Supporting Mass Customization by Generating Adjusted Repositories for Product Configuration Knowledge

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

Abstract

The process of defining a customized product is usually supported by product configurator systems. These systems use product configuration models that formally express the knowledge related to the product's configuration possibilities. In computer-based environments supporting cooperation in the activities related to defining configuration models, a repository may be used to implement information-sharing facilities. However, cooperating engineers as well as tools and methodologies used in the environment may pose diverse requirements related to the usage of repository services. In this paper, we describe our approach to defining the desired form of the services using a UML-based specification and hence generating repository managers with adjusted services on the basis of this specification.

Keywords: Product Configuration, Repositories, Cooperation, Version Management.

1 Introduction

Over the last couple of years, *mass customization* [11] has become an important trend forcing the companies to transform their business processes. The previously established trend of *mass production* allowed the companies to market a limited set of predefined products. When buying such a product, the customer had to adapt his individual needs to the product's features as defined by a company. In contrast to this, mass customization allows the customers to individually customize the product to be purchased with respect to their own needs. For example, a car manufacturer having introduced mass customization to its business processes offers its customers the possibility to individually define a car to be purchased on the basis of a set of available engine types, transmission mechanism types, security device types, sunroofs of adjustable dimensions, seat types with a set of different seating surfaces materials in various colors, and others. Similarly, a mobile telephone manufacturer aware of mass customization potential allows the customers to define their own shape of the telephone and the materials to be used for its production, adjust the shape and size of the keys, select the color of display illumination and choose optional telephone functionality, such as voice dialing.

In order for the customers to be able to define and buy individually configured products, large knowledge bases containing the information related to the product's configuration potential are needed. In Section 2 of this paper, we claim that special computer-based environments can be used to enable the engineers to *cooperatively* define this kind of information. We further focus on *repositories* as an implementation strategy for information-sharing facilities that enable the cooperating engineers to efficiently manage and share the information related to the configuration possibilities of a product. We claim that the tools, methodologies, and human factors present in the cooperation environments may pose extremely varying demands related to the usage of a repository in such environments. Section 3 of the paper introduces our approach to meeting such varying demands by formally *specifying* the desired form of the functionality to be offered by the repository. A repository meeting the demands present in the environment is then *generated* on the basis of the specification. To illustrate an application of the approach in practice, Section 4 illustrates an example of adjusting the repository's version and configuration control services. In Section 5, we draw certain conclusions and describe the ideas for the future work related to the approach.

2 Defining Product Configuration Knowledge

The customer's requirements that relate to the purchase of a customized product can be met by applying the process of *product configuration* (Fig. 1). In this process, the knowledge related to the product's configuration potential is used. The knowledge can be expressed in the form of generic product structures, also commonly called *configuration models* [8], which are used to describe a specific *product family*. A product family is a set of possible *product individuals* that allow to be instantiated generically on the basis of a given configuration model. According to Tiihonen et al. [14], configuration models involve a set of predefined product components, rules related to correctly combining these components, and rules related to meeting the customer's requirements. *Product configurators* [13] are knowledge-based systems (KBS) used to support the configuration process. They use configuration models to derive a configuration of a product individual that conforms to the generic product family description provided by the model and at the same time fulfills the requirements posed by the customer.

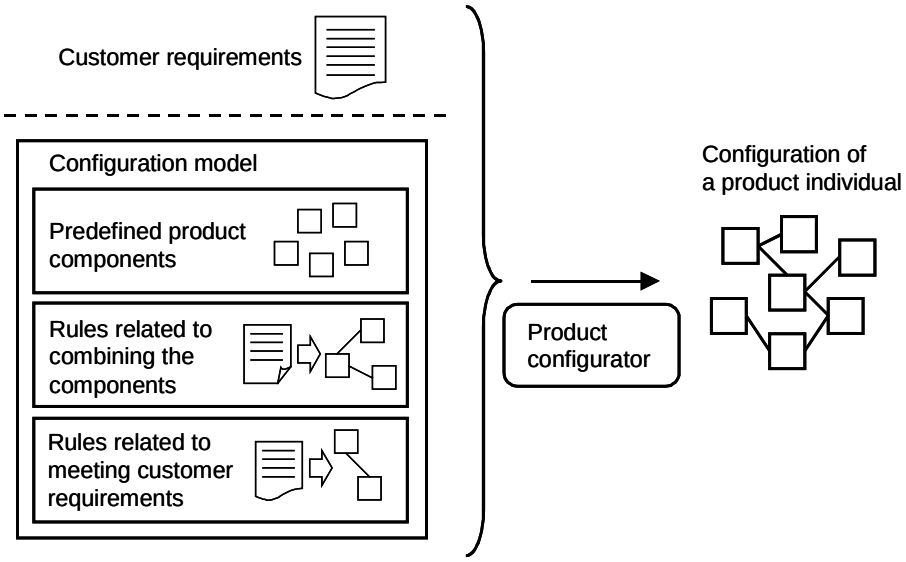


Fig. 1: Product configuration process.

In order to be able to support the configuration process, product configuration aspects have to be considered and defined in the course of product development. For example, when developing a specific product component, the engineers should consider the following:

- May the physical properties or the functionality of product parts represented by a component be adjusted in any way? In case this is true, what are the parameters supporting the adjustment? What is the range of possible values for the parameters? For example, it may be possible to adjust the dimensions of the sunroof, but only within certain limits.
- Is the presence of a component in the final configuration of the product individual mandatory or optional? For example, it is mandatory to include an engine in every configuration of a car. However, it is optional to include a security device.
- How does the presence of a component and values of its parameters affect the presence of other components and their parameter values? For example, choosing a certain car engine may require the choice of an automatic transmission mechanism.

2.1 Introducing Cooperation in the Knowledge Definition Process

The definition of configuration knowledge for a configurable product is a complex process. The knowledge is provided by product engineers familiar with the possibilities of configuring the product. The complexity of the process and the time required to accomplish it increase with respect to the product's configuration potential. A possible way to reduce the complexity and decrease the amount of time

required is the introduction of cooperation among product engineers taking part in the process. Cooperation allows the engineers to work together by sharing information related to the process and communicating the knowledge required to produce the relevant results.

Advances in information technology support computer-based formation of large-scale open cooperative environments used for the purpose of defining product configuration knowledge. Computer systems facilitating the formation of such environments specialize the general ideas and concepts of cooperation present in computer supported cooperative work (CSCW) [3, 12]. Therefore, such systems may be identified as a specialized subset of CSCW systems.

In the course of cooperatively carrying out the process of defining configuration knowledge, cooperating engineers have to be offered the possibility of sharing data directly related to defining product configuration options as well as additional data used to describe the process itself. For this purpose, special information-sharing facilities prove to be central to successful implementations of computer-based cooperative environments. By enabling the storage, retrieval and modification of shared configuration knowledge in a coordinated way, information-sharing facilities support collaboration and cooperative decision making within the process.

2.2 Using a Repository

As a supporting technology for the implementation of information-sharing facilities, the deployment of a repository offers significant advantages over other solutions that may be based purely on the usage of a database management system (DBMS) or a file system. Bernstein and Dayal [1] define a *repository* as a shared database of information on engineering artifacts. A common repository allows the tools integrated by a cooperative work environment to share information so that the individuals using the tools can work together. A *repository manager* [1] is a database application that provides services for modeling, retrieving and managing objects in a repository. It incorporates standard amenities of a DBMS (a data model, queries, views, integrity control, access control and transactions) as well as additional value-added services: *checkout/checkin*, *version control*, *configuration control*, *notification*, *context management* and *workflow control*.

Repository manager services introduce significant advantages in the process of cooperatively defining product configuration knowledge. Checkout/checkin services allow the cooperating product engineers to transfer a set of objects related to product configuration knowledge from the repository to their own workspace, perform the desired modifications and then transfer the modified objects back to the repository. Inconsistencies that may occur because of simultaneous changes committed to the same repository objects are prevented by a special concurrency control protocol. Version control services support the management of semantically meaningful snapshots of repository objects, called versions [6]. Configuration control services allow the creation of versions of composite objects by combining selected versions of their components. Notification services support the execution of specific actions related to repository objects on the basis of events occurred and conditions satisfied [1]. Context management services facilitate the definition of views to repository objects that may be used in different tasks of the engineers' work [1]. Workflow control services allow the engineers to track a repository object's state on the basis of a given workflow control model [1].

The requirements related to the usage of repository manager services in the process of cooperatively defining product configuration knowledge may differ according to tools and methodologies used in the process. Moreover, personal decisions of product engineers may influence the desired form of activities in the cooperation environment and thereby also the related form of repository manager services. For example, cooperating individuals may decide to prevent disjoint versioning of semantically complementary segments of product configuration knowledge and hence force simultaneous versioning of such segments. This decision directly affects the requirements related to version control services of the repository manager used. In the same fashion, a decision to use a specific concurrency control protocol preventing the actions that may lead to inconsistent states of stored data directly affects the repository manager's checkout/checkin services. Since repository managers used to implement information-sharing facilities usually provide a set of predefined services, it often proves impossible to entirely meet the diverse requirements posed by cooperating engineers as well as tools and methodologies used in a cooperative environment. Therefore, these requirements usually have to be adjusted so that the predefined form of the services can be used, which often degrades the configuration knowledge definition process. The advantages of cooperation might be exploited to a much greater extent if we could instead adjust the

repository manager services so that the requirements posed by cooperating engineers as well as tools and methodologies used would entirely be met.

We think that the diverse requirements related to the usage of repository manager services in the environments supporting the process of cooperatively defining product configuration knowledge can be met by generating repository managers with adjusted services. This approach is thoroughly investigated in our SERUM (Software Engineering Repositories using UML) [4, 7] project presented in this paper. The approach eliminates the need for using repository managers with predefined services. Instead, it proposes a way for cooperating engineers to formally specify the desired form of the services to be used in the cooperation process and afterwards automatically generate a repository manager with adjusted services on the basis of this specification provided. Since its services used by cooperating engineers are adjusted, the generated repository manager significantly improves the overall efficiency of the process of cooperatively defining product configuration knowledge.

3 The SERUM Approach

The approach investigated in our SERUM project aims at the generation of repository managers with adjusted services. The project is part of a long-term research effort[1] dealing with the development of large systems using generic methods. In this section, we provide an overview of the SERUM approach (Fig. 2).

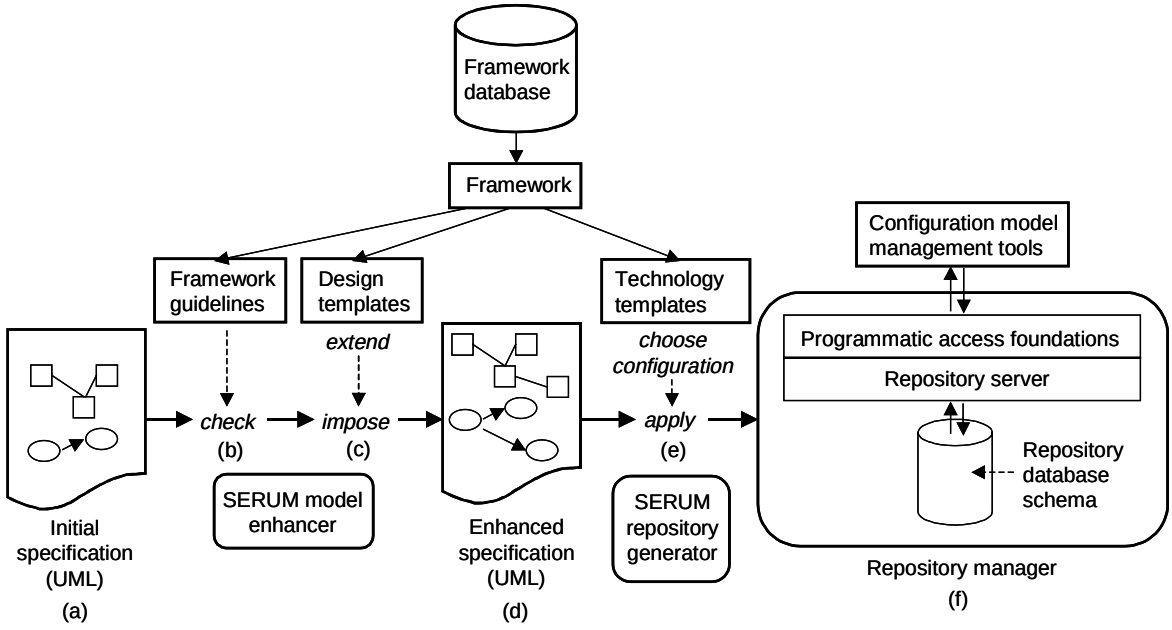


Fig. 2: Overview of the SERUM approach.

In order to apply the SERUM approach to generate a repository manager with adjusted services, a formal specification of the services is required. The specification involves various concepts that declare the adjusted form of repository manager services as desired by the cooperating engineers, and is supplied using the Unified Modeling Language (UML) [10]. 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 the structural parts of different kinds of software and non-software systems, as well as for business modeling [10]. UML offers an extensive support for object orientation which proves useful when specifying the semantics of the adjusted services of the repository manager to be generated. Using the UML, dynamic and state-oriented aspects of repository manager services may be formally specified. The visual rendering of model elements allows the specification of repository manager services to be graphically represented. In addition, UML comes with a formal Object Constraint Language (OCL) [10] used for specifying the constraints that may exist for a given model.

[1] Sonderforschungsbereich (SFB) 501, funded by the Deutsche Forschungsgemeinschaft (DFG).

3.1 Initial Specification of Repository Manager Services

Configuration knowledge is expressed through a set of related objects describing configuration possibilities of a product. In order for the generated repository manager to be able to manage these objects, a definition of object types and possible relationships among them is required. This definition forms a special *constructs model* since the object types and their relationships define the constructs that may be used to express configuration knowledge for a specific product family. The constructs model is specified in the form of a UML class diagram. By defining the model, cooperating engineers capture a combination of concepts that most efficiently meets the requirements of a specific product configuration domain. The model represents an initial part of the specification of repository manager services (Fig. 2a).

3.2 Applying SERUM Frameworks

The SERUM approach to generating repository managers with adjusted services is based on a set of half-fabricated components called *frameworks*. Johnson [5] defines a framework as a reusable design of all or a part of a system that is represented by a set of abstract classes and the way their instances interact. A framework is supplied for each aspect in which the services of a repository manager may be adjusted, such as version control and configuration control, for example. The purpose of the framework is to support the definition and integration of the demands related to the adjusted services in the UML-based specification of the services and, at the same time, provide the technology foundations needed to generate a repository manager on the basis of this specification. A SERUM framework consists of framework guidelines, design templates and technology templates (Fig. 2).

SERUM Framework Guidelines

SERUM framework guidelines incorporate a set of preconditions that have to be fulfilled by the initial specification of repository manager services in order to be able to consistently apply the framework. A special SERUM tool, called the SERUM model enhancer (SME), verifies the conformance of the initial specification to these preconditions (Fig. 2b). For example, since the framework supporting the adjustment of the repository manager's version and configuration control services is not able to deal with the concept of multiple inheritance, a specific rule of this framework's guidelines checks for the presence of this concept in the initial specification.

SERUM Design Templates

SERUM design templates are used to semi-automatically refine the UML-based specification of repository manager services so that the refined specification involves the requirements posed by cooperating engineers. To support the refinement process, design templates offer extensible technology-independent definitions of the concepts present in the services of the repository manager. The templates may be extended using their set of generic parameters and extension guidelines. By extending the template the cooperating engineers adjust the definition of the concept defined by the template and thereby influence the form of the repository manager service the concept relates to. In order to assure the presence of the adjusted concept in the services of the generated repository manager, the extended design template has to be integrated with the existing UML-based specification of repository manager services. The SME performs the integration step by imposing the extended design templates on the selected specification parts (Fig. 2c). As a result of this step, the specification of repository manager services is enhanced (Fig. 2d).

SERUM Technology Templates

A repository manager with adjusted services may be generated on the basis of different technology foundations (e.g. relational or object-relational data models, different object-oriented programming languages, etc.). For example, various DBMS products may be used to implement the underlying storage mechanism for repository objects. The services of the repository manager may be exposed using different programmatic access foundations. These can be based on the application programming interfaces (APIs)

generated for a specific programming language, such as Java or C++, or software components conforming to a specific component model.

SERUM technology templates support the presence of different technologies in the generated repository manager and enable the semi-automatic generation of repository managers with adjusted services. For example, a specific technology template may support the generation of a Java API used to programmatically access the repository manager services. Technology templates may consist of code generation rules, source code templates, and configurable ready-to-use software components used in the generation process. The usage of conflicting technologies in the generated repository manager is prevented by combining the templates into valid configurations. A special SERUM tool, called the SERUM repository generator (SRG) performs the generation process (Fig. 2e) by applying the chosen template configuration to the enhanced specification of repository manager services. Additional application-related functions of the repository manager services that cannot be generated automatically can be delivered by the cooperating engineers. Fig. 2f illustrates the main components produced in the generation process. A repository database schema enables the storage of repository objects using the underlying DBMS. The repository server enables distributed access to the services of the repository manager exposed to cooperating individuals using the programmatic access foundations.

4 Applying the SERUM Version and Configuration Control Framework

Katz [6] defines a *version* as a semantically meaningful snapshot of an object at a point in time. *Configuration* is defined as a binding between a version of a composite object and a version of each of its components [6]. Since the operations of version control and configuration control services prove to be complementary, SERUM unifies the possibilities of their adjustment in a single framework. The aim of this section is to give an example of using this framework.

As an initial part of the specification of repository manager services, a constructs model is required. Fig. 3 illustrates an intentionally very simple example of a constructs model that will be used throughout our discussion.

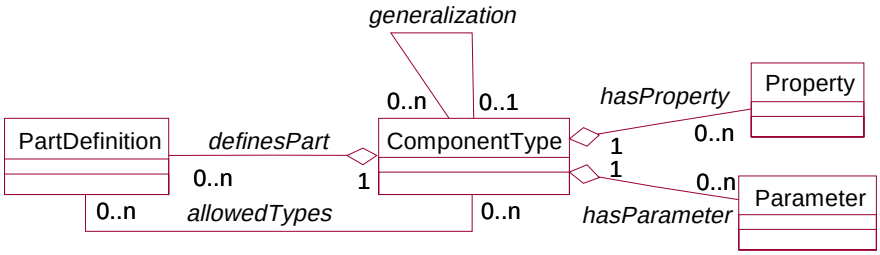


Fig. 3: A sample constructs model.

The illustrated model combines the following concepts chosen by cooperating engineers to express product configuration knowledge:

- A *component type* represents a distinguishable entity in a product [14]. An *engine* is an example of a component type present in configuration knowledge used for *car* products.
- Component types may be organized in a *classification hierarchy* allowing the specialization of component subtypes from their supertypes. For example, a *1.4L gasoline engine* component type specializes the *gasoline engine* component type. The concept is supported by the introduction of a *IsA relationship* between the component types, which denotes the specialization.
- Component types may be described by *properties*. Properties may define the units used to express their values. For example, a component type *1.4L gasoline engine* defines a property *displacement* having the value of *1390 cubic centimeters*. Component subtypes in the classification hierarchy inherit the properties of their supertypes.
- Component types may define a set of *parameters* used to adjust the characteristics of an instance of the component type. A parameter has to define a range of valid values as well as units used to express the values. For example, a component type *sunroof* defines a parameter *length* that can take values between *40 and 50 centimeters*. As with properties, component subtypes in the classification hierarchy inherit the parameters of their supertypes.

- A structured component type may involve a number of different parts, which are also represented by selected component types. A component type expresses the roles the parts may take in its structure by specifying *part definitions* [14]. A part definition includes a part name, a set of component types allowed for the part and the corresponding cardinality related to the number of component instances that may occur as parts in the structure. For example, a component type *car* specifies part roles for component types *engine* and *transmission mechanism*.

If necessary, various types of documents used to describe an instance of a component type, such as specification documents, design plans and testing results can be defined and associated with a component type. This approach extends configuration knowledge with a detailed overview of how a specific component has been developed and tested.

Additional concepts used to express limitations related to the formation of a *correct product individual* are often present in configuration knowledge. These concepts can involve the usage of *constraints*, such as compatibility or incompatibility constraints [8], or *rules* defined in a formal rule language. A product configurator has to be capable of properly interpreting the limitations expressed by the concepts while searching for a correct product individual that meets the requirements of the customer. Due to clarity, these concepts have not been included in the constructs model illustrated in Fig. 3.

4.1 Defining Versionable Units

In the course of cooperatively defining configuration knowledge, multiple versions of objects representing the knowledge may evolve. For the context of versioning, certain concepts defined by the constructs model prove to be semantically complementary. For example, cooperating engineers may not find it essential to version properties and parameters of a component type separately and then attach the versions to an appropriate version of a component type to form a valid configuration. Instead, they want to version a component type object simultaneously with its properties and parameters.

The SERUM version and configuration control framework introduces the concept of a *versionable unit* to define a set of complementary concepts in the context of versioning. Associated objects instantiated on the basis of concepts enclosed in the same versionable unit are versioned simultaneously. Versionable units are defined using a special *version definition language* (VDL) which is included in the framework. The set of defined versionable units establishes the *version management infrastructure* (VMI). Fig. 4 illustrates an example of imposing versionable units on the sample model. The versionable unit *V_PartDefinition* allows versioning of part definition objects. The versionable unit *V_ComponentType* allows versioning of component types together with their attributes and parameters. Versions formed on the basis of the versionable unit definitions may be combined to form configurations. There are two aspects of configuration involved in the VMI defined by the versionable units in Fig. 4. Configurations may be formed:

- by combining versions of part definitions with appropriate versions of component types and
- by combining versions of component subtypes with appropriate versions of component supertypes.

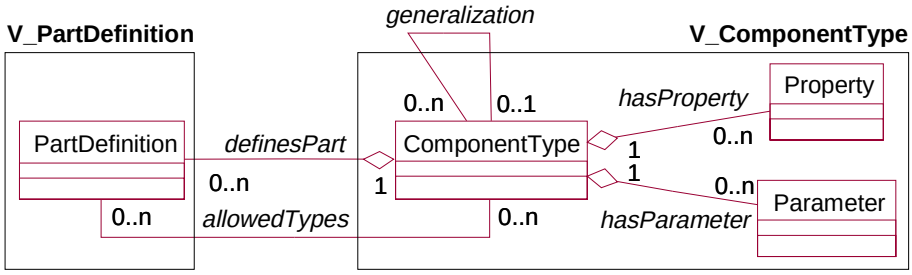


Fig. 4: Defining versionable units.

In order to include the definitions expressed by the VMI in the UML-based specification of repository manager services, the VMI has to be integrated with the constructs model. The integration step is performed by the SME, which executes the VDL statements defining the versionable units. During this step, the SME imposes the relevant SERUM design templates defining core version and configuration control concepts on the relevant parts of the model. As a result, an enhanced constructs model illustrated

in Fig. 5 is produced. The types and associations generated in the process of design pattern imposition are used to support the operations related to version and configuration control services of the repository manager. Certain extensions to the model such as additional abstract classes as well as class attribute and operation definitions have been omitted from Fig. 5 to improve clarity of the illustrated model.

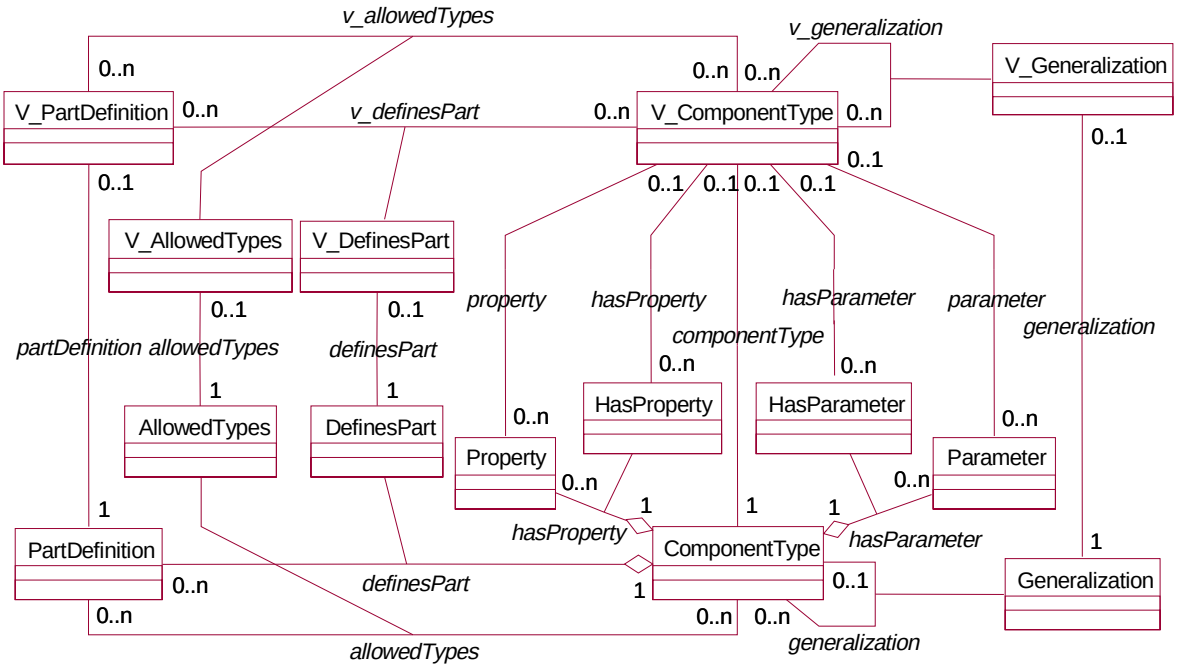


Fig. 5: Enhanced constructs model.

4.2 Generating a Repository Manager with Adjusted Services

The generation process is supported by SERUM technology templates which enable the presence of different technologies in the generated repository manager. In the generation process, the components illustrated in Fig. 6 are produced by the SRG on the basis of a UML-based enhanced specification of repository manager services and a selected configuration of technology templates.

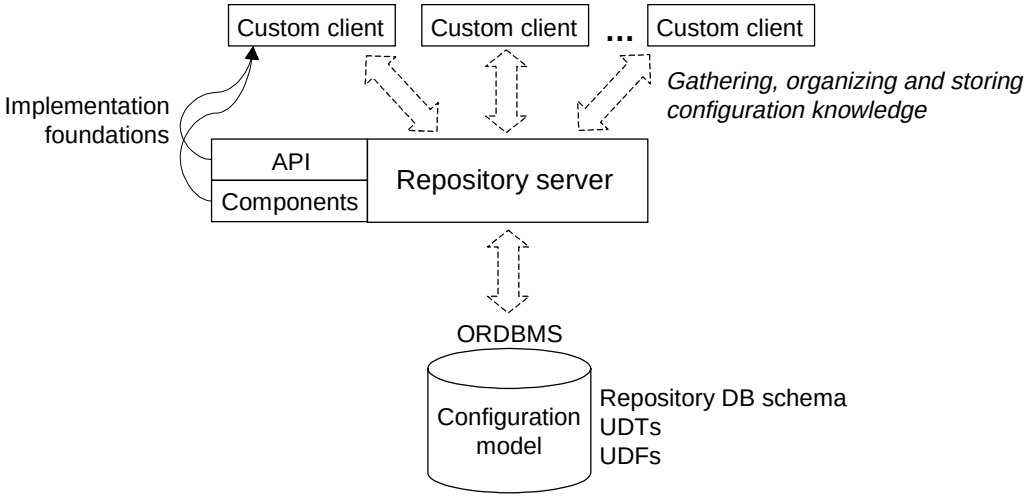
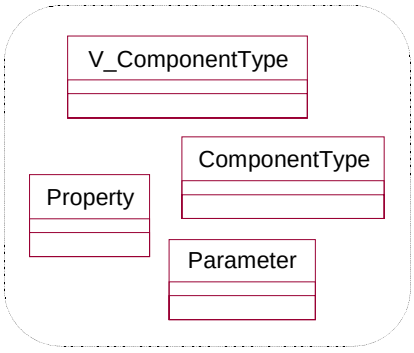


Fig. 6: Usage of the generated components.

Repository Database Schema

SERUM encourages the usage of object-relational database management systems (ORDBMS) to support the storage of repository objects. The standard DBMS amenities such as data storage and recovery capabilities are offered by the ORDBMS used. ORDBMS features enable a mapping of object types defined by the enhanced constructs model to user-defined types (UDTs). Database tables used to store repository objects are created on the basis of UDT definitions. Object-relational features of the DBMS allow the mapping of important concepts of object models, such as class hierarchy, to the repository schema. Fig. 7 illustrates a part of the generated repository schema definition. The usage of an ORDBMS also enables the definition of user-defined functions (UDFs), which can be used to implement the basic CRUD (Create, Read, Update, Delete) operations used for manipulation of repository objects as well as additional operations related to the usage of adjusted repository manager services.

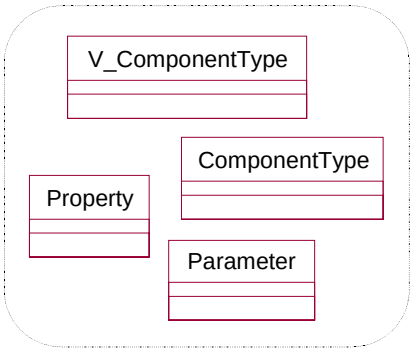



```
CREATE TYPE V_COMPONENTTYPE_TY (…);
CREATE TYPE COMPONENTTYPE_TY (…);
CREATE TYPE PROPERTY_TY (…);
CREATE TYPE PARAMETER_TY (…);
CREATE TABLE V_COMPONENTTYPE_TA
     OF TYPE V_COMPONENTTYPE_TY (…);
CREATE TABLE COMPONENTTYPE_TA
     OF TYPE COMPONENTTYPE_TY (…);
CREATE TABLE PROPERTY_TA
     OF TYPE PROPERTY_TY (…);
CREATE TABLE PARAMETER_TA
     OF TYPE PARAMETER_TY (…);
```

Fig. 7: ORDBMS schema generation.

Repository Server and Programmatic Access Foundations

A generated repository server allows access to the operations related to repository manager services to different kinds of clients used by the engineers in the process of cooperatively defining the configuration knowledge. Programmatic access foundations are used to expose the operations in the form of an API for a specific programming language. In case where the application-specific operations cannot be generated automatically, cooperating engineers have to deliver the required code segments. The foundations may as well involve the possibility of accessing the operations using software components that conform to a specific component model, such as CORBA [9] or COM [2]. Clients used by cooperating engineers to gather, organize, and transfer configuration knowledge to the repository are implemented on the basis of these foundations. The foundations can be used as an implementation framework for product configurator




```
public class V_ComponentType {…}
public class ComponentType {…}
public class Property {…}
public class Parameter {…}
```

Fig. 8: Java API generation.

systems that use the knowledge stored in the repository when searching for a product configuration that meets the customer requirements. The technologies present in the foundations are determined by the choice of SERUM technology templates used in the generation process. Fig. 8 illustrates a part of the generated API in the Java programming language.

5 Conclusion

This paper described our SERUM approach to generating repository managers with adjusted services. The repository managers may be used in the process of cooperatively defining product configuration knowledge to implement information-sharing facilities required by computer-based cooperative environments. On the basis of presented results, it is possible to make the following conclusions:

- SERUM design templates can be used to provide technology-independent definitions of concepts present in the services of a repository manager. By extending the design templates it is possible to express the adjusted form of services that meets the requirements of cooperating engineers.
- SERUM technology templates can be used to support the presence of different technologies in the generated repository manager.
- Since the services of a repository manager are adjusted to the requirements posed by cooperating engineers as well as tools and methodologies used in the process, they contribute to the efficiency of the configuration knowledge definition process.

However, certain aspects of using repository managers with adjusted services are still not explored in detail. Therefore, in our future work related to the approach, we intend to:

- further explore the possibilities of using the UML for the purpose of providing a detailed specification of adjusted services of repository managers,
- develop graphical tools supporting the usage of SERUM frameworks in order to make the process of adjusting the services easier for cooperating engineers,
- evaluate the overall SERUM approach on a large set of different cooperative environments supporting the definition of configuration knowledge for products present in various market domains.

References

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

[2] Box, D.: Essential COM, Addison-Wesley, 1998.

[3] Ellis, C.A., Gibbs, S.A., Rein, G.L.: Groupware: Some Issues and Experiences, in: Communications of the ACM 34:1, 1991, pp. 38-58.

[4] Härder, T., Mahnke, W., Ritter, N., Steiert, H.-P.: 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.

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

[6] Katz, R.H.: Towards a Unified Framework for Version Modeling in Engineering Databases, in: ACM Computing Surveys 22:4, 1990, pp. 375-408.

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

[8] Männistö, T., Peltonen, H., Sulonen, R.: View to Product Configuration Knowledge Modelling and Evolution, in: Configuration - Papers from the 1996 AAAI Fall Symposium, AAAI Press, 1996, pp. 111-118.

[9] OMG, The Common Object Request Broker: Architecture and Specification, Revision 2.4.2, OMG Document ad/01-02-01, February 2001.

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

[11] Pine, B.J.: Mass Customization: The New Frontier in Business Competition, Harvard Business School Press, 1993.

[12] Rodden, T.: A Survey of CSCW Systems, in: Interacting with Computers, 3:3, 1991, pp. 319-354.

[13] Soininen, T., Niemelä, I., Tiihonen, J., Sulonen, R.: Representing Configuration Knowledge With Weight Constraint Rules, in: Proc. of the AAAI Spring 2001 Symposium on Answer Set Programming: Towards Efficient and Scalable Knowledge, Stanford, March 2001, pp. 195-201.

[14] Tiihonen, J., Lehtonen, T., Soininen, T., Pulkkinen, A., Sulonen, R., Riitahuhta, A.: Modeling Configurable Product Families, in: Proc. of the 4th WDK Workshop on Product Structuring, Delft, Oct. 1998.