in: Proc. Int. Workshop "Web Databases", Wien, Sept. 2001

# Supporting Collaborative Authoring of Web Content by Customizable Resource Repositories

Jernej Kovse, Theo Härder, Norbert Ritter, Hans-Peter Steiert, Wolfgang Mahnke
Department of Computer Science
University of Kaiserslautern
P.O. Box 3049, 67653 Kaiserslautern, Germany
e-mail: {kovse, haerder, ritter, steiert, mahnke}@informatik.uni-kl.de

## Abstract

*The process of collaborative authoring of Web content may be supported by resource repositories which provide storage facilities as well as certain value-added services. However, since specific demands posed by the authoring process may vary according to special characteristics of such a process, resource repositories with predefined services usually address these demands only to a limited extent. To overcome this problem, we propose a way of generating customized repositories for collaborative authoring of Web content. Our approach involves a set of frameworks that can be adapted and applied to the specification of repository services in order to generate customized repositories with services tailored to the requirements of the authoring process.*

**Keywords:** *Repositories, Collaborative Authoring, Web Application Modeling, Versioning, ORDBMS*

## 1. Introduction

Over the last couple of years, the World Wide Web (Web) has significantly evolved in various aspects primarily concerning information content available to the user. As a first factor, the user demand for information has caused the number of web pages to grow, increasing the *quantity* of resources providing information. Second, information content provided by such resources has improved in *quality*. Once relatively poor content is increasingly being replaced by semantically richer content, which makes the goal of the Web to become a serious medium for information exchange closer to reality.

However, today the Web is still primarily being used for information retrieval. Accessing a resource in order to retrieve and use the provided information content is far more common than publishing a content and making it available to other users. This way, the potential of bidirectional information exchange on the Web is neglected and the majority of information flow from the user is supported by other technologies, such as electronic mail. Ideally, the Web could be used for both, the retrieval of existing information content as well as publishing of content authored by human as well as certain types of software agents.

Information content publishing may also be performed collaboratively, where multiple geographically dispersed authors contribute to the publishing of semantically related pieces of information (Fig. 1). The authors bring together expert knowledge needed to provide parts of the content. Collaboration enables the authors to work together on the authoring tasks by sharing the knowledge needed in the process of producing content. The approach poses various demands to tools and technologies supporting such cooperation, such as concurrency control (multiple authors trying to modify a certain part of the content at the same time) or version control of parts of published content, for example. Various approaches, such as the WebDAV (Web Distributed

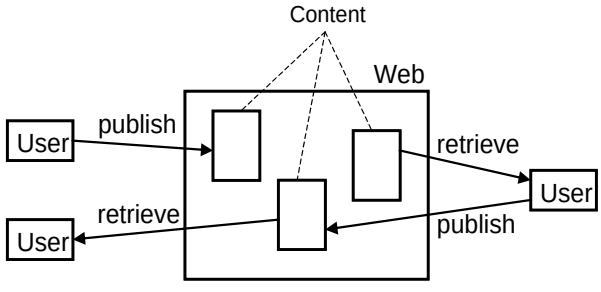


**Figure 1: Users retrieving and publishing Web content.**

Authoring and Versioning [10]) protocol, have emerged to address these demands.

The process of collaboratively authoring Web content should not be concerned merely with the authoring of the static parts of the content, such as static Web pages or graphics, but should also consider the possibilities of collaboratively authoring the functional parts needed to produce the content dynamically. This kind of parts may be provided using approaches that involve the established mechanisms addressing the dynamic content generating, such as scripts or servlets. Fig. 2 illustrates the variety of collaboratively authored items that relate either directly to the static Web content or the functionality needed to generate the content dynamically.

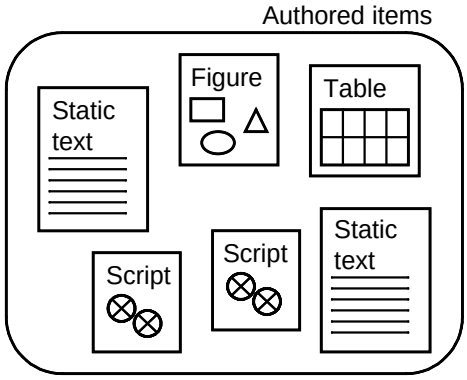


**Figure 2: Collaboratively authored items.**

An important aspect of collaborative authoring is the mechanism of storing published content as well as the items related to the functionality needed to produce the content dynamically. A storage mechanism is used as a supporting technology for providing a common information space, which enables the authors to collaborate and to cooperatively make decisions. The simplest solutions typically exploit services of a file system to implement this kind of storage. We think that special *resource repositories* fulfil the demands related to the storage mechanism to a much greater extent and also address other special requirements posed by collaborative authoring. Using a repository, it is possible to upgrade the storage mechanism with certain value-added services, which results in the overall improvement of the authoring process.

In this paper, Section 2 draws a connection between repository services and computer system support for collaborative authoring of Web content. We claim that resource repositories with sets of fixed (predefined) value-added services offer only limited functionality needed to address the requirements of collaborative authoring. Therefore, in Section 3, we propose a way of generating customized repositories to meet the demands of the authoring process in its entirety. Section 4 focuses on an example of customizing version control facilities provided by a repository to show an application of the approach described in Section 3 in practice. In Section 5, we draw certain conclusions and describe our ideas for future work related to the approach.

## 2. Repository services and collaborative authoring support

Advances in information technology may be used to implement specialized computer systems supporting the process of collaborative authoring for the Web. Such systems may involve:

- a set of authoring tools used to produce the content in the authors' own workspaces; because of the variety of possible content types, ranging from plain text parts of Web pages to graphics and scripts, users might find it necessary to use different authoring tools;
- a set of modeling tools used to model various aspects of the Web content to be collaboratively authored; models may involve static parts of the content, such as text or graphics, but may also be used to specify parts of Web applications providing the content dynamically, such as scripts or servlets, for example. Therefore, models may be used to define and document semantic entities of the content, the associations between these entities as well as behavior of Web applications providing the content and hence help designers and authors understand static and dynamic aspects of Web content to be produced;
- a set of mechanisms used to transfer the content from and to the authors' workspaces as well as store and organize parts of authored content in order to be able to access it for retrieval and further modifications;
- a set of mechanisms used to support the activities that directly relate to cooperation, such as coordination of authoring tasks, communication between authors and reviews of authored content.

Tools and mechanisms mentioned are combined to form an integrated open environment supporting collaborative authoring. In such an environment, a repository may be used to support storage facilities, as well as provide support to other services related to the authoring process. Bernstein et al. [1, 2] define a repository as a *shared database of information about engineering artifacts*. A common repository allows (design) tools to share information so that they can work together.

A repository manager [1] is a database application that provides services for modeling, retrieving and managing objects in the repository. For that purpose, a repository manager has to provide standard amenities of a DBMS (data model, queries, views, integrity control, access control and transactions) as well as some value-added services [1, 2]: checkout/checkin, version control, configuration control, notification, context management, and workflow control. Types of stored and handled objects are primarily defined using a *data model* made known to the repository. Repository managers used to support the process of collaborative authoring should meet the needs of both managing *content data*, which relates directly to the content being authored as well as *process data*, which relates to the authoring process [5].

Since general principles of cooperative work of multiple geographically distributed users are also involved in computer systems supporting collaborative authoring, these systems may be defined as a specialized subset of computer supported cooperative work (CSCW) systems. Therefore, when designing a collaborative authoring environment, several important aspects of CSCW [9] have to be considered. A non-exhaustive list of these includes:

- *group awareness* meaning that an author is aware of presence and actions of other authors;
- *internal coordination* enables the actions resulting in content formation to be coordinated;
- *communication mechanisms* enable authors to share ideas and work together with a common goal of producing content;
- *concurrency control* prevents actions that may lead to inconsistent states of authored content;
- *common information space* enables sharing of data related to the content as well as data related to the authoring process.

However, groups may pose varying requirements related to these aspects of collaboration. These may affect the properties of value-added repository manager services, such as version, configuration, and concurrency control and cause them to differ according to the specific characteristics of the authoring process. For example, the way a cooperating group of authors accesses and shares the information space directly affects the form of concurrency control services provided by the repository manager. Moreover, the variety of tools used to support the process might also lead to contradictory demands [5]. Therefore, it is extremely difficult to fulfil various requirements with a set of fixed (predefined) repository manager services. To overcome this problem, we propose a way of generating customized repository managers and hence establishing tailored repository services which fulfil specific requirements posed by the authoring process. This provides groups with the possibility to customize computer support for collaborative authoring environments in various aspects of cooperation and improve the overall efficiency of the authoring process.

## 3. The SERUM approach

Our SERUM (Software Engineering Repositories using UML) project is part of the Sonderforschungsbereich (SFB) 501, which deals with the development of large systems with generic methods. SERUM focuses on generating customizable repositories using a special *set of methods, tools, techniques, and half-fabricated components* [5, 7]. In this section, we describe the process of generating a customizable repository manager in detail.

### 3.1. Defining repository manager services

First, the *repository designer* (the person specifying application-specific semantics required by SERUM) has to provide a new or modify an existing definition of repository manager services (Fig. 3, Step a). In this step, a special *resource data model* (RDM) is provided by the repository designer. The RDM is specified using the Unified Modeling Language (UML) [8] and describes types of objects to be handled by the repository manager. The UML-based definition of repository manager services is stored in a special UML-based repository.

We decided to use the UML, because it, as we think, will play an important role as a standard language used for specification, visualization and documentation of software artifacts [8]. UML provides extensive object-oriented support for detailed modeling of class structures and relationships, which proves to be a very important feature needed to support the specification of repository services. UML also provides elements for modeling dynamic and state-oriented aspects of a software system and allows the usage of the formal object constraint language (OCL) [8]. The visual presentation of UML constructs enables the repository designer to use the graphical modeling tools in the process of providing the specification of repository manager services.

### 3.2. Applying SERUM frameworks

SERUM comes with a set of “half-fabricated” components that can be extended in such a way that the resulting repository manager fulfils specific requirements. We call such a half-fabricated component *a framework*. Johnson [6] defines a framework as a reusable design of all or a part of a system that is represented by a set of

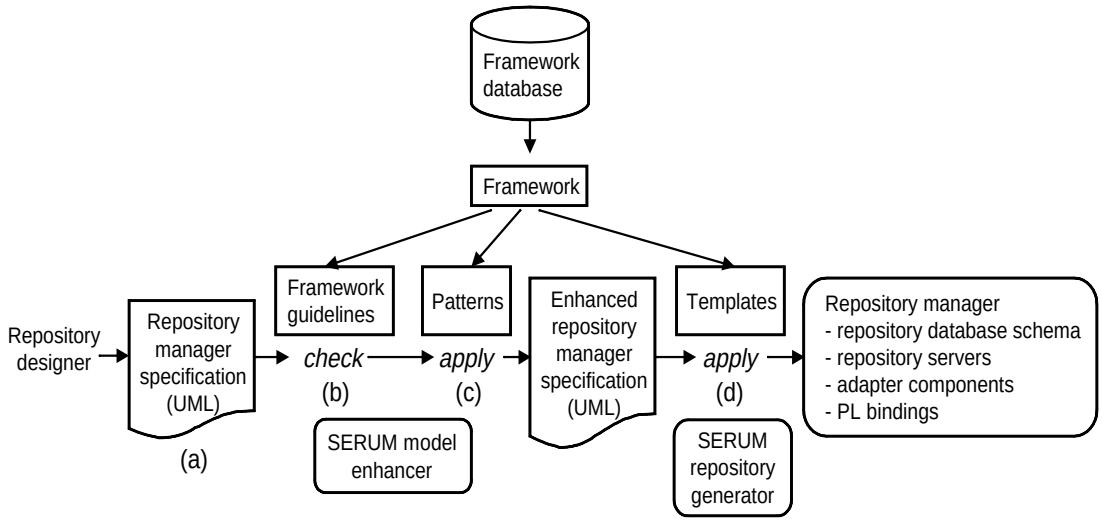


**Figure 3: Overview of the SERUM approach.**

abstract classes and the way their instances interact. SERUM provides a framework for each aspect (such as version, configuration and workflow control) in which the repository manager can be customized. Customization is achieved by *adapting* the framework properties according to specific user requirements. The application of such an adapted framework results in the generation of a customized repository manager.

A SERUM framework consists of three parts (Fig. 3): *framework guidelines*, technology-independent *design patterns*, and technology-dependent *templates.*

**SERUM framework guidelines**

Framework guidelines define a set of preconditions that have to be observed by the specification of repository manager services in order to be able to consistently apply the framework. For example, since the version control framework is not able to deal with multiple inheritance, one rule of this framework's guidelines checks for the presence of multiple inheritance in the RDM. A special SERUM tool called the *SERUM model enhancer* (SME) checks the guidelines (Fig. 3, Step b).

**SERUM patterns**

SERUM patterns enable the repository designer to influence technology-independent aspects of services of the repository manager to be generated. Gamma et al. [4] define a design pattern as descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context. A SERUM pattern is applied to the specification of repository manager services by the SME (Fig. 3, Step c) and consists of the following elements:

- *extension guidelines* and *generic parameters* are used to customize the pattern so that it addresses the customization of repository services according to the requirements;
- *constraints* are used to verify the preconditions needed for successful pattern application;
- *default structures* represent the base structures (such as abstract base classes with default attributes and behavior) used by the pattern. During pattern application the SME integrates the default structures with the existing specification of repository manager services;
- *model evolution operations* are used to integrate user specifications of repository manager services with a predefined technology-independent infrastructure of a specific framework that is specified by a pattern. The execution of model evolution operations by the SME alters the specifications and hence results in an *enhanced RDM* (ERDM).

**SERUM templates**

In contrast to SERUM patterns, SERUM templates are used to address technology-dependent aspects of repository manager services. These may include technology-specific requirements, such as demands related to the usage of programming languages and communication architecture (for example, special underlying protocols used to transfer parts of authored content from authors' workspaces to the repository). Templates provide sets of components needed to generate the new repository man-

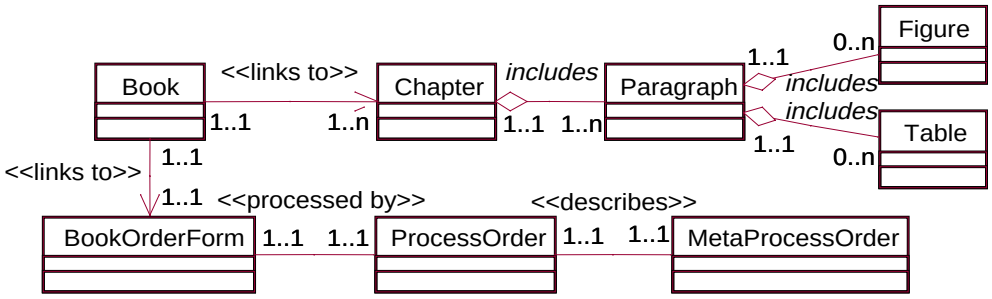


**Figure 4: A sample RDM.**

ager. Components are combined into valid configurations to prevent the usage of conflicting technologies in the generation process. Possible types of components include *code generation rules*, *source code templates* and *ready-to-use components*. A special tool called the *SERUM repository generator* (SRG) takes the specification of repository manager services that has been extended using SERUM patterns and applies the chosen templates to generate a customized repository manager to support collaborative authoring (Fig. 3, Step d). The following components (described in detail in Section 4.3) may be produced in the generation process:

- *repository database schema* enables the storage of repository objects defined by an ERDM;
- *repository servers* enable repository access of different clients, such as browsers as well as tools used by authors in the process of collaborative authoring;
- *authoring tool adapter components* needed to bridge the gap between existing authoring tools (clients) unaware of repository functionality and the repository;
- *server adapter components* needed to bridge the gap between existing servers unaware of repository functionality and the repository;
- *programming language bindings* that enable programmatic access to repository manager functionality.

## 4. Using the SERUM versioning framework

In order to give an example of using the SERUM approach in practice, this section will describe the usage of the SERUM versioning framework that is used to customize version control facilities provided by a repository manager.

### 4.1. Providing a resource data model

The repository designer has to provide an initial UML-based specification of repository manager services. This step involves defining an RDM in the form of a UML class diagram. The RDM defines the content elements, such as Web pages, hyperlinks, and dynamic content to be collaboratively authored. Conallen [3] discusses the possibilities of using UML along with the extension mechanism involving stereotypes, tagged values, and constraints to model Web application architectures.

In Fig. 4, an intentionally very simple example of an RDM that relates to the process of collaboratively authoring an electronic book to be published on the Web, is given. Due to clarity, certain association names, role names, and additional stereotype usages have been omitted. An electronic *book* consists of *chapters* which consist of *paragraphs*. *Paragraphs* may contain *figures* as well as *tables*. By using a special web page containing an order form (*BookOrderForm*), it is possible to order a printed version of the book. The order is processed by special scripted content (*ProcessOrder*).

The RDM includes a variety of different object types included in the RDM: Beside static hypertext-based objects (such as *Paragraph*), the generated repository will also manage graphical objects (*Figure*) as well as scripted content (*ProcessOrder*). However, the repository might not only be used for storing objects directly related to the content provided, but also for so-called meta objects (*MetaProcessOrder*, for example). These contain additional information about content-related objects, such as information about the author that has stored the object in the repository and the tool that has been used to create or modify the object. Of course, the description of possible object types is not extensive. For example, actual Web applications might contain far more complex object types that prove to be essential in such applications, such as servlets that are capable of producing Web content dynamically and object types providing additional business logic exploited by the servlets.

The usage of stereotypes enables the modeling of different semantic intents for content elements presented with the same type of UML construct. Since the association connecting *BookOrderForm* with *ProcessOrder* and the association connecting *Book* and *Chapter* present two

semantically different contexts of connecting content elements, the associations are denoted using different stereotypes.

For certain types of content elements, such as static hypertext-based content and scripted content, it is possible to perform content analysis to determine the internal structure of the element. In this case, the RDM may be used not only to define the basic content elements, but also to describe their structure in form of class attributes and operations. In the example illustrated in Fig. 4, operations contained by the *ProcessOrder* scripted content may be defined. After the content is authored using an existing authoring tool, the authoring tool adapter component is applied to analyze it in order to extract (Fig. 5a) the contents of the defined operations and enable their storage in the repository. Since the repository representation of the content does not directly map to the representation used by the existing authoring tool, additional meta information related to the extraction procedure has to be produced and stored to allow later recombination of content parts (Fig. 5b).

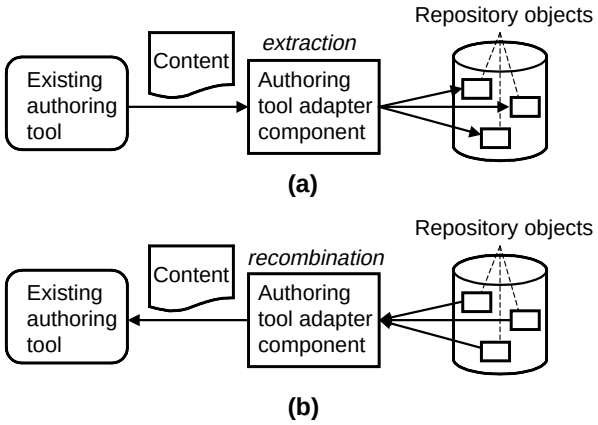


**Figure 5: (a) Extraction of repository objects and (b) content recombination.**

An important advantage of describing internal structure of content elements in the RDM is the possibility of accessing and performing modifications to the structural parts of the content using the generated programming language bindings.

The support for content analysis in the authoring tool adapted components proves to be technology-dependent. For example, various scripting languages may be used in the process of authoring scripted content. Therefore, the presence of content analysis functionality in the generated repository manager has to be supported by using specific SERUM templates addressing the technology requirements related to the functionality.

### 4.2 Applying the SERUM versioning framework

The application of the SERUM versioning framework consists of two steps:

1. Adapting the basic versioning framework
   Initially, the so-called basic versioning framework (BVF) is available to the repository designer. The BVF may be adapted by the repository designer in order to obtain an adapted versioning framework (AVF), which involves special requirements related to customized repository manager services. A detailed description of the BVF is provided in [7].
2. Applying the adapted versioning framework
   The resulting AVF is applied to the UML specification of repository manager services. As a result, a customized repository manager with the corresponding components is generated.

Version control provided by the BVF is based on the three-layer model shown in Fig. 6. The lowest layer represents collaboratively authored resource data. In the example given in Section 4.1, this involves *specific instances* (represented as ovals) of paragraphs, figures, tables, and process-order scripts, as well as *instances of connections* (represented as lines) between them. A view to this resource data, as shown by the grey excerpt in Fig. 6, is called a *version*. Types of instances are defined by the middle layer, which corresponds to the RDM provided by the repository designer. However, it is important to support not only versioning of single units of resource data, but also semantic units, which may involve several for the purpose of versioning semantically related types. Such units and the connections between them are defined using a special *version management infrastructure* (VMI) shown in the upper layer in Fig. 6 (the rectangles represent the units and the lines represent the connections between them). We call these units of versioning *versionable structures* (VS).

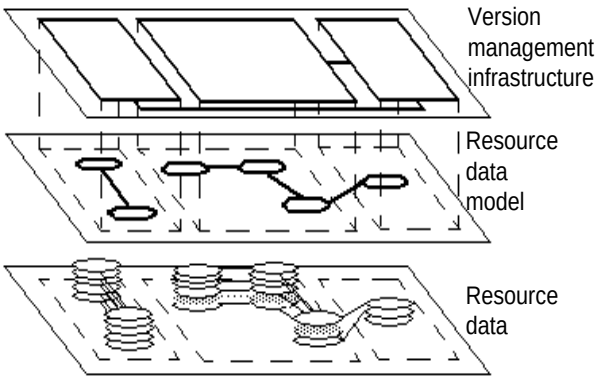


**Figure 6: Basic model of versioning.**

Note that the RDM itself is independent of various versioning aspects defined by the VMI. Therefore, in the process of defining the RDM, the versioning aspects do not have to be considered. The VMI, however, directly exploits the types defined by the RDM.

As an example of a VMI definition, we use parts of the sample RDM given in Section 4.1. Since the types *Paragraph*, *Figure* and *Table* are semantically coupled, it should not be possible to version instances of these three types individually. Therefore, we enclose the three types in a single versionable structure (*VS_Paragraph*), as illustrated in Fig. 7. *Chapters* are enclosed in their own versionable structure (*VS_Chapter*). This makes it possible to version a chapter along with its properties, such as its title and summary, independently of paragraphs the chapter contains. Since instances of chapters may contain several instances of paragraphs, this containment association should also be defined at the VMI level. We call such an association redefinition at the VMI level a *link refinement*. Instances of link refinements are used to combine instances of versionable structures into valid states, called *configurations*. For example, instances of the *vs_inclchpar* link refinement exactly specify the versions of various paragraphs (along with figures and tables) to be combined with versions of chapters.

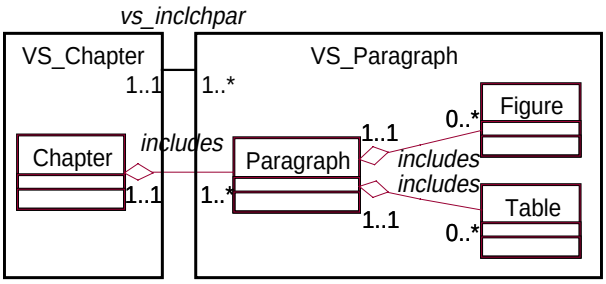


**Figure 7: A sample VMI definition.**

A special language, called the *version definition language* (VDL) is used to specify the VMI in SERUM. The VDL can be considered as a special-purpose *high-level language* (HLL) that is defined by the BVF and can later be adapted in the AVF.

The SME enhances the initial RDM using the chosen AVF and the defined VMI. The following steps are taken in this procedure:

1. *Framework guidelines* of the AVF are checked to assure the consistent AVF application.
2. *Default structures* (such as abstract base classes) of the AVF are added to the RDM.
3. VDL commands defining the VMI are applied to the RDM. In this step, SERUM *design patterns* defined by the AVF are applied to enhance the RDM.

As a result of this procedure, the initial RDM is enhanced and an ERDM is produced by the SME. Fig. 8 illustrates the most important aspects of the ERDM. Others, such as the default structures of the BVF, are omitted due to space restrictions.

## 4.3 Generating the repository manager

In this step, the repository designer chooses from a set of available templates provided by the AVF to address specific technology demands for a new repository man-

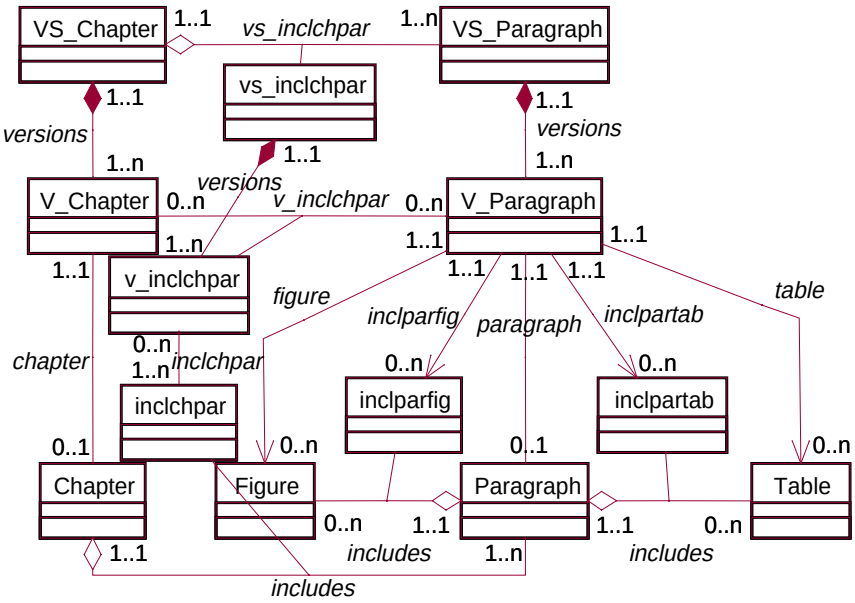


**Figure 8: Enhanced RDM.**

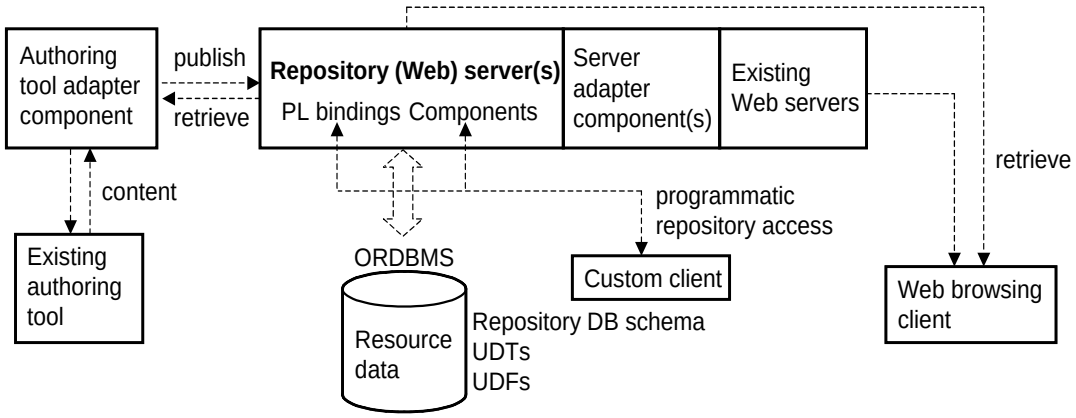


**Figure 9: Usage of the generated repository manager.**

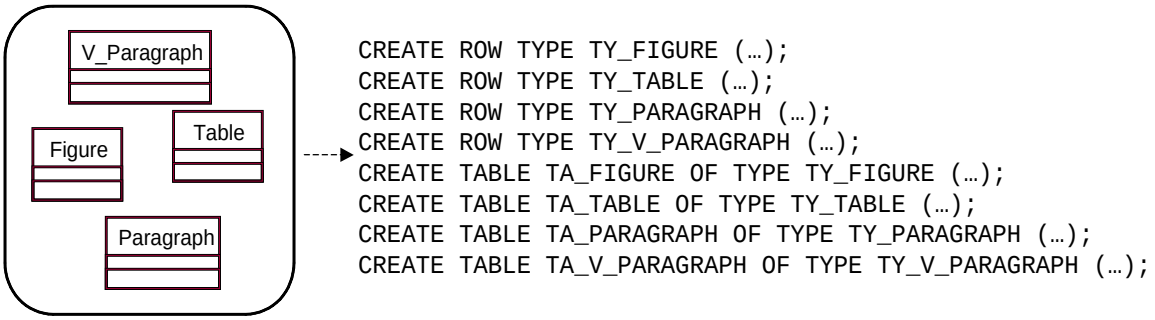


**Figure 10: ORDBMS schema generation.**

ager. Using the ERDM and the chosen templates, a new repository manager is generated. The following sections describe the results of the generation process and their possible usage (Fig. 9).

**Repository database schema**

An ORDBMS is used for the storage of repository objects. Fig. 10 illustrates an example of the generation of the repository database schema. ORDBMS features make it possible to map object types defined by the ERDM directly to user-defined types (UDT). Beside classic CRUD (Create, Read, Update, Delete) operations used for accessing and modifying the repository data, special user-defined functions (UDF) are created to provide the essential functionality related to version control. Examples of such functions include the functionality needed to lock/unlock the access to a certain version for a specific author or group of authors (perform checkout/checkin), disable further modifications of a version (freeze a version), create a successor of a specified version, find differences between versions of the same versionable structure, merge two versions of the same versionable structure, verify whether the version is a top (current) one, etc.

**Repository servers**

Repository servers provide repository manager services to various types of clients. In case of clients supporting the authoring process, the generated server is used to provide the functionality needed for content retrieval and publishing along with value-added repository manager services, such as concurrency and version control.

In case of Web browsers, the generated server is used to deliver certain repository content to the browser client. Additional aspects of value-added repository manager services might be taken into account. For example, in case version control is supported by the repository, the most current version is delivered to the browser client, which is in this case fully unaware of the repository presence.

A special type of adapter components is necessary for the case the users decide to use existing Web servers unaware of value-added repository manager services. Such components may be used to retrieve content from

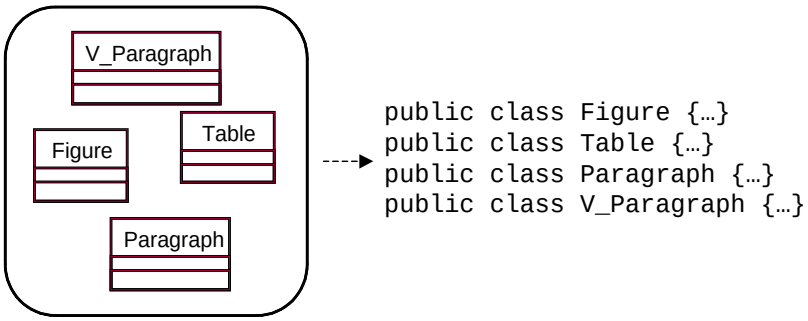


**Figure 11: API generation.**

the repository while considering aspects of version control, for example, so that the most current version is always retrieved and passed to the Web server.

**Authoring tool adapter components**

Existing tools, authors may want to use to produce different kind of content, such as plain text parts of Web pages, graphics or scripts, are unaware of value-added repository manager services. Therefore, it proves to be essential to provide special authoring tool adapter components bridging the gap between these tools and repository manager services. In this approach, authoring tools are used to create or modify parts of the content and adapter components are used to transfer the parts to or from the repository.

Since authoring tools adapter components are produced in the generation process, they recognize the types defined by the RDM and are fully aware of value-added repository manager services, such as concurrency and version control, as defined by the UML specification of these services.

In the following, we provide a short example of using adapter components in the authoring process. Assume that an author wants to modify a table as defined by the example in Section 4.1 and illustrated by the RDM in Fig. 4. Since, according to the corresponding VMI (Fig. 7), paragraphs with their figures and tables are combined in a single versionable structure, an instance of the versionable structure (a version) containing the table to be modified has to be chosen first. For this purpose, the adapter component lets the author choose from a list of versions. In case the version is frozen, further modification of this version is disabled and a successor (identical) version has to be created by the repository manager. Note that some restrictions may exist in relation to where in the version hierarchy it may actually be allowed to create a new version. These restrictions depend on the way the BVF is adapted and applied in order to customize the version control. During version modification, the access to its data from other authors or groups of authors may be restricted according to the repository manager specification of concurrency control services. A table is then retrieved from the server by the adapter component and modified in the authoring tool. If necessary, changes are also made in the paragraph containing the table. The adapter component then transfers the modified data back to the server. The ORDBMS representation of the version of the paragraph along with its figures and the modified table is updated according to the modifications made. Later, this paragraph version may be combined with a chapter version to form a valid configuration.

**Programming language bindings and component model support**

Programming language bindings provide application programming interfaces (APIs) used to enable programmatic access to repository services. The support for different programming languages corresponds to choosing templates involving the language-specific semantics. If, for example, the Java language is chosen using the templates, the SRG will provide implementations of ERDM defined types in the form of Java classes (Fig. 11).

Repository manager functionality might as well be supported using various component models, such as CORBA or COM+. If the repository manager chooses the templates providing support to the CORBA Component Model (CCM), the corresponding repository server exposes the access to the content stored in the repository as well as value-added repository manager services using CORBA components.

## 5. Conclusions

In this paper, we described the SERUM approach to generating customizable repository managers to address different demands related to the process of collaborative authoring for the Web. We use the SERUM versioning framework as an example of adapting repository man-

ager services related to version control. Based on our discussion, the following conclusions can be made:

- For each aspect of customizing repository manager services, it is possible to provide a core set of facilities that can be adapted to successfully meet specific requirements regarding repository manager services. The framework approach provides a high degree of flexibility both in aspects of reuse of existing designs and adaptation of these designs to fit such requirements.
- Technology-independent patterns enable a successful enhancement of the UML specification of these services by application-independent aspects.
- Using technology-dependent templates, it is possible to support the generation of fully operatable customized repository managers from enhanced UML specifications.

We think that the approach introduced by SERUM is very promising in the sense of managing content-related data as well as data related to the authoring process in a customized way. This introduces a significant improvement in the authoring process, since users might find it easier to focus on the core process of providing content. However, certain aspects of the area are still not explored in detail. Therefore, in our future work we intend to:

- further explore possibilities of using UML for providing specifications of repository manager services;
- explore additional possibilities of SERUM frameworks to tailor repository manager services;
- explore the relation of the SERUM approach to other emerging approaches supporting collaborative authoring, such as the WebDAV protocol [10], and existing tools that may be used to support the authoring process;
- determine how various types of tools can be used to support the application of SERUM frameworks; for example, in case of the versioning framework, graphic modeling tools may be used to support the specification of the VMI;
- evaluate the overall SERUM approach to building customizable resource repositories and estimate the benefits of this approach for the collaborative authoring process.

## 6. References


[1] P.A. Bernstein, U. Dayal: An Overview of Repository Technology, in: Proc. 20th Int. Conf. on Very Large Data Bases (VLDB‘94), J. B. Bocca et al. (Eds.), Santiago de Chile, Sept. 1994, Morgan Kaufmann, pp. 705-713.

[2] P.A. Bernstein, B. Harry, P. Sanders, D. Shutt and J. Zander: The Microsoft Repository, in: Proc. 23rd Int. Conf. on Very Large Data Bases (VLDB‘97), M. Jarke et al. (Eds.), Athens, Aug. 1997, Morgan Kaufmann, pp. 3-12.

[3] J. Conallen: Modeling Web Application Architectures with UML, in: Communications of the ACM 42:10, 1999, pp. 63-70.

[4] E. Gamma, R. Helm, R. Johnson and J. Vlissides: Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Publishing Company, 1995.

[5] T. Härder, W. Mahnke, N. Ritter and H.-P. Steiert: Generating Versioning Facilities for a Design-Data Repository Supporting Cooperative Applications, in: Int. Journal of Intelligent & Cooperative Information Systems 9:1-2, 2000, pp. 117-146.

[6] R.E. Johnson: Frameworks = Components + Patterns, in: Communications of the ACM 40:10, 1997, pp. 39-42.

[7] W. Mahnke, N. Ritter and H.-P. Steiert: Towards Generating Object-Relational Software Engineering Repositories, in: Tagungsband der GI-Fachtagung ’Datenbanksysteme in Büro, Technik und Wissenschaft’ (BTW’99), A.P. Buchmann (Ed.), Informatik aktuell, Freiburg, March 1999, Springer-Verlag, pp. 251-270.

[8] OMG, Unified Modeling Language Specification, version 1.3, OMG Document ad/00-03-01, March 2000.

[9] N. Ritter: DB-gestützte Kooperationsdienste für technische Entwurfsanwendungen, Infix, 1997.

[10] E.J. Whitehead and Jr., Y.Y. Goland: WebDAV: A network protocol for remote collaborative authoring on the Web, in: Proc. of the Sixth European Conf. on Computer Supported Cooperative Work (ESCSW´99), Bødker, S. et al. (Eds.), Copenhagen, Sept. 1999, Kluver Academic Publishers, pp. 291-310.