Architecture of Pattern Management Software System

Erki Eessaar

Department of Informatics, Tallinn University of Technology, Raja 15, 12618 Tallinn, Estonia
eessaar@staff.ttu.ee

Abstract. Patterns and models are artifacts that are assets to their owners. Storing them in a repository and using common interfaces for their management simplifies their usage. This article proposes architecture of an extensible repository system that is called Pattern Management Software System (PMSS). It permits metamodel engineer to define abstract syntax of a new modeling or pattern writing language using a metamodel. System registers metamodel in the object-relational database and creates database objects for recording corresponding artifacts and their metadata. Views provide to the users different ways of looking at the data in the database. New views are continuously created. How can user find appropriate views? This article proposes that one component of PMSS must be the subsystem that permits searching and execution of the views.

1 Introduction

This article deals with the management of artifacts that are created and used during the software development. These artifacts are for example models and patterns. Knowledge about the reoccurring practices can be extracted from the models and recorded as patterns. Artifacts are created using wide range of languages and tools. Artifacts should be easily findable and accessible in order to promote reuse and should therefore be collected to the repository. For example, following queries can be used in order to find data modeling pattern from the repository. These queries are used as an illustration and don't represent syntax that is used.

```
SET A=(<id>, <name> of artifacts WHERE metamodel is
<entity relationship model> AND artifact contains
element of type <entity> which name is <'Order'>);

SET B=(SET <A>) UNION (<id>, <name> of artifacts WHERE
type is <pattern> AND element of type <solution> is
associated with the artifacts IN the SET <A>));
```

Angle brackets "<" and ">" represent placeholders for the parameter values. User has to assign value to each parameter before executing the

query. Current example contains sample value of each parameter. System must store metadata about the artifacts and their language in order to support selection of some of these values from the repository.
The main contribution of this article is the proposal of the architecture of repository of artifacts that uses object-relational database management system.
General name of the proposed system is Pattern Management Software System (PMSS). Word "pattern" stresses that one purpose of PMSS is to promote creation of patterns based on the artifacts that are recorded in the repository.
The rest of the paper is organized as follows. Section 2 describes related works. Section 3 describes the architecture of PMSS. Section 4 discusses advantages and disadvantages of the proposed solution. Section 5 provides conclusions and describes future work with the topic.

2 Related Works

Next are described some projects for creating repositories of patterns. Greenfield and Short [1] argue that Domain Specific Languages (DSL) help automate code generation from the models. DSL uses domain specific vocabulary and needs repository of domain specific patterns. Example of such repository is repository of patterns for the electricity supply industry [2]. PsiGene [3] is a component-based domain specific development method that uses domain-specific patterns for generating application specific components for the simulation software. Halaris and Geroupoulus [4] describe the information structure of the reuse repository, which could be used to store and retrieve reusable objects of any kind. It could be used together with the different software engineering methodologies. Each of these reusable objects belongs to some class and is described by the set of attribute-value pairs. Objects can be searched using semantic similarity. There is also weakly structured Portland Pattern Repository (PPR) [5] that contains interconnected pages that everyone can edit. One disadvantage of PPR is that there is no possibility to restrict search to only some components of pattern.
Yacoub and Ammar [6] argue: "Patterns are mental building blocks that are more related to human understanding than to automatic usage."

Therefore tools that help to use patterns should also help user to understand consequences of the solution. For example, Mirbel [7] introduces context frame that describes context of the process fragment as a set of compatible criteria. Appropriate fragment can be selected by comparing context of the problem and contexts of the process fragments that represents possible solutions.
How to implement repository of reusable artifacts? One solution is to build these systems directly on top of a file-system. OMG introduces Reusable Asset Specification (RAS) [8] that helps to organize reusable assets that describe solution to the software development problem. Asset is implemented as a package file that contains artifact files that represent solution and XML file that manifests properties of the asset. Reusable assets can be collected to the central RAS repository.
Many repository systems use database management systems (DBMS) according to Dittrich and others [9]. Open source artifact management system OSCAR [10], that is part of the distributed software development environment GENESIS, stores metadata about the artifacts in the database. It keeps artifacts and associated informal materials in the files that are outside the scope of DBMS. System records and presents process information including the actors responsible for changes of artifacts and the rationale associated with those changes. Many repository systems store all the information in the database. Some systems use relational database [11, 12]. According to Dittrich and others [9] many of the repository systems use object-oriented DBMS. Date [13] gives an overview of problems of object-oriented databases and shows their weaknesses. Recently object-relational DBMS (ORDBMS) have attracted attention. SERUM [14] is an example of repository system that uses ORDBMS. SERUM provides framework for building customized repository managers.

3 Metadata Based Architecture of PMSS

Firstly purposes of the PMSS will be described. Secondly the architecture will be described that helps to achieve these purposes.

3.1 Purposes of PMSS

- Store permanently development results and make them available to the public.
- Provide environment where groups of users can develop language of some problem domain by recording and grouping patterns that describe this domain.
- Simplify recording of associations between artifacts in order to determine their context. For example, problem statement of pattern can be accompanied with "As Is" models [15] that highlight current practices. Description of the solution can be accompanied with "To Be" models [15].
- Give an overview of the creation and usage of artifacts and active contributors.
- Give an overview of the popularity of different artifact languages.
- Provide platform for the standardization of the pattern writing language.

3.2 The Architecture

Repository system contains repository manager according to Bernstein and Dayal [16] and repository (database). A repository manager provides services for modeling, retrieving, and managing the objects in a repository according to Bernstein and Dayal [16]. Repository manager must offer functions of DBMS and also additional functions.
PMSS uses object-relational DBMS (ORDBMS) PostgreSQL as its database platform. ORDBMS provides possibility to define user defined types (UDT) and user defined functions (UDF) (see SQL:1999 [17] and also SQL:2003 standard). Architecture of PMSS will be described in terms of functional subsystems and data centric subsystems that are called registers. Identification of such subsystems is part of the IS strategic development that is proposed by Roost and others [18]. A functional subsystem corresponds to one or more business processes [18] that are performed using repository manager. "A register is a logical data-centric view of a business object that holds the state and transactions data of the object and provides related recording and query services." [18]. A register corresponds to one logical part of the repository. One functional subsystem reads and modifies data in one or more registers. Subjects who have some role in

the system use services of functional subsystems that belong to the area of competence of their role [18].
Architecture of PMSS (see Figure 1) can be explained in terms of the levels of the information system that are proposed by Halpin [19].

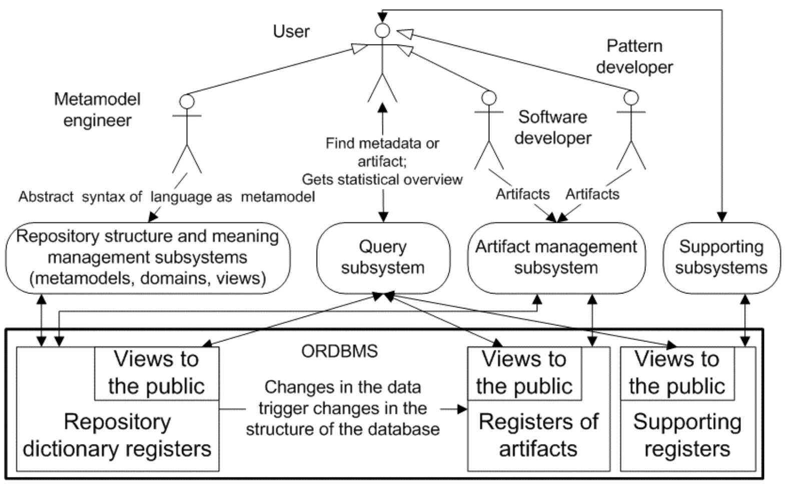

User
Pattern developer
Metamodel engineer
Software developer
Find metadata or artifact; Gets statistical overview
Abstract syntax of language as metamodel
Artifacts
Artifacts
Repository structure and meaning management subsystems (metamodels, domains, views)
Query subsystem
Artifact management subsystem
Supporting subsystems
ORDBMS
Views to the public
Repository dictionary registers
Changes in the data trigger changes in the structure of the database
Views to the public
Registers of artifacts
Views to the public
Supporting registers

Fig. 1. PMSS Architecture

Conceptual level describes structure of the application domain according to Halpin [19]. Application domains in PMSS are patterns and models. Their abstract syntax is described using metamodels and recorded in the repository dictionary registers.
Logical level is expressed in terms of data and operations that are supported in the data model according to Halpin [19]. Repository of PMSS is dynamically extensible. Changes in the metamodels trigger automatic changes in the structure of registers of artifacts. For example, new tables, columns or domains are created. New tables are needed in order to record elements of artifacts. System records also information about the database objects.
Internal level includes details about the physical storage and access structures according to Halpin [19]. PMSS is built on top of the ORDBMS and uses its data management mechanisms.

How can one retrieve artifacts? Artifact is broken to elements and each element is recorded in the database. It allows searching artifacts using SQL. Files that contain serialized artifacts can be recorded in the database and associated with the artifacts. External level specifies operations accessible to the users or group of users according to Halpin [19]. PMSS uses views that help users to find artifacts and additional (including statistical) information about them.

3.2.1 Repository Structure and Meaning Management Subsystem

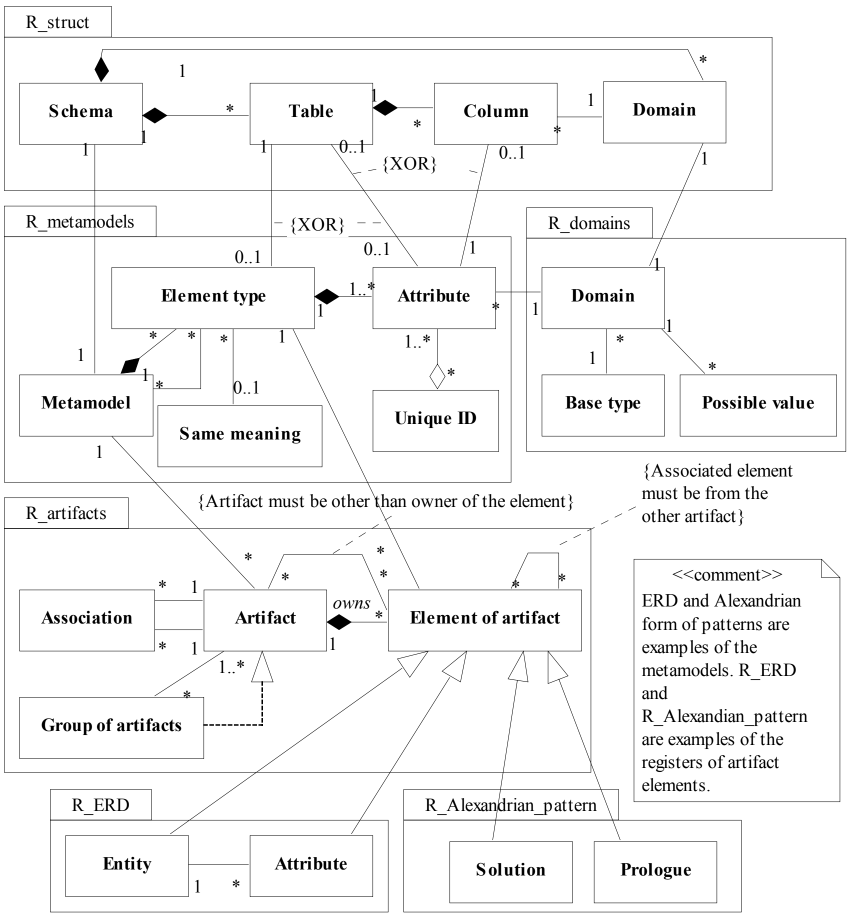
Role of the users of all these subsystems is a metamodel engineer. Subsystems use repository dictionary registers that provide a data dictionary function to the whole system. User describes abstract syntax of the modeling or pattern writing language as metamodels using subsystem of metamodels ("S_metamodels"). It can be done using forms. Metamodels are recorded in the register of metamodels ("R_metamodels). Examples of metamodels:

- Metamodel of Alexandrian form of patterns.
- Metamodel of Pattern and Component Markup Language [20].
- Metamodel of Entity Relationship model.

Metamodel engineers describe element types and their attributes, associations and generalizations. Each element type belongs to the certain metamodel. "Prologue", "Problem statement", "Discussion", "Solution", "Diagram" and "Epilogue" are examples of the element types that are used in the Alexandrian form of patterns according to PPR [5]. "Entity", "Attribute", "Relationship" are examples of the element types that are used in the Entity Relationship model. System enables registering semantics of the metamodel elements as prose. For example, problem statement has attributes "short description" and "long description". Metamodel engineers can also group element types that are part of the different metamodels but have a similar meaning. For example, element "Problem" in the Canonical pattern form corresponds to the element "Intent" that belongs to the pattern structure described by Gamma and others [21] according to PPR [5]. Element of the artifact can be associated with one or more artifacts. Metamodel determines types of artifacts with which element of the artifact can have associations.

Date [13] writes: "The sole good idea of object systems in general is proper data type support". Therefore PMSS permits creation of domains that determine possible values of attribute that have this domain. Domain can have a set of associated constraints and default value. For example, domain determines minimum and maximum amount of values that attribute can have. Multivalued attribute can have more than one value. Domain can have an explicitly defined set of values that are only allowed values of attributes that have this domain. Each attribute in the metamodel is associated with the domain. Each element type has by default an attribute that is a unique identifier and has the *systemic domain* "surrogate key" which values follow the rules of surrogate key (see Date [13]). Values of this domain are generated by the system. Element type can have more than one unique identifier. Metamodel engineer manages domains using subsystem of domains ("S_domains") and they are recorded in the register of domains ("R_domains").

Some metamodels that are recorded in "R_metamodels" are *systemic metamodels* that describe structure of the registers of PMSS: "R_metamodels", "R_domains", "R_artifacts", etc. Each not-systemic metamodel has one corresponding register of artifact elements ("R_artifact_elements"$_i$) where artifacts with this metamodel are recorded. Creation or modification of not-systemic metamodels triggers changes in the database structure. New database objects are created or existing ones are modified using rules that are described in Table 1. System also registers database objects of the repository in the register ("R_struct") (see Figure 2) and correspondence between the metamodel elements and database objects in the register of mappings ("R_map"). "R_map" is omitted from the Figure 2.



R_struct
Schema
Table
Column
Domain
{XOR}
R_metamodels
Element type
Attribute
Metamodel
Same meaning
Unique ID
R_domains
Domain
Base type
Possible value
{Artifact must be other than owner of the element}
{Associated element must be from the other artifact}
R_artifacts
Association
Artifact
owns
Element of artifact
Group of artifacts
<<comment>> ERD and Alexandrian form of patterns are examples of the metamodels. R_ERD and R_Alexandian_pattern are examples of the registers of artifact elements.
R_ERD
Entity
Attribute
R_Alexandrian_pattern
Solution
Prologue

Fig. 2. Registers for recording metamodels, domains and artifacts

Table 1. Database objects that are created based on the metamodel elements

Element type	Type of database object
Metamodel	Schema
Element type	Table (in the schema)
Single-valued attribute	Column (in the table)
Multivalued attribute	Table, columns and foreign key constraint

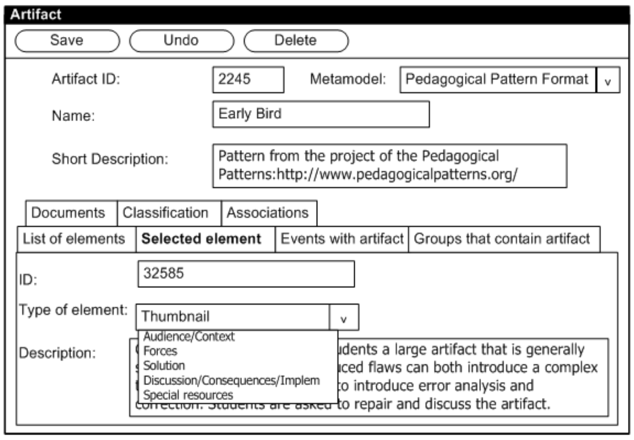
Unique identifier (ID)	Primary key or alternative-key constraint. The latter is implemented using uniqueness and NOT NULL constraint.
1:M relationship	Foreign key column and constraint
M:N relationship	Intermediate table that contains foreign key columns and has foreign key constraints
Domain	Domain. Domain "surrogate key" has corresponding sequence generator.

Registers "R_struct" and "R_map" contain also information about the database objects that correspond to the systemic metamodels. Therefore it is possible to get information about the structure and meaning of the whole repository from one source. For example, it is possible to find out that names of the metamodels can be found from the column "name" in the table "Metamodels" that belongs to the schema "Metamodels". It is advantageous if one wants to find appropriate values to the parameters of views (see section 3.1.4). Creation of the systemic metamodels, domains and corresponding database objects is part of the setup process of PMSS.

Element type has properties that determine minimum and maximum amount of corresponding elements that can be part of the artifact that has this metamodel. If new element is added to the artifact then system automatically checks this rule.

Metamodels are associated with the classification schemes that are used for the classification of the corresponding artifacts. These classification schemes are managed using subsystem of classification schemes ("S_classification_schemes") and recorded in the corresponding register ("R_classification_schemes") (see Figure 4). For example, patterns can be (good) patterns or anti-patterns. In addition artifacts can be classified by granularity, variability and articulation like in the OMG RAS [8], by maturity [4] and also by subject area (object-oriented design, data modeling etc.).

3.2.2 Subsystem for the Management of Artifacts



Artifact ID: 2245
Metamodel: Pedagogical Pattern Format
Name: Early Bird
Short Description: Pattern from the project of the Pedagogical Patterns:http://www.pedagogicalpatterns.org/
ID: 32585
Type of element: Thumbnail
Description: udents a large artifact that is generally uced flaws can both introduce a complex o introduce error analysis and correction. Students are asked to repair and discuss the artifact.

Fig. 3. Draft of the form that is used for the management of artifacts

Roles of users of this subsystem are pattern developer or software developer. They create, modify or delete artifacts using subsystem of artifacts ("S_artifacts"). "S_artifacts" uses register of artifacts ("R_artifacts") and registers of artifact elements. Each not-systemic metamodel has one corresponding register of artifact elements. Figure 3 gives an example of the artifact management form. From the document section one can select available files that contains serialized artifact.
Artifacts can be associated and grouped. Associations and groups have a type. For example, two patterns can be associated because they are alternatives. Pattern language is example of the group of patterns. Artifact group is also an artifact that has the metamodel. It gives possibility to associate additional information with the artifact language as a whole.
Software developer can also register information about the usage of the artifacts in the development work and its opinion about the artifact. This is special kind of artifact event and is recorded in the register of artifact events ("R_artifact_events"). It is background information to the potential users of the artifact.

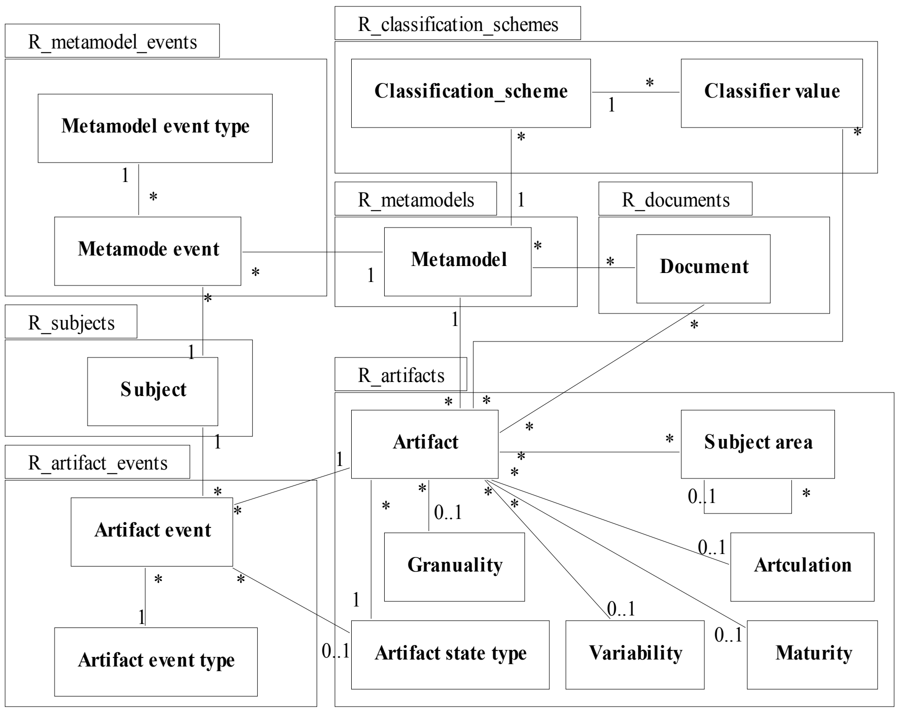


Fig. 4. Registers for recording information about the artifacts, events, documents and classification schemes

3.2.3 Supporting Subsystems

PMSS should permit registration of metadata about documents as well as storing document files. Examples of documents are documentation of development results and XMI files that contain serialized models. Documents can be associated with the artifacts in the repository. For example, pattern can be associated with the documents based on which it was worked out or documents that contain supporting or contradicting examples. Metadata about the documents and document files are recorded in the register of documents ("R_documents") using subsystem of documents ("S_documents"). PostgreSQL is object-relational DBMS and therefore allows recording large objects in a table. It makes possible saving document files to the database.

PMSS allows management of subjects. Subjects are organizations or persons. Subject can have one or more role. Administrator manages subjects who use PMSS. Software developers and pattern developers manage subjects who have somehow contributed to the creation of the models or patterns but are not users of PMSS. Data about the subjects is recorded in the register of subjects ("R_subjects") using subsystem of subjects ("S_subjects").

3.2.4 Registration of Events

PMSS records automatically information about the creation or modification of the artifact or its elements by the developers. These events are registered in the register of artifact events ("R_artifact_events") (see Figure 4). This approach is similar to the approach of active artifact that is taken by OSCAR [10]. "The active artefact records logs of who and what has accessed the artefact and (if possible) their purpose in doing so" [10]. System records time and type of the event and name of the table where this event has happened. It associates this information with the reference to the subject and artifact. Special kind of event is change of the state of the artifact. Possible states of the artifact that is not patterns are: under construction; ready for the usage; archived. Possible states of the pattern are: under construction; candidate pattern; accepted pattern; under evaluation; archived. If a state of the artifact is changed then this event is recorded and associated with the new state type.
System uses similar approach in order to record information about the events that have happened with metamodels, domains and views. System registers this information to the registers "R_metamodel_events", "R_domain_events" and "R_view_events", respectively.
System uses triggers in order to technically implement registration of events. For example, if system creates table for the registration of artifact element then it also creates triggers that are associated with the table and which tasks are registration of the events that have happened with the data in the table.

3.2.5 Subsystems for the External View to the Artifacts

Software developers and pattern developers search artifacts. Metamodel engineers search metamodels. Managers want statistical overview about the artifacts and their usage.

View is a stored query that is executed then user invokes it. PMSS must support views that use queries described by Eessaar [22]. Each subject or role has a set of views that provide access to the data in the repository. User of PMSS must have up to date information about the views that one has right to use. Therefore PMSS contains subsystem of views ("S_views") and register of views ("R_views") (see Figure 5). "S_views" permits registration of views and registration of the association between the view definition and database object that implements this view. "S_views" belongs to the category of Repository Structure and Meaning Management Subsystems.

PMSS uses parameterized and not-parameterized views. Levy and others [23] write: "A parameterized view is a conjunctive query that contains placeholders in argument positions in the body of the view, in addition to variables and constants." User of the parameterized view must give values with appropriate type to all the arguments of this view. In many cases possible values can be selected from the repository. If the parameter is associated with the domain then (a) its permitted values are all values that correspond to the rules of this domain. If the domain is associated with the set of possible values then user can choose one of these values. If the parameter is associated with the attribute (b) then its permitted values must have domain of the attribute. Additional restriction is that values must be selected from the set of existing attribute values in the repository.

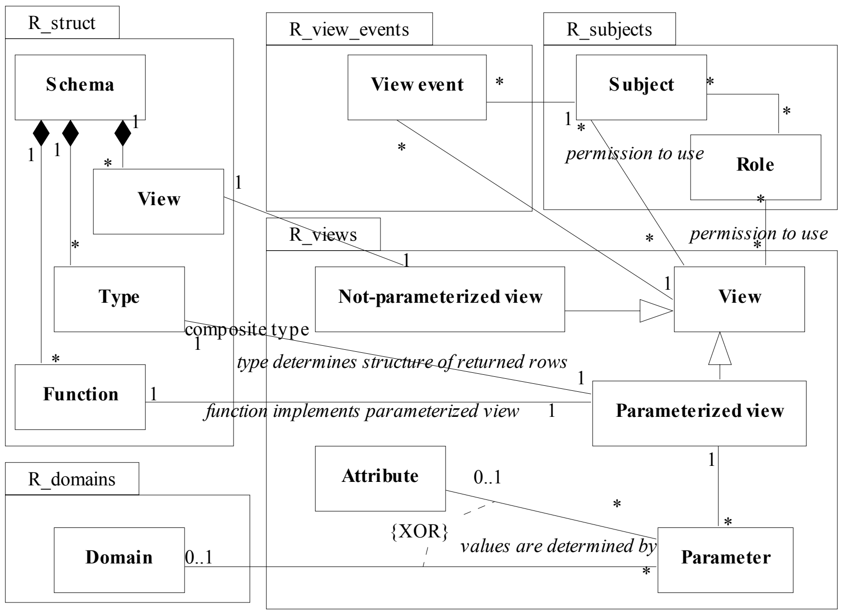


Fig. 5. Registers for recording information about the views

Next some examples of the queries will be presented. Name of the parameter and type of source of its value (in brackets) are shown between the angle brackets.

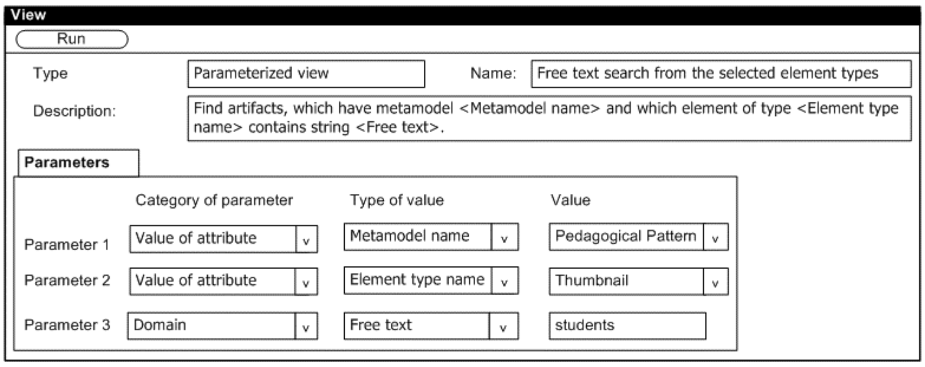
1. Find artifacts where value of attribute <Attribute id (b)> of element type <Element type id (b)> contains string <Free text (a)>.
2. Find artifacts that are associated with the element <Element type id (b)> of artifact <Artifact id (b)>.
3. Find artifacts, which elements with the meaning <Same meaning id (b)> contain string <Free text (a)>.
4. Find events with the artifact <Artifact id (b)> that have happened during last <n (a)> days.
5. For each language find amount of the artifacts that are created using this language during the period of last <n (a)> days.
6. Find amount of artifacts that are created by each subject during the period of last <n (a)> days.

Views 4-6 help to achieve goals of PMSS about the statistical overviews. Examples of the queries based on which not-parameterized views are created:

7. Find data modeling patterns.
8. Find amount of candidate patterns.

Parameterized view can be implemented in the ORDBMS using set-returning function that returns a set of records that form the answer to the question. Not-parameterized view can be implemented in the database using view object.

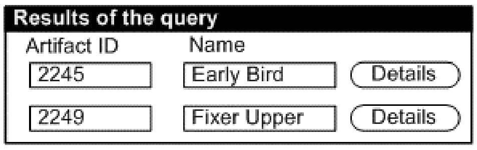
Register "R_struct" contains information about the database objects that implement views and register "R_map" contains information about the correspondence of the views and database objects. User can search the view, give values to the parameters (see Figure 6) and execute it using query subsystem "S_query". System uses corresponding database objects in order to find information and present it to the user (see Figure 7).



Type	Parameterized view	Name:	Free text search from the selected element types
Description:	Find artifacts, which have metamodel <Metamodel name> and which element of type <Element type name> contains string <Free text>.		

	Category of parameter	Type of value	Value
Parameter 1	Value of attribute	Metamodel name	Pedagogical Pattern
Parameter 2	Value of attribute	Element type name	Thumbnail
Parameter 3	Domain	Free text	students

Fig. 6. Draft of the form or page that is used for the execution of the view



Artifact ID	Name
2245	Early Bird
2249	Fixer Upper

Fig. 7. Draft of the form or page that shows the result of the view

If user wants to see details of the artifact then system opens the form that is described in the Figure 3.

4 Discussions

PMSS conforms to the four-layer metamodeling architecture of OMG Meta-Object Facility (MOF) [24]. Database objects (including constraints) in the registers "R_metamodels" and "R_domains" specify language that can be used in order to define metamodels (M3). Metamodels (M2) are recorded to the repository dictionary registers and models (M1) are recorded to the registers of artifacts (see Figure 1).

PMSS can be used during all phases of the development of information system. PMSS can be used within one organization or by the set of the collaborating organizations. It can also be used in the public domain in order to provide artifacts based on the open-source paradigm. PMSS would also be part of a development subsystem of Enterprise Information System that is proposed by Roost and others [25] and Enterprise Continuum that is proposed in the Open Group Architectural Framework [26]. Currently prototype of PMSS is under development.

Advantages of PMSS are:

- PMSS improves process of system modeling because it makes possible to search reusable artifacts as well as their metadata. Selection of some parameter values from the repository and possibility to search views makes searching more comfortable.
- PMSS records metadata about artifacts as well as artifacts themselves in the database in order to escape problems of controlling access to artifacts, which are not under control of DBMS.
- PMSS is extensible in the sense that users can define new metamodels in order to start recording new kinds of artifacts.
- Metamodel engineer can change structure of the registers of artifact elements by using concepts from the metamodel domain and not using concepts from the database design domain.
- System provides information about the meaning of the element types in the artifacts. For example, what does "prologue" mean in the context of Alexandrian form of patterns?
- Repository contains information about the purpose of database objects.
- Information about a same meaning of different element types permits searches across elements that have the same meaning but different names.

- System keeps track of events with the metamodels, domains, views and artifacts. It helps to estimate contribution of subjects and provides background information.

Disadvantages of the system are:

- PMSS distinguishes attributes and element types at the conceptual level. But Halpin [19] suggests: "an attribute-based model is inherently unstable. Even worse, applications using the model often need to be recoded when a model feature is changed." PMSS tries to overcome these problems by automatically changing database structure after changes of metamodel.
- Some drawbacks come from the fact that the current implementations of relational or object-relational DBMS's are not fully conformant with the ideas of the relational model and have problems [13]:

– SQL is defective. For example, it lacks possibilities to make queries about the hierarchical relationships. If one wants to make query in order to find all classes that are associated directly or indirectly with some class using generalization relationship then procedure has to be written.

– Each database management system has its own SQL dialect. Therefore change of DBMS means rewriting parts of PMSS.

– SQL:1999 [17] permits definition of user defined types (UDT) and creation of columns or tables that have UDT. But Date [13] argues strongly against typed tables and says that DBMS should support definition of datatypes that can be used in the definition of columns. PMSS permits definition of domains that are implemented using domain objects in order to restrict values in the columns. Creation of the domain by the metamodel engineer should cause creation of the data type and operators in the DBMS that fully conforms to the ideas of Date [13].

– Lack of possibilities to define declarative database constraints that reference two or more tables. Well-formedness rules in the metamodels could be implemented as declarative integrity constraints in the "R_metamodels" but instead triggers and control-procedures have to be created which use proprietary syntax.

– Many views that are theoretically updatable (see Date [13]) are not updatable in practice. It makes impossible to implement interface for modifying artifacts using views.

5 Conclusions and Future Work

Architecture of Pattern Management Software System was introduced in this paper. PMSS uses ORDBMS as its database platform. Definition of the abstract syntax of the artifact language causes automatic extension of the repository, in order to allow recording of artifacts. System also provides views in order to search artifacts and their metadata. Some problems of ORDBMS that restrict implementation of PMSS where also mentioned. Currently prototype of PMSS is under development.

In the future PMSS must be extended in order to support some important functionality of the traditional repository systems that are described for example by Bernstein and Dayal [16]: checkout/checkin; management of versions of artifacts; configuration management; context management. Another necessary extension is to permit user to define new views based on the metamodel descriptions. PMSS should also permit loading of existing models to the repository. Future work will also include creation of framework that generates dynamically web pages for the artifact management based on the metadata in the repository.

References

1. Greenfield, J., Short, K.: Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools. John Wiley & Sons, USA (2000)
2. Rolland, C., Loucopoulos, P., Grosz, G., Nurcan, S.: A Framework for Generic Patterns Dedicated to the Management of Change in the Electricity Supply Industry. 9th International Workshop on Database and Expert Systems Applications (1998)
3. Riegel, J. P., Kaesling, C., Schütze, M.: Modeling Software Architecture Using Domain-Specific Patterns. First Working IFIP Conference on Software Architecture, Kluwer Academic Publishers (1999)
4. Halaris, J., Geroupoulus, S.T.: Reuse Concepts and a Reuse Support Repository. IEEE Symposium and Workshop on Engineering of Computer Based Systems (1996) 27 – 34
5. Portland Pattern Repository's Wiki. Pattern Forms. (2005) http://c2.com/cgi/wiki?PatternForms
6. Yacoub, S., Ammar, H.: Pattern-oriented analysis and design: composing patterns to design software systems. Addison-Wesley, Boston (2003)
7. Mirbel, I.: A polymorphic context frame to support scalability and evolvability of information system development processes. Proceedings of the Sixth International Conference on Enterprise Information Systems, Vol. 3 (2004) 131-139
8. OMG Reusable Asset Specification. OMG Adopted Specification ptc/04-06-06

9. Dittrich, K., Tombros, D., Geppert, A.: Databases in Software Engineering: A Roadmap. The Future of Software Engineering. 22nd International Conference on Software Engineering, Assn for Computing Machinery (2000)
10. Boldyreff, C., Nutter D., Rank S.: Active Artefact Management for Distributed Software Engineering. Proceedings of the 26th Annual International Computer Software and Applications Conference, IEEE Computer Press (2002) 1081-1086
11. Blanning, R.W.: Data management and model management: a relational synthesis. Proceedings of the 20th annual Southeast regional conference (1982) 139-147
12. Park, H.C., Lee, W.B., Kim, T.G.: A relational algebraic framework for models management. Proceedings of the 26th conference on Winter simulation (1994) 649-656
13. Date, C. J.: An Introduction to Database Systems. 8th edn. Pearson/Addison Wesley, Boston (2004)
14. Mahnke, W., Ritter, N., Steiert, H.P.: Towards Generating Object-Relational Software Engineering Repositories. University of Kaiserslautern, Proc. Datenbanken in Büro, Technik und Wissenschaft, Freiburg, Germany (1999)
15. Evitts, P. A.: UML Pattern Language. Macmillian Technical Publishing (2000)
16.Bernstein, P.A., Dayal, U.: An Overview of Repository Technology. Proceedings of the 20th International Conference on Very Large Data Bases (1994) 705-713
17. Gulutzan, P., Pelzer, T.: 1999. SQL-99 Complete, Really. Miller Freeman, USA (1999)
18. Roost, M., Kuusik, R., Rava, K., Veskioja, T.: Enterprise Information System Strategic Analysis and Development: Forming Information System Development Space in an Enterprise. Proceedings of the International Conference on Computational Intelligence (2004) 215-219
19. Halpin, T.: Information Modeling and Relational Databases. From Conceptual Analysis to Logical Design. Morgan Kaufman Publishers, San Francisco (2001)
20. Pattern and Component Markup Language. Draft 3 (2003) http://www.objectventure.net/files/docs/PCMLSpecification.pdf
21. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software, Addison Wesley, Reading Mass. (1995)
22. Eessaar, E.: Methods for Searching Patterns from the Database of Patterns. The 16th Conference on Advanced Information Systems Engineering Forum Proceedings (2004) 103-111
23. Levy, A. Y., Rajaraman, A.,Ullman, J. D.: Answering Queries Using Limited External Query Processors. Proceedings of the fifteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems (1996) 227-237
24. Meta Object Facility (MOF) Specification. Version 1.4 (2002)
25. Roost, M., Kuusik, R, Rava, K., Veskioja, T.: A Model-Driven Architecture of Enterprise Information System as the Space for Information Systems Development, The 16th Conference on Advanced Information Systems Engineering Forum Proceedings (2004) 194
26. TOGAF "Enterprise Edition" Version 8.1 (2004) http://www.opengroup.org/architecture/togaf8-doc/arch