Scenario-Based Generation of Digital Library Services

Rohit Kelapure, Marcos André Gonçalves, Edward A. Fox

CS Dept.
Virginia Tech
Blacksburg, VA 24061, USA
Email: {rkelapur,mgoncalv,fox}@vt.edu

Abstract. We present 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) 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, 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, 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.

1. INTRODUCTION

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.

In this paper we describe a new generic DL generator based on the 5S (Streams, Structures, Spaces, Scenarios, Societies) formal framework [1] , focusing on support for two key aspects of DLs: "societies" and "scenarios" [2]. The first relates to support for users, user communities, collaboration, agents, resource manager routines, and other similar entities. The latter relates to services, workflow, and other descriptions and implementations of DL functionality. The principal contribution of our work is the development, implementation, and deployment of a generic DL generator that can be used by DL designers to semi-automatically produce tailored DL services from models of societies and scenarios. By doing this the generator attempts to bridge the gap between DL models and system implementation, i.e., between concept and execution, therefore partially validating the formal theory of 5S. We demonstrate the feasibility of this approach and substantiate our claims by providing two examples that illustrate the features of the generator.

This paper is organized as follows. Section 2 describes our approach and development environment. Section 3 is the core of the paper and details examples, architecture, and implementation of our digital library generator. The "Examples" subsection focuses on extensibility and reusability. Section 4 deals with related work, while section 5 concludes the paper.

2. APPROACH

Our objective is to cover the whole process of DL development, from requirements to analysis, analysis to design, 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 will be 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 (Scenario Model); and 5) the different societies of actors and managers of services that act together to carry out the DL behavior (Societal Model) [3]. We have organized and formalized these and other DL notions into the 5S (Streams, Structures, Spaces, Societies, Scenarios) framework. 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. 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 [4].

3. Using scenario based design for defining the behavior of a system. Scenarios keep design discussion focused on user activities, more specifically, they keep design discussion focused on the level of task organization that actors experience in their tasks. 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) [1] and Open Digital Libraries (ODL [2, 3]), so generation can occur atop a number of DL toolkits (initially two).

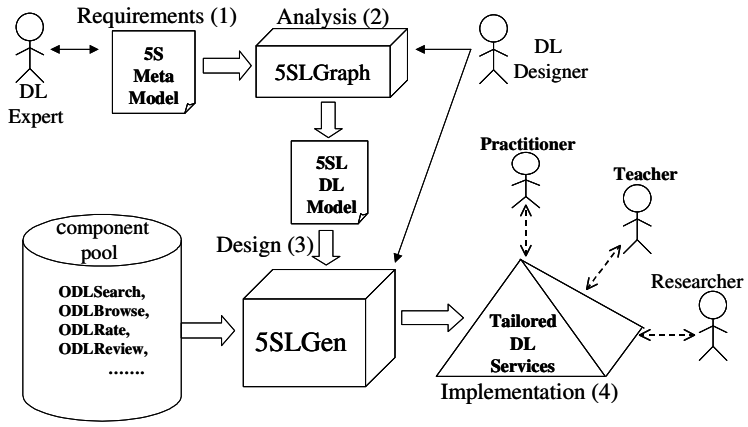

Figure 1. Overview of the architecture for DL modeling and generation

We adopt an approach shown to be highly effective in other areas of computing: develop powerful theories and (meta)models (i.e., 5S); use them to develop formal specifications (i.e., 5SL), and generate tailored systems from those specifications (using 5SLGen). We explain the approach in the context of the classical software engineering process (see Fig. 1). During *requirements gathering* (see 1 in Fig. 1) the DL designer captures all "societal" conditions and capabilities to which the DL must conform. 5S provides a common ground terminology and 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 based on 5S, which will be used for modeling the DL. In the *analysis* phase (see 2 in Fig. 1), the requirements are formally captured in 5SL. The DL designer must be aware of functional requirements — what services a community needs and what form of interaction these services should have with the users of the DL: publishers, searchers, and administrators. Modeling such a complex system using only an XML-based language requires a great deal of knowledge of the 5S theory and language syntax. Accordingly, we introduced 5SGraph [5], a visual modeling tool that helps designers to model a DL instance without knowing the theoretical foundations and the syntactical details of 5SL. The focus of the *design* phase (see 3 in Fig. 1) is to produce models that are closer to the implementation and the target architecture, but still preserve the structure of the system as captured by the analysis model. 5SLGen produces design models from 5SL models by transforming higher-level 5SL concepts into object-oriented classes and workflows. This transformation involves scenario analysis and scenario synthesis. Finally in the *implementation* phase (see 4 in Fig. 1), 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. This digital library generator, 5SLGen, is the focus of this paper.

3. THE 5SLGEN Digital Library Generator

3.1 Model

We envision the services exposed by a DL to be either of the composite or elementary type. Elementary services provide the basic infrastructure for the DL. Examples include collecting, indexing, rating, and linking. Composite services can be composed of other services (elementary or composed) by reusing or extending them. For example, a relevance feedback service extends the capabilities of a basic search service while a lesson plan building service can use already existing searching, browsing, and binding services to find and organize relevant resources.

The problem of composability of services has been studied recently, mainly in the Web community [6, 7]. 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 2 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. Elementary services do not rely on other services to fulfill their responsibilities while composite services act like umbrella structures that bring together other services, which collaborate to implement certain functionality. 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 services scenarios, including invocations of other services. Statecharts [8] and petri nets [9] are possible notations for formally representing workflows. In our implementations we chose statecharts to represent the workflow of a service. Statecharts, introduced by Harel [10], represent a compact way of describing the dynamic aspects of the system. 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.

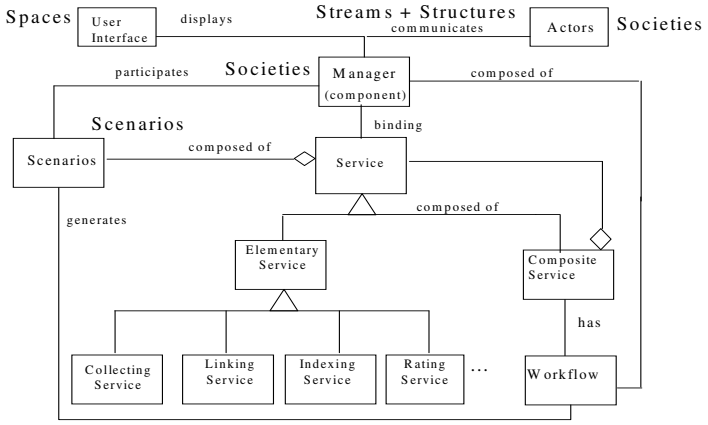

Figure 2. DL Service Composition Model or Pattern

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 generation of workflows from them;

and 3) the role of the service manager (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 should allow 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., Rocchio based expansion of a query in relevance feedback) is not defined by any component in the core pool or cannot be reused (e.g., due to incompatibility of interfaces).

More importantly, the model also shows how the 5 '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 scenarios and their interactions/relationships. Service managers 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. It is interesting to notice the connections between the service manager roles and the classical Model-View-Controller (MVC) architecture of user interfaces [11], which explicitly separates functionality, behavior, and presentation and has helped facilitate the development of user interfaces that are modular and extensible. Service managers and actors communicate through streams (e.g., protocols) and structures (e.g., structured streams such as metadata specifications and digital objects). Finally the model provides the basic architectural underpinnings for the creation of DL generators, as described in Section 3.3.

3.2 Examples

In this section we present examples of two services, a Relevance Feedback Search service and a Lesson Plan Building service, implemented using 5SLGen. The services exposed follow the model explained in Figure 2, 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. We start each subsection with the main scenario for the particular service.

3.2.1. Extensibility: A Relevance Feedback Service

Scenario 1: Relevance feedback is a well known technique to improve quality of search services. A relevance feedback service *extends* a basic search service by allowing the user to choose from the results of a search the documents that are relevant. The selected relevant documents are then used by the Relevance Feedback Manager to construct an expanded query (using the Rocchio method, for example), which is then run to retrieve the next set of documents to be presented to the user.

Figure 3 shows the relationship between the relevance feedback service and the search service (left part) and describes the relevance feedback scenario in terms of a UML sequence diagram (middle part). A sequence diagram focuses on the time ordering of events between members of societies. These members appear along the top margin of a dashed line that represents a timeline. Events can be associated with actions that the service managers perform to provide a given functionality. For the

sake of brevity we do not show the corresponding 5SL modeling in XML; the interested reader is pointed to [5SL] for syntactic details. The <<extends>> relationship specifies that an instance of a relevance feedback search service includes the behavior of search service and adds specific events, subject to specific conditions (e.g., the set of relevant documents cannot be empty). The scenario shows that all the events associated with the basic search scenario occur in the relevance feedback scenario, with the addition of the expandQuery event and synchronous response. The statechart for the Relevance Feedback Manager derived from the scenario is shown in the figure too. There are only two states: the system transitions from the default to the "expanded query" state after reception of the expandQuery event (if the condition is true) and immediately transitions back to the default state where it can receive other requests.

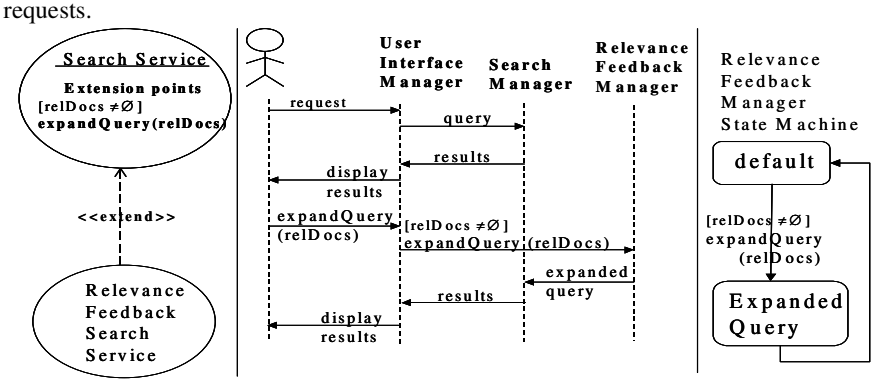

Figure 3. Relationships between services for relevance feedback

3.2.2 Reusability: Lesson Plan Building Service

Scenario 2: A *lesson plan* aggregates specific educational metadata and correlated resources (e.g., papers, simulations) available in the Computing and Information Technology Interactive Digital Electronic Library (CITIDEL; www.citidel.org) into a coherent package useful for some CS teaching activity. A specific service manager called *VIADUCT* supports this service, which can be used only by teachers registered with CITIDEL. To build a lesson plan, the teacher uses the information-seeking services of CITIDEL (i.e., searching and browsing) to look for relevant resources to a specific lesson, choses among those using any subjective criteria (e.g., by relevance, by date), assembles a number of the chosen resources together using a *binder* service, and associates descriptive metadata such as typical DC-based ones like author, identifier, language – as well as specific ones such as topic area, target audience, and time required for the whole lesson plan object. The teacher has to explicitly publish the lesson plan to allow students to view it. To allow a select group of people to view the lesson plan, the teacher saves the plan, returns to the main VIADUCT user information page, re-opens the project, and gives the project URL to whomever she wishes.

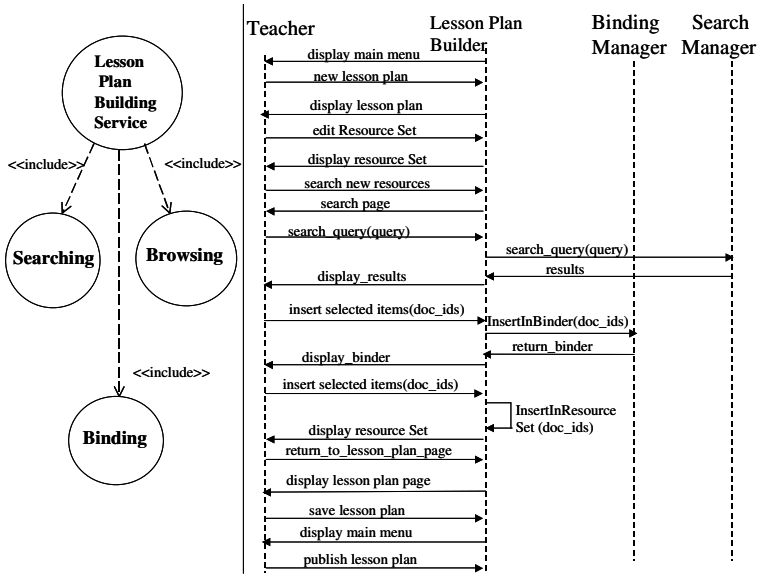

Figure 4. Relationships between the services for the lesson plan in VIADUCT

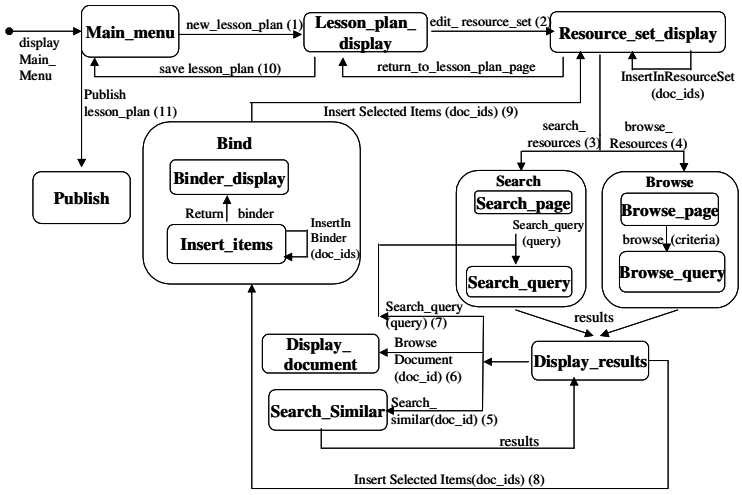

Figure 5. Statechart diagram of the VIADUCT system

Figure 4 shows that the lesson plan building (LPB) service includes three other services: Searching, Browsing, and Binding, as well as the main scenario of the LPB service. Figure 5 presents the statechart generated from scenario synthesis of the main scenario with other related scenarios of this service (not shown for brevity). The

teacher starts the construction of a new lesson plan from the main menu (see 1 in Fig. 5). The lesson plan edit page allows the teacher to fill out basic metadata about the plan and organize a number of related resources together (see 2 in Fig. 5). To locate relevant resource the teacher can either search or browse (see 3 and 4 in Fig. 5) the collection according to some criteria and sorting order. Having the results of an initial search/browse activity the user can either: 1) search for a similar document (see 5 in Fig. 5); 2) browse a particular entry for details (see 6 in Fig. 5); 3) perform another search (see 7 in Fig. 5); or 4) select a number of items to put in her binder (see 8 in Fig. 5). If the user chooses the latest option the binder is shown and she can transfers a number of resources from the binder to the resource set of the current plan (see 9 in Fig. 5). Once the plan is ready the teacher can save it and publish to the students (see 10 and 11 in Fig. 5).

3.3 5SLGen Architecture and Implementation

The architecture of 5SLGen is shown in Figure 6. The generated system is organized around the idea of a clean separation between service managers, that implement operations and carry data; views, for displaying all or a portion of the data; and controllers, for handling events that affect the data or the view(s) [11]. In the context of 5SLGen the service managers are either represented by one or more components in the pool or are generated from the 5SL-Societies model. The generated service managers may contain skeleton code for operations and capabilities not defined in any component of the pools; this code needs to be provided by the designer. Our current component pool consists of ODL components that communicate through a family of lightweight protocols based on OAI [12]. 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 service managers. The controller maps onto the workflow of the system generated from the 5SL-Scenarios model. The view corresponds to the user interface presentation. The DL designer binds the presentation elements with the service managers to complete the implementation of the generated DL services.

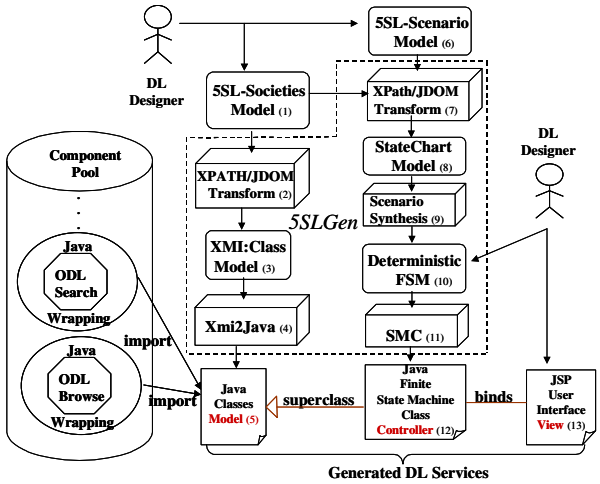

Figure 6. Architecture of 5SLGen based on MVC (expanding part of Figure 1)

This architecture for the generated DL services is achieved through the following process: The DL designer captures the structural and behavioral aspects of DL services through the 5SL-Societies model and 5SL-Scenarios model. 5SL-Societies captures the relationships among actors (those who use services) and services managers, whereas 5SL-Scenarios capture their dynamic interactions. The specifications of the DL captured in 5SL (- Societies and Scenarios) undergo a series of transformations (explained below) with the DL designer providing input at certain stages to generate the Java classes corresponding to the implementations of the service managers and the workflow of the DL. Once the presentation elements (views) are coupled with the controller, the generation of the tailored DL service is complete.

3.3.1 Generating Static Contextual Structure

The 5SL-Societies model is realized based on the relationships among actors and service managers and the set of operations that define the services' capabilities. In order to generate Java classes from the 5SL-Societies model we have chosen an intermediate step of transforming (see 2 in Figure 6) the 5SL-Societies XML Model into a XMI [13] representation model (see 3 in Figure 6) using the JDOM and XPATH XML APIs. XMI is an XML based industry standard to enable easy interchange of metadata between modeling tools and between tools and metadata repositories. Many CASE tools serialize UML diagrams to XMI. Generation of XMI files for the 5SL-Societies model enables the exchange of the 5S-Societies model among various UML modeling tools supporting modeling as well as forward and reverse engineering. Moreover, existing freeware tools (see 4 in Figure 6) enable the generation of Java code from the serialized XMI Model (XMI2Java). We use an open source XMI2Java implementation to generate Java classes that implement the service managers (see 5 in Figure 6) for the generated DL.

3.3.2. Generating Dynamic Behavior

The 5SL-Scenarios model is used to capture the dynamic behavior of services (e.g., see figures 3 and 4) as scenarios. In order to describe the whole behavior of a DL service, a great multitude of scenarios is required to capture the complete set of possible societal interactions. Scenarios can contain other scenarios and in many cases are only small variations of others. This requires an approach for scenario integration in order to capture the whole behavior of the system. Also, in order to be able to generate an implementation from the scenarios, the level of abstraction needs to be reduced to a more concrete model in terms of computational actions and change of states that occur during scenario execution. These problems can be addressed by generating a statechart model (see 8 in Figure 6) from the scenarios (see 6 in Figure 6). The mapping from scenarios to statecharts is performed according to the following rules: For any object in a sequence diagram, incoming arrows represent events received by the object and they become transitions (see Section 3.1). Outgoing arrows are actions and they become actions of the transitions leading to the states. The intervals between events become states. The object starts in the default state specified in the 5SL-Societies model [14]. This transformation (see 7 in Figure 6) is achieved by parsing the 5SL-Scenarios modeled in XML with the JDOM and XPATH XML APIs [15, 16] and implementing the rules mentioned above.

Again, since scenarios represent partial descriptions of the system behavior, an approach for scenarios composition is needed to produce a more complete specification of the service. As each scenario is mapped to a statechart we synthesize the statecharts derived in the previous step to perform scenario composition. The statecharts are synthesized (see 9 in Fig. 6) according to the following rules [14]: 1) if a transition is common to the two statecharts, it is taken only once into the final statechart; 2) 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; and 3) 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.

A statechart extends traditional a finite-state machine (FSM) with notions of hierarchy and concurrency. A FSM represents a mathematical model of a system that attempts to reduce the model complexity thereby providing a powerful manner to describe the dynamic behavior of systems and components. The synthesized statechart (Figures 3 and 5) generated using the above rules represents the FSM/workflow for the DL service (see 10 in Fig. 6). To generate code from the FSM we have extended an open source state machine compiler (see 11 in Fig. 6) that compiles the annotated FSM to generate code (Java classes — see 12 in Figure 6) for the controller of the DL services. Before compilation the FSM is annotated by providing component specific implementation details by the DL designer (see Fig. 6).

There are many techniques of implementing state machines; the most common implies some sort of switch or if-else statements for implementing state dependent behavior; however this solution is not scalable; therefore we chose to implement FSM using the state design pattern from [17]. 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 5SLGen when the service manager class receives an event, it delegates the request to its state object, which provides the appropriate state specific behavior.

The lesson plan building and relevance feedback service have been implemented using the above generation process. Manual intervention is required for: first annotating the FSM with component specific implementation details and second providing the views for the data. Building on current work, we plan to implement a fully functional and automatic version of 5SLGen before the conference starts. When functional, this will amount to about 2000 lines of code in total.

4. RELATED WORK

The first work to advocate a goal-oriented requirements analysis approach for digital libraries is [18], but that work does not propose any development tool or environment. The closest approach to our DL generator is the collection services and plug-in architecture of Greenstone [19]. However their architecture covers only portions of the Stream and Structural models of 5S with little support for modeling and generation of customized DL services (other than basic searching and browsing).

While much attention has been paid to digital library architectures and systems, very few works tackle the problem of integrated DL design, conceptual modeling, and requirements gathering. Examples of work on DL architectures and systems include:

monolithic systems (e.g., Greenstone[19], MARIAN[20]), componentized architectures (e.g, ODL[12], OpenDlib[21]), agent-based architectures (e.g., UMDL[22]), and layered architectures (e.g., Alexandria[23]). Our declarative/generative approach should be generalizable for any of those systems/architectures by taking whole or portions of those systems as part of our component pools. There is no reason why those systems and their components can not be incorporated in our component pools, given that they export clear, reusable software interfaces with accessible entry points.

Most research done in the area of code generation from requirements has not been directed towards specific domains such as DLs. Most CASE tools do not address issues raised by research in scenario-based requirements analysis and design such as scenario generation, management of scenario descriptions, analysis and integration of scenarios, and bridging the gap between scenario descriptions and software designs and implementation [2]. We have attempted to tackle the above problems and to bridge the model system gap through the 5S framework.

5. CONCLUSIONS AND FUTURE WORK

We have presented a new digital library generator capable of producing running versions of DL services from models of DL scenarios and societies and pools of components. Automatic mappings from 5SL societies and scenarios models to code have helped diminish the risk of inconsistency and incompleteness in implementations of DL services. Code generation on the basis of the 5S metamodel supports our claim for a model driven architecture. Research problems such as scenario synthesis and scenario generation have been tackled by implementing an algorithm for scenario synthesis and modeling scenarios with 5SGraph. The implementation of DL with simple Web Service-like components instead of monolithic software facilitates extensibility and reusability. The architecture of the generated DL services are based on a clear separation between model, behavior, and presentation. This separation provides for clarity of design, extensibility, and modularity of generated code.

Future work will aim especially to reduce the amount of manual intervention needed from the designer. For example, the state machine compiler (SMC) responsible for generation of Java code from the annotated state machine does not support parameterized events/actions which requires manual modification of the FSM. In the current implementation of 5SLGen the DL designer determines the component to be used for the service. We envision that the components used in 5SLGen will eventually be implemented as web services. Thus the mapping described above can be determined programatically using the WSDL [7] document exposed by the web service. We are modifying SMC to support these capabilities. Also we are investigating (semi-) automatic (rather than manual) ways of generating the user interface from the 5S-Spaces model and connecting them with the controller classes. Finally, once all those issues are solved we plan to connect the generation tool with the 5SGraph modeling tool to yield a complete CASE tool for DLs based on the 5S metamodel.

References

1. 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, CS Dept.

TR-03-04, 2003. http://eprints.cs.vt.edu:8000/archive/00000646/ (re-submitted after revision from an earlier version, for publication in ACM Transactions on Information Systems)
2. J. M. Carroll, *Scenario-based design: Envisioning work and technology in system development*. New York: John Wiley and Sons, 1995.
3. 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.
4. D. S. Batory, C. Johnson, B. MacDonald, and D. v. Heeder, "Achieving extensibility through product-lines and domain-specific languages: a case study," *TOSEM*, vol. 11, pp. 191-214, 2002.
5. Q. Zhu, "5SGraph: A Modeling Tool for Digital Libraries," Virginia Tech CS Dept., Masters Thesis, 2002. http://scholar.lib.vt.edu/theses/available/etd-11272002-210531/
6. M. H. Burstein, J. R. Hobbs, O. Lassila, D. Martin, D. V. McDermott, S. A. McIlraith, S. Narayanan, M. Paolucci, T. R. Payne, and K. P. Sycara, "DAML-S: Web Service Description for the Semantic Web," International Semantic Web Conference, 2002.
7. 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.
8. G. Booch, I. Jacobson, J. Rumbaugh, and J. Rumbaugh, *The Unified Modeling Language User Guide*: Addison-Wesley Pub Co, 1998.
9. W. Reisig, *Petri nets: an introduction*: Springer-Verlag, New York, Inc., 1985.
10. 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
11. F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture: A System of Patterns*, vol. 1: John Wiley & Son Ltd., 1996.
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. OMG, "OMG-XML Metadata Interchange (XMI) Specification, v1.2": OMG, 2002, pp. 268. http://cgi.omg.org/docs/formal/02-01-01.pdf
14. S. Vasilache and J. Tanaka, "Synthesizing Statecharts from Multiple Interrelated Scenarios," Zheng Zhou, China, 2001.
15. J. Hunter, "JDOM 1.0"Java Community Process, 2001. http://jcp.org/en/jsr/detail?id=102#4
16. W3C, "XML Path Language (XPath) Version 1.0,", J. Clark and S. DeRose, eds., 1999. http://www.w3.org/TR/xpath
17. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns*, 1st edition ed: Addison-Wesley Pub. Co., 1995.
18. D. Bolchini and P. Paolini, "Goal-Oriented Requirements Specification for Digital Libraries," *Lecture Notes in Computer Science*, vol. 2458, pp. 107-115, 2002.
19. 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.
20. M. A. Gonçalves, R. K. France, and E. A. Fox, "MARIAN: Flexible Interoperability for Federated Digital Libraries," in *Proc.s of the 5th European Conference on Research and Advanced Technology for Digital Libraries*. Darmstadt, Germany: 2001, pp. 161-172.
21. D. Castelli and P. Pagano, "OpenDLib: A Digital Library Service System," ECDL 2002, Rome, Italy, 2002.
22. P. Weinstein and G. Alloway, "Seed Ontologies: growing digital libraries as distributed, intelligent systems," in *Second ACM International Conference on Digital Libraries, Philadelphia, PA*. New York: ACM Press, 1997.
23. J. Frew, M. Freeston, N. Freitas, L. Hill, G. Janée, K. Lovette, R. Nideffer, T. Smith, and Q. Zheng, "The Alexandria Digital Library Architecture," *IJODL*, vol. 2, pp. 259-268, 2000.