Computing and Informatics, Vol. 31, 2012, 1045–1079

TRANSFORMATIONS OF CHECK CONSTRAINT PIM SPECIFICATIONS

Nikola Obrenović

Telvent DMS, Llc.
Narodnog Fronta 25 A–D
21000 Novi Sad, Serbia
e-mail: nikola.obrenovic@telventdms.com

Aleksandar Popović

University of Montenegro
Faculty of Science
Džordža Vašingtona bb
81000 Podgorica, Montenegro
e-mail: aleksandarp@rc.pmf.ac.me

Slavica Aleksić, Ivan Luković

University of Novi Sad
Faculty of Technical Sciences
Trg D. Obradovića 6
21000 Novi Sad, Serbia
e-mail: {slavica, ivan}@uns.ac.rs

Communicated by Ján Kollár

Abstract. Platform independent modeling of information systems and generation of their prototypes play an important role in software development process. However, not all tasks in this process have been covered yet, i.e. not all pieces of an information system can be designed using platform independent artifacts that are later transformable into the executable code. One of the examples is modeling of database check constraints, for which there is a lack of appropriate mechanisms to formally specify them on a platform independent level. In order to provide formal specification of check constraints at platform independent level, we developed a domain specific language and embedded it into a tool for platform independent design and automated prototyping of information systems, named *Integrated Information Systems CASE* (IIS*Case). In this paper, we present algorithms for transformation of check constraints specified at the platform independent level into the relational data model, and further transformation into the executable SQL/DDL code for several standard and commercial platforms: ANSI SQL-2003, Oracle 9i and 10g, and MS SQL Server 2000 and 2008. We have also implemented these algorithms in IIS*Case as a part of the process of generation of relational database schema.

Keywords: Check constraint, platform independent model, model-driven architecture, model-to-model transformation, model-to-code transformation, SQL/DDL generation

Mathematics Subject Classification 2010: 68U35, 68P15

1 INTRODUCTION

Data constraints are purposed to preserve data integrity as well as impose business rules over data in databases. Thus, they enrich database schemas with semantics and relieve the overlaying business applications the burden of maintaining data integrity. Database management systems (DBMS) that are based on the relational data model allow definition of various data constraints, usually providing different declarative and procedural mechanisms for that purpose. On the other hand, there is a need for platform independent specifications of constraints since the platform independent models (PIM) allow designers to specify constraints of the system being modeled regardless of the particular platform and implementation technologies. Such models provide easier maintainability and portability of database constraints between different implementation platforms compared to constraint specifications created for a particular platform only.

Our experience from a number of industrial or academic projects of developing information systems shows that obscure modeling techniques are used to design database constraint specifications. In some cases, there is even a lack of any kind of specifications of database constraints. It is particularly the case with check constraints, which represent an important concept for specifying business rules in database design, since they are frequently utilized in database design projects from various application domains, and cannot be compensated by other types of constraints at the abstraction level of data models. We believe that one of the reasons for such circumstances is the lack of domain specific, platform independent and tool-supported specification techniques for such constraints. Therefore, we have directed our research towards platform independent modeling of check constraints and automatic generation of executable code for several commercially available relational DBMSs (RDBMS).

In our previous research we developed a domain specific language (DSL) aimed at specifying check constraints at the level of PIMs. Also, we implemented a visual editor for formal specification of check constraints in a guided way. The editor and DSL are presented in detail in [1]. In this work, we extend our prior efforts by creating algorithms providing model-to-model transformations of check constraint PIM specifications into relational data model and model-to-code transformations into the executable SQL/DDL code.

We also embedded the proposed algorithms into the Integrated Information Systems CASE tool, IIS*Case for short. It is a tool for platform independent design of information systems and rapid generation of information system prototypes. Currently, the main features of IIS*Case are ([1]):

- conceptual modeling of database schemas, transaction programs, and business applications of an information system;
- automated generation of relational database subschemas in the 3^{rd} normal form (3NF);
- automated integration of independently designed subschemas into a unified database schema in the 3NF;
- automated generation of SQL/DDL code for ANSI SQL-2003 standard and various RDBMSs;
- conceptual design of common user-interface (UI) models; and
- automated generation of executable prototypes of business applications.

A PIM model of an information system in IIS*Case includes specifications of form types ([2, 3]). A form type concept is an abstraction of screen forms or documents that users utilize to communicate with the information system. By specifying form types, designers model the following information system specifications:

1. a database schema with its constraints, as it is presented in [3, 4],
2. functionality of business applications ([1]), and
3. a future user interface (UI) of business applications ([2, 5]).

An advantage of PIM of form types is that the end users are presented with a specification of the information system they can understand. This important feature of IIS*Case allows end users, software engineers and system architects to collaborate in the development of an information system, which makes IIS*Case suitable for End-User Development approach, as proposed in [6, 7]. Providing end users a possibility to participate in early stages of the software development may significantly raise the quality of information system specifications. In this way, user requirements can be met more accurately, with less effort, since frequent misunderstandings between the users and the software engineers can be avoided.

The model of an information system in IIS*Case can also be classified as a PIM in terms of Model-Driven Architecture (MDA) pattern ([8]), as it does not contain any implementation-specific details. Furthermore, the whole software development process in IIS*Case corresponds to MDA pattern, since PIM is firstly transformed via model-to-model transformations into a platform specific model (PSM), and further into the executable code by using model-to-code transformations. In [9], the application of the model driven software development (MDSD) principles in IIS*Case has been presented in detail.

In IIS*Case, the executable prototypes of business applications are completely generated from the specifications created at the level of PIM. If there is a need for a change in functionality of the prototype, it does not require long-lasting and manual programming work. Instead, the PIM specifications are modified and the prototype is generated again quickly. By this we address the problem of efficient software evolution, as considered in [10, 11].

Database schema development methodology and formal specification of database constraints supported by IIS*Case are out of the scope of this paper. They are discussed in [12] and [13], respectively. The generation of executable information system prototypes is described in detail in [5]. Further considerations about IIS*Case may be found in several authors' references, such as [14, 15].

IIS*Case allows design of various database constraints at the level of PIM, such as primary key, referential integrity, inverse referential integrity, null constraints, etc. This has been elaborated in several papers, e.g. [3, 16]. Because of the importance of check constrains in database design, our goal was to support PIM modeling of that kind of constraints. Therefore we have developed a DSL for defining check constraints at the level of PIM, which is presented in [1]. This DSL releases designers any burden about implementation details regarding relational data model and commercial RDBMSs, in accordance to the principles discussed in [17, 18]. Also, our DSL can be regarded as a meta-level language, with SQL being taken as a base level language according to [10, 11].

Consequently, we continue our efforts to provide complete lifecycle for platform independent and tool-supported modeling and implementation of check constraints. In this paper, we present algorithms for automatic transformation of check constraints specified at the level of PIMs of form types into relational data model, which is still implementation independent, and further transformation into the executable SQL/DDL code for several platforms: ANSI, Oracle and MS SQL Server. These algorithms have also been implemented in IIS*Case and tested in a selected case study.

The rest of the paper is organized as follows. In Section 2, we present the current state of the art in the design of check constraints at both academic and commercial level. In Section 3, a more detailed overview of the IIS*Case and DSL for check constraint is given. It contains details necessary to follow the main results presented in the paper. The model-to-model transformation of PIM specifications of check constraints into check constraints in relational data model is presented in Section 4. In Section 5 we present the model-to-code transformations into the executable SQL/DDL code for several RDBMSs. Conclusions and future work remarks are given in Section 6.

2 RELATED WORK

PIM design of database check constraints has been rarely explored both in academic and industrial environments. Demuth, Hussmann and Loecher in [19, 20] use the Object Constraint Language – OCL ([21]) for PIM modeling of the check constraints. They consider OCL as the natural way to design check constraints since they utilize UML models to design database schemas. On the contrary, we have opted for a user-oriented DSL for the same purpose since our approach and PIM model is entirely user-oriented. However, we share similar ideas when it comes to the automatic generation of the executable code of check constraints, where we pay special care to code optimization in order to obtain better performance in case of commercial RDBMSs.

Many commercial CASE tools allow PIM modeling of database schemas and information systems. We analyzed several CASE tools broadly utilized among software engineers: Oracle Designer [22], Sybase PowerDesigner [23], Enterprise Architect [24], IBM Rational Rose Data Modeler [25] and CA ERwin Data Modeler [26]. All of them provide entity-relationship (ER) data model for database design, but do not support the entire lifecycle of modeling and generation of check constraints. Oracle Designer and Sybase PowerDesigner allow specification of check constraints only by using SQL syntax in platform specific relational data model. On the other hand, in Enterprise Architect, check constraints are modeled at the PIM level, but they cannot span multiple relation schemes. On the contrary, IIS*Case does provide that. ERwin Data Modeler allows specification of the check constraints only at the level of attributes and generates executable code for them. Unfortunately, other types of check constraints are not supported. IBM Rational Rose Data Modeler offers the most advanced features of all analyzed tools for PIM design of check constraints. It provides SQL syntax for specification of column and table check constraints and in addition, it allows usage of the OCL for the definition of check constraints that span multiple relation schemes. However, it does not provide the transformations of constraints defined in OCL into the executable SQL code.

Cabot and Teniente claim in their survey presented in [27, 28], that modern CASE tools do not provide satisfactory modeling functionalities of check constraint at the PIM level nor appropriate automatic generation of the executable code. Consequently, the authors have set a number of goals which future MDSD tools should strive to: expressivity of PIM, efficiency of the generated code, technology-aware generation of the code, technological independence of model-to-model transformations and checking time of constraints determinable by the designer. A goal of our research is to improve IIS*Case to support all of them with a usage of a user-oriented DSL.

3 THE PIM OF THE CHECK CONSTRAINTS

At the level of PIMs, IIS*Case provides a number of concepts such as domains, attributes, user-defined functions, form types, etc. The definitions and examples of these concepts may be found in [2, 3], as well as other authors' references. In this section, we give an overview of the DSL for creating check constraints. By this, we present concepts necessary to create PIM specifications of check constraints. More details about PIM specifications of check constraints may be found in [1].

A concept of the domain is used to specify a set of allowed values of an attribute. It represents a kind of constraint present in all data models, named domain constraint. In IIS*Case, there are two types of domains: primitive and user-defined. Primitive domains correspond to the basic, built-in data types such as Boolean, integer, floating-point numbers, etc. User-defined domains are derived from primitive or previously created user-defined domains and inherit all their properties.

A user-defined domain is created by either inheriting an existing domain or grouping existing domains into a tuple or choice domain. Domain inheritance in IIS*Case is of the same nature as it is in widely used object-oriented general-purpose languages. In the following text, we refer to inheriting domains as child domains, while inherited domains are their parent domains.

Tuple and choice domains are classified as complex domains in IIS*Case. A tuple domain represents tuples of values where each value belongs to one of existing domains. On the other hand, a choice domain represents values where each value belongs to exactly one of the source domains.

To each attribute in IIS*Case, a domain specification must be assigned. By this, if the same domain is associated to more than one attribute, all values of all those attributes must be validated against the same domain constraint, regardless of different semantics expressed by such attributes.

Apart from data type and maximum data length, any domain specification may include a domain check constraint. It is specified by using our DSL and represents a logical expression over the $VALUE$ variable, which may be instantiated to any attribute value that is validated by the domain constraint. For example, a domain check constraint that allows only positive numeric values in a domain specification is specified as:

$$VALUE > 0.$$

If both parent and child domains have domain check constraint defined, the child domain inherits parent's domain check constraint and concatenates it to its own domain check constraint by means of AND logical operator.

The grammar for domain check constraints is formally specified in [1]. However, we repeat here its formal specification in order to give the reader a complete and precise overview of our DSL. The grammar is originally developed in the ANTRL notation ([29]). However, we present it in Figure 1 in the Extended Backus-Naur Form (EBNF) notation, for the sake of better readability.

```
Exp = Exp bin_operator Exp | un_operator Exp | Primary_Exp;
Primary_Exp = constant | 'VALUE'['.'  fieldName]
              | function_name '(' [Exp_List] ')' | '(' Exp ')';
Exp_List = Exp { ',' Exp_List};
```

Fig. 1. Specification of the grammar for domain check expressions

The list of standard operators that may replace bin_operator or un_operator includes the following ones:

- additive (+, −);
- multiplicative (*, /);
- comparison (<, <=, >, =>);
- equivalence (==, !=);
- concatenation (||);
- Boolean (NOT, AND, OR, XOR, ⇒, ⇔);
- inclusion (IN); and
- pattern matching (LIKE).

All the operators and parentheses are introduced with the common meaning and priorities as it is a case in any form of SQL. Apart from introducing standard arithmetic, string, comparison and logical operators existing in all general-purpose languages, we also decided to introduce the operators LIKE and IN, which are common in various forms of SQL. In this way, the language for check expressions becomes more problem oriented. The grammar in Figure 1 also provides calls of functions specified at the PIM level in IIS*Case.

The grammar in Figure 1 provides the use of constants in check expressions. The common rules for specification and interpretation of constants are applied, and accordingly we do not describe them in more detail.

The only variable symbol allowed in domain check expression is the *VALUE* variable. If the check constraint is associated with a tuple or choice domain, the *VALUE* variable may be qualified by the attribute name of a tuple or choice member. In such a case, the check constraint is validated only against the specified tuple or choice member instead of the whole domain value.

In IIS*Case, database schemas are designed to satisfy the universal relation schema assumption (URSA). Therefore, each attribute is identified only by its name throughout the whole database schema and there could not be two attributes with the same name and different semantics, or with the different names and the same semantics. By this, each attribute is specified independently of any form type or relation scheme it will belong to.

We may assign an attribute check constraint to each attribute specification. It is a logical expression over a variable representing the attribute name. In this way, an attribute check constraint must be satisfied by any attribute value, regardless of the form types in which the attribute is included. For example, an attribute check constraint that allows only numeric values from interval [5,10] to be assigned to attribute representing student's average grade, named *AvgGrade*, is specified as

$$AvgGrade >= 5 \text{ AND } AvgGrade <= 10.$$

A formal specification of the grammar for attribute check expressions, as defined in [1], is shown in Figure 2, in EBNF notation. With attName we denote a name of the attribute which a check constraint is assigned to. Expression operands have the same meaning as in the grammar specification for domain check constraints from Figure 1.

```
Exp = Exp bin_operator Exp | un_operator Exp | Primary_Exp;
Primary_Exp = constant | attName['.'  fieldName]
              | function_name '(' [Exp_List] ')' | '(' Exp ')';
Exp_List = Exp { ',' Exp_List};
```

Fig. 2. Specification of the grammar for attribute check expressions

In IIS*Case, a form type is a tree of component types ([2]). A component type is an abstraction that corresponds to real-life entity classes. A form type is a formal specification of a screen form, i.e. document used in an information system. In Figure 3 a form type FACULTY DEPARTMENTS is presented in its simplified form.

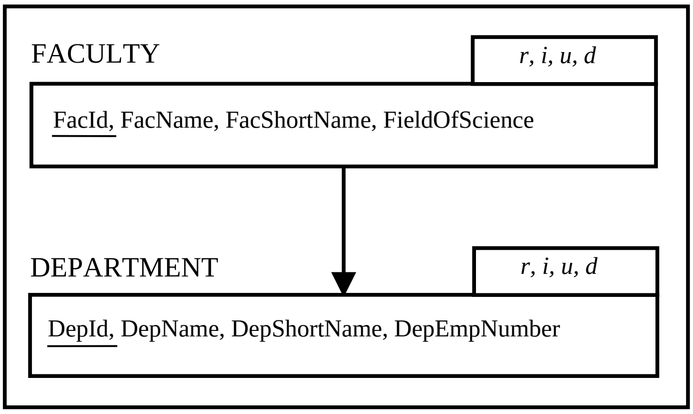

FACULTY
r, i, u, d
FacId, FacName, FacShortName, FieldOfScience
DEPARTMENT
r, i, u, d
DepId, DepName, DepShortName, DepEmpNumber

Fig. 3. Form type FACULTY DEPARTMENTS

It contains FACULTY and DEPARTMENT component types. FACULTY component type is a simple model of a faculty, which is modeled with faculty's identifier, name, abbreviated name and employee count attributes. DEPARTMENT component type is an abstraction of departments belonging to faculties and it is specified here with attributes that represent identifier, name, abbreviated name and employee count of a department. The FACULTY DEPARTMENTS form type specifies a master-detail UI form of university information system where information about a faculty is displayed in master section and information about departments belonging to the displayed faculty are viewed in detail section of the UI form.

Each component type contains a set of attributes and a set of constraints. Attributes underlined with solid lines are the keys of the component types. For each component type, a set of allowable database operations is declared. It is shown in the upper-right corner of the rectangle representing a component type. In this example, for both component types, the following database operations are allowed: r–read, i–insert, u–update and d–delete.

In [1], component type check constraints have been introduced. Informally, a component type check constraint is a logical expression that is specified at the level of a component type. It comprises variables that may reference the component type attributes, as well as the attributes belonging to the superordinated component types of the same form type. E.g., if identifier of a department has to be a positive numeric value, the following check constraint is specified at the level of DEPARTMENT component type from Figure 3:

$$\mathit{DepId} > 0.$$

At the level of the same component type, another constraint is specified to preserve that the number of employees at each faculty is greater than the number of employees at each of its belonging departments:

$$\mathit{FacEmpCount} > \mathit{DepEmpCount}.$$

A formal specification of the grammar for component type check constraints, as defined in [1], is shown in Figure 4, in EBNF notation. With cmpAttName we denote a name of the component type attribute referenced in a check constraint. Expression operands have the same meaning as in the grammar specification for domain check constraints from Figure 1.

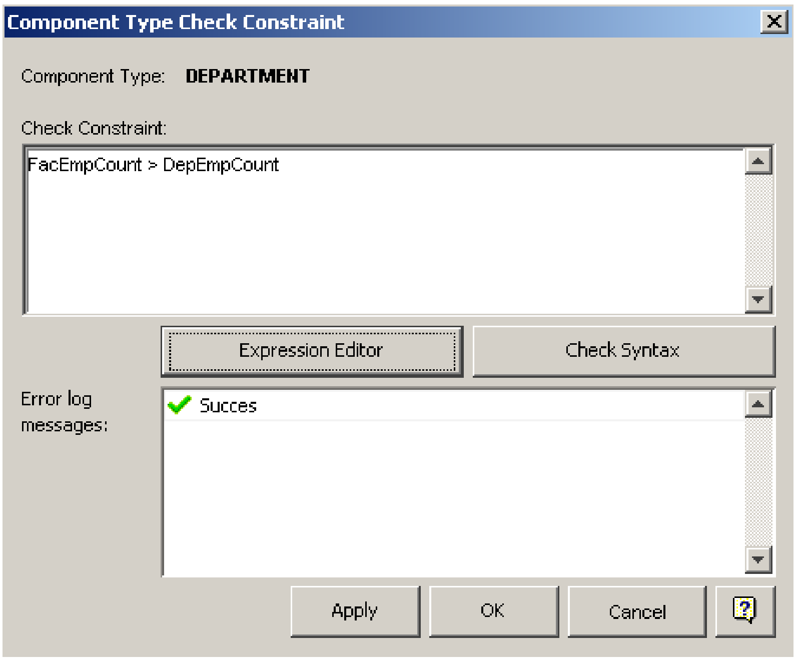
```
Exp = Exp bin_operator Exp | un_operator Exp | Primary_Exp;
Primary_Exp = constant | cmpAttName['.'  fieldName]
              | function_name '(' [Exp_List] ')' | '(' Exp ')';
Exp_List = Exp { ',' Exp_List};
```

Fig. 4. Specification of the grammar for component type check expressions

There are two ways to specify check constraints at the level of all three concepts in IIS*Case:

1. free-form and
2. guided way.

The IIS*Case form for free-form specification of component type check constraints is shown in Figure 5. By this, check constraints are specified directly by typing the logical expression, which is a more suitable option for more experienced users of IIS*Case.



```
FacEmpCount > DepEmpCount
```

```
✓ Succes
```

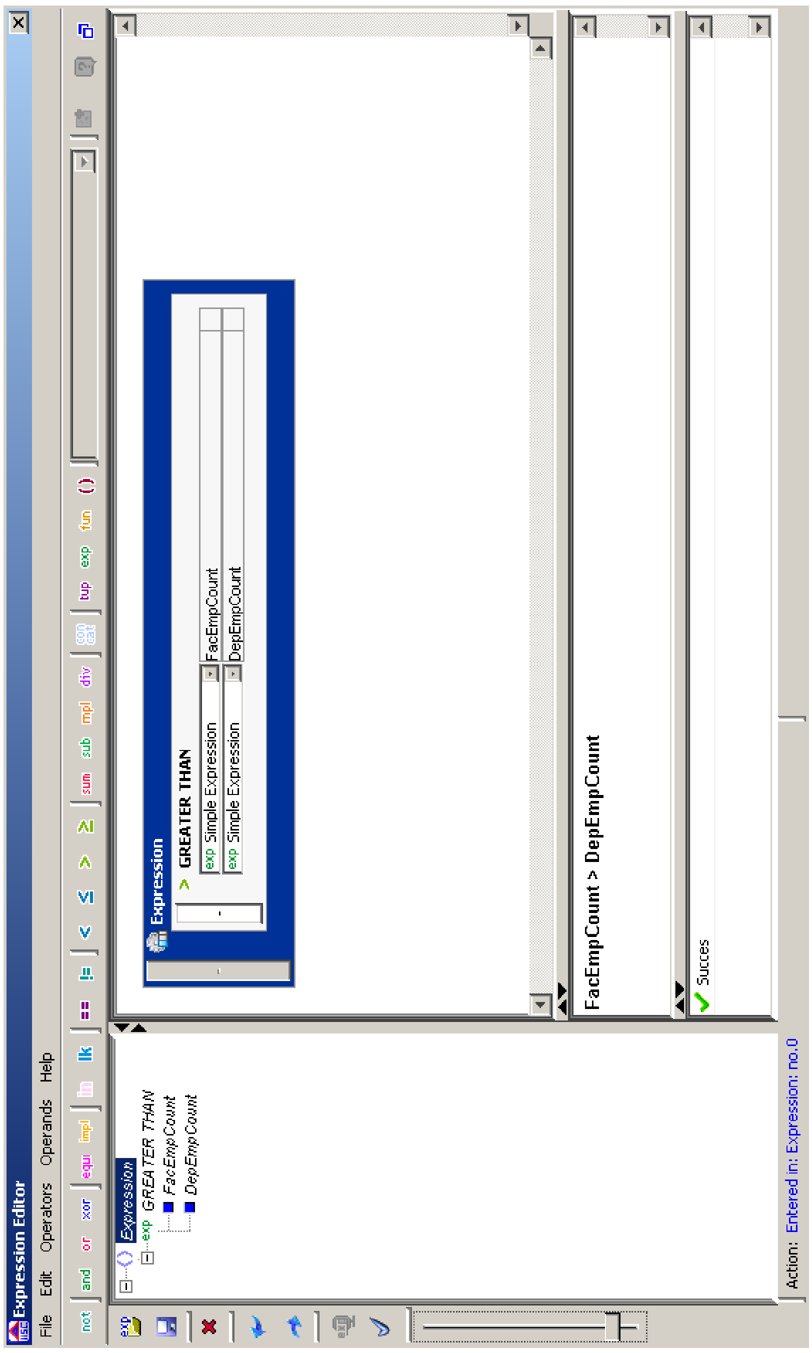
Fig. 5. A form for free-form specification of check constraints

A guided way for specification of component type check constraints is provided by the visually-oriented Expression Editor tool. The main form of Expression Editor is shown in Figure 6.

We believe that Expression Editor is more suitable option for less experienced users or users who are supposed to learn the language by using the editor. More details about Expression Editor may be found in [1].

We believe that the basic knowledge of mathematical logic or even SQL logical expressions is enough to easily learn and use the DSL for check constraints. This allows the users of SQL to learn our DSL with a minimum of effort.

Besides, we intend to formally evaluate the usability of our DSL for check constraints by using the Cognitive Dimensions Framework ([30]), as part of our future work. One example of such evaluation may be found in [31]. To perform the evaluation, we need an intensive communication with end users of IIS*Case.



Expression
exp GREATER THAN
FacEmpCount
DepEmpCount

Expression
> GREATER THAN
exp Simple Expression FacEmpCount
exp Simple Expression DepEmpCount

FacEmpCount > DepEmpCount

Succes

Action: Entered in: Expression: no.0

Fig. 6. The Expression Editor tool for visual specification of check constraints

4 TRANSFORMATIONS OF THE CHECK CONSTRAINT SPECIFICATIONS

In IIS*Case, PIM specifications of database-related modeling concepts are firstly transformed into relational data model which is still implementation independent. All the transformations of PIM to PSM specifications provided by IIS*Case are formalized through a vast number of algorithms, presented in previous authors' references, such as [2, 16, 3]. They have been specified at the rigor of mathematical formality, with proofs of their correctness included. The algorithms always produce the same output (a relational database schema, as an example) from the same input – a given set of form types. Therefore, we may consider our algorithms as transformations in the course of MDSD approaches.

The PIM to PSM transformations in IIS*Case are part of the process of database schema generation. A goal of the research presented in this paper was to enrich the database schema generation process by taking into account check constraints. Therefore, we have created transformations of PIM specifications of check constraints into the relational data model. In this paper we present these PIM to PSM transformations.

Each check constraint is defined in the scope of a PIM concept, which can be a domain, attribute or component type. Also, each check constraint includes a logical expression used for validation of the constraint. Both the PIM concept and the logical expression have to be transformed into the relational data model. First, we present the transformations of the concepts check constraints are specified upon, and then the transformations of the check constraint logical expressions.

The relational data model recognizes both concepts of the domain and the attribute. They have the same semantics in various data models, as well as in relational data model and PIM utilized by IIS*Case. Hence, the concepts which domain and attribute check constraints are specified upon remain the same through the transformations into the relational data model.

On the other hand, component types, as well as their check constraints, have to be transformed using model-to-model transformations.

The PIM of form types bears, among other constraints, functional dependencies of the modeled information system in such a way that all attributes of component types depend on their keys. By using a modified Bernstein's synthesis algorithm ([32, 33]), functional dependencies are transformed into a relational database schema. Accordingly, the set of all form types is mapped to a set of relation schemes using model-to-model transformations. This is thoroughly elaborated in [2, 3]. In the following text, we denote the set of all relation schemes obtained from the set of all form types with S.

The mapping between form types and relation schemes is many-to-many. Therefore, each form type corresponds to a set of relation schemes. We denote this set as S_F. In this way, form types represent user views onto the corresponding relation schemes. Through such views, users can retrieve and manipulate data ([2]). Algorithms for determining a set of relation schemes that a form type corresponds to, have been presented in detail in [2].

Example 1. We consider the form type UNIVERSITY ORGANIZATION given in Figure 7.

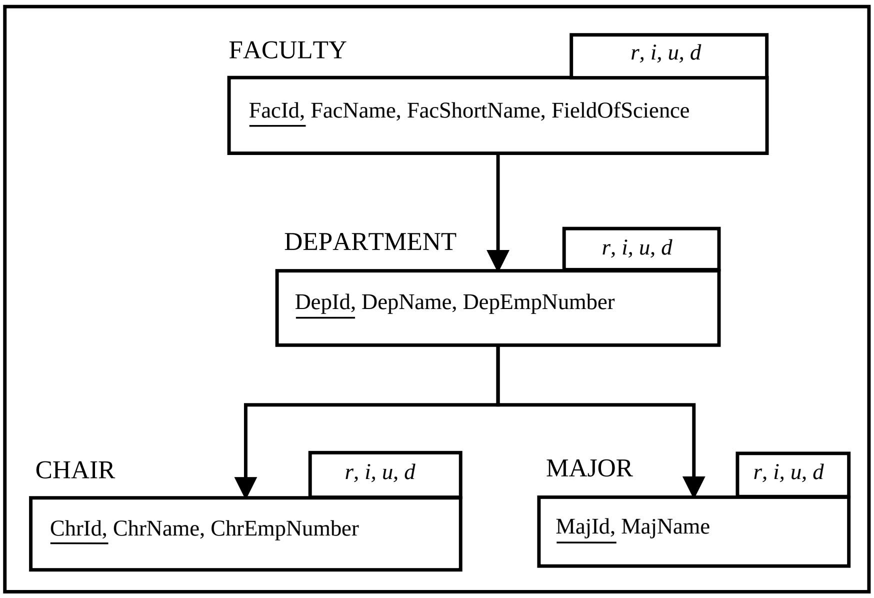

FACULTY
r, i, u, d
FacId, FacName, FacShortName, FieldOfScience
DEPARTMENT
r, i, u, d
DepId, DepName, DepEmpNumber
CHAIR
r, i, u, d
ChrId, ChrName, ChrEmpNumber
MAJOR
r, i, u, d
MajId, MajName

Fig. 7. UNIVERSITY ORGANIZATION form type

It consists of four component types: FACULTY, DEPARTMENT, CHAIR and MAJOR, where each component type models the same-named entities belonging to organizational parts of a university. Just for the sake of simplicity, let us suppose that it is the only form type of the modeled information system. The algorithms for transformation of a set of form types generate the following relation schemes given in the form $N(R, K)$, where N is the name of the relation scheme, R is the set of attributes and K is the set of keys:

- $Faculty(\{FacId, FacName, FacShortName, FieldOfScience\}, \{FacId\})$;
- $Department(\{FacId, DepId, DepName, DepEmpNumber\},$ $\{FacId + DepId\})$;
- $Chair(\{FacId, DepId, ChrId, ChrName, ChrEmpNumber\},$ $\{FacId + DepId + ChrId\})$; and
- $Major(\{FacId, DepId, MajId, MajName\}, \{FacId + DepId + MajId\})$.

Since UNIVERSITY ORGANIZATION is the only form type, its set of corresponding relation schemes S_F contains all generated relation schemes.

By extending the main idea presented in [2], we have developed an algorithm for model-to-model transformation of component type check constraints into relational check constraints. According to the algorithm, a component type check constraint may be transformed into a check constraint belonging to one relation scheme, but also it may span multiple relation schemes in the relational database schema. If a check constraint spans multiple relation schemes, it becomes an interrelation constraint. We have found the similar feature present in [19, 20] only, but not in popular or widely used CASE tools.

Relation schemes spanned by a check constraint are further denoted as context relation schemes of the constraint. We also denote a set of all context relation schemes of a check constraint as S_{CC}. For each check constraint, S_{CC} must preserve a lossless join in order to preserve semantics of the PIM specified check constraint.

The rest of this section is organized as follows. In order to introduce the algorithm, we present in Section 4.1 the notions of lossless join of relation schemes, a closure graph and a set of minimal nodes of a form type. The algorithm for model-to-model transformation of component type check constraints into relational check constraints is introduced in Section 4.2. Transformations of check constraint logical expressions into its conjunctive and disjunctive normal forms are given in Section 4.3.

4.1 Preliminary Notions

We assume that a reader is familiar with the relational data model at the level of [34]. However, to improve the readability, in the following text we present some well-known notions of relational data model. A relational database schema can be formally represented by a single relation scheme, called the universal relation scheme. As argued in [34], as well as in [35] and [33], such scheme is prone to the update anomalies because of the redundant information contained in tuples of the universal relation. In order to prevent update anomalies, universal relation scheme is decomposed into set of smaller relation schemes whose relations do not contain redundant data. Thereby, we obtain a relational database schema.

The decomposition is not performed in an arbitrary manner. The relations obtained by the decomposition need to produce back the tuples of the universal relation when natural join operation is applied over them. The relations that satisfy this condition possess the lossless join property. The word "lossless" in this case does not denote the loss of the tuples of the universal relation, but the loss of information, since the natural join operation can create more tuples than there are in the universal relation.

Formally, the lossless join property is defined in [34] as follows. A decomposition $D = \{N_1(R_1, K_1), N_2(R_2, K_2), \ldots, N_m(R_m, K_m)\}$ of a relation scheme $N(R, K)$ has the lossless join property with respect to the set of functional dependencies Γ on R if, for every relation r of $N(R, K)$ that satisfies Γ, the following holds, where $\bowtie$ is the natural join of all the relations in D:

$$\bowtie (R_1(r), \ldots, R_m(r)) = r$$

, where $R_i(r)$ denotes the projection of relation r to the set of attributes R_i.

A closure of a set of attributes X is the set of attributes that functionally depend on X, with respect to a set of functional dependencies. We also say that X is closed by its closure or that X functionally determines its closure. Formally, the closure of a set of attribute X, with respect to a set of functional dependencies Γ, is a set of attributes

$$X^+_\Gamma = A|X \rightarrow A \in \Gamma^+,$$

where Γ^+ is the set of all functional dependencies that can be inferred from Γ.

A closure graph is a directed graph where each node represents a relation scheme, generated by the synthesis algorithm. Each directed edge represents the fact that the set of attributes of superior relation scheme closes the set of attributes of subordinated relation scheme, with respect to the set of functional dependencies inferred from the set of all form types. Normally, each edge corresponds to a referential integrity constraint between the connected nodes, since a proper or improper subset of a key of a subordinated node is propagated as a foreign key into a superior node. In the following example we illustrate the notion of a closure graph.

Example 2. For the form type UNIVERSITY ORGANIZATION from Figure 7, closure graph generated by IIS*Case is given in Figure 8. Relation schemes *Major* and *Chair* are linked to *Department* since each of them closes *Department* relation scheme. Likewise, *Department* closes *Faculty* relation scheme.

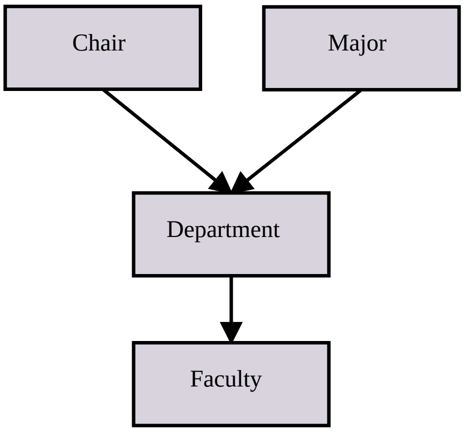


Fig. 8. Closure graph of the UNIVERSITY ORGANIZATION form type

In this example, graph edges correspond to the following referential integrity constraints:

- $Department[FacId] \subseteq Faculty[FacId]$;
- $Chair[(FacId, DepId)] \subseteq Department[(FacId, DepId)]$; and
- $Major[(FacId, DepId)] \subseteq Department[(FacId, DepId)]$.

The closure graph is introduced and discussed in [2, 16]. In [33], the closure graph is defined as graph (S, φ), where a set of relation schemes $S = N_i(R_i, K_i)|i = 1, \ldots, n$ is given, and a relation φ is defined in S^2 as

$$\begin{aligned}\varphi \;=\; & (N_i(R_i, K_i), N_j(R_j, K_j)) \in S^2 | Rj \subset (R_i)^+_\Gamma \\ & \wedge (\forall N_k(R_k, K_k) \in (S \backslash \{N_i(R_i, K_i), N_j(R_j, K_j)\}))(R_j \not\subset (R_k)^+_\Gamma \vee R_k \not\subset (R_i)^+_\Gamma),\end{aligned}$$

where $\Gamma = \{X \to A | (\exists N_i(R_i, K_i) \in S)(A \in R_i \backslash X \wedge X \in K_i)\}$.

In our proposed algorithm, we utilize the following two facts about the closure graph:

1. the set of attributes of a node closes sets of attributes of all subordinated nodes, and
2. two nodes connected in the closure graph represent two relation schemes preserving the lossless join.

For the sake of simplicity in the following text, we say that relation scheme $N_i(R_i, K_i)$ closes relation scheme $N_j(R_j, K_j)$ if the set of attributes R_i closes the set of attributes R_j. Also, we say that a relation scheme $N(R, K)$ closes a set of attributes X if the set of attributes R closes X. The set of minimal nodes of a form type is a minimal subset of S_F, such that belonging relation schemes close all other relation schemes in S_F. We denote this set as SMN_F. The SMN_F is minimal if there is no smaller subset of S_F with the same property. Formally, the set SMN_F is defined as

$$\begin{aligned}SMN_F \;=\; & \{N_i(R_i, K_i) \in S_F \mid \forall (N_j(R_j, K_j) \in (S_F \backslash N_i = (R_i, K_i))), \\ & (N_j(R_j, K_j), N_i(R_i, K_i)) \notin \varphi)\},\end{aligned}$$

where φ is the relation of the closure graph (S,φ). The algorithm for generation of SMN_F is given in [2].

In Example 1, relation schemes Major and Chair are the minimal nodes of the UNIVERSITY ORGANIZATION form type.

4.2 Algorithm for Transformation of Component Type Check Constraint

The algorithm transforms a component type check constraint into a relational check constraint. It is applied for all component type check constraints specified at the level of PIM of form types. Inputs of the algorithm are the set of relation schemes corresponding to form type S_F, the set of minimal nodes of form type SMN_F, closure graph (S, φ), and the check constraint logical expression LE_{CC}. The result of the algorithm is a set of context relation schemes of the modeled constraint S_{CC} that preserves a lossless join. Since each component type check constraint is defined in the scope of a form type, S_{CC} must be a subset of S_F. The algorithm consists of the following three steps.

Step 1. The first step is to find a relation scheme in S_F, which closes all attributes referenced by LE_{CC} such that none of its subordinated relation schemes in the closure graph has the same property. This relation scheme is named the minimal node of a check constraint. The selection of the minimal node of a check constraint is mandatory to preserve the lossless join of S_{CC}, as proven in [33].

The union of closures of all schemes from SMN_F contains all form type attributes and consequently must contain all attributes referenced by LE_{CC}. Hence, the algorithm finds the minimal node of a check constraint by checking all relation schemes of SMN_F and searching for a relation scheme that closes all attributes of LE_{CC}. When such relation scheme is found, we further check all its subordinated relation schemes in the closure graph for the same property and the step repeats recursively until none of the subordinated relation schemes satisfies the condition. The purpose of these steps is to skip all unnecessary relation schemes from S_F while preserving the lossless join of the S_{CC}.

Step 2. The next step is to determine all relation schemes that are referenced by the check constraint. It is done by finding relation schemes from S_F that have a non-empty intersection with the set of attributes used in LE_{CC}. These relation schemes are called the relevant schemes or relevant nodes.

Step 3. Finally, we find paths in the closure graph from the minimal node of the check constraint to the relevant nodes. These paths provide the lossless join between the minimal node and the relevant nodes, i.e. the lossless join between all relation schemes from S_{CC}. The nodes belonging to the paths are called the necessary nodes. Since the minimal node closes all attributes of LE_{CC}, the relevant nodes are subordinated to the minimal node in the closure graph. Hence, the search for necessary nodes is done by the modified breath-first search in the closure graph starting from the minimal node of a check constraint and moving in the direction of graph edges. At each level of the closure graph, nodes that lead to the relevant nodes are denoted as candidates for the necessary nodes. From the candidates, the algorithm chooses their smallest subset that leads to all relevant nodes and denotes them as the necessary nodes. Only the nodes subordinated to the selected ones are included in the subsequent search. The search terminates when it reaches all relevant nodes.

The output of the algorithm S_{CC} is the union of the minimal node of check constraint, the set of relevant nodes and the set of necessary nodes. The semantics of the modeled check constraint is preserved by the lossless join of S_{CC}. By this, we guarantee that the check constraint logical expression is satisfied by the natural join of relations over the schemes from S_{CC}.

For a better understanding how our algorithm works, we give one simple example in Example 3 and a bit more complex example in Example 4.

Example 3. We specify a check constraint in the form type UNIVERSITY ORGANIZATION from Example 1 with the following meaning: *"The number of employees of a department has to be greater than the number of employees of any belonging chair."*

At the PIM level, this check constraint is specified in the Chair component type as:

$$\mathit{DepEmpNumber} > \mathit{ChrEmpNumber}.$$

Our algorithm starts with finding the minimal nodes of the check constraint. In this case, the relation scheme *Chair* has to be the minimal node since

1. it is also the minimal node of the UNIVERSITY ORGANIZATION form type,
2. it closes all attributes referenced by the check constraint logical expression, and
3. there are no subordinated schemes whose closure contains all attributes from the check constraint.

Moreover, the only relevant relation scheme is *Department*, since together with *Chair* it contains all needed attributes. Since these two relation schemes are already connected in the closure graph, there is no need to find paths between them, and the relation schemes *Chair* and *Department* are the result of the algorithm.

Example 4. In the score of the form type UNIVERSITY ORGANIZATION from Example 1, we define another check constraint that imposes the following business rule: *"If the field of science of a faculty is "Engineering", the identifier of each major at the faculty is between 101 and 1000."*

This could be a business rule of a university information system. It is specified in the Major component type as

$$(\mathit{FieldOfScience} == \mathit{'Engineering'}) \Rightarrow (\mathit{MajID} >= 101 \wedge \mathit{MajID} <= 1000).$$

By using the same criteria as in Example 3, the algorithm denotes *Major* as the minimal relation scheme of the check constraint and *Faculty* as the relevant. By doing the breath-first search along the closure graph edges from Figure 8, starting from the *Major* node with *Faculty* as the goal, the algorithm also finds the *Department* scheme and adds it to the resulting set of schemes.

The algorithm is presented in Figure 9, in a form of pseudo-code. TransformCheckConstraint is the main process just encompassing the calls of three processes, where each of them corresponds to exactly one algorithm step. The first two processes are also presented in a form of pseudo-code in Figures 10 and 12, while the third one is presented in [2], in detail. In the algorithm from Figure 9, all the variables except MN_{CC}, SRN_{CC} and SNN_{CC} have the same meaning as defined earlier

```
PROCESS PROCESS TransformCheckConstraint
                    (I($S_F, (S, \varphi), SMN_F, LE_{CC}$), O($S_{CC}$), IO( ))
BEGIN TransformCheckConstraint
    CALL CheckConstraintMinimalNode ($S_F, SMN_F, (S, \varphi), LE_{CC}, MN_{CC}$)
    CALL CheckConstraintRelevantNodes ($S_F, LE_{CC}, SRN_{CC}$)
    CALL CheckConstraintNecessaryNodes
                    ($S_F, (S, \varphi), MN_{CC}, SRN_{CC}, SNN_{CC}$)
    SET $S_{CC} \leftarrow MN_{CC} \cup SRN_{CC} \cup SNN_{CC}$
END PROCESS TransformCheckConstraint
```

Fig. 9. Algorithm for transformation of a component type check constraint

in the text. MN_{CC} denotes the minimal node of the check constraint, while SRN_{CC} and SNN_{CC} denote sets of relevant and necessary nodes, respectively.

The process CheckConstraintMinimalNode from Figure 10 searches for the minimal node of a check constraint. All variables used in the pseudo-code, except $Attr(LE_{CC})$ and Γ, are defined earlier in the text. With $Attr(LE_{CC})$, we denote the set of attributes referenced by the check constraint. The set of functional dependencies inferred from all form types of the modeled information system is denoted with Γ.

To elaborate the correctness of the CheckConstraintMinimalNode process, we outline the main steps of relational database schema generation in IIS*Case. The main steps of transformation of a set of form types into a set of relation schemes are inferring functional dependencies from the form types and applying the modified Bernstein's synthesis algorithm over them, as presented in [33], in detail. Thereby, the transformation infers at least one functional dependency from each component type. Since a form type is a tree structure of component types, the left hand side of such functional dependency is a union of keys of component types belonging to the path from the root component type to the component type from which functional dependency is inferred.

Let us observe a set of relation schemes produced by the modified synthesis algorithm. For each relation scheme from this set, the algorithm provides that it corresponds to at least one component type of a form type. We call it a corresponding component type for the relation scheme. Furthermore, it holds for each key of such relation scheme and its corresponding component type that the relation scheme key is a union of the keys of all component types belonging to the path from the root component type to the corresponding component type. Also, the algorithm provides that for each component type of a form type there will be a relation scheme whose keys close the set of attributes of the component type, as well as all its superordinated component types in the form type tree.

A component type check constraint can only reference attributes of the component type it is defined at, and possibly attributes of the superordinated component types in the form type tree. By the previous considerations, we conclude that there is always at least one relation scheme that closes all attributes referenced by the

```
PROCESS CheckConstraintMinimalNode
                    (I($S_F, SMN_F, (S, \varphi), LE_{CC}$), O($MN_{CC}$), IO( ))
BEGIN PROCESS CheckConstraintMinimalNode
  SET $Cand \leftarrow SMN_F$
  DO FindMinimalNode
    SET $Found \leftarrow FALSE$
    DO CheckCandidates ($\forall N_i(R_i, K_i) \in Cand$)
      IF ($Attr(LE_{CC}) \subseteq (R_i)^+_\Gamma$) THEN
        SET $MN_{CC} \leftarrow N_i(R_i, K_i)$
        SET $Found \leftarrow TRUE$
        EXIT CheckCandidates
      END IF
    END DO CheckCandidates
    IF $Found$ THEN
      CALL Subordinated($MN_{CC}, S_F, (S, \varphi), Cand$)
    ELSE
      EXIT FindMinimalNode
    END IF
  END DO FindMinimalNode
END PROCESS CheckConstraintMinimalNode
```

Fig. 10. Process for finding the minimal node of a check constraint

check constraint. By this, relation schemes satisfying this property are candidates for the minimal node of the check constraint. Furthermore, at least one of them has to belong to S_F since functional dependencies inferred from the form type are also used by the algorithm. In other words, such relation scheme belongs to the set of minimal nodes of the form type or has to be subordinated to one of them. By this conclusion we prove the correctness of the step for searching the minimal node of a check constraint.

In Figure 11, we present the Subordinated process purposed to select directly subordinated nodes in the closure graph. All variables used in the pseudo-code are defined previously in the text. The process selects only nodes that belong to S_F because they correspond to the form type in the scope of which the check constraint is defined. Correctness of the Subordinated process is a consequence of the definition of the closure graph given in Section 4.1. For each node $N_i(R_i, K_i)$ given as an input, each relation scheme from S_F is checked just once. Hereby, we estimate the complexity of the process as $O(n)$.

The loop FindMinimalNode of the process CheckConstraintMinimalNode from Figure 10 is executed only on a part of the closure graph (S, φ), which is subordinated to SMN_F. We denote with k the number of the loop executions. Since each processed node is checked only once, it follows that $k < |S| = n$ holds. In each loop execution, the Subordinated process is called. By this, the complexity of the CheckConstraintMinimalNode process is estimated to $O(n^2)$.

```
PROCESS Subordinated (I($N_i(R_i, K_i), S_F, (S, \varphi)$), O($S_{res}$), IO( ))
BEGIN PROCESS Subordinated
    SET $S_{res} \leftarrow \emptyset$
    DO FindSubordinatedNodes($\forall N_j(R_j, K_j) \in S_F$)
        IF ($(N_i(R_i, K_i), N_j(R_j, K_j)) \in \varphi$) THEN
            SET $S_{res} \leftarrow S_{res} \cup \{N_j(R_j, K_j)\}$
        END IF
    END DO FindSubordinatedNodes
END PROCESS Subordinated
```

Fig. 11. Process for selecting subordinated nodes in the closure graph

The CheckConstraintRelevantNodes process is presented in a form of pseudo-code in Figure 12. All variables used in the pseudo-code are already introduced in the previous text.

```
PROCESS CheckConstraintRelevantNodes
                                    (I($S_F, LE_{CC}$), O($SRN_{CC}$), IO( ))
BEGIN PROCESS CheckConstraintRelevantNodes
    SET $SRN_{CC} \leftarrow \emptyset$
    DO FindRelevantNodes($\forall N_i(R_i, K_i) \in S_F$)
        IF ($R_i \cup Attr(LE_{CC}) <> \emptyset$) THEN
            SET $SRN_{CC} \leftarrow SRN_{CC} \cup \{N_i(R_i, K_i)\}$
        END IF
    END DO FindRelevantNodes
END PROCESS CheckConstraintRelevantNodes
```

Fig. 12. Process for selecting relevant nodes of a check constraint

This process identifies the relevant nodes of a check constraint by testing each relation scheme corresponding to the form type encompassing the check constraint. The process tests if the set of attributes of the relation scheme intersects with the set of attributes referenced by the check constraint. If the test is positive, the relation scheme is considered as relevant. Since the loop FindRelevantNodes is executed once for each relation scheme corresponding to the form type, the complexity of this algorithm step is $O(n)$.

The CheckConstraintNecessaryNodes process is taken in its original form from [2], where it is presented with a detailed proof of its correctness. Also in [2], its complexity is assessed to $O(n^4)$. Since all processes called from the TransformCheck-Constraint process are correct, we conclude that the algorithm for transformation of a component type check constraint is also correct. Also, we conclude that the complexity of our proposed algorithm is $O(n^4)$ since each algorithm step is executed only once.

4.3 Transformation of Check Constraint Logical Expression

The logical expression of a domain, attribute and component type check constraints in the PIM of form types is introduced and defined using grammars given in Figures 1, 2 and 4, respectively. The transformation of a check constraint logical expression is the same for logical expressions of domain, attribute and component type check constraints. Therefore, in the following text we refer only to the logical expression without differentiating the type of a check constraint it is a part of.

As a part of the PIM to PSM transformation of check constraints, the logical expression is transformed into its disjunctive (DNF) or conjunctive normal form (CNF) using the logical equivalences from propositional calculus: replacement of implication and equivalence with negation, conjunction and disjunction, double negation, De Morgan's laws and distribution laws ([36, 37]). The resulting logical expression is defined by the same grammar as the starting one, with a difference that equivalence ($\Leftrightarrow$) and implication ($\Rightarrow$) operators are not allowed.

This transformation is necessary to provide the SQL code generation, since targeted SQL platforms do not recognize implication and equivalence logical operations. On the other hand, they are allowed in PIM specifications of check constraints.

In addition, this transformation is also necessary to provide semantic comparison of check constraints and automatic identification of conflicts between PIM modeled check constraints, which is a matter of our future research work.

For example, the logical expression of the check constraint specified in Example 4 is transformed into its DNF:

$$\neg(\mathit{FieldOfScience} ==' \mathit{Engineering}') \vee (\mathit{MajID} >= 101 \wedge \mathit{MajID} <= 1000),$$

or its CNF:

$$\begin{aligned}&(\neg(\mathit{FieldOfScience} ==' \mathit{Engineering}') \vee \mathit{MajID} >= 101)\\ &\qquad \wedge (\neg(\mathit{FieldOfScience} ==' \mathit{Engineering}') \vee \mathit{MajID} <= 1000).\end{aligned}$$

5 GENERATION OF THE EXECUTABLE CODE OF CHECK CONSTRAINTS

On the basis of designed database schema, SQL generator of IIS*Case currently provides generation of SQL/DDL code for the following RDBMS platforms: ANSI SQL-2003 standard, Oracle 9i and 10g, and MS SQL Server 2000 and 2008 RDBMSs. During the research presented in this paper, we have extended the SQL generator with model-to-code transformations that provide generation of SQL/DDL code for relational check constraints for all supported platforms. SQL/DDL patterns and supplementary procedures used for code generation are presented in the following text. The input specifications for the generator of check constraints are the results of the model-to-model transformations presented in the previous section, i.e. a closure graph, logical expression of the check constraint LE_{CC} as well as a domain specification, attribute specification, or a set of context relation schemes S_{CC} in a case of domain, attribute, or component type check constraint, respectively.

In Section 5.1, we present the generation of domain and attribute check constraints. The generation of relation scheme check constraints is presented in Section 5.2.

5.1 Generation of Domain and Attribute Check Constraints

According to the ANSI standard, domain check constraints are defined declaratively by using CREATE DOMAIN statement with CHECK clause. The SQL/DDL pattern for this purpose is given in Figure 13.

```
CREATE DOMAIN <constraint_name>
  AS <data_type> CHECK (<logical_expression>);
```

Fig. 13. SQL/DDL pattern for domain check constraint for ANSI SQL-2003 standard

By `<logical_expression>` we specify the logical expression of a check constraint with a name given by `<constraint_name>`. By `<data_type>` we associate a data type to the domain in the scope of a target platform.

On the contrary, Oracle and MS SQL Server RDBMSs do not provide CREATE DOMAIN statement. Therefore, domain check constraints are generated as column level check constraints by using the CHECK clause. The pattern is given in Figure 14. Terms `<table_name>` and `<attribute_name>` denote relation scheme and attribute names, respectively. The same pattern is also used for generation of attribute check constraints for all platforms.

```
CREATE TABLE <table_name> (
  ...
  <attribute_name> <data_type> CHECK (<logical_expression>),
  ...
)
```

Fig. 14. SQL/DDL pattern for attribute check constraint for all platforms

At the level of PIM model, we may have both a domain and an attribute check constraint related to the same attribute. Also, it is possible to assign attribute check constraints both to a domain and its inherited domain. In all such situations, the same SQL/DDL pattern is used, while `<logical_expression>` of the pattern is generated by concatenating logical expressions of all relevant check constraints with AND logical operator. Consequently, the only one CHECK constraint is generated per one attribute, with a `<logical_expression>` consisting of AND-concatenated related logical expressions specified at the level of a PIM.

Example 5. Let us observe a PIM domain constraint NOT_NEG_NUM, which inherits $NUMBER$ built-in data type and constrains it to the positive numbers only by the following check constraint:

$$VALUE >= 0.$$

At the level of a PIM, *NOT_NEG_NUM* is assigned to a *PERCENTAGE* attribute, for which there is a check constraint specified by the logical expression

$$PERCENTAGE <= 100.$$

Let us further assume that *PERCENTAGE* is assigned to the *EXAMS* relation scheme by the model-to-model transformations. To implement the specifications of *NOT_NEG_NUM* and *PERCENTAGE* with their check constraints, the generators for Oracle or SQL Server platforms finally produce SQL/DDL code as is given in Listing 1.

```
CREATE TABLE EXAMS (
   ...
   PERCENTAGE NUMBER
                    CHECK (PERCENTAGE>=0 AND PERCENTAGE<=100),
   ...
)
```

Listing 1. DDL code for PERCENTAGE attribute, generated for Oracle and MS SQL Server

In a case of the ANSI standard, the generated SQL/DDL code is slightly different since *NOT_NEG_NUM* domain is specified explicitly, as illustrated in Listing 2.

```
CREATE DOMAIN NOT_NEG_NUM AS NUMBER CHECK (VALUE>=0);

CREATE TABLE EXAMS (
   ...
   PERCENTAGE NOT_NEG_NUM CHECK (PERCENTAGE<=100),
   ...
)
```

Listing 2. DDL code for NOT_NEG_NUM domain and PERCENTAGE attribute, generated for ANSI SQL-2003 standard

5.2 Generation of Relation Scheme Check Constraints

For the generation of component type check constraints, we differentiate two cases:

1. single relation scheme check constraints – having only one context relation scheme, and
2. multiple relation schemes check constraints – spanning multiple context relation schemes.

The pattern for a single relation scheme check constraint has the same form for all platforms, given in Figure 15.

```
ALTER TABLE <table_name> ADD
   CONSTRAINT <constraint_name> CHECK (<logical_expression>);
```

Fig. 15. SQL/DDL pattern for single relation scheme check constraints for all platforms

The SQL/DDL pattern for check constraints spanning multiple relation schemes is more complex since such check constraints cannot be fully declaratively specified. Despite that multiple relation scheme check constraints are generated for the ANSI platform using the declarative mechanism of assertions, the natural join of context relation schemes cannot be created declaratively. On the contrary, Oracle and SQL Server do not support assertions. Therefore, multiple relation scheme check constraints are implemented using triggers on context relation schemes. The details regarding the code generation are given in the rest of the section.

The first step in generation of multiple relation scheme check constraints, common to all targeted platforms, is to generate the natural join expression for relation schemes from S_{CC}. Since attributes in IIS*Case are defined according to the URSA rule, the same-named attributes in two different relation schemes have the same semantics and by making equality expression between the same-named attributes we specify the appropriate join conditions. Thus, the natural join expression for a set of relation schemes is created by making equality expressions between same-named attributes in each pair of relation schemes that are directly connected in the closure graph, and joining those expressions with AND logical operators. We join only relation schemes that are directly connected in the closure graph in order to avoid generation of transitive and redundant join conditions. The EBNF of the natural join expression is given in Figure 16.

```
natural_join_expression =
table_name.attribute_name '=' table_name.attribute_name
{'AND' table_name.attribute_name '=' table_name.attribute_name};
```

Fig. 16. Natural join expression in EBNF notation

In the subsequent patterns, we use a term `list_of_context_relation_schemes` specifying the list of the context relation schemes of a check constraint S_{CC}, as given in Figure 17, in EBNF notation.

```
list_of_context_relation_schemes = table_name {',' table_name};
```

Fig. 17. The list of context relation schemes of a check constraint in EBNF notation

The ANSI SQL-2003 standard provides declarative mechanisms for defining logical conditions of constraints at the level of database schema. For these purposes, the CREATE ASSERTION statement is used. We also use it to generate check constraints spanning multiple relation schemes according to the patterns given in Figure 18. The logic of the pattern is to count the tuples of the natural join of relations over S_{CC} that do not satisfy the logical expression of the check constraint. To satisfy the check constraint, the number of such tuples has to be zero.

```
CREATE ASSERTION ASSERT_<constraint_name> CHECK(
   (SELECT count(*) FROM <list_of_context_relation_schemes>
   WHERE <natural_join_expression>
      AND NOT(<logical_expression>)) = 0);
```

Fig. 18. SQL/DDL pattern for multiple relation schemes check constraint for ANSI standard

To implement the same logic under Oracle 9i/10g RDBMSs and MS SQL Server 2000/2008 RDBMSs, triggers with procedural capabilities have to be used, since these RDBMSs still do not provide CREATE ASSERTION statement. More precisely, a separate trigger has to be defined over each relation scheme from S_{CC}, which validates inserted or updated tuples against the constraint. Similar approach may be found in [20] with a difference regarding the performance optimization. The authors in [20] check all tuples of relations over S_{CC} at each insert or update operation over these relations. On the other hand, we check only newly inserted or updated tuples by joining only them with other relations from S_{CC} and validating the resulting tuples against the check constraint logical expression. The deleted tuples are not checked by the trigger since deletion cannot violate the check constraint in any way.

In Oracle 9i/10g RDBMSs, it is technically achieved by modifying the natural join expression and check constraint logical expression used in previous pattern for the ANSI standard. We modify both expressions by replacing the name of the context relation scheme of the trigger with :NEW pseudo-record, in order to reference only the current tuple being processed by the trigger. The resulting expressions are denoted simply as the modified natural join expression and modified check constraint logical expression, respectively.

Also, the context relation scheme of the trigger is removed from the list of the context schemes in order not to check all tuples of the corresponding relation. We denote such list as the reduced list of context relation schemes. The pattern for the proposed trigger is given in Figure 19, where reduced list of the context relation schemes, modified natural join expression and modified check constraint logical expression are denoted as `<reduced_list_of_context_schemes>`, `<natural_join_expression_with_:NEW>`, and `<logical_expression_with_:NEW>`, respectively.

```
CREATE OR REPLACE TRIGGER TRG_<constraint_name>
  BEFORE INSERT OR UPDATE ON <table_name>
  FOR EACH ROW
  DECLARE cnt INT;
  exc EXCEPTION;
BEGIN
  SELECT count(*) INTO cnt FROM <reduced_list_of_context_schemes>
  WHERE <natural_join_expression_with_:NEW>
    AND NOT(<logical_expression_with_:NEW>);

  IF cnt<>0 THEN
    RAISE exc;
  END IF;
EXCEPTION
  WHEN exc THEN
    RAISE_APPLICATION_ERROR(-20900,'Check constraint violated');
END
```

Fig. 19. SQL/DDL pattern for check constraint trigger for Oracle platform

In a case the number of tuples that do not satisfy the logical expression of the check constraint is not equal to zero, an exception is raised and the complete DML statement firing the trigger is rolled back.

The same logic is used for the generation of such kind of check constraints in MS SQL Server 2000/2008 RDBMS. However, details of technical implementation are different, because:

1. the Inserted table is used instead of :NEW pseudo-record for creating the modified natural join expression and modified check constraint logical expression, and
2. the context relation scheme of a trigger is replaced with the Inserted table, instead of reducing the list of context relation schemes. Thereby, we obtain the modified list of context relation schemes.

The pattern for MS SQL Server triggers is shown in Figure 20. The modified list of context relation schemes, modified natural join expression and modified check constraint logical expression are denoted as `<list_of_context_schemes_with_Inserted>`, `<natural_join_expression_with_Inserted>`, and `<logical_expression_with_Inserted>`, respectively.

In the following text, we give two examples of generated SQL/DDL code for the check constraints specified in the Examples 3 and 4.

```
CREATE TRIGGER TRG_<constraint_name>
  ON <table_name>
  FOR INSERT, UPDATE
AS
  DECLARE @cnt INT

  SELECT @cnt=count(*) INTO cnt
    FROM <list_of_context_schemes_with_Inserted>
  WHERE <natural_join_expression_with_Inserted>
    AND NOT(<logical_expression_with_Inserted>)

  IF (@cnt<>0)
  BEGIN
    RAISERROR('Check constraint violated',16,1)
    ROLLBACK TRAN
  END
GO
```

Fig. 20. SQL/DDL pattern for check constraint trigger for MS SQL Server platform

Example 6. We present the SQL/DDL code for the check constraint specified and transformed into the relational data model in Example 3, generated for the ANSI SQL-2003 standard and Oracle 9i/10g RDBMS. The SQL/DDL code for MS SQL Server platform is omitted since it is very similar to the code for Oracle platform, and is available upon request.

According to the ANSI SQL-2003, the check constraint is implemented using the CREATE ASSERTION statement (Listing 3). In the listing, the natural join expression is bolded; the logical expression of the check constraint is given in the typewriter font, whereas the list of context relation schemes is given in italics.

```
CREATE ASSERTION ASSERT_CHKC_DepCha CHECK(
  (SELECT count(*) FROM Department, Chair
   WHERE Chair.FacId=Department.FacId
         AND Chair.DepId=Department.DepId
         AND NOT (Department.DepEmpNumber >
                        Chair.ChrEmpNumber)) = 0);
```

Listing 3. The check constraint from Example 3 generated for ANSI SQL-2003 standard

For Oracle 9i/10g RDBMS, the check constraint is generated as an insert and update trigger on *Department* and *Chair* relation schemes. The triggers are given in Listing 4.

Here, the modified natural join expression is bolded, modified check constraint logical expression is given in the typewriter font and the reduced list of context relation schemes is given in italics. It should be noted that the context relation scheme of each trigger does not appear in the lists of context relation schemes, i.e. in the FROM part of the SELECT statements. Instead, the tuples of other context relations of the check constraint are joined with the :NEW pseudo-record, i.e. with the inserted or updated tuple.

```
CREATE OR REPLACE TRIGGER TRG_CHKC_DepCha_Chair
  BEFORE INSERT OR UPDATE ON Chair
  FOR EACH ROW
  DECLARE cnt INT;
          exc EXCEPTION;
BEGIN
  SELECT count(*) INTO cnt FROM Department
  WHERE :NEW.FacId=Department.FacId
        AND :NEW.DepId=Department.DepId
        AND NOT(Department.DepEmpNumber>:NEW.ChrEmpNumber);
  IF cnt<>0 THEN
    RAISE exc;
  END IF;
EXCEPTION
  WHEN exc THEN
    RAISE_APPLICATION_ERROR(-20900,
                     'Check constraint violated');
END;

CREATE OR REPLACE TRIGGER TRG_CHKC_DepCha_Department
  BEFORE INSERT OR UPDATE ON Department
  FOR EACH ROW
  DECLARE cnt INT;
          exc EXCEPTION;
BEGIN
  SELECT count(*) INTO cnt FROM Chair
  WHERE :NEW.FacId=Chair.FacId AND :NEW.DepId=Chair.DepId
        AND NOT(:NEW.DepEmpNumber>Chair.ChrEmpNumber);
  IF cnt<>0 THEN
    RAISE exc;
  END IF;
EXCEPTION
  WHEN exc THEN
    RAISE_APPLICATION_ERROR(-20900,
                     'Check constraint violated');
END;
```

Listing 4. Oracle triggers generated for the check constraint from Example 3

Example 7. The check constraint from Example 4 is generated for ANSI SQL-2003 platform with the CREATE ASSERTION statement (Listing 5). It can also be noticed here that the logical expression of the check constraint (typewriter font) has been transformed into DNF, since the ANSI standard does not support implication as a logical operation.

```
CREATE ASSERTION ASSERT_CHKC_DepMajFac CHECK(
  (SELECT count(*) FROM Department,Major,Faculty
   WHERE Department.FacId=Major.FacId
         AND Department.DepId=Major.DepId
         AND Department.FacId=Faculty.FacId
         AND NOT((NOT(Faculty.FieldOfScience='Engineering'))
                 OR(Major.MajID>=101
                 AND Major.MajID<=1000))) = 0)
```

Listing 5. The check constraint from Example 4 generated for ANSI SQL-2003 standard

On the other hand, the same check constraint is generated for Oracle platform as insert and update triggers on *Major*, *Department* and *Faculty* relation schemes. In Listing 6, we give only the trigger for the *Major* table, since other two triggers are similar. The modified natural join expression is bolded, modified check constraint logical expression is given in the typewriter font and the reduced list of context relation schemes is given in italics.

```
CREATE OR REPLACE TRIGGER TRG_CHKC_DepMajFac_Major
  BEFORE INSERT OR UPDATE ON Major
  FOR EACH ROW
  DECLARE cnt INT;
          exc EXCEPTION;
BEGIN
  SELECT count(*) INTO cnt FROM Department, Faculty
  WHERE Department.FacId=Faculty.FacId
        AND :NEW.FacId=Department.FacId
        AND :NEW.DepId=Department.DepId
        AND NOT((NOT(Faculty.FieldOfScience='Engineering'))
            OR (:NEW.MajID>=101 AND :NEW.MajID<=1000));

  IF cnt<>0 THEN
    RAISE exc;
  END IF;
EXCEPTION
  WHEN exc THEN
    RAISE_APPLICATION_ERROR(-20900,
                            'Check constraint violated');
END;
```

Listing 6. Oracle trigger on Major table for the check constraint from Example 4

6 CONCLUSION

In this paper we have presented the algorithms for transformation of check constraint PIM specifications into check constraints defined in the relational data model and generation of executable SQL/DDL code for the modeled check constraints for several standard and commercial platforms. By the MDA terminology, we have developed model-to-model transformations from PIM to PSM specifications of check constraints, as well as model-to-code transformations of check constraints from the relational data model into the executable code. We have also implemented the proposed transformation algorithms into the MDSD tool IIS*Case, and verified their applicability and validity in a case study. We extended the SQL generator of IIS*Case with a new functionality to produce the executable SQL/DDL code using the proposed patterns. To the best of our knowledge, by this we have created a convenient and user-friendly for platform independent and tool-supported development of check constraints.

During the research presented in this paper and its implementation, we have also identified several directions for future research. Development of complex information systems is always a demanding and team-oriented process, where each designer models only a part of the information system from his or her point of view and field of expertise. We have already developed algorithms for an automatic consolidation of independently developed segments of an information system and creation of a unified database schema of the information system as a whole. These algorithms are capable of identifying formal conflicts and assisting in resolving semantic conflicts at the level of relation schemes, sets of attributes, key constraints, as well as some other constraints. However, check constraints are not currently supported. One of the future research tasks is to provide algorithms for the consolidation of independently modeled check constraints, as well as the assistance in identification and resolution of semantic conflicts.

Such algorithms need to impose the following rules over the modeled check constraints. Each check constraint modeled at the level of an information system segment has to be equal to or stronger than the corresponding check constraint at the level of the unified database schema. Also, a check constraint at the level of the unified database schema has to be a logical consequence of all corresponding check constraints modeled at the level of information system segments. To test the satisfaction of the aforementioned rules the resolution method ([36]) may be applied. For example, to test the satisfaction of the second rule, we will consider the check constraints at the level of information system segments as premises, while the check constraint at the level of the unified database schema is the conclusion. By transforming check constraint logical expressions into CNF, we have created a formal basis for applying the resolution method.

A future research task is also to extend the current functionalities of automatic generation of check constraints by the coverage of user-defined functions (with rather complex functionality, often including various SQL SELECT or even DML statements) and their PIM specifications. As far as we know, this feature is not supported by the existing DBMSs directly. The most challenging issue in such a research is that the results of user-defined functions may depend on the state and updates of other database relations, not referenced directly by the check constraint being validated.

Currently IIS*Case implicitly provides modeling business rules that result in specifications of various database constraints, including check constraints. In order to extend the modeling capabilities for business rules, we are to consider various business rule technologies suitable for those purposes, as well as applying the modal logic concepts ([38]).

Taking into account a large extent of the usage of check constraints both in academic and industrial environments at one hand side, as well as the existing open questions at the other, we believe that further research efforts regarding the check constraints are fully justified.

REFERENCES

[1] Luković, I.—Popović, A.—Mostić, J.—Ristić, S.: A Tool for Modeling Form Type Check Constraints and Complex Functionalities of Business Applications. Computer Science and Information Systems (ComSIS), Vol. 7, 2010, No. 2, pp. 359–385.

[2] Luković, I.: Automated Generation of Relational Database Subschemas Using the Form Types. M. Sc. thesis (in Serbian), University of Belgrade, Faculty of Electrical Engineering, Belgrade, Serbia 1993.

[3] Luković, I.—Mogin, P.—Pavićević, J.—Ristić, S.: An Approach to Developing Complex Database Schemas Using Form Types. Software: Practice and Experience, Vol. 37, 2007, No. 15, pp. 1621–1656.

[4] Luković, I.: From the Synthesis Algorithm to the Model Driven Transformations in Database Design (Invited Talk). In: Informatics 2009, Proceedings of 10th International Scientific Conference on Informatics, Herľany, Slovak Republic, November 2009, pp. 9–18.

[5] Pavićević, J.: An Approach to Generating Executable Software Specifications of an Information System. Ph. D. thesis (in Serbian). University of Novi Sad, Faculty of Technical Sciences, Novi Sad, Serbia 2010.

[6] Burnett, M.—Cook, C.—Rothermel, G.: End-User Software Engineering. Communications of the ACM (CACM), Association for Computing Machinery, Vol. 47, 2004, No. 9, pp. 53–58.

[7] Sutcliffe, A.—Mehandjiev, N.: End-User Development. Communications of the ACM (CACM), Association for Computing Machinery, Vol. 47, 2004, No. 9, pp. 31–32.

[8] Object Management Group: MDA Guide Version 1.0.1. Available on: `http://www.omg.org/`, 2003.

[9] Luković, I.—Ristić, S.—Aleksić, S.—Popović, A.: An Application of the MDSE Principles in IIS*Case. In: Proceedings of 3rd Workshop on Model Driven Software Engineering, MDSE 2008, Berlin, Germany 2008, pp. 53–62.

[10] Kollár, J.—Porubän, J.—Václavík, P.—Bandáková, J.—Forgáč M.: Functional Approach to the Adaptation of Languages Instead of Software Systems. Computer Science and Information Systems (ComSIS), Vol. 4, 2007, No. 2, pp. 115–129.

[11] Kollár, J.—Porubän, J.: Building Adaptive Language Systems. Journal of Computer Science (INFOCOMP), Vol. 7, 2008, No. 1, pp. 1–10.

[12] Luković, I.—Ristić, S.—Mogin, P.—Pavićević, J.: Database Schema Integration Process – A Methodology and Aspects of Its Applying. Novi Sad Journal of Mathematics, Vol. 36, 2006, No. 1, pp. 115–150.

[13] Luković, I.—Ristić, S.—Mogin, P.: On the Formal Specification of Database Schema Constraints. Proceedings of 1st Serbian – Hungarian Joint Symposium on Intelligent Systems, Subotica, September 2003, pp. 125–136.

[14] Mogin, P.—Luković, I.—Karadžić, Z.: Relational Database Schema Design and Application Generating using IIS*CASE Tool. Proceedings of International Conference on Technical Informatics, Timisoara, November 1994, Vol. 5, pp. 49–58.

[15] Pavićević, J.—Luković, I.—Mogin, P.—Govedarica, M.: Information System Design and Prototyping Using Form Types. Proceedings of INSTICC I International Conference on Software and Data Technologies, ICSOFT, Setubal, September 2006, Vol. 2, pp. 157–160.

[16] Pavićević, J.: Development of a CASE Tool for Automated Design and Integration of Database Schemas. M. Sc. thesis (in Serbian). University of Montenegro, Faculty of Science, Podgorica, Montenegro, 2005.

[17] Pereira, M.—Mernik, M.—da Cruz, D.—Henriques, P.: Program Comprehension for Domain-Specific Languages. Computer Science and Information Systems (ComSIS), Vol. 5, 2008, No. 2, pp. 1–17.

[18] Kosar, T.—Oliveira, N.—Mernik, M.—Pereira, M.—Črepinšek, M.—da Cruz, D.—Henriques, P.: Comparing General-Purpose and Domain-Specific Languages: An Empirical Study. Computer Science and Information Systems (ComSIS), Vol. 7, 2010, No. 2, pp. 247–264.

[19] Demuth, B.—Hussmann, H.: Using UML/OCL Constraints for Relational Database Design. In: Robert France and Bernhard Rumpe (Eds.): "UML" '99: The Unified Modeling Language – Beyond the Standard, Second International Conference, Fort Collins, October 1999, Lecture Notes in Computer Science, pp. 598–613.

[20] Demuth, B.—Hussmann, H.—Loecher, S.: OCL as a Specification Language for Business Rules in Database Applications. In: Martin Gogolla, Cris Kobryn (Eds.): "UML" 2001 – The Unified Modeling Language, Modeling Languages, Concepts, and Tools, 4th International Conference, Toronto, October 2001, Lecture Notes in Computer Science, pp. 104–117.

[21] Warmer, J.—Kleppe, A.: The Object Constraint Language: Precise Modeling with UML. Addison-Wesley Longman Publishing Co., Inc., Boston, USA 1999.

[22] Oracle Designer web page. Available on: `http://wiki.oracle.com/page/Oracle+Designer`, September 2010.

[23] Sybase PowerDesigner web page. Available on: `http://www.sybase.com/products/modelingdevelopment/powerdesigner`, September 2010.

[24] Sparx Systems web page, Available on: `http://www.sparxsystems.com.au/`, September 2010.

[25] IBM Rational Rose web page. Available on: `http://www-01.ibm.com/software/awdtools/developer/rose`, September 2010.

[26] CA ERwin Data Modeler web page. Available on: `http://erwin.com/products/detail/ca\_erwin\_data\_modeler`, November 2010.

[27] Cabot, J.—Teniente, E.: Constraint Support in MDA Tools: A Survey. In: Arend Rensink, Jos Warmer (Eds.): Model Driven Architecture – Foundations and Applications, Second European Conference, ECMDA-FA 2006, Bilbao, July 2006, Lecture Notes in Computer Science, pp. 256–267.

[28] Cabot, J.: Constraint Support in MDA Tools: A Survey (Extended article). Available on: `http://jordicabot.com/research/OCLSurvey/index.html`, November 2010.

[29] Parr, T.: Grammars. Available on: `http://www.antlr.org/wiki/display/CS652/Grammars`, June 2011.

[30] Green, T.—Petre, M.: Usability Analysis of Visual Programming Environments: A "Cognitive Dimensions" Framework. Journal of Visual Languages and Computing, Vol. 7, 1996, No. 2, pp. 131–174.

[31] Mernik, M.—Kosar, T.—Črepinšek, M.—Henriques, P.—da Cruz, D.—Pereira, M.—Oliveira, N.: Comparison of XAML and C# Forms Using Cognitive Dimension Framework. In: Proceedings of INForum '09 – Simpósio de Informática, Faculdade de Ciéncias da Universidade de Lisboa, Lisbon, Portugal, September 2009, pp. 180–191.

[32] Bernstein, P. A.: Synthesizing Third Normal Form Relations from Functional Dependencies. ACM Transactions on Database Systems, Vol. 1, 1976, No. 4, pp. 277–298.

[33] Mogin, P.—Luković, I.—Govedarica, M.: Database Design Principles. 2nd Ed. (in Serbian). University of Novi Sad, Faculty of Technical Sciences, Novi Sad, Serbia and Montenegro, 2004.

[34] Elmasri, R.—Navathe, S.: Fundamentals of Database Systems. 4th Edition. Pearson Education/Addison-Wesley, Boston, USA 2004.

[35] Mogin, P.—Luković, I.: Database Principles. 1st Ed. (in Serbian). University of Novi Sad, Faculty of Technical Sciences, Novi Sad, Federal Republic of Yugoslavia, 1996.

[36] Colton, S.—Gow, J.: The Resolution Method. Available on: `http://www.doc.ic.ac.uk/~sgc/teaching/v231/lecture8.html`, February 2011.

[37] Mendelson, E.: Introduction to Mathematical Logic. 4th Edition. Chapman & Hall, London, United Kingdom 1997.

[38] Modal Logic, Available on: `http://en.wikipedia.org/wiki/Modal_logic`, June 2011.

Nikola Obrenović received his M. Sc. degree in applied computer science and informatics from Faculty of Technical Sciences at the University of Novi Sad. Currently, he is a Ph. D. student at the University of Novi Sad and employee with Telvent DMS Llc, Novi Sad, at senior software engineer position. His research interests include database systems, software engineering and data mining.

Aleksandar Popović graduated from Faculty of Science at the University of Montenegro. He completed his Mr (2 year) degree at the University of Novi Sad, Faculty of Technical Sciences. Currently, he is a Ph. D. student and teaching assistant at the University of Montenegro, Faculty of Science. He assists in teaching several Computer Science and Informatics courses. His research interests include software engineering, database systems and domain specific languages.

Slavica Aleksić received her M. Sc. (5 year, former Diploma) degree from Faculty of Technical Sciences in Novi Sad. She completed her Mr (2 year) degree at the University of Novi Sad, Faculty of Technical Sciences. Currently, she works as a teaching assistant at the Faculty of Technical Sciences at the University of Novi Sad, where she assists in teaching several computer science and informatics courses. Her research interests are related to information systems, database systems and software engineering.

Ivan Luković received his M. Sc. (5 year, former Diploma) degree in informatics from the Faculty of Military and Technical Sciences in Zagreb in 1990. He completed his Mr (2 year) degree at the University of Belgrade, Faculty of Electrical Engineering in 1993, and his Ph. D. at the University of Novi Sad, Faculty of Technical Sciences in 1996. Currently, he works as a Full Professor at the Faculty of Technical Sciences at the University of Novi Sad, where he lectures in several computer science and informatics courses. His research interests are related to database systems and software engineering. He is the author or coauthor of over 90 papers, 4 books, and 30 industry projects and software solutions in the fields.