USING RELATIONAL DATABASES IN THE ENGINEERING REPOSITORY SYSTEMS

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

Keywords: Relational data model, Object-relational data model, Repository, Metamodeling.

Abstract: Repository system can be built on top of the database management system (DBMS). DBMSs that use relational data model are usually not considered powerful enough for this purpose. In this paper, we analyze these claims and conclude that they are caused by the shortages of SQL standard and inadequate implementations of the relational model in the current DBMSs. Problems that are presented in the paper make usage of the DBMSs in the repository systems more difficult. This paper also explains that relational system that follows the rules of the Third Manifesto is suitable for creating repository system and presents possible design alternatives.

1 INTRODUCTION

"A repository is a shared database of information about the engineered artifacts." (Bernstein, 1998) These artifacts can be software engineering artifacts like models and patterns. Repository system contains a repository manager and a repository (database) (Bernstein, 1998). Bernstein (1998) explains that repository manager provides services for modeling, retrieving, and managing objects in the repository and therefore must offer functions of the Database Management System (DBMS) and also additional functions.

The repository manager uses custom-built data manager or general-purpose DBMS in order to manage artifacts. System Arcadia (Taylor et al., 1988) is an example of the system that uses custom-built object manager. Omega (Linton, 1984), The C Information Abstraction System (Chen et al., 1990), PARSE (Gray, 1997) and CommonKADS repository (Allsop et al., 2002) are examples of the repository systems that use a Relational DBMS (RDBMS). RDBMS in this case is a system which uses a database language that conforms to SQL:1992 standard or is even earlier relational database language. Object-Oriented DBMSs have been often seen as a suitable platform for building repository systems (Bernstein, 1998) (Dittrich et al., 2000). Object-Relational DBMS (ORDBMS) combine features of the relational data model and object-oriented programming languages. An example of an early attempt to combine relational and object technologies in one data model is ROSE (Hardwick & Spooner, 1989) that is experimental data management system for the interactive engineering applications. Bernstein (2003) envisions that object-relational systems are good platform for the model management systems. ORIENT (Zhang et al., 2001) and SFB-501 Reuse Repository (Mahnke & Ritter, 2002) are examples of the repository systems that use a commercial ORDBMS. ORDBMS in this case is a system which uses a database language that conforms to SQL:1999 or later standard.

Relational data model was introduced by Codd (1970) and has been extensively studied since then. One notable revision is the Third Manifesto (Date & Darwen, 2000), (Date, 2003). Relational model and relational systems are often criticized because they are arguably not powerful enough for the repository system.

In this work we show that these problems are caused by the shortages of SQL standard and systems that implement that standard. There exists analyzes of SQL and comparisons of it with the proposals of the Third Manifesto (Date & Darwen, 2000), (Pascal, 2000), (Date, 2003) but they don't discuss problems in the context of specific application areas like engineering repositories.

In this paper we also explain how a relational system that follows the rules of the Third Manifesto can be used in order to create a repository database.

In this case we wouldn't have problems that are present in current ORDBMSs.

We adopt definition (Date, 2003) according to which the relational data model consists of an extensible collection of scalar types, relation type generators, facilities for defining relation variables (relvars) of such types, assignment operations for assigning relation values (relations) to relvars and an extensible collection of generic relational operators for deriving relation values from other relation values.

The rest of the paper is organised as follows. Section 2 contains results of the literature study about the problems of the relational model and RDBMSs that hamper their usage in the repository systems. Section 3 explains how a RDBMS which data model follows the rules of the Third Manifesto can be used in a repository system. Nowadays much attention is paid to the ORDBMSs. Section 4 describes problems of the current ORDBMSs that make their usage in the repository systems more difficult. Section 5 summarizes and points to the future work with the current topic.

2 LITERATURE STUDY

Next we classify problems of the relational model and RDBMSs based on the literature study and analyze these problems in terms of the Third Manifesto. Researchers and developers often present these problems as reasons why relational database is not the best choice to use in the engineering systems. Table 1 presents problems of the relational *model* that have been identified by the researchers.

Some issues that have been raised by the researchers are actually orthogonal to the relational model. We adopt the approach taken by Date and Darwen (2000, p. 21): "The question as to what data types are supported is orthogonal to the question of support for the relational model." Even Codd (1970) acknowledged possibility of the *nonsimple* domains which permitted values are relations. One reason why he argues for eliminating nonsimple domains is that they require more complicated data structures at the storage level than simple domains. Transaction model is also orthogonal to the relational model. Date and Darwen (2000) have requirement for nested transactions in the section of the Other *Orthogonal* Prescriptions.

Hierarchic and networked information can be represented relationally (Pascal, 2000, chap. 7) Issue of making queries based on data that represents graph structure is addressed in the Third Manifesto. Relational Model Very Strong Suggestion no. 6 (Date & Darwen, 2000, p. 213) requires that relational language should provide shorthand for expressing generalized transitive closure query.

Table 1 Problems of the relational model.

Problem	Authors who mention that problem
It is not powerful, flexible and expressive enough.	Hardwick and Spooner (1989),Constantopoulos et al. (1995).
Fragmentation. Data about the object is in the different relations.	Liu et al. (1996), Gray (1997).
Performance problems due to fragmentation.	Hardwick and Spooner (1989), Gray (1997).
Lack of powerful type system.	Taylor et al. (1988), Miguel et al. (1990), Emmerich et al. (1992), Liu et al. (1996), Gray (1997).
Poor support to data that represents graph structures. Lack of facilities for making queries based on such data including finding transitive closure.	Hardwick (1984), Miguel et al. (1990), Katz (1990), Emmerich et al. (1992), Gray (1997), Lange et al. (2001).
Inappropriate transaction models for the engineering systems.	Hardwick and Spooner (1989)
Detailed semantics of the relvars have to be captured outside the relational system.	Engle (2003)
Lack of possibility to preserve semantics of the relationships.	Zhang et al. (2001)

Fragmentation increases complexity to the user of database according to Gray (1997). Virtual relvars (views) help to overcome this problem in the system that follows the rules of the Third Manifesto. View-defining expression can join values of relvars that contain information about the object. It can have relation-valued attributes which values are calculated using relational operator GROUP that provides relation "nest" capability (Date & Darwen, 2000). Gray (1997) writes that fragmentation may cause performance problems. But "performance is fundamentally an implementation issue, not a model issue." (Date, 2005)

Semantics of the relvar that is understandable to the human user is specified by the external predicate of the relvar (Date & Darwen, 2000, p. 179). It could well be recorded in the catalogue of the database that must be part of the database that it describes.

Semantics of the data that is understandable to the system is represented by the internal predicate of the relvar (Date, 2003, p. 262). Constraints to the value of relvar specify internal predicate. Therefore RDBMS should provide means for defining tuple-attribute-, relvar- and database constraints.

Semantics of the relationship determine constraints that are used in order to implement this relationship and operations that may be performed with the data that participate in the relationships (Zhang et al., 2001). Relational language may contain shorthand statements that cause creation of the database objects that implement particular type of relationship. An example is a generalization relationship (Pascal, 2000, p. 158).

Table 2 Problems of the RDBMSs.

Problem	Authors who mention that problem
Views (including updatable) are inadequately supported.	Haynie (1981), Emmerich et al. (1992).
Performance problems.	Linton (1984), Miguel et al. (1990), Chen et al. (1990), Lange et al. (2001).
Inappropriate transaction models.	Katz (1990), Emmerich et al. (1992), Gray (1997).
Inadequate concurrency control mechanisms.	Constantopoulos et al. (1995)
Lack of versioning facilities.	Emmerich et al. (1992), Gray (1997).
Lack of access control on a level of single tuples in a relation	Emmerich et al. (1992)
Lack of distributed and multi-database architectures	Gray (1997)
Lack of configuration management	Gray (1997)
Lack of possibilities to have cooperative work processes	Gray (1997)

Table 2 presents problems of the RDBMSs that are mentioned in the literature. Except problems with views, all other problems are *orthogonal* to the relational *model*. Problems with the views are problems of the implementation of the relational model. Problems that are mentioned in Table 1 and 2 should primarily cause improvement of the implementation and standards but not necessarily invention of new data models.

3 USING RELATIONAL DBMS IN THE REPOSITORY SYSTEM

A repository system permits management of artifacts that are created using some language that belongs to the set of its supported languages. Repository system should allow to add new languages to this set in order to be most useful. Abstract syntax of the language can be specified using metamodel (Greenfield & Short, 2004). Each repository has an *information model* that "specifies a model of the structure and semantics of the artifacts that are stored in the repository." (Bernstein, 1998)

Information model contains general and metamodel specific part (see Figure 1). The latter is union of metamodels of languages that are supported by the system.

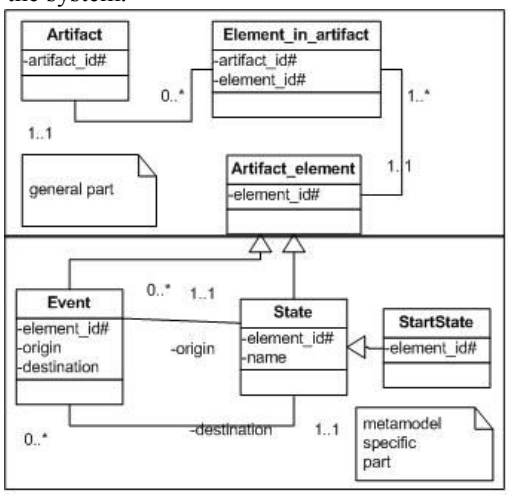

Figure 1: Example of the information model.

We propose to implement the information model in the relational database using a set of relation variables (relvars), data types (types), operators and integrity constraints. There is more than one possible design of the repository.

1. *Encapsulated artifact types*: Each artifact type has a corresponding scalar type and a base relvar in a database. Artifact is recorded as a tuple that is part of the value of this relvar.
2. *Encapsulated artifact element types*: Each artifact element type ET has a corresponding scalar type T and relvar R with an attribute that has type T.
3. *Not-encapsulated artifact element types*: Each artifact element type has a corresponding relvar where each property of the element is represented by one attribute with the appropriate type.

Artifact is recorded as a set of tuples that are part of values of more than one relvar in case of design 2 and 3. Design 1 and 2 don't eliminate complexity. They require more complex scalar type and scalar

operator specifications than design 3. In general, types should correspond to properties and relvars to entities (Date & Darwen, 2000, appendix C). Therefore we choose design alternative 3. Next we explain this design alternative more thoroughly.

Each artifact element type is implemented using at least following database objects:

1. Exactly one *relation type* RELATION {H} where H is heading of the relation in the form (C_1,..., C_n). C_1 ... C_n are pairs of type name and attribute name. Each attribute corresponds to one property of the artifact element type.
2. *Scalar types* that are used in the pairs C_1 ... C_n.
3. Set of *operators* that allow selecting and modifying components of the possible representation of these scalar types.
4. Exactly one *base relvar* with the type RELATION {C_1,...,C_n}.
5. Let's assume that we have a generalization/ specialization relationships between element types ET_1 and ET_n in the information model. ET_k is supertype and ET_{k+1} is its direct subtype ($1 \leq k < n$). Relvar R_k that corresponds to ET_k and relvar R_{k+1} that corresponds to ET_{k+1} are associated using foreign key. For each element type where $k>1$ we have to create a corresponding *virtual relvar*. For example, element type ET_{k+1} has corresponding virtual relvar V_{k+1} that joins values of relvars R_1,..., R_{k+1}. If one assigns a new value to V_{k+1}, then system must assign a new value to all the relvars R_1,...., R_{k+1}.

Integrity constraints – type-, attribute-, relval- and database constraints enforce well-formedness rules of the artifacts. Model management operations like Merge, Diff, Compose etc. (Bernstein, 2003) can be implemented using read only relation valued operators.

We illustrate our ideas by using a simple software design language *SimpleM* that was originally presented by Serrano (1999) in order to introduce *VCt* specification language. *SimpleM* specifies one diagram type. It is used for creating simple state diagrams. Diagram is a kind of artifact. Metamodel of the language is presented in the metamodel specific part in figure 1.

Serrano (1999) describes well-formedness rules of the language. We have modified rules R1 and R2.

(R1): Both StartState and State have a label with a name that is unique amongst all other states. (R2): There is at most one StartState in the repository. (R3): "The StartState can only be connected to States by outgoing Events." (R4): "Any pair of States is connected at most by two Events, one in each direction." (R5): "Loop Events, i.e. Events that connect a State to itself, are not allowed."

Next we present examples of statements for creating integrity constraints. They are written in *TutorialD* relational language and have been tested in the prototypical DBMS *Rel* (Voorish, 2005). Tutorial D language has been proposed in the Third Manifesto (Date & Darwen, 2000) and dialect used by *Rel* is based on that proposal.

```
CONSTRAINT C_2 (COUNT(StartState)<=1);

CONSTRAINT C_3 IS_EMPTY ((Event RENAME
(element_id# AS el_id#, destination AS
element_id#)     JOIN     State)     JOIN
StartState);

CONSTRAINT C_5 (IS_EMPTY (Event WHERE
origin=destination));

CONSTRAINT C_6 IS_EMPTY (Artifact_
element SEMIMINUS Element_in_artifact);
```

Each relvar must have at least one candidate key. Some well-formedness rules can be enforced by creating appropriate key constraints. Rule *R1* can be enforced by creating the relvar constraint *KEY{name}* in the relvar *State*. It declares, that name of the State is a candidate key. Rule *R4* can be enforced by creating the relvar constraint *KEY{origin, destination}* in the relvar *Event*. Constraint *KEY {artifact_id#, element_id#}* in the relvar *Element_in_artifact* ensures, that each element can participate only once in the artifact. Together with the constraint *C_2* they guarantee that each diagram (artifact) can contain at most one *StartState*.

Rules *R2* and *R5* are enforced by the relvar constraints *C_2* and *C_5*, respectively. IS_EMPTY (<relation exp>) is a scalar operator that evaluates to true if the body of the relation denoted by <relation exp> contains no tuples (Date et al., 2003). Constraints *C_3* and *C_5* could also be created using Count operator (*Count(<relation exp>)=0*).

Rule *R3* is enforced by the database constraint *C_3*. *C_3* is created based on the reformulation of *R3* to the equivalent rule *R3'*. *(R3'): StartState can't be destination of any event*. *C_3* is a database constraint and not a relvar constraint because it references to more than one relvar.

Database constraint *C_6* ensures that each element is part of at least one artifact. It uses relational operator SEMIMINUS (Date, 2003) in order to find tuples of one relation that have no counterpart in another.

If a relvar in a database gets a new value, then DBMS checks immediately conformance of this value to the integrity constraints and rejects invalid changes. Our earlier article (Eessaar, 2005) explains principles of the repository system that follows previously described principles and checks well-formedness of an artifact only if user of the system

wants that. It explains also how to implement versioning in such a system.

4 USING CURRENT ORDBMS IN THE REPOSITORY SYSTEM

Metamodel of the artifact language can be implemented in a object-relational database using a set of built-in- and user-defined data types, domains, base tables and virtual tables (views), built-in- and user defined routines, triggers, sequence generators and integrity constraints.

This section presents analysis of the problems of SQL that make usage of the current ORDBMSs in the repository systems more difficult. It is one result of this paper. Some problems are caused by the shortages of SQL standard and some are caused by the incomplete implementation of the standard in the ORDBMSs. Following description of SQL standard is based on SQL:1999 (Gulutzan & Pelzer, 1999) and SQL:2003 (Melton, 2003). We also compare existing standard and systems to proposals of the Third Manifesto.

Current ORDBMSs make it difficult to use declarative constraints in a database in order to enforce *well-formedness rules* of the artifacts. Firstly, separation of "*domain*" and "*type*" concept in SQL causes problems. Let's assume that we want to specify that names of patterns can't be empty strings or strings that contain only spaces or underscores. The Third Manifesto treats concepts "domain" and "data type" as synonyms. It prescribes that relational system should allow specification of new scalar types and scalar operators which declared types of parameters are scalar types. Type can be specified using *type constraints*. System has to enforce *strong typing* by checking that operands that participate in the operation have a right type.

In SQL a *data type* is "a set of representable values" (Melton, 2003, p. 11) and a domain is "a set of permissible values" (Melton, 2003, p. 49). According to Mattos and DeMichiel (1994) specialization by constraints should be prohibited because it requires overloading of operators. Negative implications of this approach are discussed by Date and Darwen (2000, appendix G). We add that if user wants to define a set of valid data values by adding constraints to the predefine type, then one has to create a *domain object* in SQL. It is not possible to create a new domain based on existing one. In addition, Türker and Gertz (2001) evaluate seven DBMSs that use SQL language and note that only one of them supports domain objects.

If a user wants to define a new type in SQL, based on one *predefined type* and achieve *strong typing,* then a *distinct type* has to be created. One can't use constraint definitions there. One also has to use a predefined type as a base type for distinct type and can't use distinct type as a base type for the domain. In case of using distinct or *structured types* one has to check correctness of the attribute values using the *methods* of this type. Methods can be implemented using some *imperative language* (SQL procedural extensions or other). Greenfield and Short (2004, p. 227) adopt definition: "An imperative specification describes instructions to be executed without describing the desired results of execution". Lloyd (1994) shows advantages of the *declarative* programming languages compared to imperative languages which include easier teaching, clearer semantics, improved programmer productivity and better support to meta-programming and parallelism.

Date and Darwen (2000) treat concepts *"operator"* and *"function"* as synonyms but use the term "operator". SQL (starting from SQL:1999) specifies statements for creating *user defined functions* but don't specify statement for creating operators. It is not possible to determine more convenient infix, prefix or postfix notation that is used in order to call this function. On the other hand, SQL dialect of PostgreSQL (PostgreSQL, 2005) and Oracle (Oracle, 2005) allow to create user defined operators as well as user defined functions.

Date et al. (2003, p. 22) introduces the *scalar operator IS_EMPTY* that could be built-in. For example, it is useful in order to specify constraints (see section 3). Currently there is not such built-in operator or function in SQL (Melton, 2003) and one has to program it using the Count function.

Current ORDBMSs have problems with the *relvar* and *database constraints*. Relvar constraint is associated with exactly one relvar and database constraint is associated with two or more relvars (Date, 2003). Relvar constraint can be implemented as a *CHECK constraint*. There exists ORDBMSs like PostgreSQL (PostgreSQL, 2005) and Oracle (Oracle, 2005) that don't allow to use *subqueries* in the CHECK constraint although SQL standard permits that. Relvar and database constraints can be implemented using *assertions* that constrain the set of valid values for one or more base tables in SQL (Gulutzan & Pelzer, 1999). Unfortunately Ceri et al. (2000) note that many RDBMSs don't support assertion objects although this type of object is specified in SQL standard. Türker and Gertz (2001) note in the review of integrity constraints in the different DBMS-s: "assertions are in general not available and are unlikely to be offered in the near future". Cochrane et al. (1996) write that RDBMSs don't support assertions because they are "extremely expensive to support". Maybe it means that assertion

reduces performance of the system? But performance is an issue of the implementation of the data model. Alternative method for enforcing constraints in the current ORDBMSs is to use imperative programs in the *SQL-invoked routines* or *triggers* that where both first time standardized in SQL:1999. If data in the database is changed using SQL-invoked routines, then they can check the well formedness rules. In this case routines must be the only means for modifying data. Systems like UML-repository (Marder et al., 1999), and business-rule enforcer (Zimbrão et al., 2003) use declarative OCL constraints in order to specify database constraints. They can't use assertions in order to implement these constraints and have to generate triggers. Instead of one declarative constraint we may need many triggers that are associated with different tables in order to react to all the relevant events. For example, constraint C_3 (see section 3) can be implemented using insert and update triggers that are associated with the tables Event and StartState.

If *triggers* have sufficient performance compared to assertions, then creation of the *declarative constraint* at the model level could cause automatic creation of the imperative programs (triggers) at the implementation level.

DBMS should have information about semantics of relationships between entity types in order to be able enforce properties of the relationships and answer to the queries (Zhang et al., 2001). *Generalization relationship* is an example of a generic relationship that is often used in the metamodels. For example, StartState is a State (see Figure 1). SQL standard defines language constructs for creating *subtables and supertables* that seem suitable in order to implement this kind of relationship. Both subtable and supertable must be *typed tables* and *structured type* on which subtable is defined must be subtype of the structured type on which supertable is defined (Date, 2003). As stated earlier, it is not possible to use declarative constraints in the structured type specification.

Non-standard approach is used in PostgreSQL (PostgreSQL, 2005) where subtable and supertable don't have to be typed tables. PostgreSQL implementation of subtable-supertable feature is immature because many declarative constraints of the supertable are not inherited by the subtable.

Date and Darwen (2000) show that desired functionality can be achieve in the relational system that follows the rules of the Third Manifesto, without using subtable-supertable feature and therefore raise question about usefulness it. They propose that each subtable must have corresponding *virtual relvar* that joins relations that correspond to the subtable and supertable. Value of the virtual relvar must be updatable and updates must propagate to the base relvars (Date & Darwen, 2000). Relational language could have special statement in order to create relvar that is conceptually associated with other relvar through generalization relationship (Pascal, 2000). This kind of statement causes creation of necessary base- and virtual relvars.

SQL:1992 and earlier standards don't allow to use joins in the *updatable views* (Date, 2003). Starting from SQL:1999 views defined as one-to-one or one-to-many join of two base tables are updatable (Date, 2003). Unfortunately there are ORDBMSs that don't support SQL standard in this regard. For example, in PostgreSQL all views are not-updatable without further programming (PostgreSQL, 2005). In Oracle DML statement must affect only one underlying table of the updatable join view (Oracle, 2005). But in this case system needs to add data to all the base tables that participate in the join.

Alternative is to use *rules* (PostgreSQL, 2005) or *instead-of triggers* (Oracle, 2005) that are associated with a view in order to achieve its updatability. These objects require additional programming and are not-standardized features.

Yet another possibility is to use *triggers* that are associated with the *base tables*. Let's assume that table T_{sub} is a subtable and T_{sup} is its supertable. For example, we could create delete and update triggers that are associated with T_{sub}. Their task is to delete corresponding row from T_{sup} then row in T_{sub} is deleted and update primary key of T_{sup} then corresponding foreign key is updated in T_{sub}, respectively. This approach also requires insertion of new rows into T_{sub} and T_{sub} within one transaction using two different statements. In contrast, the Third Manifesto states that constraints must be satisfied at *statement boundaries* and relational language must have *multiple form of the assignment operation* in which several individual assignments to relvars are performed in parallel as a single logical operation (Date & Darwen, 2000).

Ceri et al. (2000) notes that *handcrafted triggers* are error-prone and triggers should be created by the system. Triggers or rules that implement supertable-subtable feature must be automatically generated by the repository system then new base- or virtual table is added to the repository database. It increases complexity of the system.

Standardization of some important features that where strongly suggested by the Third Manifesto has begun in SQL:1999 or SQL:2003. It takes time before ORDBMSs start to fully implement the standard in this regard.

Recursive queries allow to find transitive closure of graph structure (introduced in SQL:1999). Information about the associations between artifacts as well as associations between elements of artifacts

is recorded in a repository. Associations and associated elements form a graph structure. Example of the query that is needed then pattern is modified: Find all patterns in the pattern language PL which directly or indirectly depend on pattern P.

Multisets (introduced in SQL:2003) could be used in the views that allow to present artifact to a user without fragmentation. SQL specifies UNNEST operator that allows to present elements of a multiset as rows of a virtual table. Integrity of these rows can be checked by the table or database constraints. As we said earlier, current DBMSs provide limited support for declarative constraints. It hampers usage of the columns that have multiset types in the base tables. Examples of the constraints are restrictions to the cardinality of a multiset or requirement that a multiset shouldn't contain repeating elements.

Table-functions (introduced in SQL:2003) allow to implement parameterized relational operators and return multiset (bag) of rows. They help to implement queries that search artifacts or statistical information from the repository.

Sequence generators generate values for the candidate keys (introduced in SQL:2003).

Multiset can contain repeating elements. It is not consistent with the Third Manifesto that prohibits duplicate tuples in a body of a relation. Developer who wants to use sets of rows instead of multisets must be continuously aware that most of SQL statements must explicitly state it.

5 CONCLUSIONS

In this paper, we showed that current RDBMSs and ORDBMs have problems that hamper their usage in the repository systems. Some necessary features like views and constraints are not *object*-oriented and are required by the earlier SQL standards. They are not implemented correctly or not implemented at all in the current DBMSs. In addition, SQL is not a correct implementation of the relational model. These shortages cause criticism towards SQL *and* relational model. They cause addition of new features to SQL standard and dialects that would be unnecessary if SQL fully conforms to the relational model. Some *object*-oriented features that are added to the ORDBMSs are actually required by the Third Manifesto. We explained how the RDBMS that follows these requirements can be used in the repository system.

Future work will include the creation of a prototype repository system that uses RDBMS that follows the rules of the Third Manifesto.

REFERENCES

Allsop, D. J., Harrison A., & Sheppard, C. (2002). A database architecture for reusable Common KADS agent specification components [Electronic version]. *Knowledge-Based Systems*, Vol. 15, No. 5, 275-283.

Bernstein, P. A. (1998). Repositories and Object-Oriented Databases [Electronic version]. *SIGMOD Rec. Vol. 27, Issue 1*, Mar. 1998), 88-96.

Bernstein, P. A., (2003). Applying Model Management to Classical Meta Data Problems [Electronic version]. In *Proceedings of the Conference on Innovative Data Systems Research 2003,* 209-220.

Ceri, S., Cochrane, R., & Widom, J. (2000). Practical Applications of Triggers and Constraints: Success and Lingering Issues [Electronic version]. In *Proceedings of the 26th international Conference on Very Large Data Bases*, 254-262.

Chen, Y.F., Nishimoto, M. Z., & Ramamoorthy, C. V. (1990). The C Information Abstraction System [Electronic version]. *IEEE Transactions on Software Engineering*, Vol.16, No. 3, March 1990, 325-334.

Cochrane, R. J., Pirahesh, H., & Mattos, N. M. (1996). Integrating triggers and declarative constraints in SQL database sytems [Electronic version]. In *Proceedings of the 22th international Conference on Very Large Data Bases*, 567–578.

Codd, E. F. (1970). A relational model of large shared data banks [Electronic version]. *Comm. ACM*, Vol. 13, No. 6, 377-387.

Constantopoulos, P., Jarke, M., Mylopoulos, J., & Vassiliou, Y. (1995). The Software Information Base: A Server for Reuse [Electronic version]. *VLDB Journal, 4* (1995), Boxwood Press, Pacific Grove, CA, 1- 43.

Date, C. J., & Darwen, H. (2000). *Foundation for Future Database Systems: The Third Manifesto,* Addison-Wesley. Reading, Massachusetts, 2nd edition.

Date, C. J. (2003). *An Introduction to Database Systems,* Pearson/Addison Wesley. Boston, 8th edition.

Date, C. J. (2005) *Database in depth : relational theory for practitioners*, O'Reilly. Beijing; Cambridge.

Date, C. J., & Darwen, H., & Lorentzos, N. A. (2003). *Temporal Data and the Relational Model*, Morgan Kaufmann. San Diego, CA.

Dittrich, K., Tombros, D., & Geppert, A. (2000). Databases in Software Engineering: a roadmap [Electronic version]. In *Proceedings of the Conference on the Future of Software Engineering*. ICSE '00. ACM Press, New York, NY, 293-302.

Eessaar, E. (2005). Truly Relational Databases as a Platform for the Artifact Management. In *Proceedings of the Fourteenth International Conference on Information Systems Development: Pre-Conference* 14-17 August 2005, Karlstad, Sweden, 207-218.

Emmerich, W., Schäfer, W., & Welsh, J. (1992). Suitable Databases for Process-centred Environments Do not yet Exist [Electronic version]. In *Proceedings of the EWSPT '92*, 94-98.

Engle, P. (2003). Data Modeling – Left and Right. *The Data Administration Newsletter*, Issue 24, April 2003. Retrieved October 07, 2005, from http://www.tdan.com/i024hy03.htm#_edn15

Gray, P. (1997). CASE tool construction for a parallel software development methodology [Electronic version]. Information and Software Technology, Vol. 39, Issue 4, 235-252.

Greenfield, J., & Short, K. (2004). *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools,* John Wiley & Sons. Indianapolis.

Gulutzan, P., & Pelzer, T. (1999). *SQL-99 Complete, Really*, Miller Freeman. Lawrence.

Hardwick, M. (1984). Extending the relational database data model for design applications [Electronic version]. In *Proceedings of the 21st Conference on Design Automation*. IEEE Press, Piscataway, NJ, 110-116.

Hardwick, M., & Spooner, D. L. (1989). The ROSE data manager: Using object technology to support interactive engineering applications [Electronic version]. *IEEE Transactions on Knowledge and Data Engineering*. Vol. 1, no. 2, 285-290.

Haynie, M. N. (1981). The relational/network Hybrid data model for Design Automation Databases [Electronic version]. In *Proceedings of the 18th Conference on Design Automation*. IEEE Press, Piscataway, NJ, 646-652.

Katz, R.H. (1990). Toward a Unified Framework for Version Modeling in Engineering Databases [Electronic version]. *ACM Computing Surveys* 22:4, 1990, 375– 408.

Lange., C., Sneed, H. M., & Winter, A. (2001). Comparing Graph-based Program Comprehension Tools to Relational Database-based Tools [Electronic version]. In *Proceedings of the 9th International Workshop on Program Comprehension*. IEEE Computer Society. Los Alamitos. 209--218.

Linton, M. (1984). Implementing relational views of programs [Electronic version]. In *Proceedings of ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, 132–140.

Liu, C., Li, H., & Orlowska, M. E. (1996). *Object-Oriented Design of Repository for Enterprise Workflows*. Retrieved October 5, 2005 from University of Queensland, DSTC Web site http://www.dstc.uq.edu.au/Research /Distributed_Databases/papers/Liu-OOD-1996.ps

Lloyd, J. (1994) Practical Advantages of Declarative Programming [Electronic version]. *Invited Lecture, GULP-PRODE '94*.

Mahnke, W., & Ritter, N. (2002). The ORDB-based SFB-501-Reuse-Repository [Electronic version]. In *Proceedings of the 8th International Conference on Extending Database Technology*, 745-748.

Marder, U., Ritter, N., & Steiert, H. P. (1999). A DBMS-based Approach for Automatic Checking of OCL Constraints [Electronic version]. In *OOPSLA'99-Workshop "Rigourous Modeling and Analysis with the UML: Challenges and Limitations", Denver, Co.*

Mattos, N., & DeMichiel, L. G., (1994). Recent design trade-offs in SQL3 [Electronic version]. *SIGMOD Rec*. 23, 4 (Dec. 1994), 84-90.

Melton, J., ISO/IEC 9075-2:2003 (E) Information technology — Database languages — SQL — Part 2: Foundation (SQL/Foundation). August, 2003. Retrieved December 26, 2004, from http://www.wiscorp.com/SQLStandards.html

Miguel, L., Kim, M. H., & Ramaroothy, C. V. (1990). A Knowledge and Data Base for Software Systems [Electronic version]. In *Proceedings of the 2nd International IEEE Conference on Tools for Artificial Intelligence*, 417-423.

Oracle® Database SQL Reference 10g Release 1 (10.1) Part Number B10759-01. Oracle Corp., Retrieved October 4, 2005, from http://download-west.oracle.com/docs/cd/B14117_01/server.101/b10759/toc.htm

Pascal, F. (2000). *Practical issues in Database Management. A Reference for the Thinking Practioner*, Addison-Wesley. Boston, Mass, 1st edition.

PostgreSQL 8.0.3 Documentation. Retrieved October 4, 2005, from http://www.postgresql.org/docs/8.0/interactive/index.html

Serrano, J. A. (1999). Formal Specifications of Software Design Methods [Electronic version]. 3rd Irish Workshop on Formal Methods.

Taylor, R.N., Belz, F.C., Clarke, L.A., Osterweil, L., Selby, R. W., Wileden, J. C., Wolf, A. L., & Young, M. (1988). Foundations for the Arcadia environment architecture [Electronic version]. In *Proceedings of the Third ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*. SDE 3. ACM Press, New York, NY, 1-13.

Türker, C., & Gertz, M. (2001) Semantic integrity support in SQL:1999 and commercial (object-) relational database management systems [Electronic version]. *The VLDB Journal, Vol. 10*, No. 4 (Dec. 2001), 241–269.

Voorish, D. (2005). An Implementation of Date and Darwen's "TutorialD". Retrieved March 26, 2005 from http://dbappbuilder.sourceforge.net/Rel.html

Zhang, N., Ritter, N., & Härder, T. (2001). Enriched Relationship Processing in Object-Relational Database Management Systems [Electronic version]. In *Proceedings of the 3rd International Symposium on Cooperative Database Systems for Advanced Applications*, 53-62.

Zimbrão, G., Miranda, R., Souza, J.M., Estolano, M.H., & Neto, F.P. (2003). Enforcement of Business Rules in Relational Databases Using Constraints [Electronic version]. *SBBD 2003*, 129-141.