Generating Collaborative Systems for Digital Libraries: a Model-Driven Approach

Alessio Malizia, Paolo Bottoni, and S. Levialdi

The design and development of a digital library involves different stakeholders, such as: information architects, librarians, and domain experts, who need to agree on a common language to describe, discuss, and negotiate the services the library has to offer. To this end, high-level, language-neutral models have to be devised. Metamodeling techniques favor the definition of domain-specific visual languages through which stakeholders can share their views and directly manipulate representations of the domain entities. This paper describes CRADLE (Cooperative-Relational Approach to Digital Library Environments), a metamodel-based framework and visual language for the definition of notions and services related to the development of digital libraries. A collection of tools allows the automatic generation of several services, defined with the CRADLE visual language, and of the graphical user interfaces providing access to them for the final user. The effectiveness of the approach is illustrated by presenting digital libraries generated with CRADLE, while the CRADLE environment has been evaluated by using the cognitive dimensions framework.

Digital libraries (DLs) are rapidly becoming a preferred source for information and documentation. Both at research and industry levels, DLs are the most referenced sources, as testified by the popularity of Google Books, Google Video, IEEE Explore, and the ACM Portal. Nevertheless, no general model is uniformly accepted for such systems. Only few examples of modeling languages for developing DLs are available,[1] and there is a general lack of systems for designing and developing DLs. This is even more unfortunate because different stakeholders are interested in the design and development of a DL, such as information architects, to librarians, to software engineers, to experts of the specific domain served by the DL. These categories may have contrasting objectives and views when deploying a DL: librarians are able to deal with faceted categories of documents, taxonomies, and document classification; software engineers usually concentrate on services and code development; information architects favor effectiveness of retrieval; and domain experts are interested in directly referring to the content of interest without going through technical jargon. Designers of DLs are most often library technical staff with little to no formal training in software engineering, or computer scientists with little background in the research findings of hypertext information retrieval. Thus DL systems are usually built from scratch using specialized architectures that do not benefit from previous experience and from research in software engineering. Wasted effort and poor interoperability can therefore ensue, raising the costs of DLs and jeopardizing the fluidity of information assets in the future.

In addition, there is a need for modeling services and data structures as highlighted in the "Digital Library Reference Model" proposed by the DELOS EU network of excellence (also called the "DELOS Manifesto");[2] in fact, the distribution of DL services over digital networks, typically accessed through Web browsers or dedicated clients, makes the whole theme of interaction between users important, for both individual usage and remote collaboration. Designing and modeling such interactions call for considerations pertaining to the fields of human–computer interaction (HCI) and computer-supported cooperative work (CSCW). As an example, scenario-based or activity-based approaches developed in the HCI area can be exploited in DL design.

To meet these needs we developed CRADLE (Cooperative-Relational Approach to Digital Library Environments),[3] a metamodel-based Digital Library Management System (DLMS) supporting collaboration in the design, development, and use of DLs, exploiting patterns emerging from previous projects. The entities of the CRADLE metamodel allow the specification of collections, structures, services, and communities of users (called "societies" in CRADLE) and partially reflect the DELOS Manifesto. The metamodel entities are based on existing DL taxonomies, such as those proposed by Fox and Marchionini,[4] Gonçalves et al.,[5] or in the DELOS Manifesto, so as to leverage available tools and knowledge. Designers of DLs can exploit the domain-specific visual language (DVSL) available in the CRADLE environment—where familiar entities extracted from the referred taxonomies are represented graphically—to model data structures, interfaces and services offered to the final users. The visual model is then processed and transformed, exploiting suitable templates, toward a set of specific languages for describing interfaces and services. The results are finally transformed into platform-independent (Java) code for specific DL applications.

CRADLE supports the basic functionalities of a DL through interfaces and service templates for managing, browsing, searching, and updating. These can be further specialized to deploy advanced functionalities as defined by designers through the entities of the proposed visual

Alessio Malizia (alessio.malizia@uc3m.es) is Associate Professor, Universidad Carlos III, Department of Informatics, Madrid, Spain; **Paolo Bottoni** (bottoni@di.uniroma1.it) is Associate Professor and **S. Levialdi** (levialdi@di.uniroma1.it) is Professor, "Sapienza" University of Rome, Department of Computer Science, Rome, Italy.

language. CRADLE is based on the entity-relationship (E/R) formalism, which is powerful and general enough to describe DL models and is supported by many tools as a metamodeling language. Moreover, we observed that users and designers involved in the DL environment, but not coming from a software engineering background, may not be familiar with advanced formalism like unified modeling language (UML), but they usually have basic notions on database management systems, where E/R is largely employed.

Literature Review

DLs are complex information systems involving technologies and features from different areas, such as library and information systems, information retrieval, and HCI. This interdisciplinary nature is well reflected in the various definitions of DLs present in the literature. As far back as 1965, Licklider envisaged collections of digital versions of scanned documents accessible via interconnected computers.[6] More recently, Levy and Marshall described DLs as sets of collections of documents, together with digital resources, accessible by users in a distributed context.[7] To manage the amount of information stored in such systems, they proposed some sort of user-assisting software agent. Other definitions include not only printed documents, but multimedia resources in general.[8] However different the definitions may be, they all include the presence of collections of resources, their organization in structured repositories, and their availability to remote users through networks (as discussed by Morgan).[9] Recent efforts toward standardization have been taken by public and private organizations. For example, a Delphi study identified four main ingredients: an organized collection of resources, mechanisms for browsing and searching, a distributed networked environment, and a set of objectified services.[10] The President's Information Technology Advisory Committee (PITAC) Panel on Digital Libraries sees DLs as the networked collections of digital text, documents, images, sounds, scientific data, and software that make up the core of today's Internet and of tomorrow's universally accessible digital repositories of all human knowledge.[11]

When considering DLs in the context of distributed DL environments, only few papers have been produced, contrasting with the huge bibliography on DLs in general. The DL Group at the Universidad de las Américas Puebla in Mexico introduced the concept of personal and group spaces, relevant to the CSCW domain, in the DL system context.[12] Users can share information stored in their personal spaces or share agents, thus allowing other users to perform the same search on the document collections in the DL. The cited text by Gonçalves et al. gives a formal foundation for digital libraries, called 5S, based on the concepts of *streams*, (data) *structures*, (resource) *spaces*, *scenarios*, and *societies*. While being evidence of a good modeling endeavor, the approach does not specify formally how to derive a system implementation from the model.

The new generation of DL systems will be highly distributed, providing adaptive and interoperable behaviour by adjusting their structure dynamically, in order to act in dynamic environments (e.g., interfacing with the physical world).[13] To manage such large and complex systems, a systematic engineering approach is required, typically one that includes modeling as an essential design activity where the availability of such domain-specific concepts as first-class elements in DL models will make application specification easier.[14]

While most of the disciplines related to DLs—e.g., databases,[15] information retrieval,[16] and hypertext and multimedia[17]—have underlying formal models that have properly steered them, little is available to formalize DLs per se. Wang described the structure of a DL system as a domain-specific database together with a user interface for querying the records stored in the database.[18] Castelli et al. present an approach involving multidimensional query languages for searching information in DL systems that is based on first-order logic.[19] These works model metadata specifications and thus are the main examples of system formalization in DL environments. Cognitive models for information retrieval, as used for example by Oddy et al.,[20] focus on users' information-seeking behavior (i.e., formation, nature, and properties of a users' information need) and on how information retrieval systems are used in operational environments.

Other approaches based on models and languages for describing the entities involved in a DL are the Digital Library Definition Language,[21] the DSpace data model[22] (with the definitions of communities and workflow models), the Metis Workflow framework,[23] and the Fedora structoid approach.[24] E/R approaches are frequently used for modeling database management system (DBMS) applications,[25] but as E/R diagrams only model the static structure of a DBMS, they generally do not deal deeply with dynamic aspects. Temporal extensions add dynamic aspects to the E/R approach, but most of them are not object-oriented.[26] The advent of object-oriented technology calls for approaches and tools to information system design resulting in object-oriented systems. These considerations drove research toward modeling approaches as supported by UML.[27]

However, since the UML metamodel is not yet widespread in the DL community, we adopted the E/R formalism and complemented it with the specification of the dynamics made available through the user interface, as described by Malizia et al.[28] Using the metamodel, we have defined a DSVL, including basic entities and

relationships for modeling DL-related scenarios and activities. The need for the integration of multiple languages has also been indicated as a key aspect of the DSVL approach.[29] In fact, complex domains like DLs typically consist of multiple subdomains, each of which may require its own particular language.

In the current implementation, the definition of DSVLs exploits the metamodeling facilities of AToM3, based on graph-grammars.[30] AToM3 has been typically used for simulation and model transformation, but we adopt it here as a tool for system generation.

Requirements for Modeling Digital Libraries

We follow the DELOS Manifesto by considering a DL as an organization (possibly virtual and distributed) for managing collections of digital documents (digital contents in general) and preserving their images on storage. A DL offers contextual services to communities of users, a certain quality of service, and the ability to apply specific policies. In CRADLE we leave the definition of quality of service to the service-oriented architecture standards we employ and partially model the applicable policy, but we focus here on crucial interactivity aspects needed to make DLs usable by different communities of users.

In particular, we model interactive activities and services based on librarians' experiences in face-to-face communication with users, or designing exchange and integration procedures for communicating between institutions and managing shared resources.

While librarians are usually interested in modeling metadata across DLs, software engineers aim at providing multiple tools for implementing services,[31] such as indexing, querying, semantics,[32] etc. Therefore we provide a visual model useful for librarians and information architects to mimic the design phases they usually perform. Moreover, by supporting component services, we help software engineers to specify and add services on demand to DL environments. To this end, we use a service component model. By sharing a common language, users from different categories can communicate to design a DL system while concentrating on their own tasks (services development and design for software engineers and DL design for librarians and information architects). Users are modeled according to the Delos Manifesto as DL End-users (subdivided into content creators, content consumers, and librarians), DL Designers (librarians and information architects), DL System Administrators (typically librarians), and DL Application Developers (software engineers).

Several activities have been started on modeling domain specific DLs. As an example, the U.S. National Science Digital Library (NSDL) program promotes educational DLs and services for basic and advanced science education as discussed by Wattenberg or Zia.[33] In the NSDL program, a new generation of services has been developed that includes support for teaching and learning; this means also considering users' activities or scenarios and not only information access. Services for implementing personal content delivery and sharing, or managing digital resources and modeling collaboration, are examples of tools introduced during this program.

The virtual reference desk (VRD) is emerging as an interactive service based on DLs. With VRD, users can take advantage of domain experts' knowledge and librarians' experience to locate information. For example, the U.S. Library of Congress Ask a Librarian service acts as a VRD for users who want help in searching information categories or to interact with expert librarians to search for a specific topic.[34]

The interactive and collaborative aspects of activities taking place within DLs facilitate the development of user communities. Social networking, work practices, and content sharing are all features that influence the technology and its use. Following Borgmann,[35] Lynch sees the future of DLs not in broad services but in supporting and facilitating "customization by community," i.e., services tailored for domain-specific work practices.[36]

We also examined the research agenda on system-oriented issues in DLs and the DELOS manifesto.[37] The agenda abstracts the DL life cycle, identifying five main areas, and proposes key research problems. In particular we tackle activities such as formal modeling of DLs and their communities and developing frameworks coherent with such models.

At the architectural level, one point of interest is to support heterogeneous and distributed systems, in particular networked DLs and services.[38] For interoperability, one of the issues is how to support and interoperate with different metadata models and standards to allow distributed cataloguing and indexing, as in the Open Archive Initiative (OAI).[39]

Finally, we are interested in the service level of the research agenda and more precisely in Web services and workflow management as crucial features when including communities and designing DLs for use over networks and for sharing content.

As a result of this analysis, the CRADLE framework features the following:

- a visual language to help users and designers when visual modeling their specific DL (without knowing any technical detail apart from learning how to use a visual environment providing diagrams representations of domain specific elements)
- an environment integrating visual modeling and code generation instead of simply providing an integrated architecture that does not hide technical details
- interface generation for dealing with different users

and designers
- flexible metadata definitions
- a set of interactive integrated tools for user activities with the generated DL system

To sum up, CRADLE is a DLMS aimed at supporting all the users involved in the development of a DL system and providing interfaces, data modeling, and services for user-driven generation of specific DLs. Although CRADLE does not yet satisfy all requirements for a generic DL system, it addresses issues focused on developing interactive DL systems, stressing interfaces and communication between users. Nevertheless, we employed standards when possible to leave it open for further specification or enhancements from the DL user community. Extensive use of XML-based languages allows us to change document information depending on implemented recognition algorithms so that expert users can easily model their DL by selecting the best recognition and indexing algorithms.

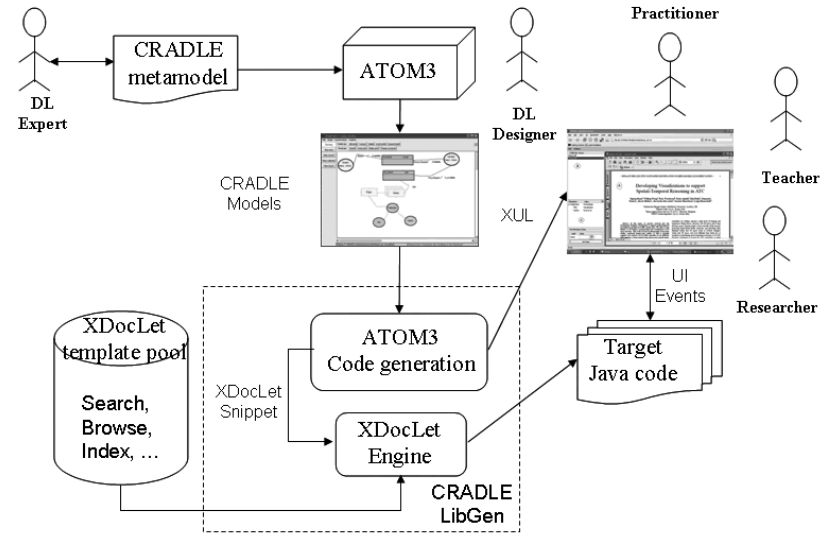

Figure 1. CRADLE architecture

CRADLE evolves from the JDAN (Java-based environment for Document Applications on Networks) platform, which managed both document images and forms on the basis of a component architecture.[40] JDAN was based on XML technologies, and its modularity allowed its integration in service-based and grid-based scenarios. It supported template code generation and modeling, but it required the designer to write XML specifications and edit XML schema files in order to model the DL document types and services, thus requiring technical knowledge that should be avoided to let users concentrate on their specific domains.

Modeling Digital Library Systems

The CRADLE framework shows a unique combination of features: it is based on a formal model, exploits a set of domain-specific languages, and provides automatic code generation. Moreover, fundamental roles are played by the concepts of society and collaboration.[41] CRADLE generates code from tools built after modeling a DL (according to the rules defined by the proposed metamodel) and performs automatic transformation and mapping from model to code to generate software tools for a given DL model.

The specification of a DL in CRADLE encompasses four complementary dimensions:

- multimedia information supported by the DL (Collection Model)
- how that information is structured and organized (Structural Model)
- the behavior of the DL (Service Model) and the different societies of actors
- groups of services acting together to carry out the DL behavior (Societal Model)

Figure 1 depicts the design approach supported by CRADLE architecture, namely, modeling the society of actors and services interacting in the domain-specific scenarios and describing the documents and metadata structure included with the library by defining a visual model for all these entities. The DL is built using a collection of stock parts and configurable components that provide the infrastructure for the new DL. This infrastructure includes the classes of objects and relationships that make up the DL, and processing tools to create and load the actual library collection from raw documents, as well as services for searching, browsing, and collection maintenance. Finally, the code generation module generates tailored DL services code stubs by composing and specializing components from the component pool.

Initially, a DL designer is responsible for formalizing (starting from an analysis of the DL requirements and characteristics) a conceptual description of the DL using metamodel concepts. Model specifications are then fed into a DL generator (written in Python for AToM3), to produce a DL tailored suitable for specific platforms and requirements. After these design phases, CRADLE generates the code for the user interface and the parts of code corresponding to services and actors interacting in the described society. A set of templates for code generation

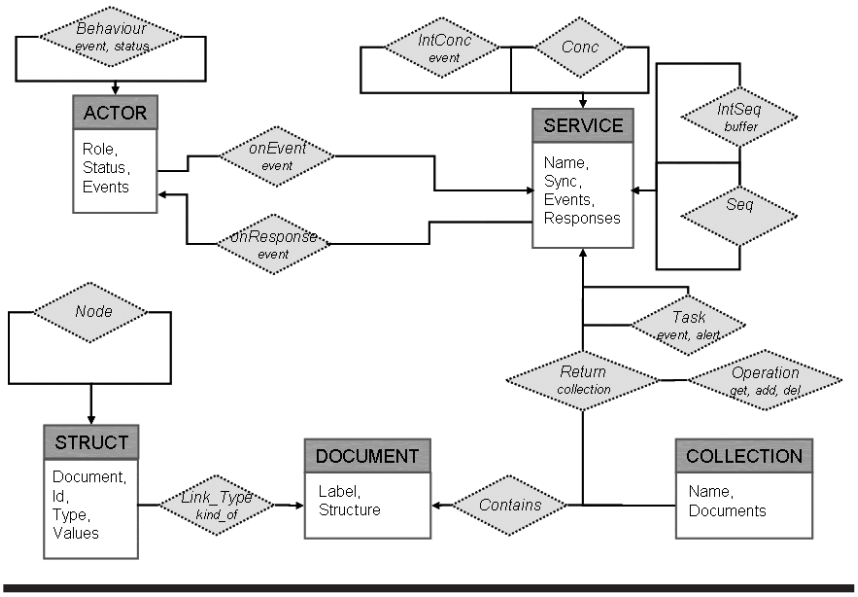

Figure 2. The CRADLE metamodel with the E/R formalism

have been built for typical services of a DL environment.

To improve acceptability and interoperability, CRADLE adopts standard specification sublanguages for representing DL concepts. Most of the CRADLE model primitives are defined as XML elements, possibly enclosing other sublanguages to help define DL concepts. In more detail, MIME types constitute the basis for encoding elements of a collection. The XML User Interface Language (XUL)[42] is used to represent appearance and visual interfaces, and XDoclet is used in the LibGen code generation module, as shown in figure 1.[43]

The Cradle Metamodel

In the CRADLE formalism, the specification of a DL includes a Collection Model describing the maintained multimedia documents, a Structural Model of information organization, a Service Model for the DL behavior, and a Societal Model describing the societies of actors and groups of services acting together to carry out the DL behavior.

A *society* is an instance of the CRADLE model defined according to a specific collaboration framework in the DL domain. A society is the highest-level component of a DL and exists to serve the information needs of its actors and to describe its context of usage. Hence a DL collects, preserves, and shares information artefacts for society members.

The basic entities in CRADLE are derived from the categorization along the *actors*, *activities*, *components*, *socioeconomic*, and *environment* dimensions. We now show in detail the entities and relations in the derived metamodel, shown in figure 2.

Actor Entities

Actors are the users of DLs. Actors interact with the DL through services (interfaces) that are (or can be) affected by the actors preferences and messages (raised events). In the CRADLE metamodel, an actor is an entity with a behavior that may concurrently generate events. Communications with other actors may occur synchronously or asynchronously. Actors can relate through services to shape a digital community, i.e., the basis of a DL society. In fact, communities of students, readers, or librarians interact with and through DLs, generally following predefined scenarios. As an example, societies can behave as query generator services (from the point of view of the library) and as teaching, learning, and working services (from the point of view of other humans and organizations). Communication between actors within the same or different societies occur through message exchange. To operate, societies need shared data structures and message protocols, enacted by sending structured sequences of queries and retrieving collections of results.

The *actor* entity includes three attributes:

1. *Role* identifies which role is played by the actor within the DL society. Examples of specific human roles include authors, publishers, editors, maintainers, developers, and the library staff. Examples of nonhuman actors include computers, printers, telecommunication devices, software agents, and digital resources in general.
2. *Status* is an enumeration of possible statuses for the actor:
   I. *None* (default value)
   II. *Active* (present in the model and actively generating events)
   III. *Inactive* (present in the model but not generating events)
   IV. *Sleeping* (present in the model and awaiting for a response to a raised event)
3. *Events* describes a list of events that can be raised by the actor or received as a response message from a *service*. Examples of events are *borrow*, *reserve*, *return*, etc. Events triggered from digital resources include *store*, *trash*, and *transfer*. Examples of response events are *found*, *not found*, *updated*, etc.

Service Entities

Services describe scenarios, activities, operations, and tasks that ultimately specify the functionalities of a DL, such as collecting, creating, disseminating, evaluating, organizing, personalizing, preserving, requesting, and selecting documents and providing services to humans concerned with fact-finding, learning, gathering, and exploring the content of a DL. All these activities can be described and implemented using scenarios and appear in the DL setting as a result of actors using services (thus societies). Furthermore, these activities realize and shape relationships within and between societies, services, and structures. In the CRADLE metamodel, the service entity models what the system is required to do, in terms of actions and processes, to achieve a task. A detailed task analysis helps understand the current system and the information flow within it in order to design and allocate tasks appropriately. The service entity has four attributes:

1. *Name* is a string representing a textual description of the service.
2. *Sync* states whether communication is synchronous or asynchronous, modeled by values *wait* and *nowait*, respectively.
3. *Events* is a list of messages that can trigger actions among services (tasks); for example, *valid* or *notValid* in case of a parsing service.
4. *Responses* contain a list of response messages that can reply to raised events; they are used as a communication mechanism by actors and services.

The Collection Entity

Collections are sets of documents of arbitrary type (e.g., bits, characters, images, etc.) used to model static or dynamic content. In the static interpretation, a collection defines information content interpreted as a set of basic elements, often of the same type, such as plain text. Examples of dynamic content include video delivered to a viewer, animated presentations, and so on. The attributes of collection are *name* and *documents*. Name is a string, while documents is a list of pairs (*DocumentName*, *DocumentLabel*), the latter being a pointer to the document entity.

The Document Entity

Documents are the basic elements in a DL and are modeled with attributes *label* and *structure*.

Label defines a textual string used by a collection entity to refer to the document. We can consider it as a document identifier, specifying a class or a type of document.

Structure defines the semantics and area of application of the document. For example, any textual representation can be seen as a string of characters, so that a text document, including scientific articles and books, becomes a sequence of strings.

The Struct Entity

A *Struct* is a structural element specifying a part of a whole. In DLs, structures represent hypertexts, taxonomies, relationships between elements, or containment. For example, books can be structured logically into chapters, sections, subsections, and paragraphs, or physically into cover, pages, line groups (paragraphs), and lines. Structures are represented as graphs, and the struct entity (a vertex) contains four attributes:

1. *Document* is a pointer to the document entity the structure refers to.
2. *Id* is a unique identifier for a structure element.
3. *Type* takes three possible values:
   I. *Metadata* denotes a content descriptor, for instance title, author, etc.
   II. *Layout* denotes the associated layout, e.g., left frame, columns, etc.
   III. *Item* indicates a generic structure element used for extending the model.
4. *Values* is a list of values describing the element content, e.g., title, author, etc.

Actors interact with services in an event-driven way. Services are connected via messages (*send* and *reply*) and can be sequential, concurrent, or task-related (when a service acts as a subtask of a macroservice). Services perform operations (e.g., *get*, *add*, and *del*) on collections, producing collections of documents as results. Struct elements are connected to each other as nodes of a graph representing metadata structures associated with documents.

The metamodel has been translated to a DSVL, associating symbols and icons with entities and relations (see "CRADLE Language and Tools" below). With respect to the six core concepts of the DELOS Manifesto (content, user, functionality, quality, policy, and architecture), content can be modeled in CRADLE as collections and structs, user as actor, and functionality as service. The quality concept is not directly modeled in CRADLE, but for quality of service we support standard service architecture. Policies can be partially modeled by services managing interaction between actors and collections, making it possible to apply standard access policies. From the architectural point of view, we follow the reference architecture of figure 1.

CRADLE Language and Tools

In this section we describe the selection of languages and tools of the CRADLE platform. To improve interoperability

and collaboration, CRADLE makes extensive use of existing standard specification languages. Most CRADLE outputs are defined with XML-based formats, able to enclose other specific languages. The basic languages and corresponding tools used in CRADLE are the following:

- *MIME type*. Multipurpose Internet Mail Extensions (MIME) constitute the basis for encoding documents in CRADLE, supporting several file formats and types of character encoding. MIME was chosen because of wide availability of MIME types, and standardisation of the approach. This makes it a natural choice for DLs where different types of documents need to be managed (PDF, HTML, Doc, etc.). Moreover, MIME standards for character encoding descriptions help keeping the CRADLE framework open and compliant with standards.
- *XUL*. The XML User Interface Language (XUL) is an XML-based markup language used to represent appearance and visual interfaces. XUL is not a public standard yet, but it uses many existing standards and technologies, including DTD and RDF,[44] which makes it easily readable for people with a background in Web programming and design. The main benefit of XUL is that it provides a simple definition of common user interface elements (widgets). This drastically reduces the software development effort required for visual interfaces.
- *XDoclet*. XDoclet is used for generating services from tagged-code fragments. It is an open-source code generation library which enables attribute-oriented programming for Java via insertion of special tags.[45] It includes a library of predefined tags, which simplify coding for various technologies, e.g., Web services. The motivation for using XDoclet in the CRADLE framework is related to its approach for template code generation. Designers can describe templates for each service (browse, query, and index) and the XDoclet generated code can be automatically transformed into the Java code for managing the specified service.
- *AToM3*. AToM3 is a metamodeling system to model graphical formalisms. Starting from a metaspecification (in E/R), AToM3 generates a tool to process models described in the chosen formalism. Models are internally represented using abstract syntax graphs. Model manipulation can then be expressed via graph grammars also specified in AToM3.

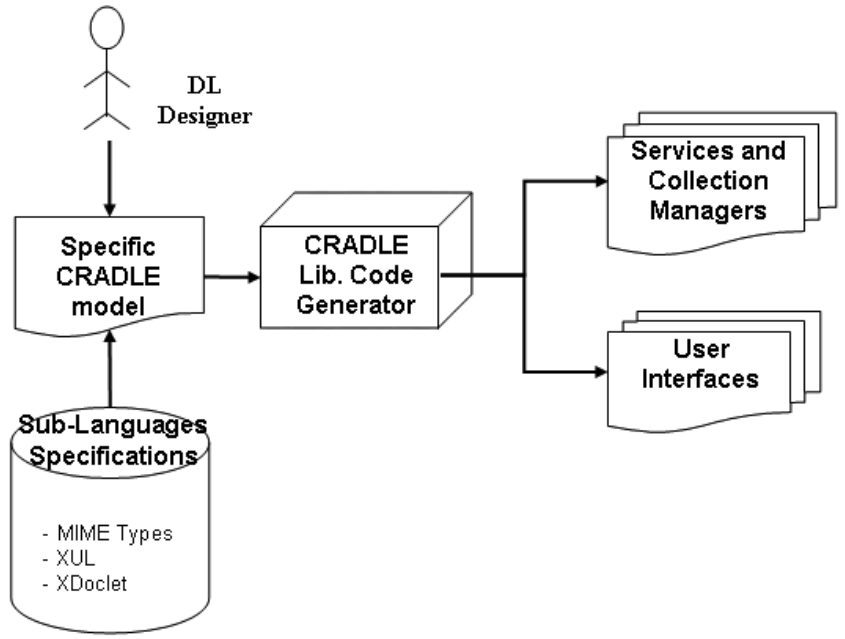

Figure 3. Cooperative DL generation process with CRADLE framework

The general process of automatic creation of cooperative DL environments for an application is shown in figure 3. Initially, a designer formalizes a conceptual description of the DL using the CRADLE metamodel concepts. This phase is usually preceded by an analysis of requirements and interaction scenarios, as seen previously. Model specifications are then provided to a DL code generator (written in Python within AToM3) to produce DLs tailored to specific platforms and requirements. These are built on a collection of templates of services and configurable components providing infrastructure for the new DL.

The sketched infrastructure includes classes for objects (tasks), relationships making up the DL, and processing tools to upload the actual library collection from raw documents, as well as services for searching and browsing and for document collections maintenance. The CRADLE generator automatically generates different kinds of output for the CRADLE model of the cooperative DL environment, such as service and collection managers.

Collection managers define the logical schemata of the DL, which in CRADLE correspond to a set of MIME types, XUL and XDoclet specifications, representing digital objects, their component parts, and linking information. Collection managers also store instances of their

classes and function as search engines for the system. Services classes also are generated and are represented as attribute-oriented classes involving parts and features of entities.

CRADLE platform

The CRADLE platform is based on a model-driven approach for the design and automatic generation of code for DLs. In particular, the DSVL for CRADLE has four diagram types (collection, structure, service, and actor) to describe the different aspects of a DL.

In this section we describe the user interface (UI) and service templates used for generating the DL tools. In particular, the UI layout is mainly generated from the structured information provided by the document, struct, and collection entities. The UI events are managed by invoking the appropriate services according to the imported XUL templates. At the service and communication levels, the XDoclet code is generated by the service and actor entities, exploiting their relationships. We also show how code generation works and the advanced platform features, such as automatic service discovery. At the end of the section a running example is shown, representing all the phases involved in using the CRADLE framework for generating the DL tools for a typical library scenario.

User Interface Templates

The generation of the UI is driven by the visual model designed by the CRADLE user. Specifically, the model entities involved in this process are document, struct and collection (see figure 2) for the basic components and layout of the interfaces, while linked services are described in the appropriate templates.

The code generation process takes place through transformations implemented as actions in the AToM3 metamodel specification, where graph-grammar rules may have a condition that must be satisfied for the rule to be applied (preconditions), as well as actions to be performed when the rule is executed (postconditions). A transformation is described during the visual modeling phase in terms of conditions and corresponding actions (inserting XUL language statements for the interface in the appropriate code template placeholders). The generated user interface is built on a set of XUL template files that are automatically specialized on the basis of the attributes and relationships designed in the visual modeling phase.

The layout template for the user interface is divided into two columns (see figure 4). The left column is made of three boxes: (1) the collection box (2) the metadata box, and (3) the metadata operations box.

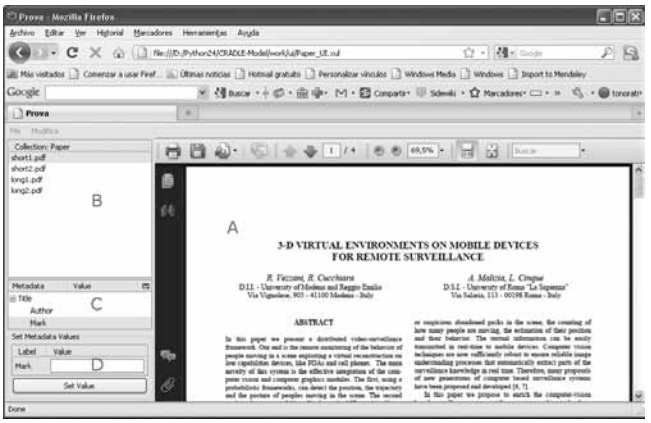

Figure 4. An example of an automatically generated user interface. (A) document area; (B) collection box; (C) metadata box; (D) metadata operations box.

The right column manages visualization and multimedia information obtained from documents. The basic features provided with the UI templates are document loading, visualization, metadata organization, and management.

The layout template, in the collection box, manages the visualization of the documents contained in a collection, while the visualization template works according to the data (MIME) type specified by the document. Actually, by selecting a document included in the collection, the corresponding data file is automatically uploaded and visualized in the UI.

The metadata visualization in the code template reflects the metadata structure (a tree) represented by a struct, specifying the relationship between parent and child nodes. Thus the XUL template includes an area (the metadata box) for managing tree structures as described in the visual model of the DL. Although the tree-like visualization has potential drawbacks if there are many metadata items, there should be no real concern with medium loads.

The UI template also includes a box to perform operations on metadata, such as insert, delete, and edit. Users can select a value in the metadata box and manipulate the presented values. Figure 4 shows an example of a UI generated from a basic template.

Service Templates

To achieve automated code generation, we use XDoclet to specify parameters and service code generation according to such parameters. CRADLE can automatically annotate Java files with name–value pairs, and XDoclet provides a syntax for parameter specification. Code generation is

based on code templates. Hence service templates are XDoclet templates for transforming XDoclet code fragments obtained from the modeled service entities.

The basic XDoclet template manages messages between services, according to the event and response attributes described in "CRADLE Language and Tools" above. In fact, CRADLE generates a Java application (a service) that needs to receive messages (event) and reply to them (response) as parameters for the service application. In XDoclet, these can be attached to the corresponding field by means of annotation tags, as in the following code segments:

```
public class MSGArguments {
. . . . . .
/*
* @msg arguments.argname name="event "
   desc="event_string "
*/ protected String eventMsg = null;
/*
* @msg arguments.argname name="response"
* desc="response_string "
*/ protected String responseMsg = null;
}
```

Each msg arguments.argname related to a field is called a field tag. Each field tag can have multiple parameters, listed after the field tag. In the tag name msg arguments .argname, the prefix serves as the namespace of all tags for this particular XDoclet application, thus avoiding naming conflicts with other standard or customized XDoclet tags. Not only fields can be annotated, but also other entities such as class and functions can have tags too.

XDoclet enables powerful code generation requiring little or no customization (depending on how much is provided by the template). The type of code to be generated using the parameters is defined by the corresponding XDoclet template.

We have created template files composed of Java codes and special XDoclet instructions in the form of XML tags. These XDoclet instructions allow conditionals (if) and loops (for), thus providing us with expressive power close to a programming language. In the following example, we first create an array containing labels and other information for each argument:

```
public class <XDtClass : classOf>
<XDtClass : className/>Impl</XDtClass :
   classOf> extends
<XDtClass : classOf><XDtClass : className/>
   </XDtClass : classOf> {
public static String[ ][ ] argumentNames = new
   String[ ][ ] {
<XDtField : forAllFields>
<XDtField : ifHasFieldTag tagName=
   "msg arguments.argname">
{ "<XDtField : fieldName/>" ,
"<XDtField : fieldTagValue tagName=
   "msg arguments.argname"
paramName="name"/>"
"<XDtField : fieldTagValue tagName=
   "msg arguments.argname"
paramName=" desc "/>"
} ,
</XDtField : ifHasFieldTag>
</XDtField : forAllFields> };
```

The first two lines declare a class with a name class nameImpl that extends the class name. The XDoclet template tag XDtClass:className denotes the name of the class in the annotated Java file. All standard XDoclet template tags have a namespace starting with "XDt."

The rest of the template uses XDtField : forAllField to iterate through the fields. For each field with a tag named msg arguments.argname (checked using XDtField : ifHasFieldTag), it creates a subarray of strings using the values obtained from the field tag parameters. XDtField : fieldName gives the name of the field, while XDtField : fieldTagValue retrieves the value of a given field tag parameter. Characters that are not part of some XDoclet template tags are directly copied into the generated code. The following code segment was generated by XDoclet using the annotated fields and the above template segment:

```
public class MSGArgumentsImpl extends
   MSGArguments {
public static String[ ][ ] argumentNames = new
   String[ ][ ]{ {
"eventMsg" ,
" event " ,
" eventstring "
} ,
{
" responseMsg " ,
" response " ,
" responsestring "
} ,
};
}
Similarly, we generate the getter and setter
   methods for each field:
<XDtField : forAllFields > <XDtField : ifHasFieldTag
tagName="msg arguments.argname">
public <XDtField : fieldType/> get <XDtField :
   fieldName />() {
return <XDtField : fieldName />;
}
public void set <XDtField : fieldName />
   ( String value ) {
```

```
setValue ( "<XDtField : fieldName/>" , value ) ;
}<
/XDtField : ifHasFieldTag>
</XDtField : forAllFields >
This translates into the following generated code:
public java.lang.String get eventMsg ( ) {
return eventMsg ;
}
public void set eventMsg ( String value ) {
setValue ( "eventMsg" , value ) ;
}
public java.lang.String getresponseMsg ( ) {
return getresponseMsg ;
}
public void setresponseMsg ( String value ) {
setValue ( " responseMsg " , value ) ;
}
```

The same template is used for managing the name and sync attributes of service entities.

Code Generation, Service Discovery, and Advanced Features

A service or interface template only describes the solution to a particular design problem—it is not code. Consequently, users will find it difficult to make the leap from the template description to a particular implementation even though the template might include sample code. Others, like software engineers, might have no trouble translating the template into code, but they still may find it a chore, especially when they have to do it repeatedly. The CRADLE visual design environment (based on AToM3) helps alleviate these problems. From just a few pieces of information (the visual model), typically application-specific names for actors and services in a DL society along with choices for the design tradeoffs, the tool can create class declarations and definitions implementing the template. The ultimate goal of the modeling effort remains, however, the production of reliable and efficiently executable code. Hence a code generation transformation produces interface (XUL) and service (Java code from XDoclet templates) code from the DL model.

We have manually coded XUL templates specifying the static setup of the GUI, the various widgets and their layout. This must be complemented with code generated from a DL model of the systems dynamics coded into services. While other approaches are possible,[46] we employed the solution implemented within the AToM3 environment according to its graph grammar modeling approach to code generation.

CRADLE supports a flexible iterative process for visual design and code generation. In fact, a design change might require substantial reimplementation because different design choices in the template can lead to vastly different code. We have included an incremental mechanism by which users can modify the visual model of a DL and regenerate (XUL interface) code only for the modifications. By employing this solution, librarians and DL designers can work as they would on paper by designing the visual scheme and collaboratively updating and changing it. They can generate the code, verify the implementation, and, if something has to be changed, go back to the visual model, apply modification, and generate code in a new iteration of the process. Once the visual model has been modified, the system incrementally updates the code by examining only those model parts affected by the edit and modifying the corresponding parts of the generated code.

The same approach could be used for services but with a different technique. In fact, predefined templates exist for basic services, e.g., indexing, uploading, and querying. To allow service providers to add new code to the rest of the service component list, we have implemented a registry listing the available service templates. When the user runs the code generation process, a routine verifies if the service templates included in the model are available in the registry and loads it into memory for the code generation process.

We are planning to support a standard mechanism based on the Universal Description, Discovery, and Integration registry.[47] Moreover, we have developed an advanced interface template that embeds validation code into the XUL templates for the interfaces to look up the list of services made available by the interface at run-time. If there are services embedded in the interface but not available, the interface is modified to prevent access to them. For instance, suppose that an interface is specified with buttons to access to the document upload and edit services. If, at run-time, the check does not find the edit service available, the interface will present only the button for the upload service.

Generating a Digital Library Environment

As a first step in designing the digital library environment in the CRADLE framework, designers model the society involved in the specific scenario. We define a running example, called Library, to show the process, starting from the basic entities of the model. We consider modeling a simple DL environment. The involved actors are students and librarians. The DL Collection consists of Digital Paper Documents with publication, author, and title metadata information (struct entities). In figure 5, the CRADLE environment (a society) is shown together with the defined entities. Circles represent actors in the model, rectangles render services, multiple rectangles represent

collections, while a single rectangle connected to a collection represents a document entity; the circles linked to the document entity are the struct (metadata) entities. Metadata entities are linked to the node relationships (organized as a tree) and linked to the document entity by a metadata LinkType relationship.

The search service is synchronous (*sync* attribute set to "wait"). It queries the document collection (*get* operation) looking for the requested document (using metadata information provided by the borrow request), and waits for the result of get (a collection of documents). Based on this result, the service returns a Boolean message "Is_Available," which is then propagated as a response to the librarian and eventually to the student, as shown in figure 5.

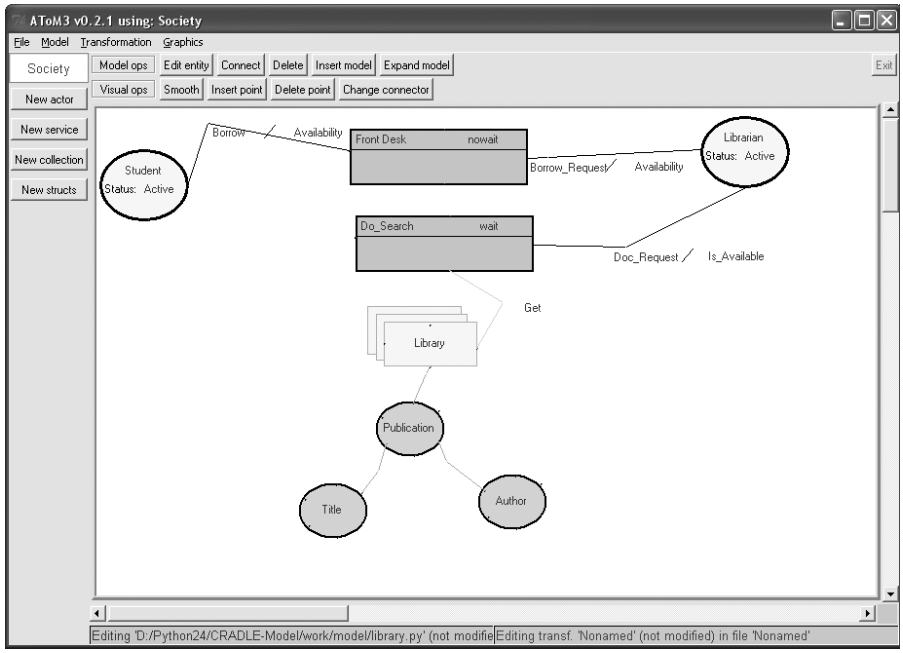

Figure 5. The Library model, alias the model of the Library society

When the library designer has built the model, the transformation process can be run, executing the code generation actions associated with the entities and services represented in the model. The code generation process is based on template code snippets generated from the AToM3 environment graph transformation engine, following the generative rules of the metamodel. We also use pre– and postconditions on application of transformation rules to have code generation depend on verification of some property.

The generated UI is presented in figure 6. On the right side, the document area is presented according to the XUL template. Documents are managed according to their MIME type: the PDF file of the example is loaded with the appropriate Adobe Acrobat Reader plug-in.

On the left column of the UI are three boxes, according to the XUL template. The collection box—figure 6(B)—presents the list of documents contained in the collection specified by the documents attribute of the library collection entity, and allows users to interact with documents. After selecting a document by clicking on the list, it is presented in the document area—figure 6(A)—where it can be managed (edit, print, save, etc.).

In the metadata box—figure 6(C)—the tree structure of the metadata is depicted according to the categorization modeled by the designer. The XUL template contains all the basic layout and action features for managing a tree structure. The generated box contains the parent and child nodes according to the attributes specified in the corresponding struct elements. The user can click on the root for compacting or exploding the tree nodes; by selecting one, the UI activates the metadata operations box—figure 6(D). The selected metadata node will then be presented in the lower (metadata operations) box, labeled "set MetaData Values," replacing the default "None" value as shown in figure 6. After the metadata item is presented, the user can edit its value and save it by clicking on the "set value" button. The associated action saves the metadata information and causes its display in the intermediate box (tree-like structure), changing the visualization according to the new values.

The code generation process for the Do_Search and Front Desk services is based on XDoclet templates. In particular, a message listener template is used to generate the Java code for the Front Desk service. In fact, the Front Desk service is asynchronous and manages communications between actors. The actors classes are generated also by using the services templates since they have attributes, events, and messages, just like the services. The Do_Search service code is based on the producer and consumer templates, since it is synchronous by definition in the modeled scenario. A get method retrieving a collection of documents is implemented from the getter template.

The routine invoked by the transformation action for struct entities performs a breadth-first exploration of the metadata tree in the visual model and attaches the corresponding XUL code for displaying the struct node in the correct position within the graph structure of the UI.

Designing and Generating Advanced Collaborative DL Systems

In this section we show the use of CRADLE as an analytical tool helpful in comprehending specific DL phenomena, to present the complex interplays that occur between CRADLE components and DL concepts in a real DL application, and to illustrate the possibility of using CRADLE as a tool to design and generate advanced tools for DL development.

Modeling Document Images Collections

With CRADLE, the designer can provide the visual model of the DL Society involved in document management and the remaining phases are automatically carried out by CRADLE modules and templates. We have provided the user with basic code templates for the recognition and indexing services, the data-entry plug-in, and archive release. The designer can thus simply translate the particular DL society into the corresponding visual model within the CRADLE visual modeling editor.

As a proof of concept, figure 7 models the JDAN architecture, introduced in "Requirements for Modeling Digital Libraries," exploiting the CRADLE visual language. The Recognition Service performs the automatic document recognition and stores the corresponding document images, together with the extracted metadata in the Archive collection. It interacts with the Scanner actor, representing a machine or a human operator that scans paper documents. Designers can choose their own segmentation method or algorithm; what is required to be compliant with the framework is to produce an XDoclet template. It stores the document images into the Archive collection, with its different regions layout information according to the XML metadata schema provided by the designer. If there is at least one region marked as "not interpreted," the Data-Entry service is invoked on the "not interpreted" regions.

The Data-Entry service allows Operators to evaluate the automatic classification performed by the system and edit the segmentation for indexing. Operators can also edit the recognized regions with the classification engine (included in the Recognition service) and adjust their values and sizes. The output of this phase is an XML description that will be imported in the Indexing service for indexing (and eventually querying).

The Archive collection stores all of the basic information kept in JDAN, such as text labels, while the Indexing service, based on a multitier architecture, exploiting JBoss 3.0, has access to them. This service is responsible for turning the data fragments in the Archive collection into useful forms to be presented to the final users, e.g., a report or a query result.

The final stage in the recognition process could be to release each document to a content management or workflow system. The Release collection maintains the image files in a permanent storage, while data is written to the target database or content management software, together with XML metadata snippets (e.g., to be stored in XML native DBMS).

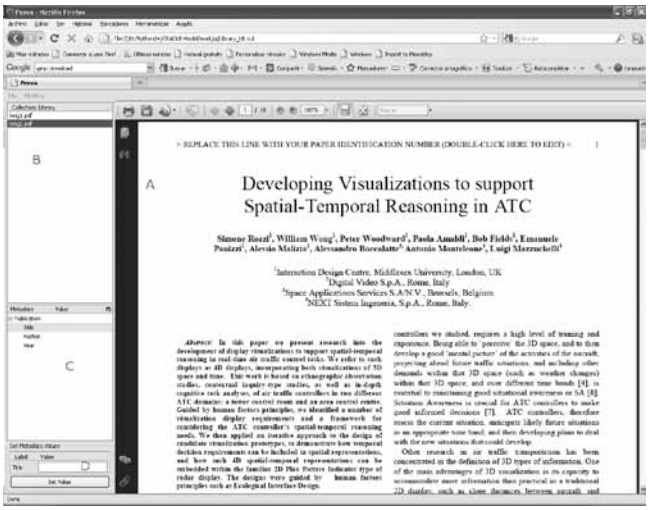

Figure 6. The UI generated by CRADLE transforming the Library model in XUL and XDocLet code

A typical configuration would have the Recognition service running on a server cluster, with many Data-Entry services running on different clients (Web browsers directly support XUL interfaces). Whereas current document capture environments are proprietary and closed, the definition of an XML-based interchange format allows the suitable assembly of different component-based technologies in order to define a complex framework.

The realization of the JDAN DL system within the CRADLE framework can be considered as a preliminary step in the direction of a standard multimedia document managing platform with region segmentation and classification, thus aiming at automatic recognition of image database and batch acquisition of multiple multimedia documents types and formats.

Personal and Collaborative Spaces

A personal space is a virtual area (within the DL society) that is modeled as being owned and maintained by a user including resources (document collections, services, etc.), or references to resources, which are relevant to a task, or set of tasks, the user needs to carry out in the DL. Personal spaces may thus contain digital documents in multiple media, personal schedules, visualization tools, and user agents (shaped as services) entitled with various tasks. Resources within personal spaces can be allocated

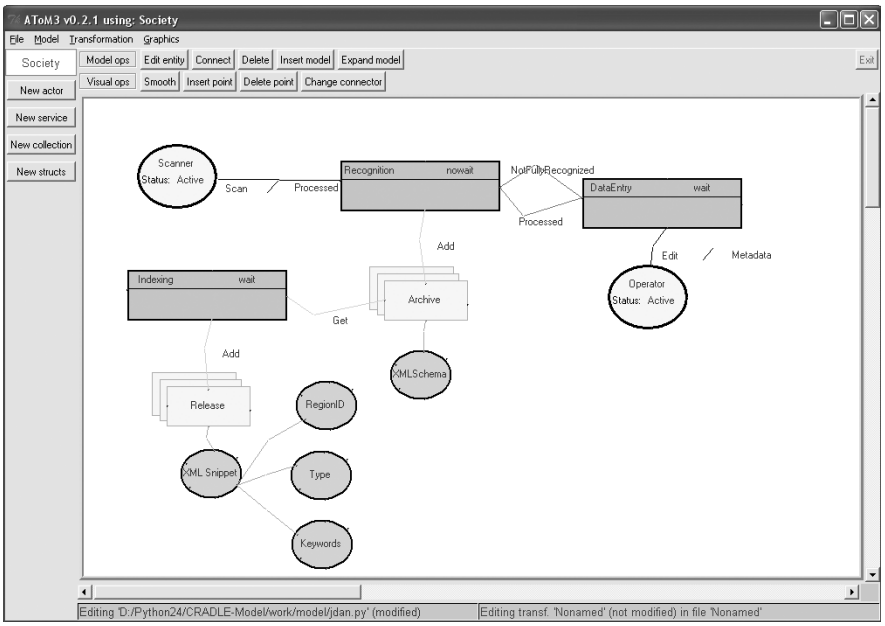

Figure 7. The CRADLE model for the JDAN framwork

according to the user's role. For example, a conference chair would have access to conference-specific materials, visualization tools and interfaces to upload papers for review by a committee. Similarly, we denote a group space as a virtual area in which library users (the entire DL society) can meet to conduct collaborative activities synchronously or asynchronously. Explicit group spaces are created dynamically by a designer or facilitator who becomes (or appoints) the owner of the space and defines who the participants will be. In addition to direct user-to-user communication, users should be able to access library materials and make annotations on them for every other group to see. Ideally, users should be able to act (and carry DL materials with them) between personal and group spaces or among group spaces to which they belong.

It may also be the case, however, that a given resource is referenced in several personal or group spaces. Basic functionality required for personal spaces includes capabilities for viewing, launching, and monitoring library services, agents, and applications. Like group spaces, personal spaces should provide users with the means to easily become aware of other users and resources that are present in a given group space at any time, as well as mechanisms to communicate with other users and make annotations on library resources.

We employed this personal and group space paradigm in modeling a collaborative environment in the Academic Conferences domain, where a Conference Chair can have a personal view of the document collections (resources) and metadata, but also can share information with the various committees collaborating for certain tasks.

Evaluation

In this section we evaluate the presented approach from three different perspectives: usability of the CRADLE notation, its expressiveness, and usability of the generated DLs.

Usability of CRADLE Notation

We have tested it by using the well known Cognitive Dimensions framework for notations and visual language design.[48] The dimensions are usually employed to evaluate the usability of a visual language or notation, or as heuristics to drive the design of innovative visual languages. The significant results are as follows.

Abstraction Gradient

An abstraction is a grouping of elements to be treated as one entity. In this sense, CRADLE is abstraction-tolerant. It provides entities for high-level abstractions of communication processes and services. These abstractions are intuitive as they are visualized as the process they represent (services with events and responses) and easy to learn as their configuration implies few simple attributes. Although CRADLE does not allow users to build new abstractions, the E/R formalism is powerful enough to provide basic abstraction levels.

Closeness of Mapping

CRADLE elements have been assigned icons to resemble their real-world counterparts (e.g., a collection is represented as a set of paper sheets). The elements that do not have a correspondence with a physical object in the real world have icons borrowed from well-known notations (e.g., structs represented as graph nodes).

Consistency

A notation is consistent if a user knowing some of its structure can infer most of the rest. In CRADLE, when two elements represent the same entity but can be used either as input or as output, then their shape is equal but incorporates an incoming or an outgoing message in order to differentiate them. See, for example, the icons for services or those for graph nodes representing either a

struct or an actor, with different colors.

Diffuseness/Terseness

A notation is diffuse when many elements are needed to express one concept. CRADLE is terse and not diffuse because each entity expresses a meaning on its own.

Error-Proneness

Data flow visualization reduces the chance of errors at a first level of the specification. On the other hand, some mistakes can be introduced when specifying visual entities, since it is possible to express relations between source and target models which cannot generate semantically correct code. However, these mistakes should be considered "programming errors more than slips," and may be detected through progressive evaluation.

Hidden Dependencies

A hidden dependency is a relation between two elements that is not visible. In CRADLE, relevant dependencies are represented as data flows via directed links.

Progressive Evaluation

Each DL model can be tested as soon as it is defined, without having to wait until the whole model is finished. The visual interface for the DL can be generated with just one click, and services can be subsequently added to test their functionalities.

Viscosity

CRADLE has a low viscosity because making small changes in a part of a specification does not imply lots of readjustments in the rest of it. One can change properties, events or responses and these changes will have only local effect. The only local changes that could imply performing further changes by hand are deleting entities or changing names; however, this would imply minimal changes (just removing or updating references to them) and would only affect a small set of subsequent elements in the same data flow.

Visibility

A DL specification consists of a single set of diagrams fitting in one window. Empirically, we have observed that this model usually involves no more than fifteen entities. Different, independent CRADLE models can be simultaneously shown in different windows.

Expressiveness of CRADLE

The paper has illustrated the expressiveness of CRADLE by defining different entities end relationships for different DL requisites. To this end, two different applications have been considered: (1) a basic example elaborated with the collaboration of the Information Science School of "Sapienza" University of Rome (undergraduate students), shown in figure 5, and (2) an application employed with a project of Records Management in a collaboration between the Computer Science and the Computer Engineering Department of "Sapienza" University, as shown in figure 7.

Usability of the Generated Tools

Environments for single-view languages generated with AToM3 have been extensively used, mostly in an academic setting, in different areas like software and Web engineering, modeling, and simulation; urban planning; etc. However, depending on the kind of the domain, generating the results may take some time. For instance, the state reachability analysis in the DL example takes a few minutes; we are currently employing a version of AToM3 that includes Petri-nets formalism where we can test the services states reachability.[49] In general, from application experience, we note the general agreement that automated syntactical consistency support greatly simplifies the design of complex systems. Finally, some users pointed out some technical limitations of the current implementation, such as the fact that it is not possible to open several views at a time.

Altogether, we believe this work contributes to make more efficient and less tedious the definition and maintenance of environments for DLS. Our model-based approach must be contrasted with the programming-centric approach of most CASE tools, where the language and the code generation tools are hard-coded so that whenever a modification has to be done (whether on the language or on the semantic domain) developers have to dive into the code.

Conclusions and Future Work

DLs are complex information systems that integrate findings from disciplines such as hypertext, information retrieval, multimedia, databases, and HCI. DL design is often a multidisciplinary effort, including library staff and computer scientists. Wasted effort and poor interoperability can therefore ensue. Examining the related bibliography, we noted that there is a lack of tools or automatic systems for designing and developing cooperative DL systems. Moreover, there is a need for modeling interactions between DLs and users, such as scenario or activity-based approaches.

The CRADLE framework fulfills this gap by providing a model-driven approach for generating visual interaction tools for DLs, supporting design and automatic generation of code for DLs. In particular, we use a metamodel made of different diagram types (collection, structures, service, and

society), which describe the different aspects of a DL. We have built a code generator able to produce XUL code from the design models for the DL user interface. Moreover, we use template code generation integrating predefined components for the different services (XDoclet language) according to the model specification.

Extensions of CRADLE with behavioral diagrams and the addition of analysis and simulation capabilities are under study. These will exploit the new AToM3 capabilities for describing multiview DSVLs, to which this work directly contributed.

References

1. A. M. Gonçalves, E. A Fox, "5SL: a language for declarative specification and generation of digital libraries," *Proc. JCDL '02* (New York: ACM, 2002): 263–72.

2. L. Candela et al., "Setting the Foundations of Digital Libraries: The DELOS Manifesto," *D-Lib Magazine* 13 (2007), http://www.dlib.org/dlib/march07/castelli/03castelli.html (accessed Oct 18, 2010).

3. A. Malizia et al., "A Cooperative-Relational Approach to Digital Libraries," *Proc. ECDL 2007*, LNCS 4675 (Berlin: Springer, 2007): 75–86.

4. E. A. Fox and G. Marchionini, "Toward a Worldwide Digital Library," *Communications of the ACM* 41 (1998): 29–32.

5. M. A. Gonçalves et al., "Streams, Structures, Spaces, Scenarios, Societies (5s): A Formal Model for Digital Libraries," *ACM Transactions on Information Systems* 22 (2004): 270–312.

6. J. C. R. Licklider, *Libraries of the Future* (Cambridge, Mass.: MIT Pr., 1965).

7. D. M. Levy and C. C. Marshall, "Going Digital: A Look at Assumptions Underlying Digital Libraries," *Communications of the ACM* 38 (1995): 77–84.

8. R. Reddy and I. Wladawsky-Berger, "Digital Libraries: Universal Access to Human Knowledge—A Report to the President," 2001, www.itrd.gov/pubs/pitac/pitac-dl-9feb01.pdf (accessed Mar. 16, 2010).

9. E. L. Morgan, "MyLibrary: A Digital Library Framework and Toolkit," *Journal of Information Technology & Libraries* 27 (2008): 12–24.

10. T. R. Kochtanek and K. K. Hein, "Delphi Study of Digital Libraries," *Information Processing Management* 35 (1999): 245–54.

11. S. E. Howe et al., "The President's Information Technology Advisory Committee's February 2001 Digital Library Report and Its Impact," In *Proc. JCDL '01* (New York: ACM, 2001): 223–25.

12. N. Reyes-Farfan and J. A. Sanchez, "Personal Spaces in the Context of OA," *Proc. JCDL '03* (IEEE Computer Society, 2003): 182–83.

13. M. Wirsing, *Report on the EU/NSF Strategic Workshop on Engineering Software-Intensive Systems*, 2004, http://www.ercim.eu/EU-NSF/sis.pdf (accessed Oct 18, 2010)

14. S. Kelly and J.-P. Tolvanen, *Domain-Specific Modeling: Enabling Full Code Generation* (Hoboken, N.J.: Wiley, 2008).

15. H. R. Turtle and W. Bruce Croft, "Evaluation of an Inference Network-Based Retrieval Model," *ACM Transactions on Information Systems* 9 (1991): 187–222.

16. R. A. Baeza-Yates, B. A. Ribeiro-Neto, *Modern Information Retrieval* (Reading, Mass.: Addison-Wesley, 1999).

17. D. Lucarella and A. Zanzi, "A Visual Retrieval Environment for Hypermedia Information Systems," *ACM Transactions on Information Systems* 14 (1996): 3–29.

18. B. Wang, "A Hybrid System Approach for Supporting Digital Libraries," *International Journal on Digital Libraries* 2 (1999): 91–110,.

19. D. Castelli, C. Meghini, and P. Pagano, "Foundations of a Multidimensional Query Language for Digital Libraries," in *Proc. ECDL '02*, LNCS 2458 (Berlin: Springer, 2002): 251–65.

20. R. N. Oddy et al., eds., *Proc. Joint ACM/BCS Symposium in Information Storage & Retrieval* (Oxford: Butterworths, 1981).

21. K. Maly, M. Zubair et al., "Scalable Digital Libraries Based on NCSTRL/DIENST," in *Proc. ECDL '00* (London: Springer, 2000): 168–79.

22. R. Tansley, M. Bass and M. Smith, "DSpace as an Open Archival Information System: Current Status and Future Directions," *Proc. ECDL '03*, LNCS 2769 (Berlin: Springer, 2003): 446–60.

23. K. M. Anderson et al., "Metis: Lightweight, Flexible, and Web-Based Workflow Services for Digital Libraries," *Proc. 3rd ACM/IEEE-CS JCDL '03* (Los Alamitos, Calif.: IEEE Computer Society, 2003): 98–109.

24. N. Dushay, "Localizing Experience of Digital Content via Structural Metadata," In *Proc. 2nd ACM/IEEE-CS JCDL '02* (New York: ACM, 2002): 244–52.

25. M. Gogolla et al., "Integrating the ER Approach in an OO Environment," *Proc. ER, '93* (Berlin: Springer, 1993): 376–89.

26. Heidi Gregersen and Christian S. Jensen, "Temporal Entity-Relationship Models—A Survey," *IEEE Transactions on Knowledge & Data Engineering* 11 (1999): 464–97.

27. B. Berkem, "Aligning IT with the Changes using the Goal-Driven Development for UML and MDA," *Journal of Object Technology* 4 (2005): 49–65.

28. A. Malizia, E. Guerra, and J. de Lara, "Model-Driven Development of Digital Libraries: Generating the User Interface," *Proc. MDDAUI '06*, http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-214/ (accessed Oct 18, 2010).

29. D. L. Atkins et al., "MAWL: A Domain-Specific Language for Form-Based Services," *IEEE Transactions on Software Engineering* 25 (1999): 334–46.

30. J. de Lara and H. Vangheluwe, "AToM3: A Tool for Multi-Formalism and Meta-Modelling," *Proc. FASE '02* (Berlin: Springer, 2002): 174–88.

31. J. M. Morales-Del-Castillo et al., "A Semantic Model of Selective Dissemination of Information for Digital Libraries," *Journal of Information Technology & Libraries* 28 (2009): 21–30.

32. N. Santos, F. C. A. Campos, and R. M. M. Braga, "Digital Libraries and Ontology," in *Handbook of Research on Digital Libraries: Design, Development, and Impact,* ed. Y.-L. Theng et al. (Hershey, Pa.: Idea Group, 2008): 1:19.

33. F. Wattenberg, "A National Digital Library for Science, Mathematics, Engineering, and Technology Education," *D-Lib Magazine* 3 no. 10 (1998), http://www.dlib.org/dlib/october98/wattenberg/10wattenberg.html (accessed Oct 18, 2010); L. L. Zia, "The NSF National Science, Technology, Engineering, and Mathematics Education Digital Library (NSDL) Program: New Projects and a Progress Report," *D-lib Magazine*, 7, no. 11 (2002), http://www.dlib.org/dlib/november01/zia/11zia.html (accessed Oct 18, 2010).

34. U.S. Library of Congress, Ask a Librarian, http://www.loc

.gov/rr/askalib/ (accessed on Mar. 16, 2010).

35. C. L. Borgmann, "What are Digital Libraries? Competing Visions," *Information Processing & Management* 25 (1999):227–43.

36. C. Lynch, "Coding with the Real World: Heresies and Unexplored Questions about Audience, Economics, and Control of Digital Libraries," In *Digital Library Use: Social Practice in Design and Evaluation*, ed. A. P. Bishop, N. A. Van House, and B. Buttenfield (Cambridge, Mass.: MIT Pr., 2003): 191–216.

37. Y. Ioannidis et al., "Digital Library Information-Technology Infrastructure," *International Journal of Digital Libraries* 5 (2005): 266–74.

38. E. A. Fox et al., "The Networked Digital Library of Theses and Dissertations: Changes in the University Community," *Journal of Computing Higher Education* 13 (2002): 3–24.

39. H. Van de Sompel and C. Lagoze, "Notes from the Interoperability Front: A Progress Report on the Open Archives Initiative," *Proc. 6th ECDL, 2002,* LNCS 2458 (Berlin: Springer 2002): 144–57.

40. F. De Rosa et al., "JDAN: A Component Architecture for Digital Libraries," *DELOS Workshop: Digital Library Architectures*, (Padua, Italy: Edizioni Libreria Peogetto, 2004): 151–62.

41. Defined as a set of actors (users) playing roles and interacting with services.

42. Mozilla Developer Center, XUL, https://developer.mozilla.org/En/XUL (accessed Mar. 16, 2010).

43. XDoclet, Welcome! What is XDoclet? http://xdoclet.sourceforge.net/xdoclet/index.html (accessed Mar. 16, 2010).

44. W3C, Extensible Markup Language (XML) 1.0 (Fifth Edition), http://www.w3.org/TR/2008/REC-xml-20081126/ (accessed Mar. 16, 2010); W3C, Resource Description Framework (RDF), http://www.w3.org/RDF/ (accessed Mar. 16, 2010).

45. H. Wada and J. Suzuki, "Modeling Turnpike Frontend System: A Model-Driven Development Framework Leveraging UML Metamodeling and Attribute-Oriented Programming," *Proc. MoDELS '05*, LNCS 3713 (Berlin: Springer, 2005): 584–600.

46. I. Horrocks, *Constructing the User Interface with Statecharts* (Boston: Addison-Wesley, 1999).

47. Universal Discover, Description, and Integration OASIS Standard, Welcome to UDDI XML.org, http://uddi.xml.org/ (accessed Mar. 16, 2010).

48. T. R. G. Green and M. Petre, "Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions Framework,'" *Journal of Visual Languages & Computing* 7 (1996): 131–74.

49. J. de Lara, E. Guerra, and A. Malizia, "Model Driven Development of Digital Libraries—Validation, Analysis and Formal Code Generation," *Proc. 3rd WEBIST '07* (Berlin: Springer, 2008).