIdRetrieval | return_OaiIds<br>AndURLs_InCategory | OaiIdDisplay | mui.transformUrls |
| OaiIdDisplay | doGet<br>get_Metadata<br>get_List_Metadata<br>back | OaiIdDisplay<br>MetadataRetrieval<br>MetadataRetrieval<br>CategoryDisplay | uimfsm.parseRequest<br>mcb.getMetadata<br>mcb.getListMetadata<br>uimfsm.getBackToCategoryDis<br>play |
| MetadataRetrieval | return_Metadata<br>return_List_Metadata | MetadataDisplay<br>MetadataDisplay | mui.transformMetadata<br>mui.transformMetadata |
| MetadataDisplay | doGet<br>back | MetadataDisplay<br>OaiIdDisplay | uimfsm.parseRequest<br>uimfsm.getBackToOaiIdDispla<br>y |

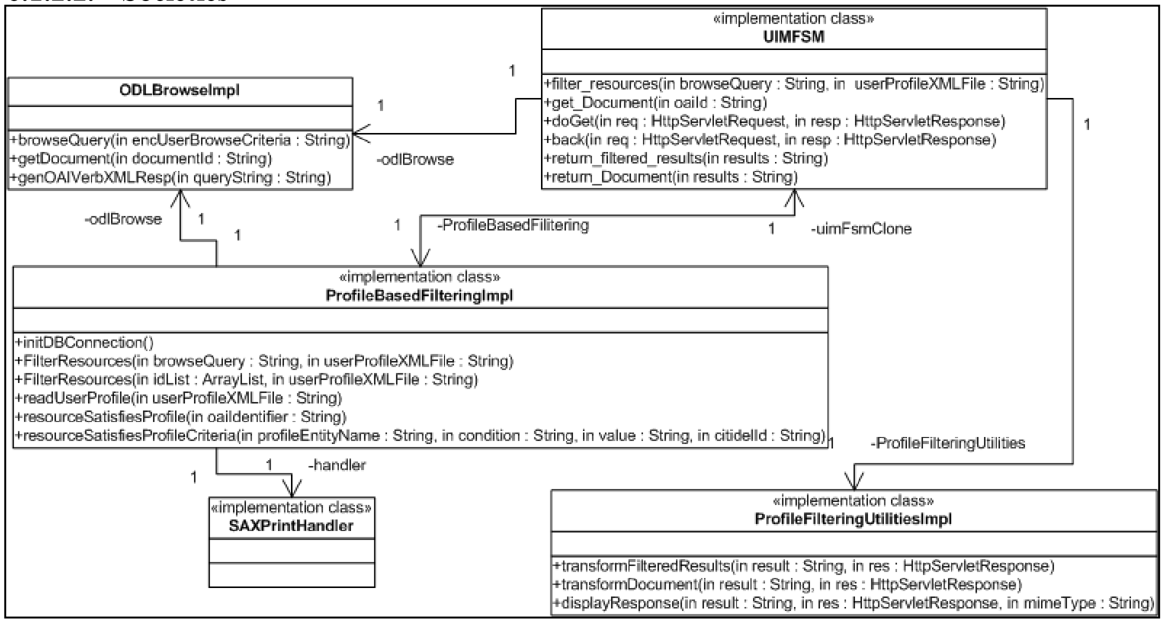
### 6.1.2. CITIDEL: Profile Based Filtering Service

The purpose of modeling this service is to illustrate the 5SLGen approach of reusing existing components wherever possible, and building on their functionalities to implement composite DL services.

#### 6.1.2.1. Functionality

The CITIDEL profile based filtering service accommodates multiple user sub-communities by means of filtering profiles that tell the filtering system what view of the content should be provided to specific users. Each user's profile maps onto a particular view of the records in the CITIDEL database. The profile is constituted from a vocabulary defined by CITIDEL. The user does not see the records in the CITIDEL database that lack filtering element metadata.

#### 6.1.2.2. Societies



«implementation class»
UIMFSM
+filter_resources(in browseQuery : String, in userProfileXMLFile : String)
+get_Document(in oaiId : String)
+doGet(in req : HttpServletRequest, in resp : HttpServletResponse)
+back(in req : HttpServletRequest, in resp : HttpServletResponse)
+return_filtered_results(in results : String)
+return_Document(in results : String)
ODLBrowseImpl
+browseQuery(in encUserBrowseCriteria : String)
+getDocument(in documentId : String)
+genOAIVerbXMLResp(in queryString : String)
-odlBrowse
-ProfileBasedFilitering
-uimFsmClone
«implementation class»
ProfileBasedFilteringImpl
+initDBConnection()
+FilterResources(in browseQuery : String, in userProfileXMLFile : String)
+FilterResources(in idList : ArrayList, in userProfileXMLFile : String)
+readUserProfile(in userProfileXMLFile : String)
+resourceSatisfiesProfile(in oaiIdentifier : String)
+resourceSatisfiesProfileCriteria(in profileEntityName : String, in condition : String, in value : String, in citidelId : String)
-handler
-ProfileFilteringUtilities
«implementation class»
SAXPrintHandler
«implementation class»
ProfileFilteringUtilitiesImpl
+transformFilteredResults(in result : String, in res : HttpServletResponse)
+transformDocument(in result : String, in res : HttpServletResponse)
+displayResponse(in result : String, in res : HttpServletResponse, in mimeType : String)
1

**Figure 6.3 5SLSocieties model for the profile based filtering service**

This composite service implemented by the ProfileBasedFilteringImpl SM extends the browsing service provided by the ODLBrowse component. The ODLBrowse SM serves as a Java wrapper for the ODLBrowse component.

The ProfileBasedFilteringImpl SM is responsible for implementing the profile based filtering service. The ProfileBasedFilteringImpl SM expects the user profile in the form of an XML file. The user profile is a set of conditions on the metadata of the resource. The schema for the profile is defined using the CITIDEL filtering element metadata vocabulary. The ProfileBasedFilteringImpl SM parses the user profile XML file and generates a set of SQL queries that filter the results from the ODLBrowse component response. Thus, the ProfileBasedFilteringImpl SM builds upon the browsing functionality provided by the ODLBrowse component to offer profile based filtering.

In Figure 6.3, one can see three sets of SMs, the controller (UIMFSM), the view SM (ProfileFilteringImpl), and the model SMs (ProfileBasedFilteringImpl, SAXPrintHandler, ODLBrowseImpl). All the SMs together constitute the 5SFramework classes. This demonstrates the architecture of 5SFramework classes based on the MVC design pattern.

#### 6.1.2.3. Scenarios

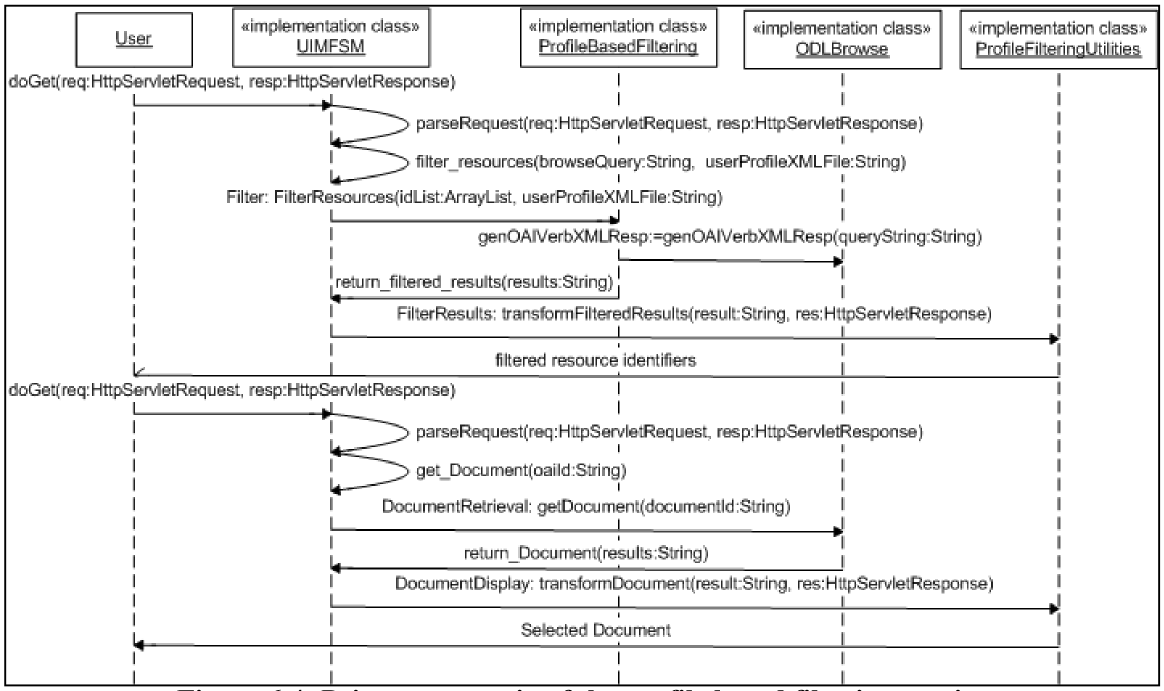

User
«implementation class» UIMFSM
«implementation class» ProfileBasedFiltering
«implementation class» ODLBrowse
«implementation class» ProfileFilteringUtilities
doGet(req:HttpServletRequest, resp:HttpServletResponse)
parseRequest(req:HttpServletRequest, resp:HttpServletResponse)
filter_resources(browseQuery:String, userProfileXMLFile:String)
Filter: FilterResources(idList:ArrayList, userProfileXMLFile:String)
genOAIVerbXMLResp:=genOAIVerbXMLResp(queryString:String)
return_filtered_results(results:String)
FilterResults: transformFilteredResults(result:String, res:HttpServletResponse)
filtered resource identifiers
doGet(req:HttpServletRequest, resp:HttpServletResponse)
parseRequest(req:HttpServletRequest, resp:HttpServletResponse)
get_Document(oaiId:String)
DocumentRetrieval: getDocument(documentId:String)
return_Document(results:String)
DocumentDisplay: transformDocument(result:String, res:HttpServletResponse)
Selected Document

**Figure 6.4 Primary scenario of the profile based filtering service**

Figure 6.4 illustrates the sequence diagram for the primary scenario of the profile based filtering service. The scenario for the profile based filtering service is simple. The user first issues a query for browsing the resources. The ProfileBasedFilteringImpl SM then reads the user profile and filters the browsed resources. After returning a list of filtered resources to the user, the user requests for the metadata of a particular resource. Please note the interaction among the ProfileBasedFiltering object and the ODLBrowse component. The remaining scenarios modeled are for navigation across different states of the controller. The navigation scenarios enable the controller to transition from the DocumentDisplay state to the FilterResults state and from the FilterResults state to the MainMenu state (see Figure 6.5).

#### 6.1.2.4. Controller Statechart

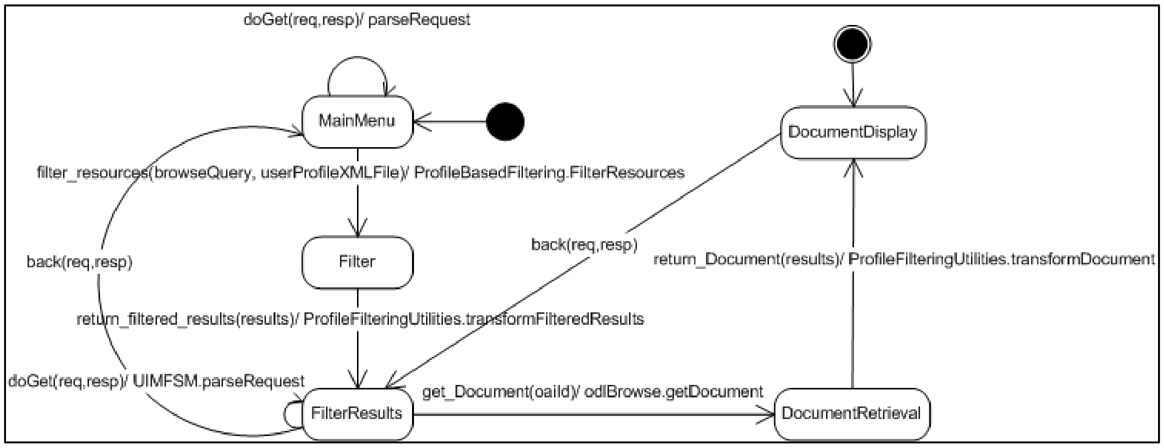

doGet(req,resp)/ parseRequest
MainMenu
DocumentDisplay
filter_resources(browseQuery, userProfileXMLFile)/ ProfileBasedFiltering.FilterResources
back(req,resp)
Filter
back(req,resp)
return_Document(results)/ ProfileFilteringUtilities.transformDocument
return_filtered_results(results)/ ProfileFilteringUtilities.transformFilteredResults
doGet(req,resp)/ UIMFSM.parseRequest
get_Document(oaiId)/ odlBrowse.getDocument
FilterResults
DocumentRetrieval

**Figure 6.5 Statechart for the controller of the profile based filtering service**

Figure 6.5 illustrates the Statechart for the controller of the modeled service. In addition to the primary scenario, the Statechart incorporates scenarios of navigation across different states of the controller. All the interactions in the sequence diagram (see Figure 6.4) can be mapped onto either events or actions of the Statechart. For instance, the doGet event on the controller, initiated by the user, maps to the first event of the Statechart. The parseRequest event, invoked in response to the doGet method call, maps to an action in the Statechart The filter_resources internal event, and the FilterResources event of the ProfileBasedFilteringImpl SM in the sequence diagram, both map to the transition of Statechart from the MainMenu state to the Filter state.

### 6.1.3. CITIDEL: Relevance Feedback Search Service

The purpose of modeling this service is to illustrate the extensibility of services implemented with 5SLGen.

#### 6.1.3.1. Functionality

Relevance feedback is a well-known technique to improve quality of search services. A relevance feedback search service extends a basic search service by allowing the user to refine the search using relevant documents from the basic search results. The selected relevant documents are then used to construct an expanded query, which is then used to retrieve the next set of documents to be presented to the user.

#### 6.1.3.2. Societies

Figure 6.6 shows the relationship between the SMs for the relevance feedback search service. The UIMFSM SM serves as the controller of the service. The RelevanceFeedbackSearchImpl SM implements the relevance feedback service and extends the ODLSearchImpl SM that serves as the wrapper for the ODLSearch component providing the basic search service. The RelevanceFeedbackUtilitiesImpl SM transforms the results returned by the RelevanceFeedbackSearchImpl SM and presents them to the users. It serves as the view of the DL service. The RelevanceFeedbackSearchImpl SM extends the basic functionality provided by the ODL component to implement relevance feedback.

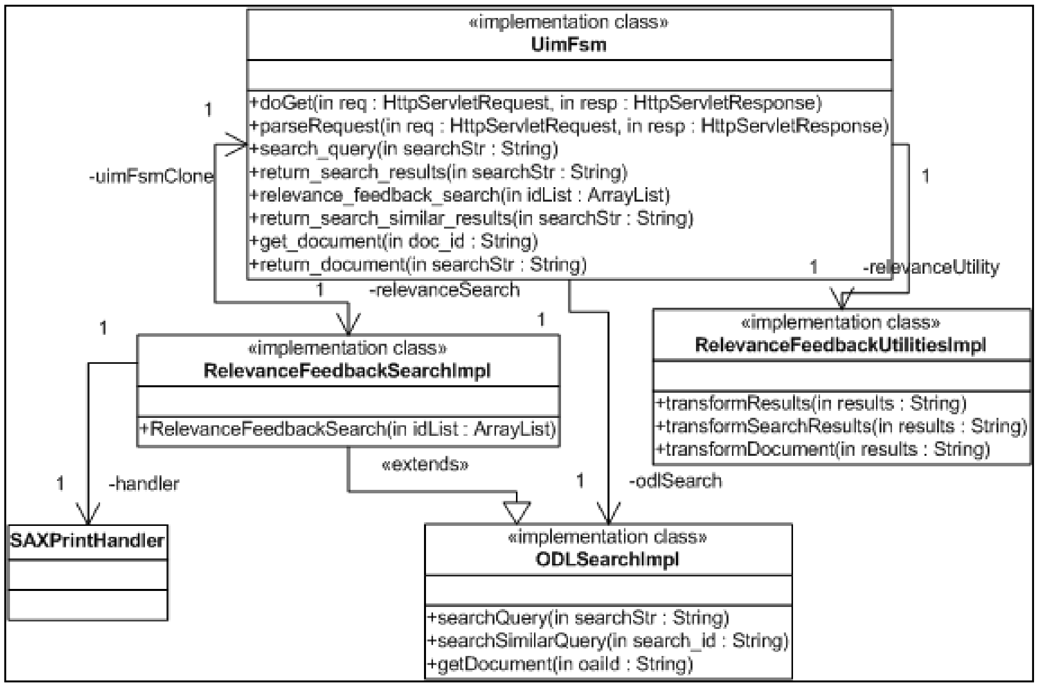

«implementation class»
UimFsm
+doGet(in req : HttpServletRequest, in resp : HttpServletResponse)
+parseRequest(in req : HttpServletRequest, in resp : HttpServletResponse)
+search_query(in searchStr : String)
+return_search_results(in searchStr : String)
+relevance_feedback_search(in idList : ArrayList)
+return_search_similar_results(in searchStr : String)
+get_document(in doc_id : String)
+return_document(in searchStr : String)
-uimFsmClone
-relevanceSearch
-relevanceUtility
«implementation class»
RelevanceFeedbackSearchImpl
+RelevanceFeedbackSearch(in idList : ArrayList)
«implementation class»
RelevanceFeedbackUtilitiesImpl
+transformResults(in results : String)
+transformSearchResults(in results : String)
+transformDocument(in results : String)
«extends»
-handler
-odlSearch
SAXPrintHandler
«implementation class»
ODLSearchImpl
+searchQuery(in searchStr : String)
+searchSimilarQuery(in search_id : String)
+getDocument(in oaiId : String)

**Figure 6.6 5SLSocieties model for the relevance feedback search service**

### 6.1.3.3. Scenarios

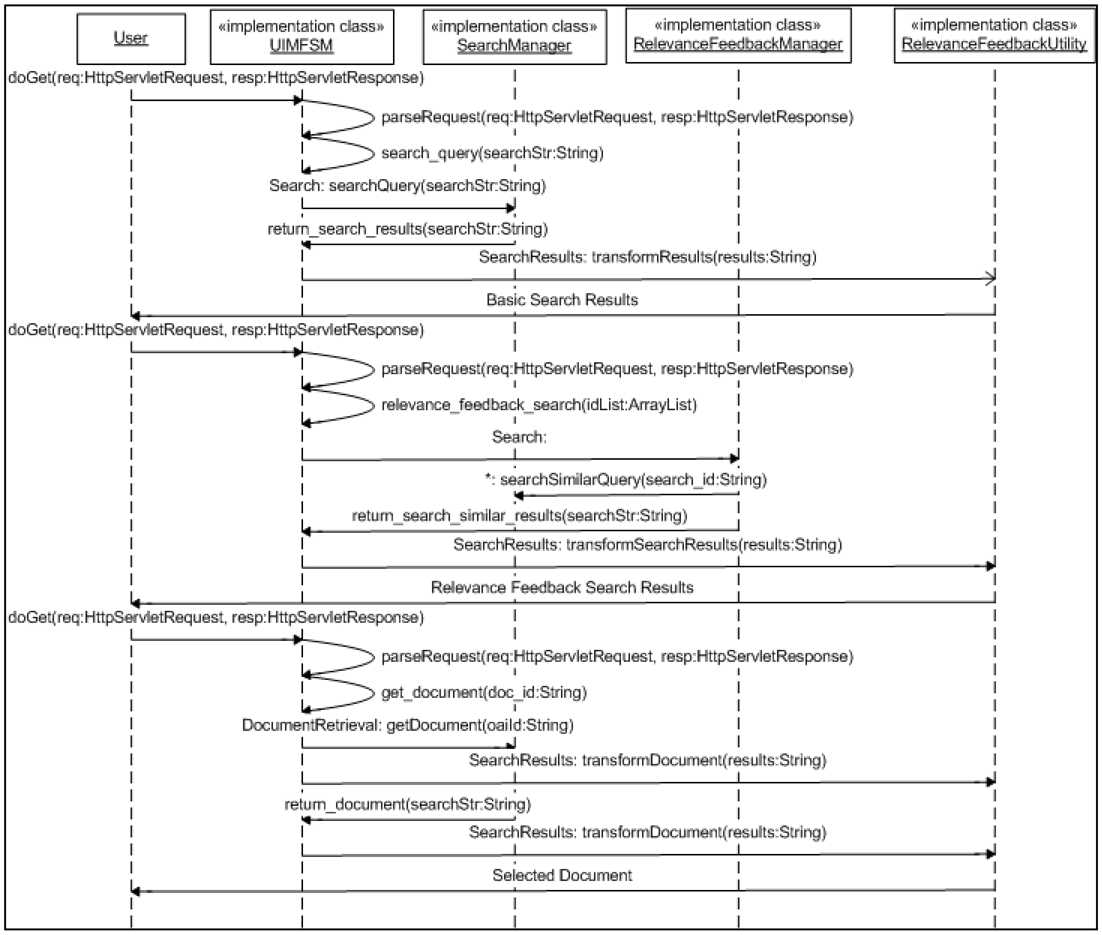

User
«implementation class» UIMFSM
«implementation class» SearchManager
«implementation class» RelevanceFeedbackManager
«implementation class» RelevanceFeedbackUtility
doGet(req:HttpServletRequest, resp:HttpServletResponse)
parseRequest(req:HttpServletRequest, resp:HttpServletResponse)
search_query(searchStr:String)
Search: searchQuery(searchStr:String)
return_search_results(searchStr:String)
SearchResults: transformResults(results:String)
Basic Search Results
doGet(req:HttpServletRequest, resp:HttpServletResponse)
parseRequest(req:HttpServletRequest, resp:HttpServletResponse)
relevance_feedback_search(idList:ArrayList)
Search:
*: searchSimilarQuery(search_id:String)
return_search_similar_results(searchStr:String)
SearchResults: transformSearchResults(results:String)
Relevance Feedback Search Results
doGet(req:HttpServletRequest, resp:HttpServletResponse)
parseRequest(req:HttpServletRequest, resp:HttpServletResponse)
get_document(doc_id:String)
DocumentRetrieval: getDocument(oaiId:String)
SearchResults: transformDocument(results:String)
return_document(searchStr:String)
SearchResults: transformDocument(results:String)
Selected Document

**Figure 6.7 Primary scenario of the relevance feedback search service**

The scenario (see Figure 6.7) shows the primary scenario of the service. Note that all the events associated with the basic search scenario occur in the relevance feedback scenario, in addition to other events responsible for the relevance feedback. This is the simple scenario wherein the user first issues a basic search query. On receiving the results from the basic search the user selects the relevant documents and issues a relevance feedback search query. On receipt of the relevance feedback search results, the user retrieves the metadata of the document of interest. The rest of the scenarios modeled are for user navigation across different controller states.

#### 6.1.3.4. Controller-Statechart

We present the Statechart of the relevance feedback service in a tabular form (see Table 6.2). The Statechart represented in the table defines the permissible actions for the user at different stages in the retrieval process. For instance, the user cannot issue a relevance feedback search query before doing a basic search as the relevance_feedback_search event is permitted only in the SearchResults state. The SearchResults state can only be entered upon after the user has completed a basic search. Thus, the controller-Statechart provides a mechanism to enforce valid user behavior.

**Table 6.2 Statechart for the controller of the relevance feedback search service**

| **Initial state**: MainMenu, **SMs:** RelevanceFeedbackSearchImpl rfb, RelevanceFeedbackUtilitiesImpl rui, UIMFSM uimfsm, ODLSearchImpl odl | | | |
|---|---|---|---|
| **Current state** | **Event** | **Next State** | **Action** |
| | | | |
| MainMenu | doget<br>search_query | MainMenu<br>Search | uimfsm.parseRequest<br>odl.searchQuery |
| | | | |
| Search | return_search_results<br><br>return_search_similar_results | SearchResults<br><br>SearchResults | rui.transformResults<br><br>rui.TransformRelevance Results |
| | | | |
| SearchResults | doget<br>get_Document<br>search_query<br>relevance_feedback_search<br>back | SearchResults<br>DocumentRetrieval<br>Search<br>Search<br>MainMenu | uimfsm.parseRequest<br>odl.getDocument<br>odl.searchQuery<br>rfb. relevanceFeedbackSearch<br>uimfsm.getBackToMainMenu |
| | | | |
| DocumentRetrieval | return_document | SearchResults | rui.transformDocument |
| | | | |

### 6.1.4. CITIDEL: Binder Service

The purpose of modeling the Binder service is to complete the set of services needed for implementing the CITIDEL with 5SLGen.

#### 6.1.4.1. Functionality

The Binder service provides a personalized space for the user where he can maintain the resources of his interest. A user can only access his binder after authenticating himself. The service allows a user to: insert resources into the binder, delete resources from the binder, and view all the resources of the binder. The resources used for creation of an instructional activity in VIADUCT [85] can only be inserted from the binder. Every user in CITIDEL starts off with an empty binder, which is later populated, with resources of interest. The binder service serves to bind the services offered by VIADUCT to CITIDEL.

#### 6.1.4.2. Societies

The SMs that implement the Binder service can be classified as the controller SM (UIMFSM), and the model SMs (BinderServiceImpl, AuthenticationServiceImpl). The model SMs expose functionalities for user authentication and binder actions such as insertion, deletion, and listing. This service assumes the existence of a user database that can be used for user authentication. The 5SLSocieties model for the binder service is shown in Figure 6.8.

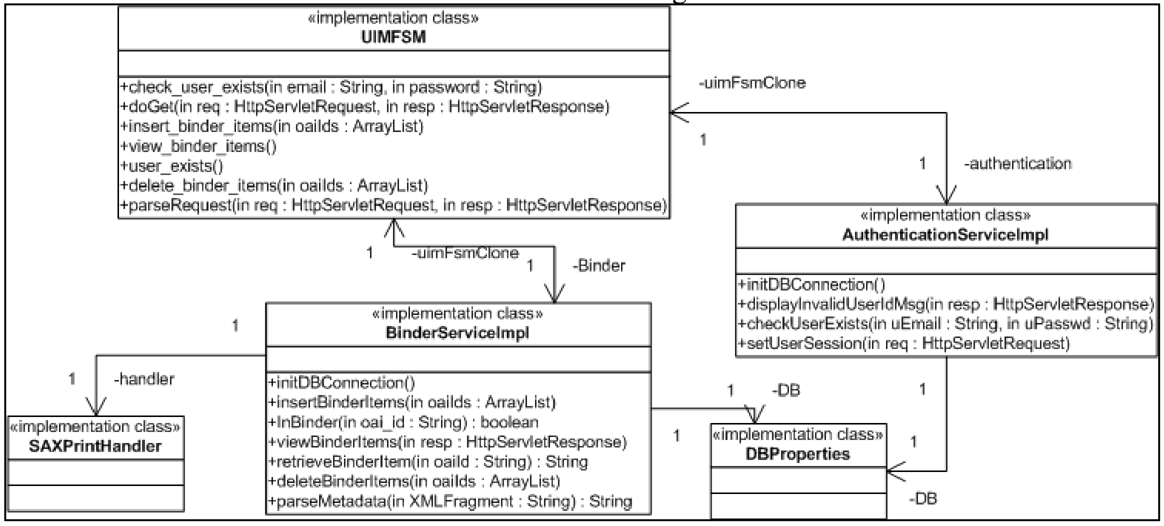

«implementation class»
UIMFSM
+check_user_exists(in email : String, in password : String)
+doGet(in req : HttpServletRequest, in resp : HttpServletResponse)
+insert_binder_items(in oaiIds : ArrayList)
+view_binder_items()
+user_exists()
+delete_binder_items(in oaiIds : ArrayList)
+parseRequest(in req : HttpServletRequest, in resp : HttpServletResponse)
-uimFsmClone
1
1
-authentication
«implementation class»
AuthenticationServiceImpl
+initDBConnection()
+displayInvalidUserIdMsg(in resp : HttpServletResponse)
+checkUserExists(in uEmail : String, in uPasswd : String)
+setUserSession(in req : HttpServletRequest)
1
-uimFsmClone
1
-Binder
«implementation class»
BinderServiceImpl
+initDBConnection()
+insertBinderItems(in oaiIds : ArrayList)
+InBinder(in oai_id : String) : boolean
+viewBinderItems(in resp : HttpServletResponse)
+retrieveBinderItem(in oaiId : String) : String
+deleteBinderItems(in oaiIds : ArrayList)
+parseMetadata(in XMLFragment : String) : String
1
1
-handler
«implementation class»
SAXPrintHandler
1
-DB
1
1
«implementation class»
DBProperties
1
-DB

**Figure 6.8 5SLSocieties model for the binder service**

#### 6.1.4.3. Scenarios

We have modeled four scenarios that capture all the possible means of using the binder service. Please note that the word resource and item in this context mean the same thing, i.e., a digital object from the CITIDEL.

1. The user authenticates himself and views all the items in the binder (see Figure 6.9).
2. The user authenticates himself and inserts some items in the binder.
3. The user authenticates himself and deletes some items in the binder.
4. The user is not authenticated.

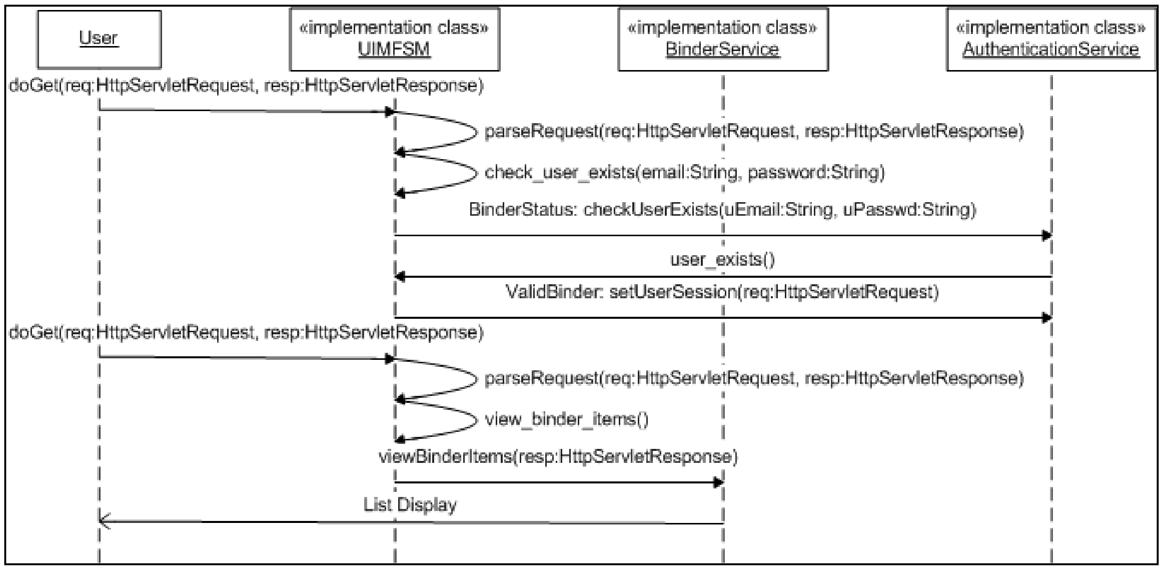

User
«implementation class» UIMFSM
«implementation class» BinderService
«implementation class» AuthenticationService
doGet(req:HttpServletRequest, resp:HttpServletResponse)
parseRequest(req:HttpServletRequest, resp:HttpServletResponse)
check_user_exists(email:String, password:String)
BinderStatus: checkUserExists(uEmail:String, uPasswd:String)
user_exists()
ValidBinder: setUserSession(req:HttpServletRequest)
doGet(req:HttpServletRequest, resp:HttpServletResponse)
parseRequest(req:HttpServletRequest, resp:HttpServletResponse)
view_binder_items()
viewBinderItems(resp:HttpServletResponse)
List Display

**Figure 6.9 First primary scenario of the binder service**

### 6.1.4.4. Statechart

The Statechart of the Binder service contains three states, i.e., MainMenu, BinderStatus, and ValidBinder (see Table 6.3). The controller of the service starts in the MainMenu state. The controller transitions from the MainMenu state to the ValidBinder state once the user is authenticated. In this state, the user can insert, delete, or view the binder resources. BinderStatus serves as the intermediate state in the transition to the ValidBinder state.

**Table 6.3 Statechart for the controller of the binder service**

| **Initial state**: MainMenu,<br>**SMs:** BinderServiceImpl bsi, AuthenticationServiceImpl asi, UIMFSM uimfsm | | | |
|---|---|---|---|
| **Current state** | **Event** | **Next State** | **Action** |
| | | | |
| MainMenu | doget<br>check_user_exists | MainMenu<br>BinderStatus | uimfsm.parseRequest<br>asi.checkUserExists |
| | | | |
| BinderStatus | user_exists<br>user_does_not_exist | ValidBinder<br>MainMenu | <br>asi.<br>displayInvalidUserIdMsg |
| | | | |
| ValidBinder | doget<br>view_binder_items<br>insert_binder_items<br>delete_binder_items | ValidBinder<br>ValidBinder<br>ValidBinder<br>ValidBinder | uimfsm.parseRequest<br>bsi.viewBinderItems<br>bsi.insertBinderItems<br>bsi.deleteBinderItems |

## 6.2. DLs Implemented using 5SLGen

### 6.2.1. Union Catalog

The purpose of modeling this DL is to demonstrate the ease of implementing a DL that exposes all elementary services through the components of the component pool. This was the first DL implemented using 5SLGen and serves as proof of concept for the modeling and generation process.

#### 6.2.1.1. Functionality

The Union Catalog DL provides for searching and browsing the items in the catalog. This catalog is created through the process of metadata harvesting using the OAI-PMH protocol. The searching and browsing services are provided by the ODLBrowse and ODLSearch ODL components from the component pool.

#### 6.2.1.2. Societies

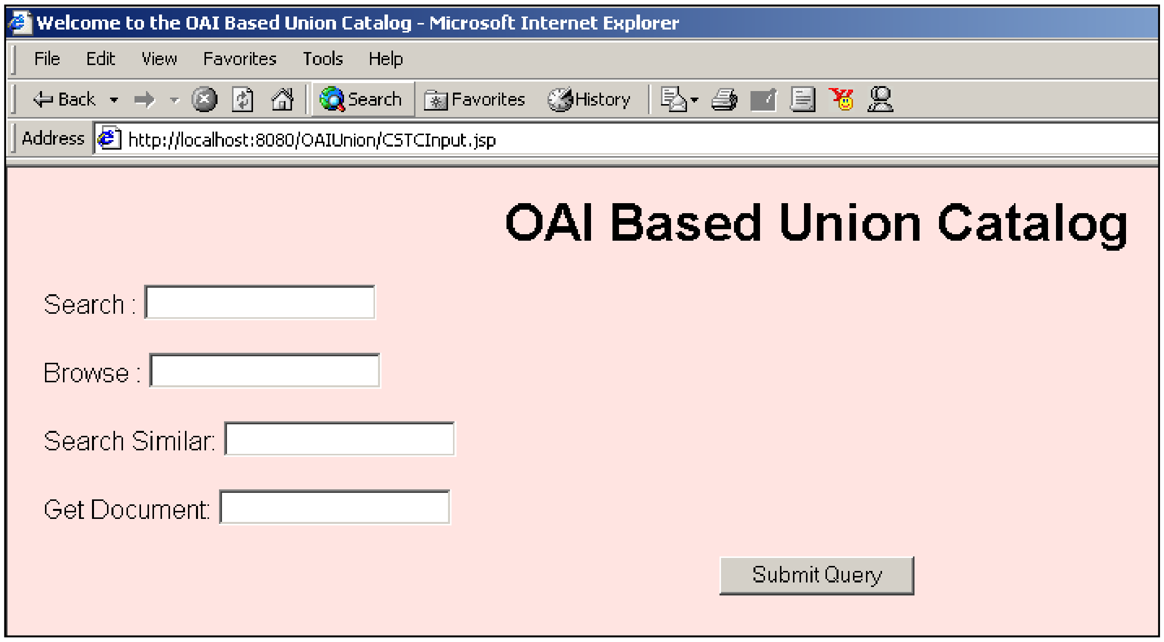
The 5SLSocieties model of this DL consists of the controller SM, the view SM, and the component wrapper SMs. The DL designer only needs to write code to implement the skeleton SMs that constitute the view of the DL and modify the generated controller SM to parse user requests. The component wrapper SMs are used as-is in the implementation.

#### 6.2.1.3. Scenarios

This DL is modeled with scenarios for searching the resources in the catalog, browsing the resources in the catalog, retrieving a particular document from the catalog, and user navigation.

#### 6.2.1.4. Screenshot

Figure 6.10 shows the main menu of the DL. The search service expects a word input. Other options also can be specified such as presence or absence of terms. The search service follows the odlsearch1 protocol that defines the precise semantics of the search input. The search-similar service implements relevance-feedback search and expects multiple record-identifiers as input. The precise semantics is defined by the odlsearch2 protocol. The browse service expects browse queries as defined by the odlbrowse protocol. The document retrieval service retrieves any valid resource in the catalog that has an OAI-identifier. The ODL component protocols are defined in [11].


Welcome to the OAI Based Union Catalog - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://localhost:8080/OAIUnion/CSTCInput.jsp

OAI Based Union Catalog

Search :

Browse :

Search Similar:

Get Document:

Submit Query

**Figure 6.10 Screenshot of the Union Catalog DL**

### 6.2.2. CITIDEL

The purpose of implementing CITIDEL using 5SLGen is to demonstrate the construction of existing DLs using the 5SFramework. The modeling and implementation of 5SLGen represents a union of all the modeling and generation efforts of its composite services. In the previous sections, we have modeled and implemented all the composite services of CITIDEL using 5SLGen. We now integrate all our 5SLSocieties and 5SLScenarios models created in the previous phase to create a single model for CITIDEL that reuses these services and serves as the input to 5SLGen.

#### 6.2.2.1. Functionality

The CITIDEL implementation represents an aggregation of all the services modeled in this work. The functionality exposed by CITIDEL includes

1. Multi-Classification Browsing
2. Profile Based Filtering
3. Basic Keyword Search
4. Relevance Feedback Search
5. Binder

#### 6.2.2.2. Societies

The 5SLSocieties model of CITIDEL includes all the SMs that constitute the application logic of its component services. In addition to this, the CITIDEL implementation has SMs that are responsible for the control and the view of the CITIDEL interface. The list of the SMs that implements CITIDEL and the services they expose is provided below. The numbers in the right column match the functionalities listed in section 6.2.2.1.

**Table 6.4 SMs and their roles in the CITIDEL implementation**

| SMs | Service |
|---|---|
| RelevanceFeedbackSearchImpl | Relevance Feedback Search (4) |
| ProfileBasedFilteringImpl<br>UserProfile | Profile Based Filtering (2) |
| MultiClassificationBrowsingImpl | Multi-classification browsing (1) |
| AuthenticationServiceImpl | User Authentication |
| BinderServiceImpl | Binder Service (5) |
| ODLBrowseImpl | Flat Browsing based on metadata fields |
| ODLSearchImpl | Basic Search (3) |
| UIMFSM | Controller |
| RelevanceFeedbackUtilitiesImpl<br>ProfileBasedUtilitiesImpl<br>MultiClassUtilitiesImpl | View |
| Debug, DBProperties, SAXPrintHandler | Helpers |

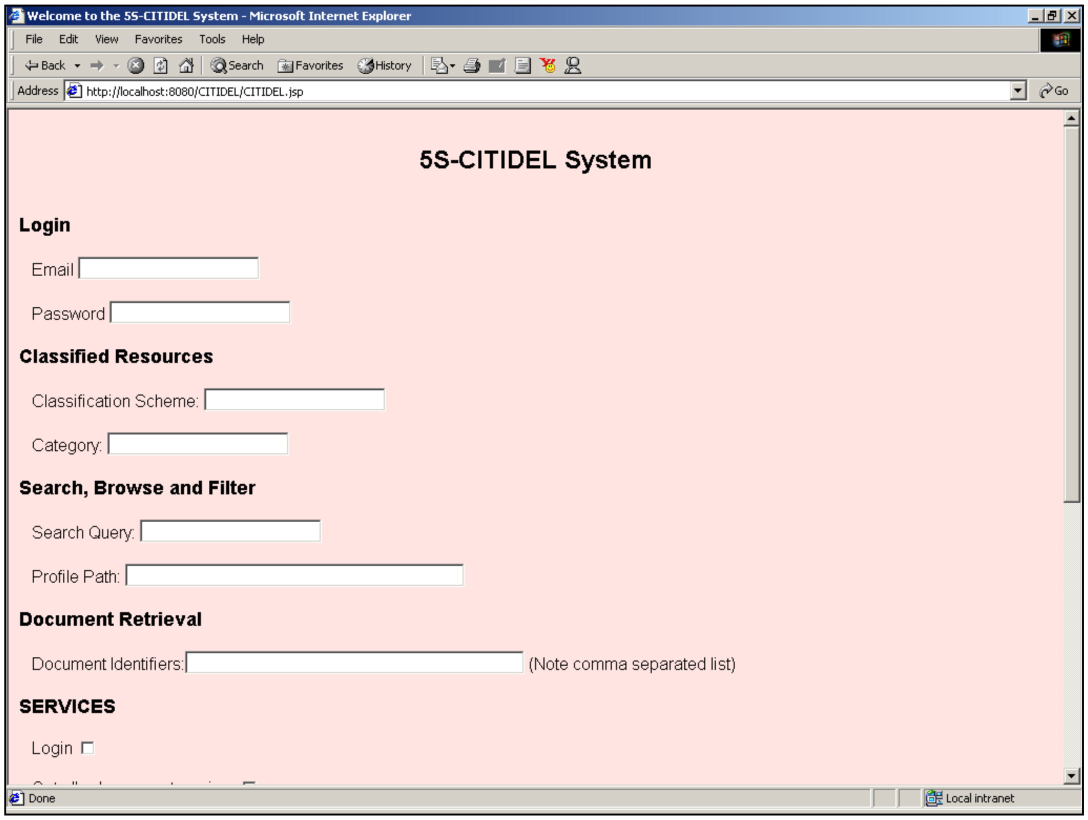
### 6.2.2.3. Scenarios

The scenarios of CITIDEL represent all the possible ways in which the user can interact with the system. We have ensured that the controller-Statechart allows all user actions that are permitted in the real world CITIDEL. The 5SLScenarios model of CITIDEL is a union of the scenarios of its composite services and scenarios of navigation among its composite services.

The large number of scenarios in the 5SLScenarios model makes it impractical to include them in the body of this thesis document. For the controller Statechart of CITIDEL, please refer to appendix B**.**

### 6.2.2.4. Screenshot

Figure 6.11 shows the user-interface of the CITIDEL implementation. The first screenshot illustrates all the input fields, which are required by the services exposed. The second screenshot illustrates all the services and sub-services exposed. For instance, the “Get all schemes categories”, “Get all categories within a scheme”, “Get All sub-categories within the scheme”, and the “Get All record-identifiers/URLs within the category” checkboxes and their associated input text fields expose the CITIDEL multi-classification browsing service.


Welcome to the 5S-CITIDEL System - Microsoft Internet Explorer

Address http://localhost:8080/CITIDEL/CITIDEL.jsp

5S-CITIDEL System

**Login**

Email

Password

**Classified Resources**

Classification Scheme:

Category:

**Search, Browse and Filter**

Search Query:

Profile Path:

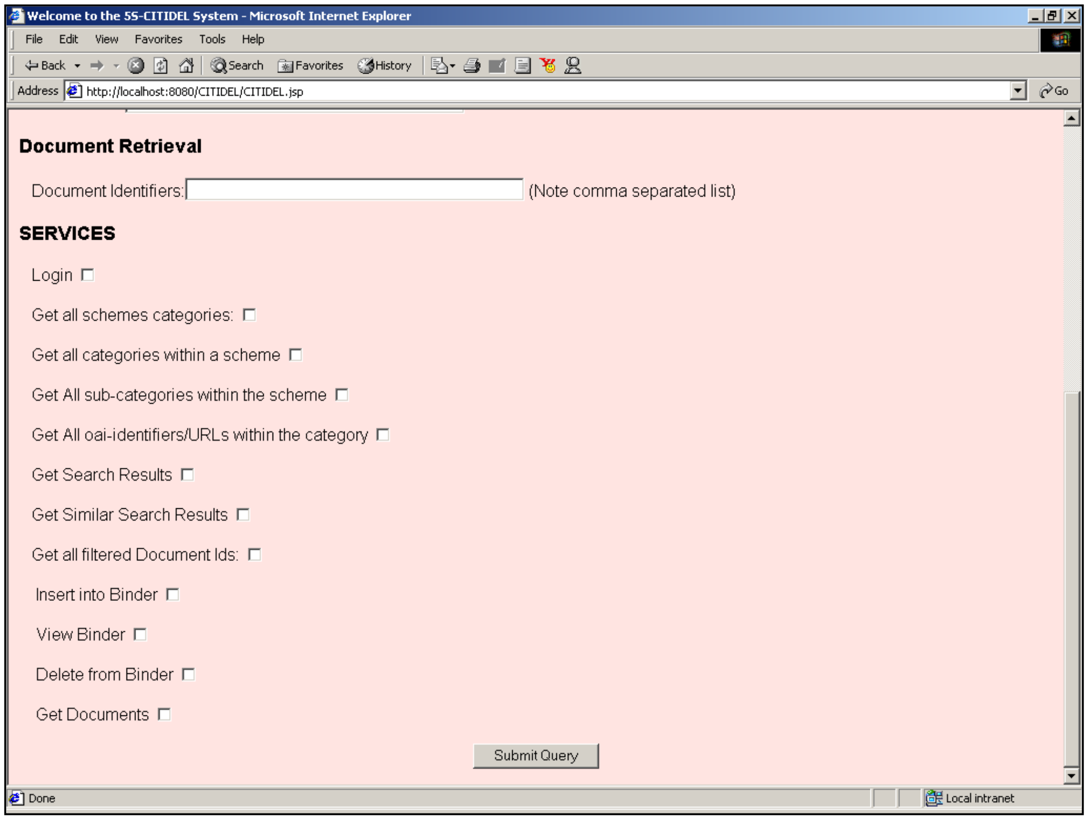
**Document Retrieval**

Document Identifiers: (Note comma separated list)

**SERVICES**

Login ☐

Done | Local intranet


Welcome to the 5S-CITIDEL System - Microsoft Internet Explorer

Address http://localhost:8080/CITIDEL/CITIDEL.jsp

**Document Retrieval**

Document Identifiers: (Note comma separated list)

**SERVICES**

Login ☐

Get all schemes categories: ☐

Get all categories within a scheme ☐

Get All sub-categories within the scheme ☐

Get All oai-identifiers/URLs within the category ☐

Get Search Results ☐

Get Similar Search Results ☐

Get all filtered Document Ids: ☐

Insert into Binder ☐

View Binder ☐

Delete from Binder ☐

Get Documents ☐

Submit Query

Done | Local intranet

**Figure 6.11 Screenshot of the CITIDEL interface**

### 6.2.3. VIADUCT

The purpose of implementing the VIADUCT DL is to complete the implementation of CITIDEL and all its dependent DLs. This example also demonstrates the reusability and the extensibility of services in the 5SFramework.

#### 6.2.3.1. Functionality

The VIADUCT DL provides for the creation of instructional activity for teachers. An instructional activity can be a lesson plan, exercise, laboratory plan, or any related type of activity that would be useful in teaching. All resources for a VIADUCT instructional activity are CITIDEL resources. Before inclusion in an instructional activity, they have to be present in the user's binder.

#### 6.2.3.2. Societies

VIADUCT is implemented by the ActivitiesImpl and Activity SMs. The AuthenticationServiceImpl SM service created while implementing the binder service is reused here to provide the authentication service. The rest of the SMs serve as the controller, view, and helpers of the implemented DL. No ODL components are used in the implementation of VIADUCT. The ActivitiesImpl and Activity Java skeleton classes created by 5SLGen have to be fleshed out by the DL designer. The VIADUCT implementation is responsible for creation of the activity, and saving the activity to the CITIDEL database.

#### 6.2.3.3. Scenarios

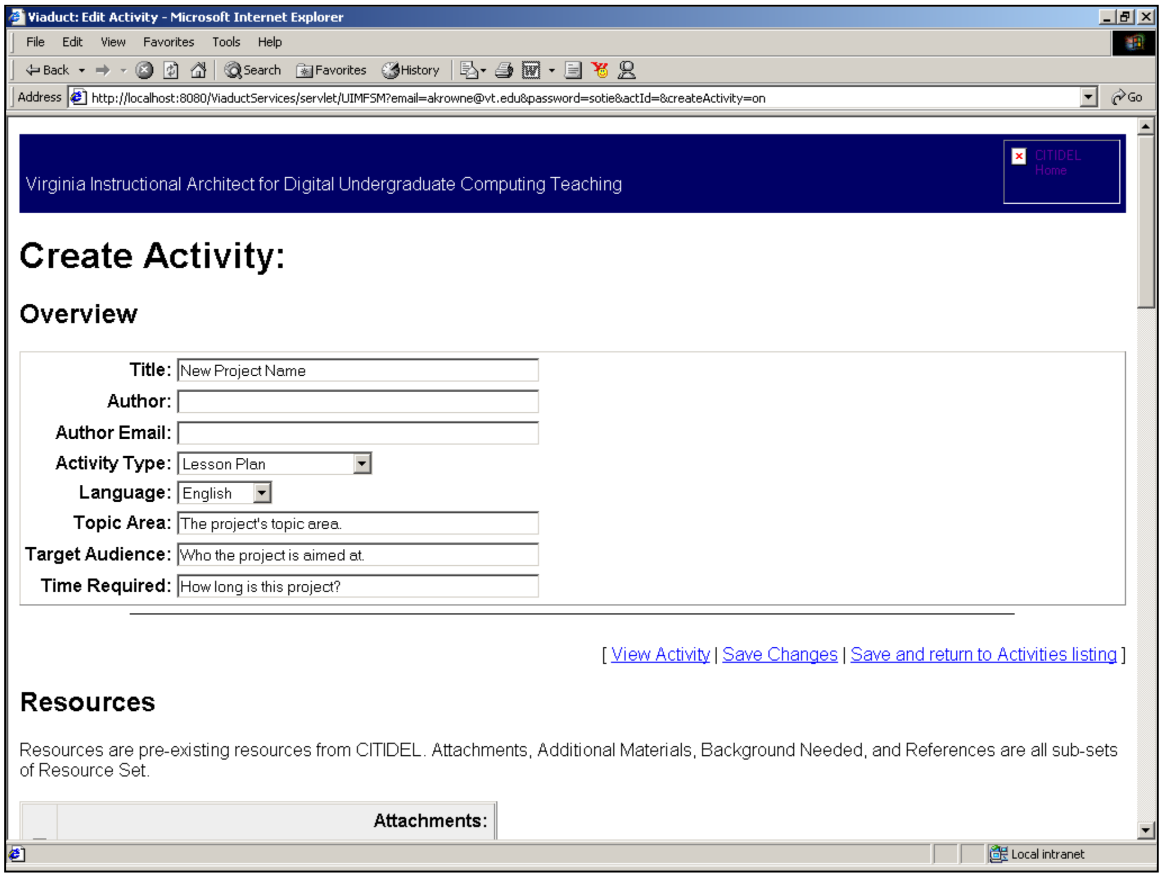
The scenarios modeled for VIADUCT are listed below:

- The user logs in, creates a new activity, and saves the instructional activity.
- The user logs in and sees a listing of the current instructional activities.
- The user logs in and sees a particular instructional activity.
- Plumbing scenario to complete the Statechart for the controller.

Other scenarios of usage of VIADUCT, in conjunction with CITIDEL, are elaborated in [88]. The emphasis in [88] is on the process of seeking resources and creation of a specific instructional activity. In this document we have attempted to differentiate the functionalities of VIADUCT from CITIDEL, so that VIADUCT can be abstracted as a component and used in the modeling and implementation of other DLs.

### 6.2.3.4. Screenshot

Figure 6.12 illustrates the user-interface for the activity creation sub-service of the VIADUCT DL.



Viaduct: Edit Activity - Microsoft Internet Explorer

http://localhost:8080/ViaductServices/servlet/UIMFSM?email=akrowne@vt.edu&password=sotie&actId=&createActivity=on

Virginia Instructional Architect for Digital Undergraduate Computing Teaching

Create Activity:

Overview

| | |
|---|---|
| Title: | New Project Name |
| Author: | |
| Author Email: | |
| Activity Type: | Lesson Plan |
| Language: | English |
| Topic Area: | The project's topic area. |
| Target Audience: | Who the project is aimed at. |
| Time Required: | How long is this project? |

[ View Activity | Save Changes | Save and return to Activities listing ]

Resources

Resources are pre-existing resources from CITIDEL. Attachments, Additional Materials, Background Needed, and References are all sub-sets of Resource Set.

Attachments:

**Figure 6.12 Screenshot of the VIADUCT DL**

## 6.3. Observations on the Modeling and the Generation of DLs

After the generation of 5SFramework classes by 5SLGen and inclusion of component wrappers from the component pool, the DL designer writes code primarily to implement the new functionalities desired by the service or the DL. The time taken to implement a DL is proportional to the number of components in the component pool utilized by the DL. An analysis of the lines of code (LOC) and the time taken to implement the DLs is shown in Table 6.5. Consider the VIADUCT and the Union Catalog DLs. In the case of the Union Catalog, 77% of the total code was reused, whereas there was 43% code reuse with VIADUCT. The time taken to implement VIADUCT was 7 days, whereas the time taken to implement the Union Catalog was 2 days. A similar analysis for CITIDEL reveals similar trends. This proves that the greater the number of components utilized, the less the time taken to implement the DL.

Once a particular service is implemented using 5SLGen, it can be considered as another component in the component pool. Thus, the modeling and generation process with 5S results in an ever-increasing number of components in the component pool, which translates into a reduction in implementation times for constructing the next DL. Each service implemented simplifies the task of generating the next service as components from the previous service can be reused. This sort of piggybacking was seen in the implementation of the VIADUCT system. After implementation of CITIDEL, we could reuse the authentication mechanism of CITIDEL while implementing the VIADUCT system.

As most scenarios are slight variations of one another, a lot of copy-paste activity from other scenarios was observed while creating 5SLScenarios. This contributes to the verbosity of the 5SLScenarios model. The complexity and the verbosity of the 5SLScenarios model indicates the need for a visual modeling tool to construct these models. We envision that 5SGraph could be extended to do this modeling.

**Table 6.5 Evaluation of the generated DLs**

| Metrics | Union Catalog | CITIDEL | VIADUCT |
|---|---|---|---|
| Total lines of code (LOC) | 910 | 2400 | 1003 |
| LOC implemented by the DL designer | 210 | 500 | 574 |
| LOC reused from components and composite services | 700 | 2000 | 429 |
| Time taken | 2 day | 9 days | 7 days |

# 7. Conclusions and Future Work

## 7.1. Conclusions

In this thesis, we have implemented a DL generator that yields a framework of classes that can be customized for implementing DLs and DL services. This DL generator is based on the 5S metamodel for DLs. The 5S metamodel of DLs is derived from the 5S theory, which provides the theoretical foundation for our work. The 5SL models developed for capturing the 5S abstractions represent the requirements of a DL. Our work in particular focuses on the scenarios and societies aspects of DLs. In 5SLGen, societies models were used for generation of the static contextual structure and the scenarios models were used to implement the dynamic behavior of DLs and their services.

### 7.1.1. Contributions of 5SLGen

- The implementation of CITIDEL and other DLs along with their composite services has partially validated the theory of 5S.
- Through this thesis, we have attempted efforts in DL interoperability with DL modeling. This marriage has resulted in the birth of 5SLGen, which introduces a scenario-based approach to the generation of componentized DLs.
- The DL services and DLs implemented using the 5S approach prove that our work is not in the realm of theory alone. The proof of the pudding lies in the eating, and through the implementation of DLs that mirror existing production systems we have shown that high quality complex DLs can be built on the basis of a formal theory.
- We believe that our work with the 5S metamodel, 5SL, and 5SLGEN, and implemented DLs such as CITIDEL, has provided a unifying framework for the specification and generation of DLs.
- We have attempted to loosely couple the different artifacts that are generated in the process of modeling and generation of DLs. This allows the 5S approach for the generation of DLs to use tools other than 5SLGen for the code generation.
- The 5SFramework incorporates the latest developments in the field of DL interoperability and software modeling. This adherence to open standards and established design patterns ensures that our work is extensible and relevant in the field of DLs and software in general.

## 7.2. Future Work

### 7.2.1. Integration of 5SLGen with 5SGraph

Both 5SLGen and 5SGraph are based on the 5S metamodel. 5SLGen is a tool for generation of DL services, whereas 5SGraph is a visual tool for modeling DLs. 5SGraph in its current state does not support the modeling of 5SLScenarios and 5SLSocieties. 5SGraph needs to provide support for the creation and reuse of the 5SLScenarios and 5SLSocieties models to realize the integration of both tools. This integration of 5SGraph and 5SLGen will result in the creation of a complete CASE tool based on the 5S theory that will provide for the complete lifecycle development of DLs.

### 7.2.2. Incorporating the uPortal Framework into the 5SFramework

uPortal is a distributed multi-tiered Internet application framework for developing a web portal and developing content for display within that portal [89]. The uPortal framework consists of a number of uPortal components or portlets communicating together via XML. uPortal has a number of components that provide services such as LDAP authentication, single-sign on, etc. The 5SFramework could incorporate the uPortal framework for the presentation of DL content and include its components into the component pool thereby allowing the 5S approach to build on the advantages offered by this open-source uPortal technology. As uPortal components talk to one another through XML the integration with 5SLGen should be a smooth one.

### 7.2.3. Improvements to the 5SFramework

#### 7.2.3.1. Scalability of the Generated DLs and DL Services

Each DL or DL service implemented using the 5SFramework has a single controller. Thus if two or more users use the service simultaneously the single controller for the service cannot maintain a consistent state. The solution to this is to create an instance of the controller for every user. In this way, every user has his view of the DL. This involves adding another layer to the classes of the 5SFramework. This additional layer will be responsible for instantiation of the controller for every user of the DL.

#### 7.2.3.2. Automated Construction of User-Interfaces

We have previously explored the automatic generation of user-interfaces from scenarios [51]. However, the approaches explored yielded results that were less than satisfactory. Ian Horrocks in [90] provides ideas for constructing the user-interface with Statecharts. As 5SLGen generates Statecharts in the process of DL generation, the application and implementation of Ian’s ideas has the potential to provide an additional dimension of user-interface-prototyping to the 5S approach of generating DL libraries.

#### 7.2.3.3. Support for Transaction Scoping and Error handling

Modern web applications have to carry out conversational transactions with the client. This requires the server to have a firm idea of the client state and a strong knowledge of the transaction boundaries. We need to formulate an approach for scoping and enforcing the transaction boundaries within the presentation layer of the 5SFramework.

In the DL web applications the client, using the browser, can take the controller to an unknown state. The server must be able to cope comfortably with unexpected events from the client because the user has altered the client state using the browser “Back” button or the browser “Goto” bookmark facility. Again, an approach is needed for having the server keep track of the state of the client and still cope adequately with unexpected events from the client. The uidesign.net magazine (in [91]) provides solutions to the above problems, using a rigorous UML Statechart modeling approach to the presentation layer.

#### 7.2.3.4. Web Services

Web services are self-contained, self-describing, modular applications that can be published, located, and invoked across the web. Web services perform functions, ranging from simple requests to complicated business processes. The ODL experiments have shown how web-based services can be used to effectively and efficiently create distributed systems to meet the needs of a particular community [28]. The 5SFramework needs to include such web-based services into the component pool. We envision that, eventually, DLs can be modeled and implemented with the 5SFramework using web services, ODL components, uPortal components, and other componentized systems that communicate in XML.

### 7.2.4. Model Validation

We need to carry out 5SL model validation to ensure the correctness and consistency of the generated DLs. This will involve comparison of the real systems and the modeled systems on a number of metrics including time for implementation, complexity of generated code, functionality, maintainability, reliability, etc. An alternative way to perform model validation would be to carry out Wizard of Oz tests [92].

### 7.2.5. Personalization of the 5S Approach for Generation of DLs

Personalization entails customizing information access, structure, and presentation to the DL end-user. PIPE is an approach to personalizing information-seeking interactions by transforming programmatic representations. PIPE models personalization as a form of partial evaluation [93], an automatic technique for specializing programs given some of their input. PIPE models can be mapped onto 5SL models allowing the 5S approach to bring the full power of transformation to bear upon diverse information resources. Integration of PIPE models into 5SL has been pursued in [94]. The implementation of 5SLGen needs to adapt to accept personalized5SL DL models as input so as to generate personalized 5SFramework classes that can be customized to implement a personalizable DL (see Figure 7.1 taken from [95]).

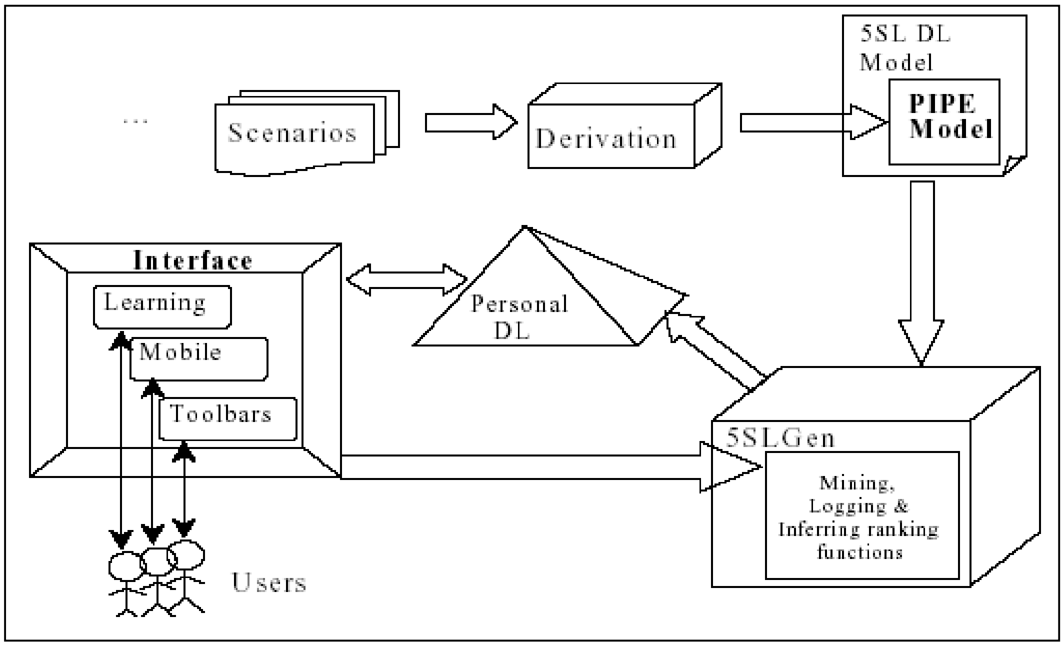


**Figure 7.1 Personalization in DLs**

## Appendix A. XML Schemas

### A.1. XML Schema for the 5SLSocieties model

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">

    <!-- The Schema Starts here -->
    <xs:element name="Societies5SL">
        <xs:complexType>

            <!-- List of Service Managers -->
            <xs:sequence>
                <xs:element name="TaggedValue" type="TaggedValueType" minOccurs="0"/>
                <xs:choice minOccurs="0" maxOccurs="unbounded">
                    <!-- Each Service Manager -->
                    <xs:element name="ServiceManager">
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element name="TValue" type="TValueType" minOccurs="0"/>
                                <xs:choice minOccurs="0" maxOccurs="unbounded">
                                    <xs:element name="Attribute" type="AttributeType"/>
                                    <xs:element name="Operation" type="OperationType"/>
                                    <xs:element name="Extends" type="GeneralizationType"/>
                                    <xs:element name="Association" type="AssociationType"/>
                                    <xs:element name="Dependency" type="DependencyType"/>
                                </xs:choice>
                            </xs:sequence>
                            <xs:attribute name="NAME" type="xs:string" use="required"/>
                            <xs:attribute name="TYPE" type="xs:string" use="required"/>
                            <xs:attribute name="ABSTRACT" default="false">
                                <xs:simpleType>
                                    <xs:restriction base="xs:NMTOKEN">
                                        <xs:enumeration value="true"/>
                                        <xs:enumeration value="false"/>
                                    </xs:restriction>
                                </xs:simpleType>
                            </xs:attribute>
                            <xs:attribute name="VISIBILITY" use="optional">
                                <xs:simpleType>
                                    <xs:restriction base="xs:NMTOKEN">
                                        <xs:enumeration value="public"/>
                                        <xs:enumeration value="private"/>
                                    </xs:restriction>
                                </xs:simpleType>
                            </xs:attribute>
                        </xs:complexType>
                    </xs:element>
                </xs:choice>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <!-- Documentation for the Service Manager and the 5SLSocieties file -->
    <xs:complexType name="TaggedValueType">
```

```
        <xs:sequence minOccurs="0" maxOccurs="unbounded">
            <xs:element name="Tag" type="xs:string"/>
            <xs:element name="Value" type="xs:string"/>
        </xs:sequence>
</xs:complexType>
```

**<!-- Attribute -->**

```
<xs:complexType name="AttributeType">
        <xs:attribute name="VISIBILITY" use="required">
            <xs:simpleType>
                <xs:restriction base="xs:NMTOKEN">
                    <xs:enumeration value="public"/>
                    <xs:enumeration value="protected"/>
                    <xs:enumeration value="private"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="TYPE" type="xs:string" use="required"/>
        <xs:attribute name="NAME" type="xs:string" use="required"/>
        <xs:attribute name="INITVAL" type="xs:string"/>
        <xs:attribute name="CONSTRAINT" default="changeable">
            <xs:simpleType>
                <xs:restriction base="xs:NMTOKEN">
                    <xs:enumeration value="changeable"/>
                    <xs:enumeration value="addOnly"/>
                    <xs:enumeration value="frozen"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="CLASS-SCOPE" default="false">
            <xs:simpleType>
                <xs:restriction base="xs:NMTOKEN">
                    <xs:enumeration value="true"/>
                    <xs:enumeration value="false"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="MULTIPLICITY" type="xs:string"/>
</xs:complexType>
```

**<!-- Operation -->**

```
<xs:complexType name="OperationType">
        <xs:choice minOccurs="0" maxOccurs="unbounded">
            <xs:element name="Parameter">
                <xs:complexType>
                    <xs:attribute name="TYPE" type="xs:string" use="required"/>
                    <xs:attribute name="NAME" type="xs:string" use="required"/>
                    <xs:attribute name="DEFAULTVAL" type="xs:string"/>
                </xs:complexType>
            </xs:element>
        </xs:choice>
        <xs:attribute name="VISIBILITY">
            <xs:simpleType>
                <xs:restriction base="xs:NMTOKEN">
                    <xs:enumeration value="public"/>
                    <xs:enumeration value="protected"/>
                    <xs:enumeration value="private"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="NAME" type="xs:string" use="required"/>
        <xs:attribute name="RETURN" type="xs:string" use="required"/>
        <xs:attribute name="CLASS-SCOPE" default="false">
            <xs:simpleType>
                <xs:restriction base="xs:NMTOKEN">
                    <xs:enumeration value="true"/>
                    <xs:enumeration value="false"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="CONCURRENCY" default="sequential">
            <xs:simpleType>
                <xs:restriction base="xs:NMTOKEN">
                    <xs:enumeration value="isQuerys"/>
                    <xs:enumeration value="sequential"/>
                    <xs:enumeration value="guarded"/>
                    <xs:enumeration value="concurrent"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="EXCEPTION" type="xs:string"/>
</xs:complexType>
```

**<!-- Generalization/Inheritance -->**

```
<xs:complexType name="GeneralizationType">
    <xs:attribute name="FROM" type="xs:string" use="required"/>
</xs:complexType>
```

**<!-- Association/Includes -->**

```
<xs:complexType name="AssociationType">
    <xs:attribute name="MULTIPLICITY" type="xs:string" use="required"/>
    <xs:attribute name="NAME" type="xs:string" use="optional"/>
    <xs:attribute name="ORDERING" default="unordered">
        <xs:simpleType>
            <xs:restriction base="xs:NMTOKEN">
                <xs:enumeration value="ordered"/>
                <xs:enumeration value="unordered"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="QUALIFIER" type="xs:string"/>
    <xs:attribute name="ROLENAME" type="xs:string"/>
    <!-- Existence of rolename indicates that the asssoc.is navigable-->
    <xs:attribute name="CLASS-SCOPE" default="false">
        <xs:simpleType>
            <xs:restriction base="xs:NMTOKEN">
                <xs:enumeration value="true"/>
                <xs:enumeration value="false"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="CONSTRAINT" default="changeable">
```

```
            <xs:simpleType>
                <xs:restriction base="xs:NMTOKEN">
                    <xs:enumeration value="changeable"/>
                    <xs:enumeration value="addOnly"/>
                    <xs:enumeration value="frozen"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="PEER" type="xs:string" use="required"/>
    </xs:complexType>
```

**<!-- Dependency  -->**

```
    <xs:complexType name="DependencyType">
        <xs:attribute name="PEER" type="xs:string" use="required"/>
        <!-- Two types (use ...package import),(implements ...For interfaces)-->
        <xs:attribute name="DEPKIND" type="xs:string" use="required"/>
    </xs:complexType>

</xs:schema>
```

## A.2. XML Schema for the 5SLScenarios model

```
<?xml version="1.0" encoding="UTF-8"?>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified" attributeFormDefault="unqualified">
```

**<!-- The Schema starts here -->**
**<!-- Service -->**

```
    <xs:element name="SERVICE">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="SCENARIO" type="SCENARIOTYPE"
                maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="NAME" type="xs:string" use="optional"/>
        </xs:complexType>
    </xs:element>
```

**<!-- Each scenario of the service -->**

```
    <xs:complexType name="SCENARIOTYPE">
        <xs:sequence>
            <xs:element name="NOTE" type="xs:string"/>
            <xs:element name="INTERFACEOBJECT" type="xs:string"/>
            <xs:element name="STARTMESSAGE" type="xs:string"/>
            <xs:element name="LISTOFEVENTS" type="LISTOFEVENTTYPE"
            maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="SC_NUMBER" type="xs:integer" use="required"/>
        <xs:attribute name="SC_NAME" type="xs:string" use="optional"/>
    </xs:complexType>
```

**<!-- List of Events -->**

```
    <xs:complexType name="LISTOFEVENTTYPE">
        <xs:sequence>
            <xs:element name="EVENT" type="EVENTTYPE" maxOccurs="unbounded"/>
```

```
		</xs:sequence>
	</xs:complexType>
	<xs:complexType name="EVENTTYPE">
		<xs:sequence>
			<xs:element name="SENDER" minOccurs="0"/>
			<xs:element name="RECEIVER" minOccurs="0"/>
			<xs:element name="MESSAGE" type="MESSAGETYPE" minOccurs="0"/>
			<xs:element name="LISTOFACTIONS" type="LISTOFACTIONSTYPE" minOccurs="0"/>

		</xs:sequence>
		<xs:attribute name="SEQNO" type="xs:string" use="optional"/>
	</xs:complexType>

	<!-- List of Messages -->
	<xs:complexType name="MESSAGETYPE">
		<xs:sequence>
			<xs:element name="LISTOFARGUMENTS" type="LISTOFARGUMENTSTYPE"
			minOccurs="0"/>
			<xs:element name="LISTOFEXCEPTIONS" type="LISTOFEXCEPTIONSTYPE"
			minOccurs="0"/>
		</xs:sequence>
		<xs:attribute name="NAME" type="xs:string" use="required"/>
		<xs:attribute name="METHOD" type="xs:string" use="optional"/>
	</xs:complexType>

	<!-- List of Arguments -->
	<xs:complexType name="LISTOFARGUMENTSTYPE">
		<xs:sequence>
			<xs:element name="ARGUMENT" type="xs:string" maxOccurs="unbounded"/>
		</xs:sequence>
	</xs:complexType>

	<!-- List of Exceptions -->
	<xs:complexType name="LISTOFEXCEPTIONSTYPE">
		<xs:sequence>
			<xs:element name="EXCEPTION" type="xs:string" maxOccurs="unbounded"/>
		</xs:sequence>
	</xs:complexType>

	<!-- List of Actions -->
	<xs:complexType name="LISTOFACTIONSTYPE">
		<xs:sequence>
			<xs:element name="ACTION" type="ACTIONTYPE" minOccurs="0"/>
		</xs:sequence>
	</xs:complexType>
	<xs:complexType name="ACTIONTYPE">
		<xs:sequence>
			<xs:element name="ARGUMENT" type="xs:string"  minOccurs="0"
			maxOccurs="unbounded"/>
			<xs:element name="EXCEPTION" type="xs:string" minOccurs="0"
			maxOccurs="unbounded"/>
		</xs:sequence>
		<xs:attribute name="NAME" type="xs:string" use="required"/>
	</xs:complexType>
</xs:schema>
```

## Appendix B. Synthesized Statechart for the CITIDEL

```
InitialState:  MainMenu
{
    Source state
    MainMenu
    {
        Event                              Destination state  Actions
        doGet                              MainMenu           parseRequest
        check_user_exists                  UserStatus         checkUserExists
    }

    UserStatus
    {
        user_exists                        validUser          viewBinderItems
        user_does_not_exist                MainMenu           displayInvalidUserIdMsg
    }


    validUser
    {
        doGet                                  validUser      parseRequest
        search_query                           ListRetrieval  searchQuery
        get_Class_Schemes                      ListRetrieval  getClassSchemes
        get_Class_Scheme_Categories            ListRetrieval  getClassSchemeCategories
        view_binder_items                      ListRetrieval  viewBinderItems
        back                                   MainMenu       getBackToMainMenuState
    }

    ListRetrieval
    {
        doGet                                  ListRetrieval  parseRequest
        return_search_results                  Display        transformSearchResults
        return_Class_Schemes                   Display        transformClassSchemes
        return_Class_Scheme_Categories         Display        transformCategories
    }

    Retrieval
    {
        doGet                                  ListRetrieval  parseRequest
        return_search_results                  Display        transformSearchResults
        return_Class_Schemes                   Display        transformClassSchemes
        return_Class_Scheme_Categories         Display        transformCategories
        return_OaiIdsAndURLs_InCategory        Display        transformURLs
        return_Sub_Categories                  Display        transformCategories
        return_Metadata                        Display        transformMetadata
        return_List_Metadata                   Display        transformListMetadata
        return_filtered_results                Display        transformFilteredResults
        return_relevance_documents             Display        transformListMetadata
    }
    Display
    {
        doGet                           Display     parseRequest
        get_Metadata                    Retrieval   getMetadata
        get_List_Metadata               Retrieval   getListMetadata
        filter_resources                Retrieval   FilterResources
        relevance_feedback_search       Retrieval   relevanceFeedbackSearch
        search_query                    Retrieval   searchQuery
        get_Class_Schemes               Retrieval   getClassSchemes
        get_Class_Scheme_Categories     Retrieval   getClassSchemeCategories
        get_Sub_Categories              Retrieval   getSubCategories
        get_OaiIdsAndURLs_InCategory    Retrieval   getOaiIdsAndURLsInCategory
        view_binder_items               Retrieval   viewBinderItems
        insert_binder_items             Retrieval   {insertBinderItems viewBinderItems}
        delete_binder_items             Retrieval   {deleteBinderItems viewBinderItems}
        back                            validUser   getBackToValidUserState
    }

}
```

## References

[1] V. Bush, “As We May Think,” *Atlantic Monthly*, vol. 176, pp. 101--108, 1945.

[2] J. C. R. Licklider, *Libraries of the Future*. Cambridge, MA: MIT Press, 1965.

[3] B. M. Leiner, “The Scope of the Digital Library”. Website. DLib Working Group on Digital Library Metrics, October 15, 1998, 1998. http://www.dlib.org/metrics/public/papers/dig-lib-scope.html

[4] D. Waters, “What Are Digital Libraries?,” *Council on Library and Information Resources*, vol. 4, 1998. http://www.clir.org/pubs/issues/issues04.html#dlf

[5] E. Fox, “Digital Library Definitions”, 2003. http://ei.cs.vt.edu/~dlib/def.htm

[6] E. A. Fox, “Sourcebook on Digital Libraries: Report for the National Science Foundation”. Technical Report TR-93-35. Blacksburg, VA: Dept. of Computer Science, Virginia Tech, December, 1993. Available by FTP from directory pub/DigitalLibrary on fox.cs.vt.edu

[7] M. A. Goncalves and E. A. Fox, “5SL -- A Language for Declarative Specification and Generation of Digital Libraries,” in *Proceedings JCDL'2002*, G. Marchionini, Ed. Portland, OR: ACM, 2002.

[8] J. M. Carroll, *Scenario-based design: Envisioning work and technology in system development*. New York: John Wiley and Sons, 1995.

[9] J. Ryser and M. Glinz, “Dependency Charts as a Means to Model Inter-Scenario Dependencies,” presented at Modellierung 2001, Workshop der Gesellschaft fur Informatik e. V. (GI), 28.-30. Marz 2001 in Bad Lippspringe, Germany, 2001.

[10] E. A. Fox, R. Akscyn, R. Furuta, and J. Leggett, “Digital Libraries,” *Communications of the ACM*, vol. 38, pp. 22-28, 1995.

[11] H. Suleman, “Open Digital Libraries,” Virginia Tech Department of Computer Science, Blacksburg, Ph. D. Disseration, 2002. http://scholar.lib.vt.edu/theses/available/etd-11222002-55624/

[12] H. Suleman and E. A. Fox, “A Framework for Building Open Digital Libraries,” *D-Lib Magazine*, vol. 7, 2001. http://www.dlib.org/dlib/december01/suleman/12suleman.html

[13] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns*, 1st ed: Addison-Wesley Pub Co, 1995.

[14] G. Krasner and S. Pope, “A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80,” *JOOP*, vol. 1, 1998.

[15] M. A. Goncalves, E. A. Fox, L. T. Watson, and N. A. Kipp, “Streams, Structures, Spaces, Scenarios, Societies (5S): A Formal Model for Digital Libraries,” Virginia Tech, Department of Computer Science TR-03-04, 2003. http://eprints.cs.vt.edu:8000/archive/00000646/

[16] D. S. Batory, C. Johnson, B. MacDonald, and D. Heeder, “Achieving extensibility through product-lines and domain-specific languages: a case study,” *TOSEM*, vol. 11, pp. 191--214, 2002.

[17] J. M. Carroll, *Making use: Scenario-based design of human-computer interactions*. Cambridge, Massachusetts: MIT Press, 2000.

[18] C. Lagoze, H. Van de Sompel, M. Nelson, and S. Warner, “The Open Archives Initiative Protocol for Metadata Harvesting - Version 2.0, Open Archives Initiative”, 2002. http://www.openarchives.org/OAI/2.0/openarchivesprotocol.htm

[19] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language For Object-Oriented Development, Documentation Set Version 1.0*. Santa Clara, CA: Rationale Software Corporation, 1997.

[20] “Definition and Purposes of a Digital Library”, vol. 2003. Website. 1995. http://www.arl.org/sunsite/definition.html

[21] G. Marchionini, “Research and Development in Digital Libraries”. Website. 2003. http://ils.unc.edu/~march/digital_library_R_and_D.html

[22] CITIDEL, “CITIDEL Homepage”, in *Computing and Information Technology Interactive Digital Educational Library,*. Website. Blacksburg, VA, USA: Virginia Tech, 2001. www.citidel.org

[23] M. A. Gonçalves, R. K. France, and E. A. Fox, “MARIAN: Flexible Interoperability for Federated Digital Libraries,” presented at Proceedings of the 5th European Conference on Research and Advanced Technology for Digital Libraries, Darmstadt, Germany, 2001.

[24] E. Fox, “NDLTD: Networked Digital Library of Theses and Dissertations”. Website. 2000. http://www.ndltd.org

[25] J. R. Davis and C. Lagoze, “NCSTRL: Design and Deployment of a Globally Distributed Digital Library,” *J. American Society for Information Science*, vol. 51, pp. 273--280, 2000.

[26] S. Payette and C. Lagoze, “Flexible and Extensible Digital Object and Repository Architecture,” presented at ECDL, Heraklion, Crete, Greece, 1998.

[27] D. Castelli and P. Pagano, “OpenDLib: A Digital Library Service System,” presented at ECDL, Rome, Italy, 2002.

[28] H. Suleman, E. Fox, R. Kelapure, A. Krowne, and M. Luo, “Building Digital Libraries from Simple Building Blocks,” Virginia Tech, Blacksburg, Technical Report, TR-03-09, 2003.

[29] M. A. Gonçalves, P. Mather, J. Wang, Y. Zhou, M. Luo, R. Richardson, R. Shen, L. Xu, and E. A. Fox, “Java MARIAN: From an OPAC to a Modern Digital Library System,” in *Proceedings of 9th String Processing and Information Retrieval Symposium (SPIRE 2002)*. Lisbon, Portugal, 2002.

[30] I. H. Witten, R. J. McNab, S. J. Boddie, and D. Bainbridge, “Greenstone: A Comprehensive Open-Source Digital Library Software System,” in *Proceedings of the Fifth ACM Conference on Digital Libraries: DL '00, June 2-7, 2000, San Antonio, TX.* New York: ACM Press, 2000, pp. 113--121.

[31] D. Schwabe, G. Rossi, and S. D. J. Barbosa, “Systematic Hypermedia Application Design with OOHDM,” in *Proceedings of the Seventh ACM Conference on Hypertext*, 1996, pp. 116 -- 128.

[32] P. Fraternali and P. Paolini, “Model-driven development of Web applications: the AutoWeb system,” *ACM Transactions on Information Systems*, vol. 18, pp. 323-382, 2000.

[33] S. Ceri, P. Fraternali, and A. Bongio, “Web Modeling Language (WebML): a modeling language for designing Web sites,” *Computer Networks (Amsterdam, Netherlands: 1999)*, vol. 33, pp. 137--157, 2000.

[34] W. Bing, “A hybrid system approach for supporting digital libraries,” *International Journal on Digital Libraries*, vol. 2, pp. 91--110, 1999.

[35] L. A. Kalinichenko, N. A. Skvortsov, D. O. Briukhov, D. V. Kravchenko, and I. A. Chaban, “Designing Personalized Digital Libraries,” *Programming and Computer Software*, vol. 26, pp. 123--133, 2000.

[36] D. Castelli, C. Meghini, and P. Pagano, “Foundations of a Multidimensional Query Language for Digital Libraries,” *Lecture Notes in Computer Science*, vol. 2458, pp. 251--265, 2002.

[37] G. Booch, I. Jacobson, J. Rumbaugh, and J. Rumbaugh, *The Unified Modeling Language User Guide*: Addison-Wesley Pub Co, 1999.

[38] G. Engels, R. Heckel, and S. Sauer, “UML - A Universal Modeling Language?,” presented at ICATPN, Aarhus, Denmark, 2000.

[39] “Borland® Together® ControlCenter™ for Collaborative Application Development”. Website. 2003. http://www.togethersoft.com/products/index.jsp

[40] "Rational Rose Corporation". Website. www.rational.com
[41] "ArgoUML:A UML design tool with cognitive support". Website. Tigris.org: Open Source Software Engineering. http://argouml.tigris.org
[42] OMG, "OMG-XML Metadata Interchange (XMI) Specification, v1.2": OMG, 2002, pp. 268. http://cgi.omg.org/docs/formal/02-01-01.pdf
[43] J. M. Carroll, "Five reasons for scenario-based design," presented at Proceedings of the 32nd Hawaii International Conference on Systems Sciences, 1999.
[44] F. Bordeleau and J. Corriveau, "From Scenarios to Hierarchical State Machines: A Pattern-Based Approach," presented at OOPSLA, Minnesota, 2000.
[45] H. Behrens, "Requirements Analysis and Prototyping using Scenarios and Statecharts," presented at International Conference on Software Engineering, Orlando, Florida, USA, 2002.
[46] K. Koskimies, T. Systa, J. Tuomi, and T. Mannisto, "Automatic support for modeling OO software.," *IEEE Software*, vol. 15, pp. 42--50, 1998.
[47] J. Desharnais, M. Frappier, R. Khédri, and A. Mili, "Integration of Sequential Scenarios," *Transactions on Software Engineering*, vol. 24, pp. 695--708, 1998.
[48] M. Glinz, "An integrated formal model of scenarios based on statecharts,", 1995. http://citeseer.nj.nec.com/glinz95integrated.html
[49] I. Khriss, M. Elkoutbi, and R. K. Keller, "Automating the Synthesis of UML Statechart Diagrams from Multiple Collaboration Diagrams," presented at UML'98: Beyond the Notation, Mulhouse, France, 1998.
[50] M. A. Gonçalves, Q. Zhu, R. Kelapure, and E. A. Fox, "Rapid Modeling, Prototyping, and Generation of Digital Libraries - A Theory-Based Approach," Virginia Tech, Blacksburg, Technical Report, TR-03-16, 2002.
[51] L. Lobo, V. Colaso, A. Shah, and P. Shastri, "Generation of a User Interface Prototype from an Integrated Scenario Specification," Virginia Tech, Blacksburg, Technical Report, TR-02-33, 2002.
[52] A. Prabhune, R. Mahajan, and M. Singhal, "Scenario/Class Diagram Synthesis," Virginia Tech, Blacksburg, Technical Report, TR-03-15, 2002.
[53] J. Whittle and J. Schumann, "Generating statechart designs from scenarios,", Limerick, Ireland, 2000. http://citeseer.nj.nec.com/whittle00generating.html
[54] S. Vasilache and J. Tanaka, "Synthesizing Statecharts from Multiple Interrelated Scenarios,", Zheng Zhou, China, 2001.
[55] D. Connolly and H. Thompson, "XML Schema"Cambridge, MA: W3C, 2000. http://www.w3.org/XML/Schema
[56] P. V. Biron and A. Malhotra, "XML Schema Part 2: Datatypes," W3C, work-in-progress, 2000. http://www.w3.org/TR/2000/WDxmlschema-2-20000407/
[57] H. S. Thompson_et_al., "XML Schema Part 1: Structures," W3C, Cambridge, MA, working-in-progress, 2000. http://www.w3.org/TR/2000/WDxmlschema -1-20000407/
[58] W3C, "World Wide Web Consortium (W3C) Home Page": W3C, 2000. http://www.w3.org/
[59] J. Clark, "XSL Transformations (XSLT),", W3C Recommendation, 1999. http://www.w3.org/TR/xslt/
[60] J. Hunter, "JDOM 1.0,"Java Community Process, 2001. http://jcp.org/en/jsr/detail?id=102#4
[61] D. Brownell, "Simple API for XML". Website. 2003. http://www.saxproject.org/
[62] D. Akehurst and S. Kent, "A Relational Approach to Defining Transformations in a Metamodel," presented at UML, Dresden, Germany, 2002.
[63] Dublin-Core-Community, "Dublin Core Metadata Initiative", in *The Dublin Core: A Simple Content Description Model for Electronic Resources*. WWW site. Dublin, Ohio: OCLC, 1999. http://purl.org/dc/

[64] A. Tsiolakis, “Semantic Analysis and Consistency Checking of UML Sequence Diagrams,” University of Berlin, Berlin, Technical Report, 2001.
[65] R. Winston, “Managing the Development of Large Software Systems:Concepts and Techniques,” presented at International Conference on Software Engineering, Pittsburgh, PA, USA, 1989.
[66] R. Carter, A. Antón, A. Dagnino, and L. Williams, “Evolving Beyond Requirements Creep: A Risk-Based Evolutionary Prototyping Model,” presented at IEEE 5th International Symposium on Requirements Engineering, 2001.
[67] B. Boehm, “A Spiral Model of Software Development and Enhancement,” *IEEE Computer*, vol. 21, pp. 61--72, 1988.
[68] R. Pressman, *Software Engineering : A practitioner's approach*, 3rd ed. New York: McGrawHill Inc, 1992.
[69] K. Beck, “Embracing Change with Extreme Programming,” *Computer*, vol. 32, pp. 70--77, 1999.
[70] M. Gonçalves, G. Panchanathan, U. Ravindranathan, A. Krowne, E. A. Fox, F. Jagodzinski, and L. Cassel, “The XML Log Standard for Digital Libraries: Analysis, Evolution, and Deployment,” presented at Third Joint Conference in Digital Libraries, Houston, Texas, 2003.
[71] W3C, “Web Services Activity”. Web site. 2003. http://www.w3.org/2002/ws/
[72] B. Benatallah, M. Dumas, Q. Z. Sheng, H. Anne., and H. Ngu, “Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web Services,” presented at International Conference on Data Engineering, San Jose, California, USA, 2002.
[73] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, “Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI,” *IEEE Internet Computing*, vol. 6, pp. 86--93, 2002.
[74] R. Wolfgang, *Petri nets: an introduction*: Springer-Verlag New York, Inc., 1985.
[75] D. Harel, “Statecharts: A Visual Formalism for Complex Systems,” *Science of Computer Programming*, vol. 8, pp. 231--274, 1987. http://citeseer.nj.nec.com/harel87statecharts.html
[76] R. Johnson and B. Foote, “Designing Reusable Classes,” *Journal of Object-Oriented Programming*, vol. 1, pp. 22-35, 1988.
[77] T. Grose, G. Doney, and S. Brodsky, *Java programming with XMI, XML and UML*. New York: John Wiley & Sons, Inc., 2002.
[78] S. Sarkar and C. Cleaveland, “XML based document transform applied to application software development projects”: The ServerSide J2EE Community, 2002, pp. 20. http://citeseer.nj.nec.com/sarkar01xml.html
[79] J. Gosling, B. Joy, and G. Steele, *The Java(TM) Language Specification*, 2-nd ed: Addison-Wesley Pub Co, 1996.
[80] J. M. Carroll, C. W. Choo, D. R. Dunlap, P. L. Isenhour, S. T. Kerr, A. MacLean, and M. B. Rosson, “Knowledge Management Support for Teachers,” *Educational Technology Research and Development*, 2003.
[81] “Metadata Repository project (MDR)”. Website. 2000. http://mdr.netbeans.org/
[82] IBM, “Eclipse modeling framework”. Website. 2000. http://www.eclipse.org/emf/
[83] “Poseidon for UML”. Website. 2003. http://www.gentleware.com/
[84] J. Cooper, *The design patterns Java companion*: Addison-Wesley, 1998.
[85] J. Pryor, “Virginia Instuctional Architect for Digital Undergraduate Computing Teaching”. Website. Virginia Tech, 2001. http://citidel-dev.dlib.vt.edu/viaduct/app_user/
[86] “The ACM Computing Classification System”. website. Association for Computing Machinery, Inc, 2003. http://www.acm.org/class/1998/
[87] “Computing Curricula 2001”: Association for Computing Machinery, 2003. http://www.computer.org/education/cc2001/report/appendix-a.html

[88] R. Kelapure, M. A. Goncalves, and E. Fox, "Scenario-Based Generation of Digital Library Services," presented at European Conference of Digital Libraries, Trondheim, Norway, 2003.

[89] "uPortal by JASIG". Website. Java Architectures Special Interest Group, 2000. http://mis105.mis.udel.edu/ja-sig/uportal/

[90] I. Horrocks, *Constructing the User Interface with Statecharts*, 1st edition ed: Addison-Wesley Pub Co, 1999.

[91] "Server-side MVC Architecture: Clients, Transactions and Exceptions", in *uidesign.net*. website. uidesign.net, 2000. http://www.uidesign.net/2000/papers/webmvc1a.html

[92] J. Gould, J. Conti, and T. Hovanyecz, "Composing Letters with a Simulated Listening Typewriter," *CACM*, vol. 26, pp. 295--308, 1983.

[93] N. Ramakrishnan, "PIPE: Web Personalization By Partial Evaluation," *IEEE Internet Computing*, vol. 4, pp. 21--31, 2000.

[94] M. A. Gonçalves, A. A. Zafer, N. Ramakrishnan, and E. A. Fox, "Modeling and Building Personalized Digital Libraries with PIPE and 5SL," in *Proceedings of the Joint DELOS-NSF Workshop on Personalization and Recommender Systems in Digital Libraries*. Dublin, Ireland: DELOS, 2001.

[95] E. Fox, J. Carroll, P. Fan, L. Cassel, Z. Mohammed, K. Maly, G. McMillan, N. Ramakrishnan, and M. Halbert, "Science of Digital Libraries," Virginia Tech, Blacksburg, NSF Proposal, TR-03-13, February 2, 2003.

## VITA

Rohit Kelapure was born on November 23, 1979 in Mumbai, India.

Rohit Kelapure earned a Bachelor of Science degree in Computer Science & Applications from the University of Mumbai, Mumbai in May 2001. He stood fourth in the merit list for the last two years of engineering study and graduated with Honors.

He started his graduate study in the Computer Science Department at Virginia Tech, Virginia in August 2001. He completed his Master of Science degree requirements in June 2003.

From June to August 2002, Rohit Kelapure interned with IBM in the Extreme Blue internship program. After graduation, he will continue his professional career with IBM in the Network Dispatcher Development software group, in Raleigh, North Carolina.