SCIENTIFIC PAPERS, UNIVERSITY OF LATVIA, 2010. Vol. 757
COMPUTER SCIENCE AND INFORMATION TECHNOLOGIES 31–42 P.

A Collaborative Web-based and Database-based Meta-CASE Environment

Rünno Sgirka[1],
[1] Department of Informatics, Tallinn University of Technology, Raja 15, 12618 Tallinn, Estonia
runno.sgirka@gmail.com

Abstract. Meta-CASE systems are used for the creation of CASE (Computer Aided Software Engineering) systems. In this paper, a web-based meta-CASE system is presented that uses an object-relational database system (ORDBMS) as its enabling technology. It allows us to integrate different parts of the system and to simplify the creation of meta-CASE and CASE systems. The proposed system includes a query subsystem, which allows developers to use queries to evaluate and gradually improve artifacts.

Keywords: CASE, meta-CASE, object-relational database system, SQL, queries, consistency, completeness, software measure.

1 Introduction

Meta-CASE systems are used for the creation of CASE systems. In [1], [2] and [3], we have introduced a web-based and database-based meta-CASE system, which allows us to create web-based and database-based CASE systems. As a web-based system, it allows distributing the CASE systems more widely, opposed to many traditional CASE systems which are thick-client programs that must be installed and run on the host computer. In addition, web-based systems make the group collaboration between users of the systems more available. Delen et al. [4] write: "Second, there is a need for a completely Web-based integrated modeling environment for distributed collaborative users."

As a database-based system, our meta-CASE system uses an object-relational database management system (ORDBMS) as its *enabling technology*. More precisely, it uses an ORDBMS, the database language of which conforms more or less to SQL:2003 standard [5]. Fugetta [6] explains that *enabling technologies* are integrating platforms that provide extensions to traditional operating systems and support "the creation of logically integrated but physically distributed systems". If a system uses a DBMS as its enabling technology, then developers of the system can use the system-defined (built-in) features and extensibility mechanism of the DBMS to simplify the creation of the entire system. Dittrich et al. [7] analyze the use of DBMSs in the field of software engineering

and conclude that modern systems provide "a rich set of functionality that is able to meet several requirements of software engineering applications in a satisfactory manner".

The author of this paper has chosen to continue the research and development of the aforementioned meta-CASE system in his Doctoral thesis. The system has been extended and improved since then. In [3], we introduced the query subsystem of the meta-CASE system and CASE systems that have been created by using the system. It allows us to take advantage of the powerful query mechanism of the ORDBMS and to create queries that can be executed based on artifacts. For instance, queries can be used to identify consistency and completeness problems of artifacts, to find how well developers have followed modeling guidelines, and to calculate values of software measures.

In this paper, we will demonstrate the use of the meta-CASE system by presenting a small case study. The metamodel of the *Family Tree Modeling Language*, introduced in the tutorial website [8] of one of the leading meta-CASE systems in the market, MetaEdit+ [9], is used as an example. Our meta-CASE system is mainly intended to be a research vehicle and not an industrial strength system and it currently provides less functionality compared to more matured meta-CASE systems.

The rest of the paper is organized as follows. In Section 2, our web-based and database-based meta-CASE system is described, together with its query subsystem. A small case-study using the *Family Tree* metamodel is presented to demonstrate it in practice. In Section 3, we outline topics regarding the future work with meta-CASE systems in general and with the current system. It is planned to study the topics in the Doctoral thesis of the author of this paper. Finally, some conclusions are drawn.

2 A Web-based and Database-based Meta-CASE System

In this section, the proposed meta-CASE system is introduced. Both the meta-CASE system and CASE systems that are created by using the meta-CASE system use an ORDBMS as their enabling technology. A prototype of the system has been created, using PostgreSQL 8.0.4 ORDBMS and PHP 5.0.5 scripting language.

In our system, there are two types of users. An *administrator* has to create a new CASE system by specifying a metamodel. In addition, an administrator has to register settings of the user interface and user identification of the CASE system. Administrators also have to manage queries. *Developers* (end-users) use the CASE systems in order to manage (create, read, update, and delete) artifacts. Developers can execute queries based on artifacts.

Fig. 1 presents the UML class diagram of the *Family Tree* metamodel presented as a case study in our system. We have extended the metamodel compared to the example in [8], by adding *parent_relation* and *parent_relation_type* classes. The former class represents the relationship between two parents; the latter class is a classifier which characterizes the relationship. These classes were added for better demonstration of using the queries to evaluate the artifacts.

In our meta-CASE system, each *metamodel* is used to describe the modeling language of a CASE system. For every metamodel, there is a corresponding SQL schema (an *artifact base*) in the database. For example, if an administrator creates a metamodel *family_tree*, then the meta-CASE system creates a database schema *family_tree,* which

is then used for storing the artifacts of the *family_tree* metamodel. The different schemas have been used to avoid possible name conflicts.

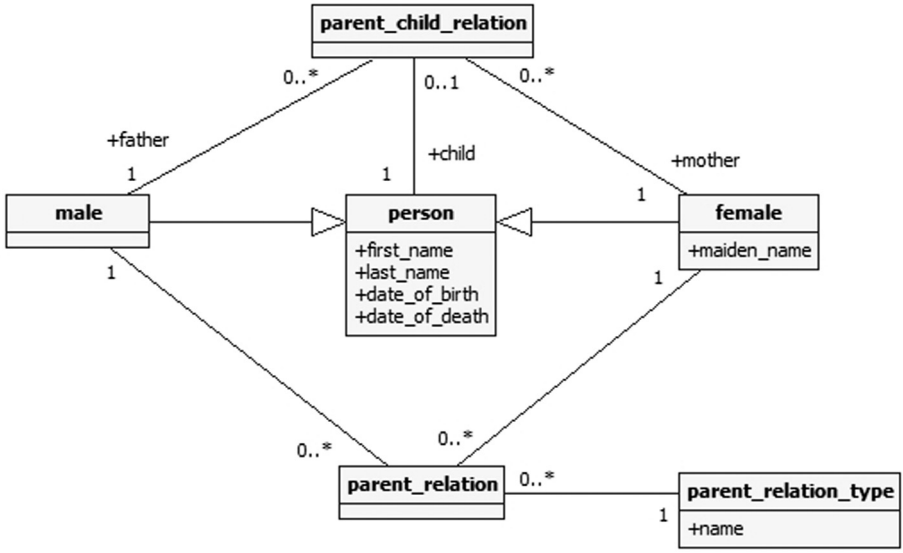

Fig. 1. An UML class diagram representing a Family Tree metamodel.

Each metamodel consists of zero or more *objects*. For every object, there is a corresponding *table* in the database schema corresponding to the metamodel. For example, if an administrator creates object *person* in the metamodel *family_tree*, then the meta-CASE system creates table *person* in the database schema *family_tree*.

Each object has exactly one type. Possible types are *main*, *inherited main*, *relationship,* and *classifier*. A *main object* is the most common one in our meta-CASE system, representing an element in the metamodel. It does not have the constraints describing a relationship object or a classifier object. For example, in the context of the *Family Tree* metamodel, the object *person* is a main object. An *inherited main object* is a main object which is derived from another main object, inheriting all the latter's characteristics. Objects *male* and *female* can be classified as inherited main objects, because both are derived from some other main object, in this case from object *person*. Our system makes use of the INHERITS clause of the PostgreSQL ORDBMS in the table definition statement. The non-standard INHERITS clause specifies "a list of tables from which the new table automatically inherits all columns" [10].

Each *classifier object* is used to characterize other objects in a metamodel. For example, object *parent_relation_type* is a classifier object, which characterizes *parent_relation* object. Possible values for *parent_relation_type* object can be *commitment*, *open marriage*, *marriage* and *divorce*.

Each *relationship object* represents a relationship between two or more main, inherited main, or classifier objects, thus supporting n-ary relationships. Currently our system does not support relationship objects, which connect a relationship object and another object. For example, to achieve the most accurate portrayal of the family *relationship* and parent/child *roles* described in MetaEdit+ [8], the counterpart in our

system *should* be built as follows: a relationship object (e.g. *family*), connecting a *male* and a *female* object, and a relationship object (e.g. *family_child*), connecting the *family* object and a *person* object. Since our system does not currently support relationship objects connecting to another relationship object, we have settled to build the family relationship as shown in Fig. 1: a relationship object *parent_child_relation* connects foreign objects *father*, *mother* and *child* (referencing *male*, *female* and *person* objects, respectively). In addition, to represent the relationship between parents, we have created relationship object *parent_relation*, which connects *male*, *female* and *parent_relation_type* objects.

Fig. 2 represents the screenshot of the objects of the *family_tree* metamodel in our meta-CASE system. Note that there are some additional objects generated by the system itself. Object *sysg_artifact* allows the developers to create different artifacts, which are all recorded in the same artifact base. All these artifacts have the same metamodel. Objects *sysg_user* and *sysg_user_artifact* are used for end-user (developer) management. Object *sysg_file* is used for storing files in the artifact base.

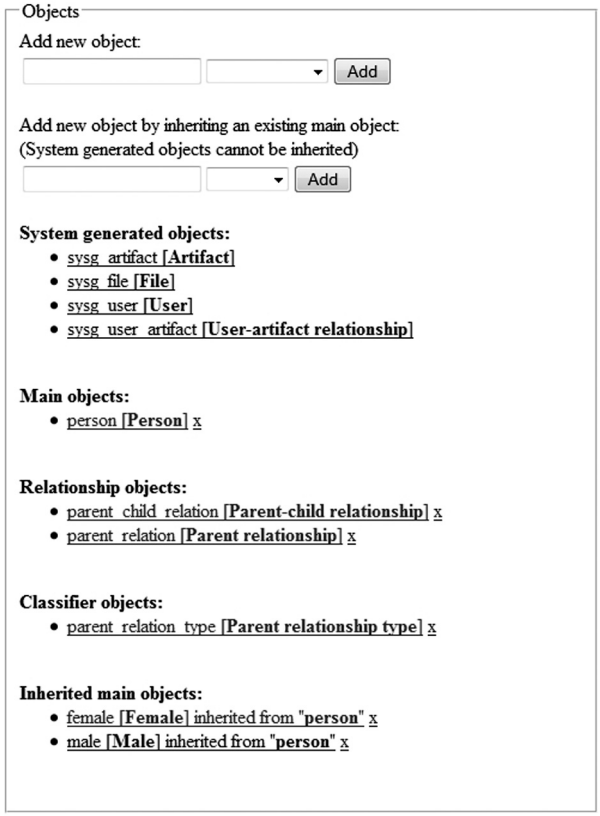

Fig. 2. A screenshot of the objects of the *family_tree* metamodel.

Every object consists of zero or more *sub-objects*. For every sub-object, there is a corresponding *column* in the database table corresponding to the object. The *type*

of the sub-object specifies the type of the corresponding column. For example, if an administrator creates sub-objects *first_name* and *last_name* with the type *varchar(100)* and sub-objects *date_of_birth* and *date_of_death* with the type *date* to object *person*, then the system generates columns *first_name* and *last_name* with the type *character varying(100)* and columns *date_of_birth* and *date_of_death* with the type of *date* to table *person*. If the sub-object is used to reference another object, then the system will generate a foreign key constraint to the corresponding column. This sub-object is called *foreign object* in our system. For example, if an administrator creates the foreign objects *father*, *mother* and *child* to object *parent_child_relation*, referencing to *male*, *female* and *person* objects, respectively, then the system generates columns *father*, *mother,* and *child* to table *parent_child_relation*, and adds foreign key constraint to each of them, referencing the tables *male*, *female,* and *person*, respectively.

Fig. 3 represents the sub-objects of the *parent_child_relation* object in our meta-CASE system. Note that all of the foreign objects are also part of the *additional unique identifier*. For relationship objects, this is done automatically by the system. Each column corresponding to the sub-objects in the additional unique identifier is included to the UNIQUE constraint of the table corresponding to the object. For objects other than relationship object, managing the additional unique identifier and therefore the UNIQUE constraint of the corresponding table is the responsibility of the administrator. If at least one sub-object is added there, the system automatically includes the sub-object *sysg_artifact_id* as well.

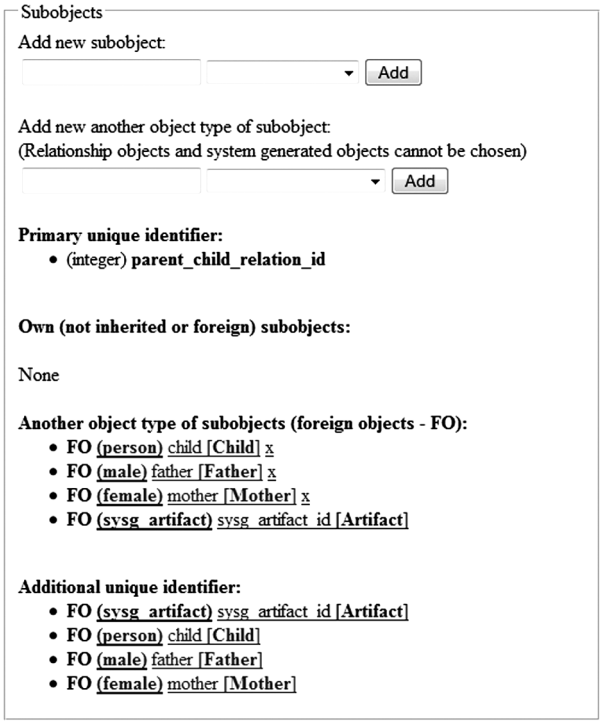

Fig. 3. A screenshot of the sub-objects of the relationship object parent_child_relation.

There are no sub-objects other than foreign objects in object *parent_child_relation*. However, note that in addition to the sub-objects added by an administrator, there is also a system generated sub-object present (*sysg_artifact_id*). This sub-object references the *sysg_artifact* object and is included in every main, inherited main and relationship object. It ensures that the corresponding tables in each artifact base contain identifiers of artifacts. It simplifies the creation of queries about a single artifact – the queries must perform the restriction operation based on column *sysg_artifact_id*.

Fig. 4 represents the objects of inherited main object *female* in our meta-CASE system. Note that the only sub-object not inherited is the sub-object *maiden_name*. Every other sub-object is derived from object *person* (including foreign object *sysg_artifact_id*).

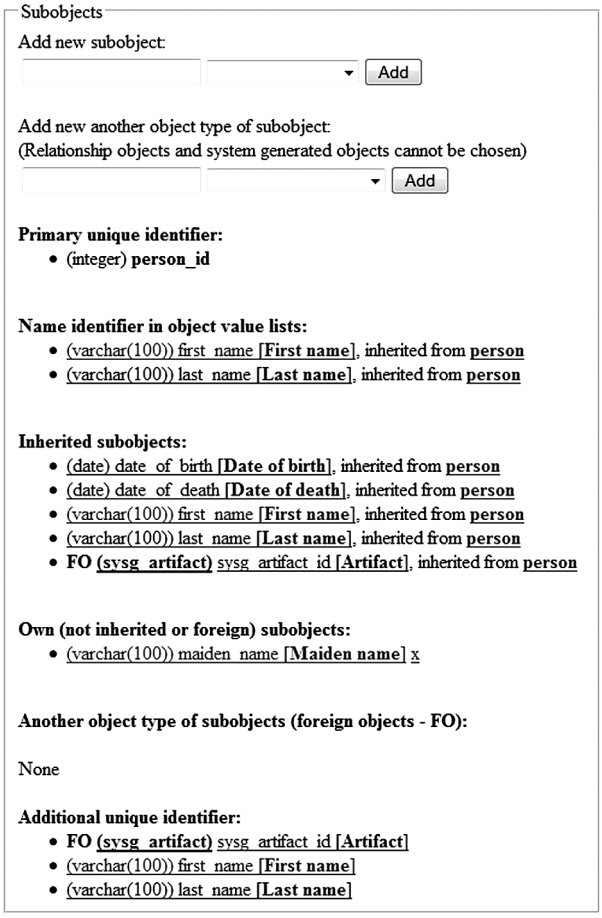

Fig. 4. A screenshot of the sub-objects of the inherited main object *female*.

The *name identifier* is used in our systems to specify the sub-objects which "represent" the objects to the developers in different lists (main menu, combo boxes etc). For example, an administrator has included *first_name* and *last_name* sub-objects

as name identifier for object *person* and the settings are consequently derived by objects *male* and *female*. If an administrator chooses to add object *person* or objects *male* or *female* to the main menu of the *Family Tree* CASE system, then the object instances are displayed there by using the values of sub-objects *first_name* and *last_name* (e.g. *Homer Simpson*, *Marge Simpson* etc). The name identifier is a decorative functionality only and has no corresponding constraint in the database, except that all sub-objects in the name identifier are also added to the additional unique identifier.

Before developers start using a CASE system, an administrator should define the *metamodel settings*. These include *setting up the main menu* of the CASE system, *managing the user settings,* and *adding classifier instances* that correspond to classifier objects. Setting up the main menu allows the administrator to determine the menu tree which is then displayed in the CASE system. By default, the highest element of the menu tree is always the *sysg_artifact* object and it cannot be changed, which means that developers have to choose an artifact instance in order to be able to manage artifact elements. Fig. 5 presents the main menu of the CASE system based on the *family_tree* metamodel. An administrator has added object *person* under the *Artifact* menu item, which means that *person* instances will be shown for each *sysg_artifact* instance, using the name identifier.

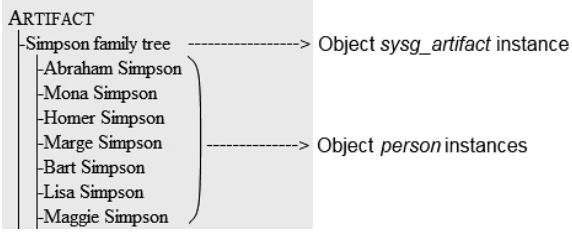

Fig. 5. A screenshot of the main menu of the CASE system based on metamodel *family_tree*.

The user settings management allows administrators to define whether all developers are allowed to see all the artifacts in the artifact base or only those, which are created by the developers themselves. In addition, an administrator can choose if the developers have to register first before accessing the CASE system or the CASE system is available to everyone without registering. System generated objects *sysg_user* and *sysg_user_artifact* are used to handle the necessary data. For example, an administrator states that the CASE system based on *family_tree* metamodel and all artifacts inside are available to all users without registering.

Finally, an administrator must add the classifier instances for each classifier object for developers to use. For example, an administrator adds instances *Commitment*, *Open marriage*, *Marriage* and *Divorce* to classifier object *parent_relation_type*.

Fig. 6 displays objects *parent_child_relation* and *parent_relation* in the CASE system based on the *family_tree* metamodel. A developer has added an artifact *Simpson family tree*, which contains object *male* instances *Abraham Simpson*, *Homer Simpson* and *Bart Simpson*, and object *female* instances *Mona Simpson*, *Marge Simpson*, *Lisa Simpson* and *Maggie Simpson*. All these values are also instances of object *person*.

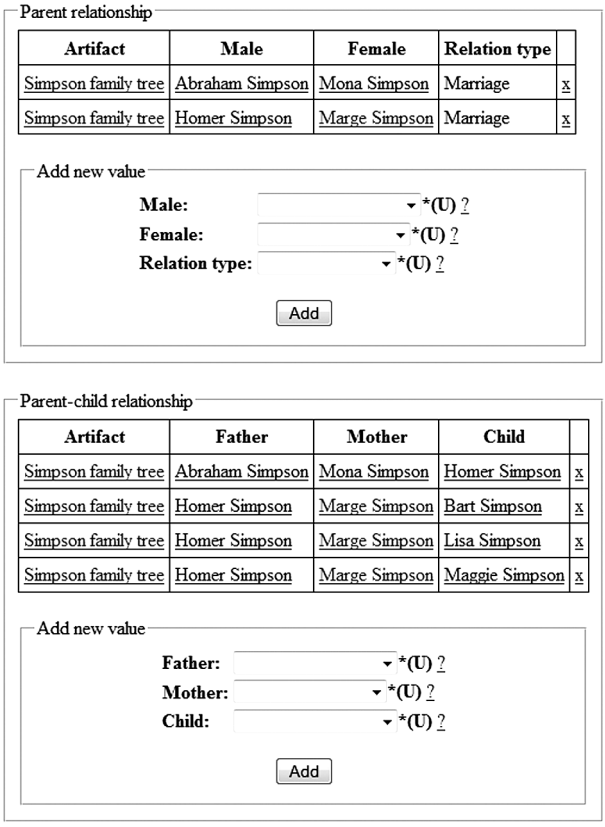

Fig. 6. A screenshot of objects *parent_child_relation* and *parent_relation* in the CASE system.

Administrators can use the *query subsystem* of the meta-CASE system to create queries that help developers to understand and improve artifacts. The queries (SQL SELECT statements) will be executed based on artifacts that are recorded in an artifact base. Simple and consistent syntax of a database language makes it easier to construct such queries and hence makes this kind of system more useful.

It is possible to create queries for different *purposes*, with different *scope* and expecting different *type* of *results*. Query purposes can include, for example, finding violations of constraints that give information about the semantics of the underlying metamodel of a CASE system, finding violations of design guidelines, or calculating values of software measures and general queries that cannot be classified to any of the previous category.

The scope of the queries can be exactly one artifact or all artifacts that are in one artifact base. In the first case, administrators can use generic placeholder #A# when specifying an artifact, using *sysg_artifact_id = #A#*. If a developer selects an artifact *af* and executes the query, then the system replaces the placeholder with the identifier of *af* before executing the query.

The expected results of queries can be either scalar values (for instance, with type Boolean; the results are tables with one column and zero or one row) or tables where

there are one or more columns and zero or more rows. For example, developers can use these queries to find elements of artifacts that have problems.

All the queries that can be used in the system have to be written by using the SQL dialect of the underlying DBMS (PostgreSQL™ in our case). The system allows administrators to save and check only SELECT statements. The statements cannot contain reserved words like INSERT, UPDATE or DELETE that are not delimited (not between “”). It is needed to prevent SQL *injection* attacks.

Choinzon and Ueda [11] present a set of software design measures together with their thresholds of undesirable value. Our meta-CASE system supports the use of *measure defect queries* that allow developers to determine design defects based on the values of measures. For each such query *q* there are one or more non-overlapping ranges of scalar values that have the same type as the expected results of *q*.

Administrators can construct *tests* based on Boolean queries and measure defect queries. The scope of the queries that belong to a test must be one artifact. Developers can execute these tests in order to evaluate artifacts. For example, a test *t* contains queries *q1,..., qn*. Each query in a test is a subtest of this test. For each query that belongs to *t*, administrators have to define a set of acceptable values. If a developer executes *t* in case of an artifact, then *t* fails if at least one query, which belongs to t, fails.

We have created some queries, which allow the developers to check the artifacts created by using the *family_tree* metamodel.

Let us assume that two parents have to have a relationship in order to have children. The following query checks whether there are parent instances in an artifact that have children, but have no relationship. If there are any, the query will return TRUE. In our case study (the *Simpson family tree* artifact), there are no such parents and therefore this query returns FALSE.

```
SELECT public.is_empty('SELECT pr.* FROM parent_relation
pr, parent_child_relation pcr WHERE pr.male = pcr.father
AND pr.female = pcr.mother AND pcr.sysg_artifact_id =
#A#') AS result;
```

IS_EMPTY is a user-defined function that has one parameter, the expected value of which is a query (SELECT statement) *q*. If the execution of *q* results with a table that has zero rows, then the function returns TRUE. If the execution of *q* results with a table that has more than zero rows, then the function returns FALSE.

The following query is an example of a query, the answer of which will be found based on all the artifacts that are recorded in an artifact base. This query allows developers to find the average amount of children in a family over the whole artifact base of *Family Tree* models. In our case study, this query returns “2.0”.

```
SELECT avg(children) AS result FROM (SELECT count(*) AS
children FROM parent_child_relation GROUP BY father,
mother) AS foo;
```

The table names in the query statements are *unqualified* names. If a user executes a query that is associated with a metamodel *m*, then the system modifies the schema search path in order to ensure that the result will be found based on tables that are in the schema that corresponds to *m*.

The following *general query*, the scope of which is one artifact, can be used to find children who are still underage (< 18).

```
SELECT p.* FROM person p, parent_child_relation pcr WHERE
p.person_id = pcr.child AND extract(year from age (p.date_
of_birth)) < 18 AND pcr.sysg_artifact_id = #A#;
```

3 Future Work and Conclusions

The proposed meta-CASE system is completely web-based. Users of the system need only a web browser and do not have to install additional plug-ins. An administrator must specify a metamodel and determine settings of the CASE system that is created by using the meta-CASE system. Based on this information the system is able to create data structures and dynamically generate web pages, through which it is possible to manage models. Dynamic generation means that if a modeler requests a web page, then the system determines the structure of the page based on data that is stored in the metamodel base. The system is in this sense similar to Oracle Application Express ™ development environment [12], which stores specifications of entire applications in an Oracle ™ database. The meta-CASE system is currently form-based and does not allow users to specify metamodels and artifacts by using a visual language.

There are several fields of research, which will further extend the functionality of the meta-CASE system described in this paper. The author of the paper considers the following topics as his future work and possible sections of his Doctoral Thesis. These topics are divided into three groups: *basic research*, *empirical studies* and *extending the functionality of the system*.

Basic research, the results of which are relevant in case of all meta-CASE systems

- *Ontological analysis* of the meta-metamodel of our meta-CASE system to find out whether it is flexible and powerful enough compared to the meta-metamodels of other meta-CASE systems. One has to define the methodology of such analysis.
- *Comparing* other existing meta-CASE systems to the system described in this paper, outlining positive and negative aspects. For evaluation one can use the Analytical Hierarchy Process decision framework [13] and also define a decision model, which can be then used to compare meta-CASE systems.
- Defining a set of *design pattern languages* for creating web-based CASE systems and using the patterns as building blocks of CASE systems.

Empirical studies

- *Empirical studies*, where different CASE systems will be defined using the meta-CASE system. Based on the feedback left by the users of these CASE systems, there might be a need to adjust the meta-CASE system. Examples of the CASE systems in question can be a *patterns management system* [14] or a system enabling the *framework for EIS strategic analysis* [15]. The latter does not currently have a specialized CASE tool.

Extending the functionality of the system

- *Version management subsystem*, which allows the developers to manage different revisions of artifacts, to restore an older state of an artifact, and to merge different

revisions to a new artifact. This can include *managing changes in a metamodel* to support the evolution of information systems and CASE systems that are used to develop evolving information systems. A metamodel is an artifact as well and the system should be able to manage its revisions like with every other artifact.

- The system should allow developers to use *conversion rules*, which allow the system to display the artifacts in some alternate way (HTML webpage, XMI file etc.) or to generate program code, tests, documentation etc., based on the artifacts. The latter is an actual problem in Model Driven Development context where developers often leave the model aside to focus on writing program code; the added code is later left unsynchronized with the model.
- *Interconnection between different metamodels* (and therefore, CASE tools) by linking different artifacts. These artifacts can be based on the same metamodel or several different ones.
- Using *templates* in the query subsystem to simplify the composition of queries. There can be a number of base queries. For each of these base queries one can create a template, which consists of a constant part and a variable part. For creating a particular query an administrator can select a template, value the parameters and the system will generate a query based on this information. Many of these templates can be created based on generalized integrity constraints that one can use in conceptual data models. For instance, one can create a query template based on a multiplicity constraint [Parent]-1-----1.*-[Child] to check whether each parent has at least one child.
- Managing metamodels and artifacts by using a diagramming environment.
- Further development of the query subsystem for administrators to be able to define queries to check the *completeness and consistency of metamodels* (CASE systems). These queries can be run based on the system catalog of the DBMS.

The most important keywords regarding the meta-CASE system described in this paper are “web-based” and “database-based”. Using web-based systems can create more opportunities for group collaboration between and within development teams of information systems of globalised and networked organizations. Such web-based systems can also be used as the basis of communities that are interested in the study and use of a particular modeling language. Database-based systems provide built-in functionality of the DBMS as well as quite mature extension mechanism to simplify the development of systems.

The author of this paper has not yet found any other meta-CASE system which while being web-based and database-based itself, allows creating web-based and database-based CASE systems. There are many existing meta-CASE tools (e.g. MetaEdit+, MetaMOOSE [16], Pounamu [17]), which provide much more functionality than the meta-CASE system described in this paper. However, they all use a desktop application as a modeling tool, which requires the user to install and update the program. There are some web-based extensions, for example in Pounamu, but they focus more on using the metamodels (CASE systems) over the web instead of developing them.

There are many new modeling languages developed nowadays. However, many of them lack a development environment, which allows creating models using the language. Thus, a demand of this kind of environment as described in this paper, still exists.

Acknowledgments. This research was supported by the Department of Informatics in Tallinn University of Technology and the Doctoral Studies and Internationalisation Programme DoRa.

References

1. Sgirka, R.: The Design and Prototyping of Meta-CASE Analysis Environment (in Estonian). Master's Thesis. Tallinn University of Technology, Tallinn, Estonia (2008)
2. Eessaar, E., Sgirka, R.: A Database-Based and Web-Based Meta-CASE System. In: International Conference on Systems, Computing Sciences and Software Engineering (SCSS 08) (2009) (in press)
3. Eessaar, E., Sgirka, R.: A SQL Database Based Meta-CASE System and its Query Subsystem. In: International Conference on Systems, Computing Sciences and Software Engineering (SCSS 09) (2009) (to appear)
4. Delen, D., Dalal, N.P., Benjamin, P.C.: Integrated modeling: the key to holistic understanding of the enterprise. Commun. ACM 48, 4, pp.107--112 (2005)
5. Melton, J.: ISO/IEC 9075-2:2003 (E) Information technology - Database languages - SQL - Part 2: Foundation (SQL/Foundation) (2003)
6. Fugetta, A.: A Classification of CASE Technology. Computer 26, December 1993, 25--38 (1993)
7. Dittrich, K., Tombros, D., Geppert A.: Databases in Software Engineering: A Roadmap. The Future of Software Engineering. In: 22nd International Conference on Software Engineering, pp. 293--302. ACM New York (2000)
8. MetaEdit+ Evaluation Tutorial, *http://www.metacase.com/support/45/manuals/evaltut/et-Title.html*
9. Tolvanen, J., Rossi, M.: MetaEdit+: defining and using domain specific modeling languages and code generators. In: The International Conference on Object Oriented Programming, Systems, Languages and Applications 2003, pp. 92--93 (2003)
10. PostgreSQL 8.3.9 Documentation, *http://www.postgresql.org/docs/8.3/static/*
11. Choinzon, M., Ueda, Y.: Design Defects in Object Oriented Designs Using Design Metrics. In: 7th Joint Conference on Knowledge-Based Software Engineering, pp. 61--72 (2006)
12. Oracle Application Express 3.1 Documentation.
13. Saaty, T.L.: The Analytic Hierarchy Process: Planning, Priority Setting, Resource Allocation. McGraw-Hill (1980)
14. Zub, D., Eessaar, E.: Pattern-Based Usability Evaluation of E-Learning Systems. In: International Conference on Engineering Education, Instructional Technology, Assessment, and E-learning (EIAE 07), pp. 117--122 (2008)
15. Roost, M., Kuusik, R., Rava, K., Veskioja, T.: Enterprise Information System Strategic Analysis and Development: Forming Information System Development Space in an Enterprise. In: International Conference on Computational Intelligence. pp. 215--219 (2004)
16. Ferguson, R., Parrington, N., Dunne, P., Archibald, J., Thompson, J.: MetaMOOSE – an object-oriented framework for the construction of CASE tools. In: The 1st International Symposium on Constructing Software Engineering Tools (1999)
17. Zhu, N., Grundy, J., Hosking, J.: Pounamu: A Meta-Tool for Multi-View Visual Language Environment Construction. In: 2004 IEEE Symposium on Visual Languages - Human Centric Computing, pp.254--256 (2004)