A Model Repository Description Language - MRDL

Brahim Hamid

A Model Repository Description Language -MRDL

Brahim Hamid

IRIT, University of Toulosue
118 Route de Narbonne, 31062 Toulouse Cedex 9, France
hamid@irit.fr

Abstract. Repository-based development of software systems has gained more attention recently by addressing new challenges such as security and dependability. However, there are still gaps in existing modeling languages and/or formalisms dedicated to define model repositories and the way how to reuse them in the automation of software development. Thus, there is a strong requirement for defining a model repository description language not only as a modeling approach, but also as a suitable instrument to support system and software engineers in the activity of search and retrieval of appropriate models beyond keyword-based search. Moreover, modeling approaches allow using tools for the specification and the exploitation of the designed artifacts (e.g. analysis and evaluation).
The goal of this paper is to advance the state of the art in model repository description for software and systems engineering. In particular, we have designed a flexible and extensible modeling language, by means of an OMG style metamodel, to specify model repositories for modeling artifacts, and we have defined an operational architecture for development tools. In particular, we show the feasibility of our own approach by reporting some preliminary prototype providing a model-based repository of security and dependability (S&D) pattern models.
Keywords. Model Repository, Metamodel, Model-Driven Engineering, Software System Engineering.

1 Introduction

Our society has become increasingly dependent on software-intensive systems, such as Information and Communication Technology (ICT) systems, not only in safety-critical areas (e.g., defense, transportation, nuclear power generation, space exploration), but also in areas such as finance, medical information management and systems that use web applications (e.g., cloud computing systems). ICT systems are a type of socio-technical systems, which include not only technical systems but also operational processes and the people who use and interact with those technical systems. However, the shift from traditional computer systems toward the Internet of Things, i.e., devices connected via the Internet, wireless communication or other interfaces, requires a reconsideration of software system engineering processes. As a result, new recommendations should be considered to develop novel methods capable of handling the complexity and reducing the cost of the development of these systems. We believe that the specification and packaging of software modeling artifacts can provide an efficient means of addressing these problems, improving industrial efficiency and fostering technology *reuse* across domains (the reuse of models at different levels), thus reducing the time and effort required to design a complex system [1, 2].

During system development lifecycles, modeling artifacts may be used in various forms such as domain models, design patterns, component models, code modules, test and code generators [3–5]. Repositories of modeling artifacts have recently gained increased attention as a means of encouraging reuse in software engineering. In fact, repository-centric development processes are more widely adopted in software system development than are other approaches, such as architecture-centric or pattern-centric development processes. According to Bernstein and Dayal [6], a repository is a shared database of information regarding engineered artifacts. These authors note that a repository possesses (1) a *Manager* for modeling, retrieving, and managing the objects in the repository; (2) a *Database* to store the data; and (3) *Functionalities* to enable interaction with the repository. In our work, we go one step further: we conceptualize a model-based repository to support the specifications, definitions and packaging of a set of modeling artifacts. This paper addresses the challenges of creating a flexible repository of modeling artifacts and managing the models in that repository while providing assistance with the selection of appropriate modeling artifacts in the various stages of the system engineering lifecycle. We propose an abstract syntax, by means of an OMG-style metamodel, for constructing the modeling language for model repository models, build in an incremental approach. The abstract syntax is based on the requirements for the model repository modeling language, describing various concerns, such as engineering concepts, repository interactions, modeling artifact management and reuse. To specify model repositories conforming to the proposed metamodel, we develop a concrete syntax. In addition to this task, several services dedicated to repository features are developed. The objective is to integrate multiple features through model-based repository engineering coupled with Model-Driven Engineering (MDE) technology, making it possible to leverage the reuse of model building blocks from the repository.

The remainder of this paper is organized as follows. Section 2 describes several previous approaches to repository building and to system development based on repositories. In Section 3, we describe our approach to designing a model repository for software system engineering. Then, Section 4 presents the design process of a model-based repository. Section 5 describes the architecture of the tool suite and an example of the implementation of a repository. Finally, Section 6 presents our conclusions and suggests possible directions for future work.

2 Related Work

Several methodologies use repositories for the storage of reusable artifacts. In general, these approaches rely on repositories to resolve code dependencies during the development process. The usage of such repositories (of code, code libraries, or binaries) is widespread in development and deployment processes. Code or code library repositories are used for dependency management. For example, the available repositories in the Java domain include Ivy [7] and Maven [8], and recently emerging approaches such as Gradle [9] or Bundler [10] for Ruby are prominent representatives of the usage of these repositories. One example of repositories for binaries (e.g., deployable components) is the Eclipse Platform, which uses the Equinox p2 repositories [11]. Code, code library and binary repositories are generally developed in close collaboration with the tools

through which they are accessed. Maven and other Java-based tools may use software repositories such as Apache Archiva or Sonatype Nexus, for example.

In Model-Driven Development (MDD), model repositories [12–15, 6] are used to facilitate the exchange of models through tools for managing modeling artifacts. In the field of biology, the CellML Model Repository [16] provides free access to over 330 biological models. The CellML Model Repository provides versioning capabilities and stores the version information at the model level. When a model is modified, a new version is created and added to the repository. Model repositories are often built as a layer on top of existing technologies (for instance, databases). To facilitate querying of the repository, metadata can be added to assist in the selection of the desired artifacts. Therefore, certain repositories exist that are composed solely of metadata. For instance, as presented in the ebXML standard [17] and in the ebXML Repository Reference Implementation [18], a service repository can be regarded as a metadata repository that contains metadata about location information to assist in finding a service.

In [14], the authors propose a reusable architecture decision model for establishing model and metadata repositories. Their purpose is the design of data model and metadata repositories. In addition, several helpful tools are included in the product to assist in the selection of a basic repository technology, appropriate repository metadata, and suitable modeling levels for the model information stored in the repository. In [19], the authors propose a repository implementation with support for artifact storage and management. The supported types of artifacts are metamodels, models, constraints, specifications, transformation rules, code, templates, configuration or documentation information, and their associated metadata.

Another issue of concern is the generation of graphical modeling tools, as studied in the GraMMi project [20]. In this project, the repository is based on three levels of abstraction (meta-metamodel, metamodel and model). The repository stores both metamodels (notation definitions) and models (instantiation definitions). The repository is accessed through a self-provided interface. GraMMi's kernel permits the management of persistent objects. Thus, the purpose of this kernel is to convert the objects (models) into an understandable form for the user via the graphical interface.

Gomes et al. [21] have proposed a centralized knowledge base that can be used through case-based reasoning, a paradigm for reusing past knowledge stored in the form of cases. In this context, a case is a UML diagram that is enriched with certain identifiers. WordNet is used as a common-sense ontology to provide a classification of software projects described in UML. Searches are performed based on similarity metrics, which consider the relationships between the UML elements (packages, classes, interfaces, attributes, methods and relationships) and WordNet elements. Different metrics must exist, one for each type of UML element, and the use of the ontology allows more relaxed matching to be performed. The proposed framework cannot be used with any metamodels; only UML class models are supported.

The ReMoDD (Repository for Model-Driven Development) project [15] focuses on MDD to reduce the effort involved in the development of complex software by raising the level of abstraction at which software systems are developed. This approach is based on a repository that contains artifacts that support research and education in MDD. The ReMoDD platform provides a set of tools with which to interact with the repository.

Concretely, ReMoDD artifacts include documented MDD case studies, examples of models that reflect good and bad modeling practices, modeling exercises and problems that can be used to develop classroom assignments and projects.

Moogle [22] is a model search engine that uses a UML or DSL metamodel to create indexes that facilitate the evaluation of complex queries. Its key features include the ability to search through different kinds of models, as long as their metamodels are provided. The index is built automatically, and the system attempts to present only the relevant portion of the results, for example, to remove the XML tags or other unreadable characters to improve readability. The model element types, the model attributes and the hierarchy among model elements can be used as search criteria. Models are searched by using keywords (Simple Search), by specifying the types of model elements to be returned (Advanced Search) and by using filters organized into facets (Browse). To properly use such advanced search engines, the user must have knowledge of the metamodel elements. Moogle uses the Apache SOLR ranking policy for its results. The most important information contained in the results is highlighted for clarity to the user.

ModelBus [23] provides a framework for model storage and transformation as well as a directory of model services. Similarly, the MORSE project [24] offers a Model-Aware Service Environment repository to facilitate dynamic, reflective model searches. MORSE addresses two common problems in MDD systems: traceability and collaboration. The model repository is the main component of MORSE and was designed with the intent of abstraction from specific technologies. MORSE focuses on runtime services and processes and on their integration and interaction with the repository.

The technique described in [25] is a general-purpose approach that uses graph query processing to search a repository of models represented as graphs. First, the repository models are translated into directed graphs. Then, the system receives a query that conforms to the considered DSL metamodel. To reduce the matching problem to one of graph matching, the submitted query is also transformed into a graph. Matches are calculated by finding a mapping between the query graph and the project graphs or sub-graphs, depending on the granularity. The results are ranked based on the graph edit distance metric using the A-Star algorithm. The prototype addresses the case of the domain-specific WebML language.

Most process model repositories are linked to business process management systems and business process editors. In addition to these systems, APROMORE [26] is an Advanced Process Model Repository supported by a tool infrastructure that integrates a set of features for the analysis, management and use of process models. The work presented in [27] provides a survey of business process model repositories and their related frameworks. This work addresses the management of large collections of business processes that use repository structures and provide common repository functions such as storage, search capabilities and version management. It targets the process model designer to facilitate the reuse of process model artifacts. A comparison of process model repositories is presented to highlight the degree of reusability of artifacts.

Repository-centric engineering implies multiple kinds of interactions with repositories, e.g., management, population and access, to assist in software system development through reuse. To our knowledge, there are no existing methodological tools to support the definition and description of these interactions in repository-based approaches. The

workflows, especially those for the development of systems with dependency resolutions and the deployment of components, are well documented. However, the ability to describe these approaches in MDE is lacking, and there is also a shortage of available approaches to the use of models that possess integrated model management functionalities. Our work aims to provide a new engineering approach to facilitate the reuse of these solutions (infrastructures and approaches). Regarding compliance with a specific repository, the approach described in this paper, including the description languages and the methodology, may be used to specify the management and use of several of the aforementioned types of repositories. Finally, our approach focuses not on implementation but rather on the methodology for the development and use of model repositories. We have used EMF and a repository based on the Eclipse Connected Data Objects (CDO) framework to implement the structure and interfaces of a prototype repository. However, other existing platforms for repository implementation may also be targeted using our proposed approach.

3 Conceptual model of a model-based repository

The proposed model-based repository will effectively support reuse and the integration of the processes of model specification and system development using models. The resulting modeling framework reduces the time/cost of understanding and analyzing system artifact descriptions by virtue of its abstraction mechanisms and reduces the cost of the development process by virtue of its generation mechanisms. Concretely, a repository system is a structure that stores specification languages as well as models and the relationships among them, coupled with a set of tools to manage, visualize, export and instantiate these artifacts for use in engineering processes. In this section, we introduce the requirements for a repository system and how they influence the design decisions regarding the specification language of the model-based repository.

3.1 Requirements

Here, we address the set of requirements involved in the design of a model repository description language. These requirements are primarily focused on defining the engineering process for the development of repository-based applications (the definition, integration, transformation, and validation of a modeling artifact for use in an application). Additionally, these requirements identify ways in which the engineering process may serve the engineer in the implementation of this process and, finally, identify the restrictions that must be considered in the specification of a model repository.

The following is a list of high level requirements which were established by the Semco Project [28] and the TERESA Project [29] and which also target lacks in existing model repository building processes. The following set of requirements does not represent a complete set of requirements for any arbitrary model-based repository, as such a set depends on the application domain. As we will demonstrate, these requirements are identified based on an analysis of the challenges that the repository should address and range from the top-down specification of the repository structure to its implementation and deployment with the intent of facilitating reuse in the process of model-based

system and software engineering. As we will see, the metamodel proposed in this paper tries to fulfill most, if not all, of the requirements elicited during the projects. The requirements elicited are categorized into five categories (Table 1) and are outlined in Table 2.

Category	**Abbreviation**
Repository content and organization	RC
Populating the repository	PR
Managing the repository	MR
Accessing the repository	AR
Specification of the repository structure and interfaces	SR
Implementing the repository software system	IR

Table 1. Categories of requirements for the metamodel

Category	**Requirement**	**Description**
IR	R-01. Repository construction	-Domain-independent vs. domain-specific repository construction
RC	R-02. Repository organization	-Domain-independent vs. domain-specific repository organization
SR	R-03. Artifact description	-Repository-compliant description of artifacts -Support for multiple types of artifacts
PR	R-04. Artifact publication	-Support for the artifact insertion operation - Support for the incomplete artifact description insertion operation
AR	R-05. Artifact retrieval	-Domain-independent vs. domain-specific artifact retrieval -Domain- and role-based artifact searching -Support for the artifact search function
MR	R-06. Dependencies	-Support for dependencies between artifacts -Assurance of consistency in artifact dependencies
AR	R-07. Artifact visualization	-Support for the visualization of artifacts -Support for the visualization of the internal structure of the repository (browsing) -Role-dependent visualization of the content of the repository -Adaptation of the visualization mode depending on the user role
AR	R-08. Repository interaction	-Guidelines for easy interaction with the repository -Multiview support for domain- and actor-specific knowledge -Multiview support for domain- and actor-specific tools
MR	R-09. User access control	-Mechanisms to support access control for artifacts -Mechanisms to support access control for dependencies
MR	R-10. Administration	-Support of administrative functions -Support of maintenance and reorganization of the repository content

Table 2. List of requirements - overview

3.2 System and software artifact repository conceptual model (SARM)

An analysis of the identified requirements leads to the definition of a set of concepts on which to base the modeling languages. We have identified three main concepts for this purpose, as shown in Figure 1.

(1) SARM content. The repository structure includes a storage space that is used to describe multiple storage targets, in which different types of content are stored. The repository is model-based, and its main content consists of models and their related

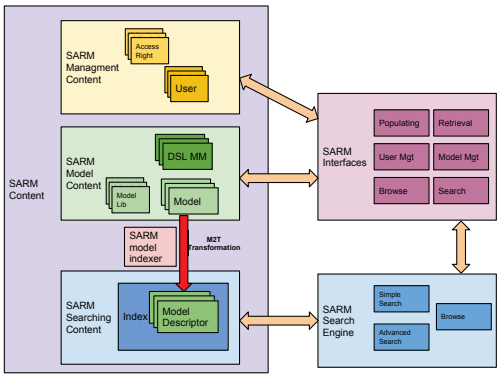

Fig. 1. SARM Architecture

libraries. The remainder of the repository content is used to store necessary elements for the functioning of the repository.

- *SARM Model Content.* To store models, the repository structure requires a dedicated storage space for model content, including models, metamodels and model libraries.
- *SARM Search Content.* To facilitate model searching, the repository must store metadata for both models and metamodels. To create such metadata, an indexer component examines models from the repository and extracts all information needed for the search.
- *SARM Management Content.* The repository must also store data regarding its users. The SARM Management Content stores user profiles and the associated access rights.

(2) SARM search engine. The repository structure includes a search engine, which is used to search the contents of the repository. The search engine uses metamodeling information and indexing mechanisms to enable the performance of more sophisticated queries. The SARM repository offers three modes of searching for models: Simple Search, Advanced Search and Browsing.

- *Simple Search.* Designed to perform general-purpose queries using keywords. This type of search is best suited to earlier stages of development, when the developer is looking for models related to domain concepts.
- *Advanced Search.* Designed to perform more complex queries. In this case, the user can specify the types of model elements to be returned.
- *Browsing.* Similar to Advanced Search but without the need to specify keywords to be searched. Instead, all elements matching the specified filters are shown. The user can select one or more filters and combine them for more precise browsing results.

(3) SARM interfaces. The repository structure includes interfaces (APIs) to enable interaction with its content. Six types of necessary interfaces can be identified.

- *Populating interface.* To populate the repository with models, SARM provides a set of interfaces (APIs) for interaction with external tools. This interface interacts with the *SARM Model Content*.
- *Retrieval interface.* SARM provides an interface to search for, select and instantiate models within a specific development environment. This interface interacts with the *SARM Model Content* and depends on the *Search interface*.
- *Model Management interface.* For management of the models, SARM provides an interaction interface that offers a set of functions for incorporating new metamodels into the repository, specifying relationships between models, and adding other information. This interface interacts with the emphSARM Model Content.
- *User Management interface.* For user management in the repository, SARM provides an interaction interface that offers a set of functions for managing user data. This interface interacts with the *SARM Management Content*.
- *Search interface.* SARM offers a search interface that uses the *SARM Search Engine* to search for models in the repository.
- *Browse interface.* SARM offers a browsing interface that uses the *SARM Browsing* functionality of the search engine to browse models.

3.3 Metamodel of the repository structure.

We propose an abstract syntax, by means of an OMG-style metamodel, for constructing the MRDL modeling language for model repositories build in an incremental approach. The abstract syntax is based on the previous requirements (Table 2), describing various concerns, such as engineering concepts, repository interactions, artifact-based engineering and reuse. We specify our model repository modeling language using the Meta Object Facility (MOF) constructs [30], a Object Management Group (OMG) standard for describing modeling languages such as the Unified Modeling Language (UML) [31], and using the open-source Eclipse Modeling Framework (EMF) [32] environment. EMF provides an implementation of EMOF (Essential MOF), a subset of MOF, called Ecore [1]. EMF offers a set of tools to specify metamodels in Ecore and to generate other representations of them. The principal classes of the metamodel of the repository are described using Ecore notation in Figure 2. Greater detail is provided below with regard to the meanings of the principal concepts used to specify the structure of the repository.

- SarmRepository. The core element used to define the repository.
- SarmModelContent. Represents the model content, including models, metamodels and model libraries.
- SarmDslMetamodel. Represents the specification languages (metamodels) for the modeling artifacts.
- SarmMmConcept. Represents a specific concept from a specific metamodel. In general, a concept represents a modeling artifact that will be stored in the repository.
- SarmAttribut. Defines a property of a specific concept.
- SarmManagementContent. Represents the management content, including user data.

[1] Ecore is a meta-metamodel

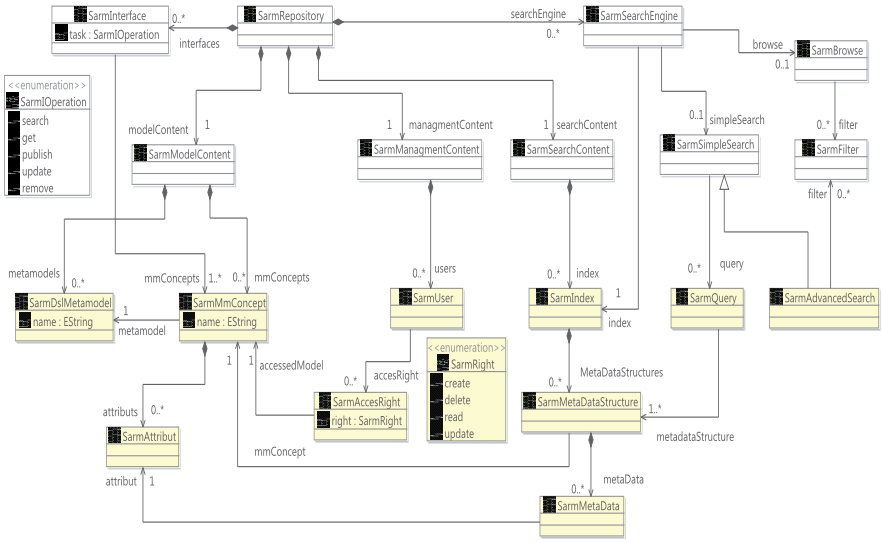

Fig. 2. SARM - Structure

- SarmUser. Used to define user profiles.
- SarmAccesRight. Used to define characteristics regarding access rights to the repository and its contents.
- SarmSearchContent. Represents search contents, including the model metadata for indexing.
- SarmIndex. A structured collection of data used to store a list of model descriptors. The index is used by SarmSearchEngine to perform model searches.
- SarmMetaDataStructure. Used to describe each stored model. This class defines all information to be searched and all metadata used for the Advanced Search functionality.
- SarmMetaData. Used to define specific metadata with regard to a model.
- SarmInterface. Provides a specification of the interfaces (APIs) for visualizing the contents of the repository and for repository management.
- SarmSearchEngine. Uses metamodeling information and indexing mechanisms to enable the performance of more sophisticated queries.

4 Definition of a repository model

The objective of this step is to specify the repository structure in accordance with the conceptual model defined in Section 3.3. To create model instances of the proposed metamodel, we choose to use a tree-structured concrete syntax provided by Eclipse Modeling Framework Technology (EMFT) [2]. It provides graphical, but not-diagrammatic notations, to specify Ecore models. It allows to represent a model as a

[2] https://eclipse.org/modeling/emft/

nested, collapsible structure with composite and leaf elements having text labels and/or symbols. A concrete syntax for the SARM language supports to instantiate the SARM metamodel to create the model of the repository comprising the creation of the meta-model compartments, the modeling artifact compartments, the user list, the interfaces, the search engines, and so on.

In addition, other features are supported, such as the specification of views of the repository according to its interfaces, its organization and the needs of the targeted system engineering processes. The structure of the repository and its interfaces can then be made available (1) to modelers for populating and managing the repository and (2) to access tools for reusing the repository's content. Thus, we provide an EMF-based tree editor for specifying repository models, as visualized in Figure 3.

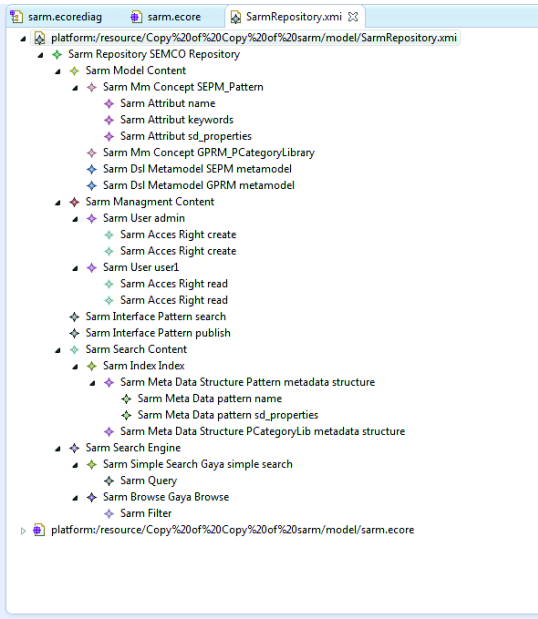

Fig. 3. Example of a repository model

In our example, we define *SemcoRepository* as an instance of SarmRepository: a model- based repository of S&D patterns and their related property models. To support these S&D pattern models and property models, we use *SEPM* and *GPRM*, as instances of SarmDslMetamodel, which uses a set of instances of SarmMmConcept to store the pattern modeling language and property modeling language concepts, respectively. The Generic PRoperty Metamodel (GPRM) [33], which is described using Ecore notation in Figure 4, is a metamodel that defines a new formalism (i.e., language) for describing property libraries, including units, types and property categories. For instance,

S&D attributes [34] such as authenticity, confidentiality and availability are defined as categories. These categories require a set of measure types (degree, metrics,...) and units (Boolean, float,...). These models are used as external model libraries to type the properties of the patterns.

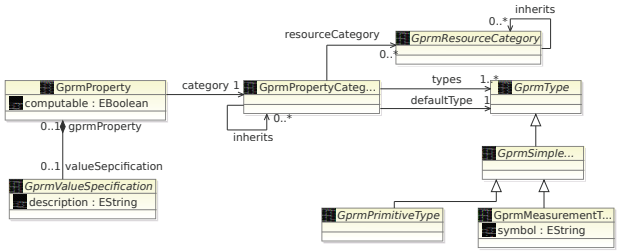

Fig. 4. The (simplified) GPRM

The System and Software Engineering Pattern Metamodel (SEPM) [35] is a metamodel that defines a new formalism for describing S&D patterns and constitutes the basis of our pattern modeling language. Here, we consider patterns to be subsystems that provide access to services (via interfaces) and manage S&D and resource properties (via features), offering a unified means of capturing meta-information related to a pattern and its context of use. Figure 5 describes the principal concepts of the SEPM using Ecore notation.

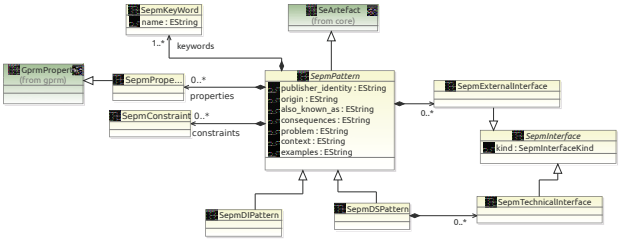

Fig. 5. The (simplified) SEPM

In addition to the repository structure, we define a model of the interfaces used to visualize and manage the contents of the repository. For instance, *Patternsearch* and *Patternpublish* are defined as instances of SarmInterface for pattern search and publication, respectively.

5 Tool support

In this section, we propose an MDE tool chain to support the proposed approach and hence assist the developers of Model-based Repository Software Systems. As we will see, the proposed tool chain is designed to support the proposed metamodels, and hence, the tool chain and the remainder of the activities involved in the approach may be developed in parallel. Appropriate tools for supporting our approach must fulfill the following key requirements:

- Enable the creation of the UML class diagrams used to describe metamodels in our approach.
- Support the model-based repository development process.
- Enable the creation of modeling artifacts and the publication of the results into the repository using the appropriate repository interface.
- Enable the creation of model libraries for artifact classification.
- Enable the creation of model libraries for the classification of the relationships between artifacts.
- Support the administration of the repository.
- Enable the creation of visualizations of the repository to facilitate its access.

To satisfy the above requirements, we define four integrated sets of software tools:

- *Tool set A* for populating the repository,
- *Tool set B* for retrieval from the repository,
- *Tool set C* to serve as the repository software and
- *Tool set D* for managing the repository.

There are several environments that can be used to build an MDE tool chain. In the context of our work, we have chosen to use EMFT to build the support tools for our approach. All metamodels used are specified using EMF. The design tools were semi-automatically generated from these metamodels. Several enhancements have been added to the generated code, such as creation wizards, to guide the modeling artifact designer in populating the repository. Visual enhancements have been added to facilitate the recognition of different concepts, as a first step toward a future visual syntax. To describe the model transformations, the QVT Operational language [3] is used. The structure of the repository is derived from the repository structure model and implemented using Java and the Eclipse CDO [4] framework.

We have successfully applied our approach in the context of the FP7 TERESA (Trusted computing Engineering for Resource constrained Embedded Systems Applications) project by applying the approach to Pattern-Based System Engineering (PBSE). Specifically, we have developed SEMCOMDT [5](SEMCO Model Development Tools) as an MDE tool chain to support all steps of our approach. As visualized in Figure 6, SEMCOMDT offers the following features:

- *Gaya* as a repository platform (structure and interfaces) that conforms to *SARM*,

[3] http://www.omg.org/spec/QVT/
[4] http://www.eclipse.org/cdo/
[5] http://www.semcomdt.org

– *Tiqueo* for specifying models of S&D properties that conform to *GPRM*,
– *Arabion* for specifying patterns that conform to *SEPM*,
– *Admin* for repository management,
– *Retrieval* for repository access.

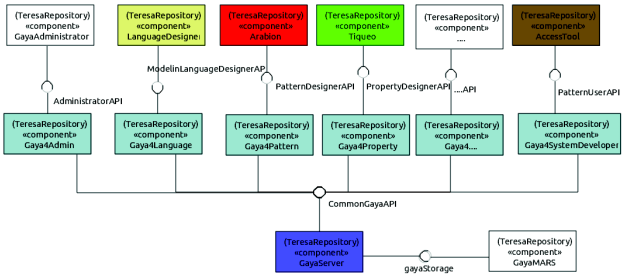

Fig. 6. An overview of the tool components

The server portion of *Gaya* consists of two components: (1) *GayaServer*, which provides the implementation of the common API, and (2) *GayaMARS*, which provides the storage mechanisms. The client portion of *Gaya* provides interfaces, such as *Gaya4Pattern* (which implements API4PatternDesigner), *Gaya4Property* (which implements API4PropDesigner), *Gaya4Admin* (which implements API4Admin) and *Gaya4SystemDeveloper* (which implements API4PatternUser).

For populating the repository, we have built two design tools: (1) the property design tool (Tiqueo), to be used by a property designer, and (2) the pattern design tool (Arabion), to be used by a pattern designer. Arabion (resp. Tiqueo) interacts with the Gaya repository for publication purposes through the *Gaya4Pattern* (resp. *Gaya4Property*) API. Furthermore, Arabion includes mechanisms for verifying the conformity of the pattern with the *SEPM* metamodel and for publishing the results to the repository.

For management of the repository by a repository manager, the *Admin* features provide a set of tools for the organization and usage and usage of the repository through the *Gaya4Admin* API. We also provide basic features such as user and artifact management. Moreover, we provide features to support the management of the relationships among artifact specifications and between artifact specifications and their complementary models.

For access to the repository by a system engineer, the Retrieval tool provides a set of functionalities to assist in the search, selection and sorting of patterns. The access tools interact with the Gaya repository through the *Gaya4SystemDeveloper* API. For instance, as shown on the right-hand side of Figure 7, the tool offers assistance in selecting appropriate patterns through key word searches and lifecycle stage searches. The results are displayed in the search result tree as System, Architecture, Design and Implementation patterns. The tool includes features for export and tailoring using dialogs, primarily based on model transformation techniques, to adapt pattern models to

the target development environment. Moreover, the tool includes dependency-checking mechanisms. For example, a pattern cannot be tailored when a property library is missing; an error message will be thrown.

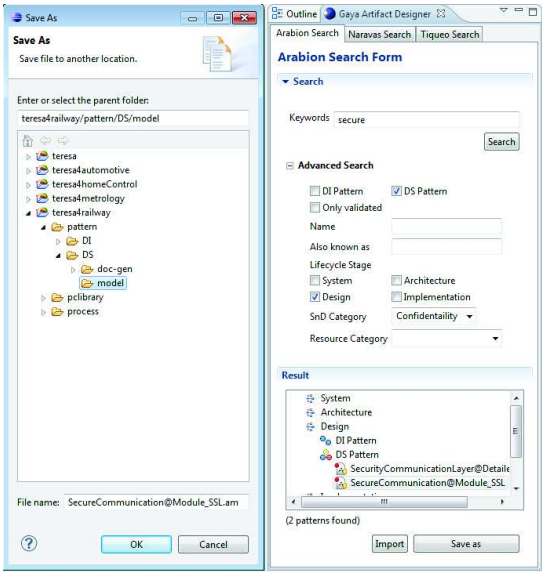

Fig. 7. Access tool/pattern tailoring

6 Conclusion and future work

The promoted model-based approach for software application development relies on a repository of models and focuses on the problem of software system engineering through a design philosophy that fosters reuse. The primary purpose of such a repository is to share expertise in a manner that interacts with existing engineering processes to facilitate the construction of applications for various domains. The proposed framework for building a repository is based on metamodeling techniques that enable the specification of the repository structure and interfaces, on content in the form of modeling artifacts, and on model transformation techniques for the purposes of generation. We begin by specifying a conceptual model of the desired model-based repository and proceed by designing modeling languages that are appropriate for the content. The results of these efforts are then used to specify and build the repository. Furthermore, we propose an operational architecture for a tool suite to support the proposed approach. We have successfully applied our approach in a case study of PBSE. Specifically, we have developed a tool suite, named SEMCOMDT , was built using EMFT and a CDO-based repository and is currently provided in the form of Eclipse plugins. In addition,

the tool suite promotes the separation of concerns during the development process by distinguishing the roles of the stakeholders. Primarily, access to the repository is customized with regard to the development phases and the stakeholders' domain and system knowledge.

In our future work, we plan to complete the case study and assess whether domain experts agree on the benefits of adopting the our specification language in a real industrial context. This requires an instantiation of the full software engineering tool and method and an evaluation across the experiences of many users across many domains. We intend to validate the feasibility and effectiveness of the proposed specification and design frameworks - how our approach may significantly reduce the cost of engineering a system compared to current practice. In addition, we will study the automation of the model search and tailoring tasks, and a framework to allow the simpler specification of constraints would be beneficial. Our vision is for modeling artifacts to be inferred from the browsing history of users and constructed from a set of already developed applications. In addition, we will study the customization of the tool suite, primarily the access tool, with regard to the development phases and the stakeholders' domain and system knowledge. We would also like to study the integration of our tools with other MDE tools. Concurrently, more sophisticated techniques to manage artifacts relationships at a metamodel level can be implemented.

References

1. C. McClure, Software Reuse Techniques: Adding Reuse to the System Development Process, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1997.
2. W. Agresti, Software Reuse: Developers' Experiences and Perceptions, Journal of Software Engineering and Applications 4 (1) (2011) 48–58.
3. C. Krueger, Software Reuse, ACM Compututing Survey 24 (2) (1992) 131–183.
4. M. Morisio, M. Ezran, C. Tully, Success and Failure Factors in Software Reuse, IEEE Transactions on Software Engineering 28 (4) (2002) 340–357.
5. W. Frakes, K. Kang, Software Reuse Research: Status and Future, IEEE Transactions on Software Engineering 31 (7) (2005) 529–536.
6. P. A. Bernstein, U. Dayal, An Overview of Repository Technology, in: Proceedings of the 20th International Conference on Very Large Data Bases, VLDB '94, Morgan Kaufmann Publishers Inc., 1994, pp. 705–713.
7. Apache Software Foundation, Ivy, http://ant.apache.org/ivy/ (2015).
8. Apache Software Foundation, Maven, https://maven.apache.org/what-is-maven.html (2015).
9. GRADLE INC., Gradle, https://gradle.org/why/robust-dependency-management/ (2015).
10. Bundler Core Team, Bundler, http://bundler.io/ (2015).
11. D. L. Berre, P. Rapicault, Dependency management for the eclipse ecosystem: eclipse p2, metadata and resolution, in: Proceedings of the 1st international workshop on Open component ecosystems, ACM, 2009, pp. 21–30.
12. P. Sriplakich, X. Blanc, M. Gervais, Supporting transparent model update in distributed CASE tool integration, in: Proceedings of the 2006 ACM symposium on Applied computing, SAC '06, ACM, New York, NY, USA, 2006, pp. 1759–1766.
13. G. Kramler, G. Kappel, T. Reiter, E. Kapsammer, W. Retschitzegger, W. Schwinger, Towards a semantic infrastructure supporting model-based tool integration, in: Proceedings of the 2006 international workshop on Global integrated model management, GaMMa '06, ACM, New York, NY, USA, 2006, pp. 43–46.

14. C. Mayr, U. Zdun, S. Dustdar, Reusable Architectural Decision Model for Model and Metadata Repositories, in: FMCO, 2008, pp. 1–20.
15. R. B. France, J. M. Bieman, B. H. C. Cheng, Repository for Model Driven Development (ReMoDD), in: MoDELS Workshops'06, 2006, pp. 311–317.
16. C. M. Lloyd, J. R. Lawson, P. J. Hunter, P. F. Nielsen, The CellML Model Repository, Bioinformatics/computer Applications in The Biosciences 24 (2008) 2122–2123.
17. ebXML: Oasis Registry Services Specification v2.5 (2003).
18. freebXML: Oasis ebxml registry reference implementation project, http://ebxmlrr.sourceforge.net/ (2007).
19. N. Milanovic, R.-D. Kutsche, T. Baum, M. Cartsburg, H. Elmasgünes, M. Pohl, J. Widiker, Model&Metamodel, Metadata and Document Repository for Software and Data Integration, in: MoDELS, 2008, pp. 416–430.
20. C. Sapia, M. Blaschka, G. Höfling, GraMMi: Using a Standard Repository Management System to Build a Generic Graphical Modeling Tool, in: Proceedings of the 33rd Hawaii International Conference on System Sciences-Volume 8 - Volume 8, HICSS '00, IEEE Computer Society, 2000, p. 8058.
21. P. Gomes, F. Pereira, P. Paiva, N. Seco, P. Carreiro, J. Ferreira, C. Bento, Using WordNet for Case-based Retrieval of UML Models, AI Commun. (2004) 13–23.
22. D. Lucrédio, F. de Mattos, P. Renata, J. Whittle, MOOGLE: A Model Search Engine, in: MoDELS, Springer, 2008, pp. 296–310.
23. C. Hein, T. Ritter, M. Wagner, Model-driven tool integration with modelbus, in: Workshop Future Trends of Model-Driven Development, 2009, pp. 50–52.
24. T. Holmes, U. Zdun, S. Dustdar, MORSE: A Model-Aware Service Environment (2009).
25. B. Bislimovska, A. Bozzon, M. Brambilla, P. Fraternali, Graph-Based Search over Web Application Model Repositories, in: ICWE, Springer, 2011, pp. 90–104.
26. M. L. Rosa, H. R. W. V. D. Aalst, R. Dijkman, J. Mendling, M. Dumas, L. GarcíA-BañUelos, APROMORE: An advanced process model repository, Expert Systems with Applications 38 (6) (2011) 7029–7040.
27. Z. Yan, R. M. Dijkman, P. Grefen, Business process model repositories - framework and survey, Information & Software Technology 54(4) (2012) 380–395.
28. B. Hamid, SEMCO Project (System and software Engineering for embedded systems applications with Multi-COncerns support), http://www.semcomdt.org.
29. TERESA, TERESA Project (Trusted Computing Engineering for Resource Constrained Embedded Systems Applications), http://www.teresa-project.org/.
30. OMG, MetaObject Facility 2.4.2, Specification, http://www.omg.org/spec/MOF/2.4.2/ (2014).
31. OMG, OMG Unified Modeling Language (OMG UML), Superstructure, http://www.omg.org/spec/UML/2.4.1 (August 2011).
32. D. Steinberg, F. Budinsky, M. Paternostro, E. Merks, EMF: Eclipse Modeling Framework 2.0, 2nd Edition, Addison-Wesley Professional, 2009.
33. A. Ziani, B. Hamid, S. Trujillo, Towards a Unified Meta-model for Resources-Constrained Embedded Systems, in: 37th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), IEEE, 2011, pp. 485–492.
34. A. Avizienis, J.-C. Laprie, B. Randell, C. Landwehr, Basic Concepts and Taxonomy of Dependable and Secure Computing, IEEE Transactions on Dependable and Secure Computing 1 (2004) 11–33.
35. B. Hamid, S.Gurgens, C. Jouvray, N. Desnos, Enforcing S&D Pattern Design in RCES with Modeling and Formal Approaches, in: J. Whittle (Ed.), ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS), Vol. 6981, Springer, 2011, pp. 319–333.