Resolving Database Constraint Collisions Using IIS*Case Tool

Sonja Ristić, Assistant Professor, Ph.D., B.Sc. in Math., B.Sc. in Ecc.,
University of Novi Sad, Faculty of Technical Sciences,
Trg D. Obradovića 6, 21000 Novi Sad, Serbia,
e-mail: sdristic@uns.ns.ac.yu

Ivan Luković, Full Professor, Ph.D., Inf. Eng.,
University of Novi Sad, Faculty of Technical Sciences,
Trg D. Obradovića 6, 21000 Novi Sad, Serbia,
e-mail: ivan@uns.ns.ac.yu

Jelena Pavićević, M.Sc., B.Sc. in Math.,
Internet Crna Gora d.o.o and University of Montenegro, Faculty of Science,
Bulevar Svetog Petra Cetinjskog 2, 81000 Podgorica, Montenegro,
e-mail: jelenap@cg.yu

Pavle Mogin, Senior Lecturer, Ph.D., El. Eng.,
Victoria University of Wellington, School of Mathematical and Computing Sciences,
P.O. Box 600, Wellington, New Zealand
e-mail: pmogin@mcs.vuw.ac.nz

Abstract. *IIS*Case (Integrated Information Systems*Case, R.6.21) is a CASE tool that we developed to support automated database (db) schema design, based on a methodology of gradual integration of independently designed subschemas into a database schema. It provides complete intelligent support for developing db schemas and enables designers to work together and cooperate reaching the most appropriate solutions.*

*The process of independent design of subschemas may lead to collisions in expressing the real world constraints and business rules. IIS*Case uses specialized algorithms for checking the consistency of constraints embedded in the database schema and the subschemas. IIS*Case supports designers in reviewing and validating results obtained after each step of the design process. The paper outlines the process of resolving collisions. A case study based on an imaginary production system is used to illustrate the application of IIS*Case. Different outcomes and their consequences are presented.*

Keywords. *Database Schema Design and Integration; CASE tool; Constraint Collisions; IIS*Case.*

1. Introduction

There are two common basic approaches to the process of database (db) schema design: (a) the direct approach, and (b) the approach of a gradual integration of external schemas [3].

In the direct approach, user requirements are processed all at once and this approach may be appropriate only in cases of design of small db schemas.

The second approach is used when the number and complexity of user requirements are beyond the designer's power of perception.

IIS*Case (Integrated Information Systems* Case, R.6.21) is a CASE tool, relying on the second approach. It is developed to support an automated database (db) schema design, based on the concepts end-users are familiar with. It is based on a methodology of gradual integration of independently designed subschemas into a db schema ([4], [5], [3], [18]). IIS*Case is designed to provide complete support for developing db schemas and to give an intelligent support during that process. It enables designers to work together and cooperate reaching the most appropriate solutions.

The process of independent design of subschemas may lead to collisions in expressing the real world constraints and business rules. If the collisions exist, at least one subschema is formally not consistent with the db schema. Programs made over an inconsistent subschema do not guarantee safe database updates. IIS*Case uses specialized algorithms for checking the consistency of constraints embedded in the database schema and the subschemas. The nature of the most of the

collisions is such that the designers must resolve them themselves, at the conceptual level, but we believe that IIS*Case may considerably improve the process of their resolving.

The paper presents a way of applying IIS*Case in the process of resolving constraint collisions, for a selected case study. The case study represents a simplified, imaginary production system. We discuss in the paper some alternative designer's decisions and analyze their possible consequences. Not all of the alternatives are always applicable, and we highlight such particular cases in the paper.

Our approach is based on the concept of form type ([2], [4], [18]). Unlike some other similar approaches ([6], [13]), we do not use Entity-Relationship (ER) data model for conceptual modelling. Instead, we focus on straightforward generating relational db schemas using form type specifications. Although the approach is not a sole one [19], we have not found references covering all the aspects of our approach. Some crucial differences between our approach and the other ones are: (i) IIS*Case generates not only relation scheme keys and basic referential integrity constraints, but also unique constraints and other interrelation constraints; (ii) IIS*Case provides algorithms for integrating independently designed subschemas into a unified db schema; (iii) Unified db schema and subschemas are aimed not only for queries, but also for safe updates that guarantee database consistency. A more detailed discussion of related works may be found in [18].

Apart from Introduction and Conclusion, the paper consists of three sections. In Section 2 we discus a survey of the approach. Section 3 is devoted to the constraint collisions. Section 4 presents a case study, and illustrates applying IIS*Case in detecting and resolving collisions. We particularly cover resolving collisions of key, unique, null value, and referential integrity constraints.

2. An Outline of the Approach

Generally, design of a complex db schema is based on a gradual integration of external schemas. An *external schema* is a structure that, at the conceptual level, formally specifies a user view on a db schema. The first step of a db schema design process is designing separate external schema for each group of similar end users business tasks. Each transaction program that supports a user requirement is based on an external schema, which is associated to it.

A db schema design in the IIS*Case environment is organized by decomposing the whole project into application systems. An *application system* is a specification of a subsystem of a future information system. The set of all application systems of an information system is organized as a tree structure. It is the *application system tree* of the information system. Thus, each application system may include one or more child application systems (*application subsystems*). Fig. 1 depicts two different application system trees in IIS*Case: *Factory* and *Faculty Organization*. Application system *Administration* has three apllication subsystems: *Personnel, Working_Unit* and *Working_Orders*.

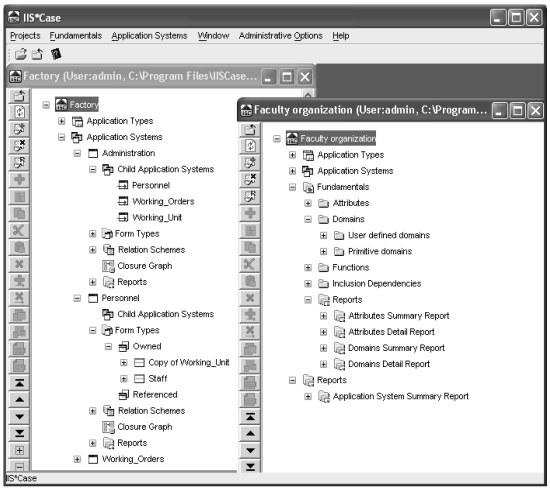


Figure 1. Application system trees in IIS*Case

External schemas in IIS*Case are expressed by sets of the *form types*. A form type generalizes a document type, i.e. screen or report forms that users utilize to communicate with an information system [18]. Each form type is designed in the context of an application system. Therefore, a set of form types is a part of an application system, and represents an input specification for the process of the db schema design.

Fig. 2 depicts steps of the db schema design process in IIS*Case. Texts written in italic style denote the outputs of the preceding steps. Conceptual modelling is performed by creating sets of form types, one for each application subsystem.

After being created, external schemas should be integrated into a conceptual db schema. In contrast to other (conceptual) data models, relational data model offers much wider possibilities to formalize and automate the process of db schema integration [18]. Therefore, db schema integration in IIS*Case is done at the implementation level, where a db schema is expressed by the relational data model. A db schema is obtained by the gradual integration of *subschemas*. A *subschema* is obtained by expressing an external schema by the concepts of the relational data model and

applying the synthesis algorithm [3], [12]. A formal specification of a subschema may be found in [16]. Step 2 generates a *subschema* for each directly subordinated application subsystem of the selected application system. Step 3 generates a relational db schema for the selected application system. It is called a *potential database schema*.

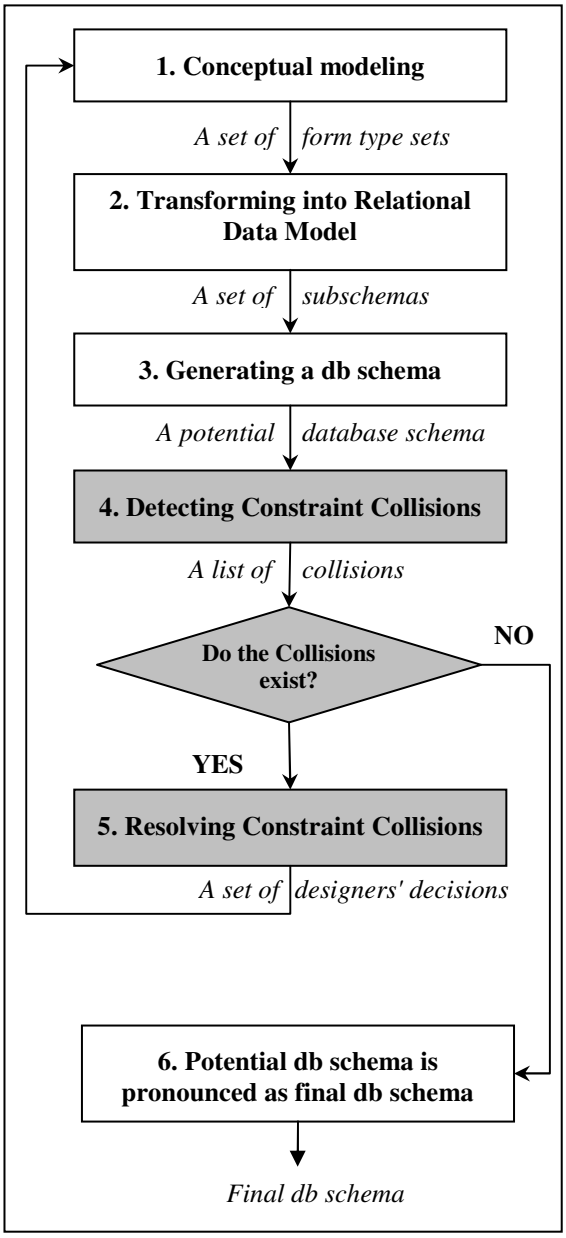


Figure 2. Steps of the db schema design process

The process of independent design of external schemas may lead to collisions in expressing the real system constraints. If such collisions exist, at least one subschema is inconsistent with the potential database schema. The programs made over inconsistent subschemas do not guarantee *safe database updates*. (The problem of safe database updates is discussed in [9].) Therefore, the appropriate procedures for resolving collisions, which arise as a result of independent modelling of subschemas, must be applied. The process of detecting and resolving constraint collisions is called a *consolidation* of a db schema and its subschemas. Shaded rectangles in Fig. 2 represent steps of the consolidation process.

Db schema design is an iterative process, ending when all of the subschemas are consistent with the potential db schema. The potential db schema becomes a formal specification of an implementation db schema (Step 6).

IIS*Case supports a designer in reviewing and validating results obtained after each step of the design process. For example, the designer may review generated relation schemes and constraints, and check the compatibility with the subschemas. If the designer is not satisfied with generated results, or there are some incompatibilities, he or she can go one or more steps back, make changes in form types and repeat the process.

A more detailed explanation of db design process in IIS*Case may be found in [18].

3. Collisions of Constraints

Our approach to the integration is based on detecting and resolving constraint collisions that may arise among a potential db schema and subschemas of an application system. In this section the principles of the process of resolving constraint collisions are presented.

Let P_k be the subschema from one of the application subsystems of the selected application system. In step 3 of the db design process, a potential database schema for the application system is generated.

A db schema constraint is said to be *relevant constraint* for subschema P_k, if the operation that might violate it, is allowed in P_k.The operations that might violate a constraint are called *critical operations*.

A database schema constraint is said to be *embedded* into subschema P_k if it is a logical consequence of the set of subschema constraints.

A constraint that is relevant for a subschema P_k may be embedded, or not embedded into P_k. A constraint that is relevant for P_k but not embedded into it, may be:

- *Includible*, if it can be expressed using the existing concepts and structure of the subschema P_k; or

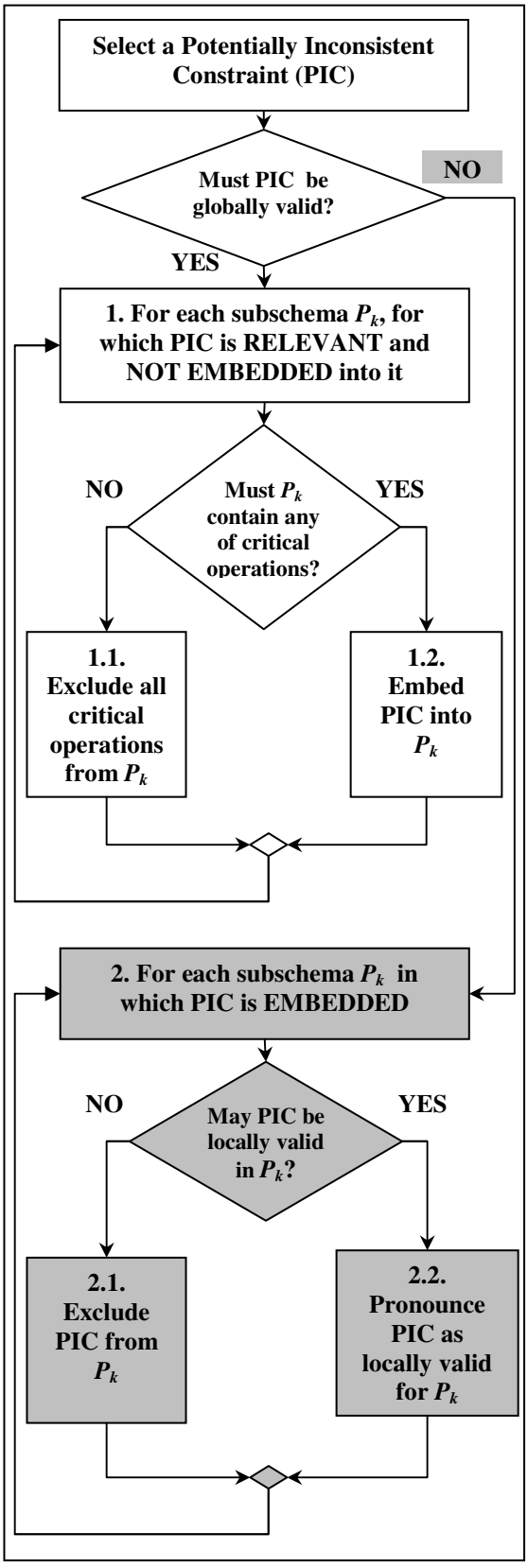


Figure 3. Steps of the process of resolving constraint collisions

· *Extending*, otherwise.

In order to embed an extending constraint into a subschema P_k, we must add some new concepts (new attributes, or even new relation schemes) into the subschema. Embedding an includible constraint does not require any changes of the structure of a subschema.

A database constraint is *potentially inconsistent* if it is relevant for at least one subschema P_k, but not embedded into it.

Constraint inconsistencies are also called *constraint collisions*.

The integration process may successfully pass from step 3 trough step 6 (Fig. 2), only if all the subschemas contain compatible sets of constraints i.e. if an empty list of collisions is generated in step 4. Otherwise, the integration process stops, and the collisions must be resolved. In the process of resolving collisions, colliding constraints may be embedded into subschemas for various reasons. The main one is independent modelling of their form types. Thus, the appropriate procedures for resolving collisions must be applied in step 5 (Fig. 2).

Fig. 3 depicts steps of the process of resolving constraint collisions. For each potentially inconsistent constraint (PIC), a designer has to decide whether it should be embedded into the db schema. Subschema constraints that are embedded into the db schema are considered as *globally valid*.

If a designer decided to embed a PIC into a db schema, it must be also embedded into all the subschemas, for which it is relevant. Therefore, for each subschema for which selected PIC is relevant and not embedded into it, designer has two possible solutions:

· To embed selected PIC into P_k; or

· To exclude all critical operations for selected PIC from P_k. Accordingly, PIC is no longer relevant constraint for P_k.

Otherwise, a PIC must not be embedded into the set of database constraints. It is important to emphasize here that subschema constraints must not be less restrictive than the corresponding database constraints, but may be more restrictive. Consequently, some subschema constraints may not be embedded into the db schema. A subschema constraint is considered as *locally valid* if it is embedded into a subschema, but not embedded into the db schema. Some constraints could not be locally valid. Unique constraint is one of them, as it is illustrated in Section 4.2. Therefore, a selected PIC has to be

· excluded from, or

· pronounced as locally valid in

all the subschemas from which it stems.

In the first iteration of the db schema design process, all constraints of a subschema are pronounced as globally valid. Some of them may be pronounced as locally valid in the subsequent iterations.

IIS*Case uses specialized algorithms to check the consistency of constraints embedded in a db schema and the corresponding subschemas. Each execution of the consistency checking algorithm processes all constraints of a selected type. Therefore, consistency checking should be performed for each constraint type separately. Currently, IIS*Case supports detecting collisions of attribute sets, and the constraints of the following types: key and unique constraints, null value constraints, and referential integrity constraints. It generates the reports on detected collisions. Resolving collisions may lead to producing a new version of a db schema. In the following Section, we demonstrate applying IIS*Case in detecting collisions, together with an analysis of related reports and possible designer's actions.

4. Detecting and Resolving Collisions of Constraints in IIS*Case – A Case Study

We use a case study of an imaginary production system to illustrate applying IIS*Case in detecting and resolving collisions. The example is purposely simplified, in order to clearly present the process of detecting and resolving constraint collisions.

We identified three groups of similar user requirements:

- Personnel – managing personnel data, i.e. supporting insert, update and delete data about staff members;
- Working Units (WU) – managing WU data, i.e. supporting insert, update and delete data about working units and update some data about staff members belonging to a particular WU;
- Working Orders (WO) – supporting delete data about working orders.

For each of those groups, a set of form types is designed. Descriptions of the sets of form types designed in IIS*Case for the sake of this case study may be found in [10]. As an illustration, the IIS*Case form for specifying form types is shown in Fig. 4. It presents the form type *Staff* from the external schema *PERSONNEL*.

IIS*Case generates the following non-trivial inclusion dependencies at the level of the attributes of a Universal Relation Scheme (URS):

{[*ManagWU*] $\subseteq$ [*SSN*], [*Sign*] $\subseteq$ [*SSN*], [*Manag*] $\subseteq$ [*SSN*]},

as a reaction on a designer's decision to introduce new attributes by the renaming of existing ones, as follows: *ManagWU* from *SSN* (*Social Security Number*) for working unit's manager, *Manag* from *SSN* for an employee's manager, and *Sign* from *SSN* for an employee who signed up a working order.

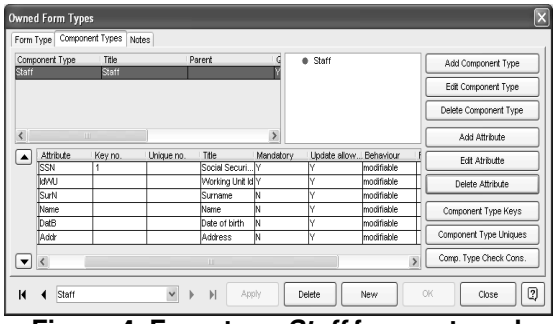


Figure 4. Form type *Staff* from external schema *PERSONNEL*

For each group of user requirements, IIS*Case maps form types into a relational subschema by inferring attributes and constraints from the form types and embedding them into a relational subschema. It also generates the appropriate reports about the db schema design progress. In this way, we obtain three subschemas: *PERSONNEL*, *WORKING_UNIT* and *WORKING_ORDER*. Each of them is presented in the following text, with its sets of relation schemes and interrelation constraints, where each relation scheme is represented as a named triple, with the following components: a set of attributes, a set of keys, and a set of unique constraints [17].

PERSONNEL

- *Staff*({*IdWU*, *SurN*, *DatB*, *Addr*, *SSN*, *Name*}, {*SSN*}, {})
- NULL values allowed only for the attributes: *SurN*, *Addr*, *Name*, *Datb*,
- Operations allowed in the relation scheme *Staff*: *read, insert, update, delete*
- *WU*({*IdWU*, *NamWU*, *ManagWU*},{*IdWU*}, {})
- NULL values allowed only for the attributes: *ManagWU*
- Operations allowed in the relation scheme *WU*: *read*, *insert*, *update*
- *WU*[*ManagWU*] $\subseteq$ *Staff*[*SSN*]
- *Staff*[*IdWU*] $\subseteq$ *WU*[*IdWU*]

WORKING_UNIT

· *Staff*({*IdWU*, *SurN*, *DatB*, *Addr*, *SSN*, *Name*, *School*, *IdS*, *Manag, CelTel*}, {*IdWU+IdS*, *SSN*}, {})

· NULL values allowed only for the attributes: *Addr*, *CelTel*

· Operations allowed in the relation scheme *Staff*: *read, insert, update, delete*

· *WU*({*WRoom*, *IdWU*, *NamWU*, *ManagWU*}, {*IdWU*}, {*NamWU*})

· NULL values not allowed for all the attributes

· Operations allowed in the relation scheme *WU*: *read*, *insert, update, delete*

· *WU*[*ManagWU*] ⊆ *Staff*[*SSN*]

· *Staff*[*IdWU*] ⊆ *WU*[*IdWU*]

· *Staff*[*Manag*] ⊆ *Staff*[*SSN*]

WORKING_ORDER

· *WO*({*IdWO*, *DatWO*, *Amount*, *IdPR*, *Sign*}, {*IdWO*}, {})

· NULL values allowed only for the attribute: *Sign*

· Operations allowed in the relation scheme *WO*: *read, delete*. □

IIS*Case produces the first version of a db schema (i.e. a potential db schema) by using synthesis algorithm, and independently designed subschemas. The order of integration is irrelevant.

IIS*Case performs the consistency checking over the potential db schema and all the specified subschemas, for each constraint type, separately. The order of selecting constraint types in the consistency checking procedure is relevant. IIS*Case imposes the following order of constraint types in consistency checking: checking of the attribute sets, the key and unique constraints, the null value constraints, and finally the referential integrity constraints. Successful execution of the procedure for a selected constraint type is a prerequisite for initiating the procedure for the subsequent constraint type. The consistency checking for the subsequent constraint types cannot be initiated, whereas the detected collisions are not resolved. The reports on detected collisions contain the explanations, how to interpret collisions. The structure of those reports, for different constraint types, will be presented in the following subsections.

The first condition that a db schema and a subschema have to satisfy is that the set of attributes of each relation scheme of the subschema must be a subset of the attribute set of at least one relation scheme of the db schema. A selected relation scheme satisfying the aforementioned condition is called the *corresponding database relation scheme*. Checking the collisions of the sets of attributes is the first step of the consistency checking process. Further discussion of the collision is omitted from the paper. More information and the examples may be found in [10], [18].

4.1. Key Constraint Collisions

A potential database schema *ADMINISTRATION* is generated using the subschemas *PERSONNEL*, *WORKING_UNIT* and *WORKING_ORDER*. It is structured as follows:

· *Staff*({*IdWU*, *SurN*, *DatB*, *Addr*, *SSN*, *Name*, *School*, *IdS*, *Manag*, *CelTel*}, {*IdWU+IdS*, *SSN*}, {})

· *WU*({*WRoom*, *IdWU*, *NamWU*, *ManagWU*}, {*IdWU*}, {*NamWU*})

· *WO*({*IdWO*, *DatWO*, *Amount*, *IdPR*, *Sign*}, {*IdWO*}, {})

· *WU*[*ManagWU*] ⊆ *Staff*[*SSN*]

· *Staff*[*IdWU*] ⊆ *WU*[*IdWU*]

· *Staff*[*Manag*] ⊆ *Staff*[*SSN*]

· *WO*[*Sign*] ⊆ *Staff*[*SSN*]. □

The analysis of the attribute set collisions finishes successfully, and the process continues by initiating the consistency checking of key constraints. A key constraint collision is detected, the process stops, and an appropriate report is generated. The first part of the report is shown in Fig. 5.

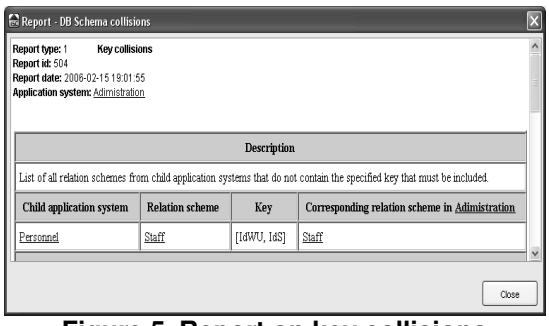


Figure 5. Report on key collisions

Relation scheme *Staff* in the subschema *PERSONNNEL* has *SSN* as the sole key, while its corresponding relation scheme *Staff* in the potential db schema *ADMINISTRATION* has two keys: *SSN* and *IdWU+IdS*. Operations *insert* and *update* are allowed for *Staff* in the subschema *PERSONNEL*. Since they may violate the key constraints, the key constraint *IdWU+IdS* is relevant for the subschema *PERSONNEL*, but it is not embedded into it. Furthermore, it cannot be

expressed using the concepts of subschema *PERSONNEL*, since its relation schemes do not contain the attribute *IdS*. This constraint is extending one. A designer initiates the process of resolving the collision.

According to Fig. 3, a designer may choose between four alternatives, in common. However, some of them may be inapplicable in the specific situation.

1. **The key constraint *IdWU+IdS* needs to be embedded into the db schema.**
   Consequently, the subschema *PERSONNEL* must be changed in one of the following ways.
   - **1.1. The operations *insert* and *update* of *IdWU* have to be removed from the relation scheme *Staff* in *PERSONNEL*.**
     A designer may decide so if he or she finds that the operations *insert* and *update* of *IdWU* are obsolete for *Staff* in *PERSONNEL*.
   - **1.2. The key constraint *IdWU+IdS* must be embedded into the subschema *PERSONNEL*.**
     A designer may decide so if he or she finds that the operations *insert* and *update* of *IdWU* are mandatory for *Staff* in *PERSONNEL*. Since the key constraint *IdWU+IdS* is extending one, in order to embed it into the subschema, a designer need to add the attribute *IdS* into the relation scheme *Staff* in *PERSONNEL*.
2. **The key constraint *IdWU+IdS* does not need to be embedded into the db schema.**
   Consequently, the subschema *WORKING_UNIT* must be changed in one of the following ways.
   - **2.1. The key constraint *IdWU+IdS* must be excluded from the relation scheme *Staff* in *WORKING_UNIT*.**
     A designer may decide so if he or she finds that the key constraint *IdWU+IdS* is obsolete for *Staff* in *WORKING_UNIT*.
   - **2.2. The key constraint *IdWU+IdS* must be pronounced as a locally valid one for the relation scheme *Staff* in *WORKING_UNIT*.**
     A designer may decide so if he or she finds that the key constraint *IdWU+IdS* is mandatory for *Staff* in *WORKING_UNIT*. The case needs an additional explanation. Namely, if the key constraint would be embedded into the subschema, but would not be embedded into the db schema, then it might cause duplicate values for *IdWU+IdS* in a db relation *WU*. If it would happen, a Database Management System (DBMS) could not select the tuples making the virtual relation over *WU* in *WORKING_UNIT*, unambiguously. A solution is to pronounce the key constraint *IdWU+IdS* as a locally valid in *WORKING_UNIT*, and also to embed the unique constraint *IdWU+IdS* into the relation scheme *WU* in the subschema *PERSONNEL*. In order to do that, since the constraint is extending one, a designer has to add the attribute *IdS* into the relation scheme *Staff* in *PERSONNEL*.

Selecting the one of the aforementioned alternatives depends on a designer's judgment. After selecting the most appropriate alternative and modifying the appropriate form types, IIS*Case generates a new subschema *PERSONNEL*, and/or a new subschema *WORKING_UNIT*, and also a new potential db schema *ADMINISTRATION*. Suppose that a designer selects the solution 1.1, in order to resolve the collision.

4.2. Unique Constraint Collisions

After resolving the key collision, reinitiated analyses of the attributes sets, and key collisions finish successfully, and the process continues by initiating the consistency checking of unique constraints. A unique constraint collision is detected, the process stops, and an appropriate report is generated. The first part of the report is shown in Fig. 6.

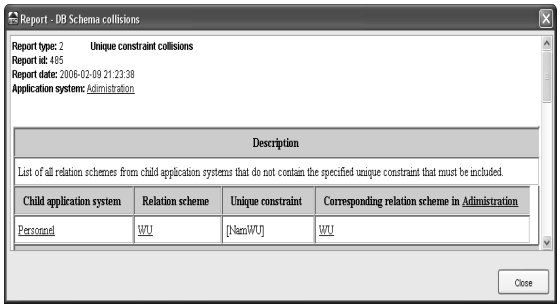


Figure 6 Report on unique constraint collisions

We may notice that the attribute *NamWU* in the relation scheme *WU* in the potential db schema *ADMINISTRATION* must have unique values. This constraint stems from the subschema *WORKING_UNIT*, and has been built into the db schema. However, it has not been embedded into the subschema *PERSONNEL*. There are four alternatives:

1. **The unique constraint *NamWU* needs to be embedded into the db schema.**
   Consequently, the subschema *PERSONNEL* must be changed in one of the following ways.

1.1. The operations *insert* and *update* of *NamWU* have to be removed from the relation scheme *WU* in *PERSONNEL*.
The solution is analogous to the solution 1.1. from Subsection 4.2, and therefore it is not commented here.

1.2. The unique constraint *NamWU* must be embedded into the subschema *PERSONNEL*.
Since the unique constraint *NamWU* is includible one, it is sufficient to add it in the appropriate form type in the external schema *PERSONNEL*.

2. Unique constraint *NamWU* does not need to be embedded into the db schema.
Consequently, the subschema *WORKING_UNIT* must be changed in one of the following ways.

2.1. The unique constraint *NamWU* must be excluded from the relation scheme *WU* in *WORKING_UNIT*.
A designer may decide so if he or she finds that *NamWU* is obsolete for the relation scheme *WU* in *WORKING_UNIT*.

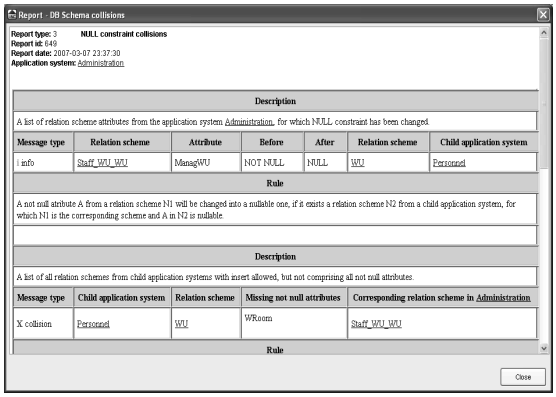
2.2. The unique constraint *NamWU* must be pronounced as locally valid in the subschema *WORKING_UNIT*.
In this particular case, this solution cannot be applied. Namely, if the unique constraint would be embedded into the subschema, but would not be embedded into the db schema, then it might cause duplicate values for *NamWU* in a db relation *WU*. If it would happen, a Database Management System (DBMS) could not select the tuples making the virtual relation over *WU* in *WORKING_UNIT*, unambiguously.

Therefore, a designer may choose between the first three alternatives, since the fourth one (2.2) is not a valid choice in the particular case. After selecting the most appropriate alternative, and modifying the appropriate form types, IIS*Case generates a new subschema *PERSONNEL*, or a new subschema *WORKING_UNIT*, and also a new potential db schema *ADMINISTRATION*.

Suppose that the collision is resolved by excluding the unique constraint over *NamWU* from the appropriate form type (2.1), and consequently from the relation scheme *WU* in the application system *WORKING_UNIT*.

4.3. Null Value Constraint Collisions

After resolving the unique constraint collision, reinitiated analyses of the attributes sets, key collisions and unique constraint collisions finish successfully, and the process continues by initiating the consistency checking of null value constraints. Some of the detected null value constraint collisions are automatically resolved. In the appropriate report, those changes are reported by the messages of type "info" (Fig. 7). The attribute *ManagWU* may have null values in the subschema *PERSONNEL*, whereas in the subschema *WORKING_UNIT* it must not. IIS*Case resolves the collision automatically by converting attribute *ManagWU* in db schema into the attribute with null values allowed. This change does not affect the form types from external schemas *PERSONNEL* and *WORKING_UNIT*. The null value constraint over the attribute *ManagWU* becomes a locally valid in the subschema *WORKING_UNIT*. Such a solution is formally valid, because a DBMS can select tuples making the virtual relation over *WU* in *WORKING_UNIT*, unambiguously.


Report - DB Schema collisions

Report type: 3 NULL constraint collisions
Report id: 649
Report date: 2007-03-07 23:37:30
Application system: Administration

Description
A list of relation scheme attributes from the application system Administration, for which NULL constraint has been changed.

Message type	Relation scheme	Attribute	Before	After	Relation scheme	Child application system
i info	Staff_WU_WU	ManagWU	NOT NULL	NULL	WU	Personnel

Rule
A not null attribute A from a relation scheme N1 will be changed into a nullable one, if it exists a relation scheme N2 from a child application system, for which N1 is the corresponding scheme and A in N2 is nullable.

Description
A list of all relation schemes from child application systems with insert allowed, but not comprising all not null attributes.

Message type	Child application system	Relation scheme	Missing not null attributes	Corresponding relation scheme in Administration
X collision	Personnel	WU	WRoom	Staff_WU_WU

Rule

Close

Figure 7 Report on NULL constraint collisions

Apart from automatic resolving collisions of the null value constraints, IIS*Case detects collisions of the null value constraints with insert operations, on all the relation schemes in child application systems that are declared for inserts. A collision arises if there is a relation scheme in child application system with insert operation allowed, but not containing all the not null attributes from the corresponding relation scheme. In the appropriate report (Fig. 7), such collisions are represented with the "collisions" message type.

In the case study, an *insert* operation is allowed for the relation scheme *WU* in the subschema *PERSONNEL*. Apparently, *WU* does not contain the attribute *WRoom*, whereas it is a not null attribute in the corresponding relation scheme in *ADMINISTRATION*. Possible designer's alternatives are:

1. The null value constraint for *WRoom* must be preserved in the db schema.
WRoom must be a not null attribute in *ADMINISTRATION*. Consequently, the subschema *PERSONNEL* must be changed in one of the following ways.

1.1. The operation *insert* has to be removed from the relation scheme *WU in PERSONNEL.*

1.2. The null value constraint *WRoom* must be embedded into the subschema *PERSONNEL.*

Since the null value constraint *WRoom* is extending one, in order to embed it into the subschema, designer needs to add the attribute *WRoom* into the relation scheme *WU* in the subschema *PERSONNEL.*

2. The null value constraint for *WRoom* must be removed from the db schema.

WRoom must be an attribute with nulls allowed in *ADMINISTRATION*. Consequently, the subschema *WORKING_UNIT* must be changed in one of the following ways.

2.1. The null value constraint *WRoom* must be removed from the relation scheme *WU* in *WORKING_UNIT.*

The attribute *WRoom* must be pronounced as optional one in the appropriate form type.

2.2. The null value constraint *WRoom* must be pronounced as locally valid in the subschema *WORKING_UNIT.*

The attribute *WRoom* may be pronounced as optional one in the appropriate form type, but with the operation "nullify a not null value" disallowed.

Suppose that the alternative (1.2) is chosen. After modifying the appropriate form types, IIS*Case generates a new subschema *PERSONNEL* and a new potential db schema *ADMINISTRATION*.

4.4. Referential Integrity Constraint Collisions

Reinitiated analyses of the attributes sets, key collisions, unique constraint collisions and null value collisions finish successfully. The final step is the consistency checking of the referential integrity constraints. After detecting collisions, IIS*Case produces an appropriate report (Fig. 8).

Two different message types may appear in the report: warnings and collisions. Collisions must be resolved, while warnings need not.

A warning is generated only if a subschema contains a referenced relation scheme but not the referencing one, and *delete* is an allowed operation for the referenced relation scheme in the subschema. There are two possible alternatives to resolve the warning: (i) disallowing the delete operation, or (ii) including the referencing relation scheme in the subschema. Selecting the second alternative may cause a repetitive including of a vast number of new relation schemes into the subschema. It may cause a subschema "overloading". Therefore, IIS*Case allows a designer to decide weather to resolve, or to ignore collisions of type "warning". A more detailed explanation of this problem may be found in [18].

In the subschema *WORKING_ORDER* the relation scheme *WO* contains the attribute *Sign.* Since the URS inclusion dependency [*Sign*] ⊆ [*SSN*] exists, the db schema *ADMINISTRATION* contains the referential integrity *WO*[*Sign*] ⊆ *Staff*[*SSN*], despite that it does not exist in the subschemas *PERSONNEL*, *WORKING_UNIT* and *WORKING_ORDER*. Delete is an allowed operation for the relation scheme *Staff* in the subschema *PERSONNEL*, as well as in the subschema *WORKING_UNIT*. A designer decides not to resolve the warnings, since it does not reflect either on the database consistency, or "commodity" of end users.

In the report from Fig.8, there is a collision concerning referential integrity constraint *R1*: *Staff*[*Manag*] ⊆ *Staff*[*SSN*]. In the subschema *WORKING_UNIT*, the relation scheme *Staff* contains the attribute *Manag*, and participates the constraint *R1*. Consequently, the db schema contains the same referential integrity constraint, and *Manag* belongs to the set of attributes of the db relation scheme *Staff*. The operation *delete* is allowed for the relation scheme *Staff* in the subschema *PERSONNEL*. Therefore, the referential integrity constraint *R1* is relevant for *PERSONNEL*, and the operation delete may violate it. Possible designer's alternatives are:

1. The referential integrity constraint *R1* must be preserved in the db schema.

Consequently, the subschema *PERSONNEL* must be changed in one of the following ways.

1.1. The operation *delete* has to be removed from the relation scheme *Staff* in *PERSONNEL.*

1.2. The referential integrity constraint *R1* must be embedded into the subschema *PERSONNEL.*

Since the referential integrity constraint *R1* is extending one, in order to embed it into the subschema, a designer has to add the attribute *Manag* into the relation scheme *Staff* in the subschema *PERSONNEL*. The referential integrity constraint *R1* is generated automatically, during the db schema design.

2. The referential integrity constraint *R1* must be removed from the db schema.

Consequently, the subschema *WORKING_UNIT* must be changed in one of the following ways.

2.1. The referential integrity constraint *R1* must be excluded from the relation scheme *Staff in WORKING_UNIT.*

One option is to exclude the attribute *Manag* from the relation scheme *WU* by changing appropriate form type. Another one is to delete the URS inclusion dependency [*Manag*] ⊆ [*SSN*], although it is not advisable, since the set of URS constraints is changed.

2.2. The referential integrity constraint *R1* must be pronounced as locally valid in the subschema *WORKING_UNIT*.

In this particular case, this solution cannot be applied. Namely, the existence of basic and extended referential integrity constraints is a consequence of the primary key propagation [18]. Referential integrity constraints based on non-trivial inclusion dependencies arise from the URS non-trivial inclusion dependencies that a designer defines at the level of the set of all information system attributes [18]. Consequently, it is not possible to pronounce a referential integrity as locally valid.

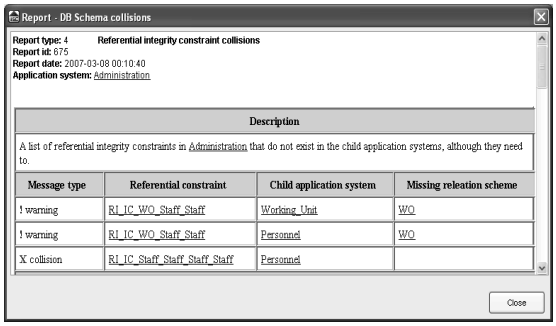


Figure 8 Report on referential integrity collisions

In this case, a designer may choose only between first three alternatives. After selecting the most appropriate alternative, and modifying the appropriate form types, IIS*Case generates a new subschema *PERSONNEL*, and/or a new subschema *WORKING_UNIT*, and also a new potential db schema *ADMINISTRATION*.

Suppose a designer chooses the alternative 1.2. The following final versions of subschemas *PERSONNEL*, *WORKING_UNIT* and *WORKING-_ORDER*, and the final db schema *ADMINISTRATION* are obtained, where the differences with respect to the previous version are written in bold style:

PERSONNEL

- *Staff*({*IdWU*, *SurN*, *DatB*, *Addr*, *SSN*, *Name*, ***Manag***}, {*SSN*}, {})
- NULL values allowed for the attributes: *SurN*, *Addr*, *Name*
- Operations allowed in the relation scheme: ***read, delete***
- *WU*({***WRoom***, *IdWU*, *NamWU*, *ManagWU*}, {*IdWU*}, {})
- NULL values allowed for the attributes: *ManagWU*
- Operations allowed in the relation scheme: *read*, *insert*, *update*
- *WU*[*ManagWU*] ⊆ *Staff*[*SSN*]
- *Staff*[*IdWU*] ⊆ *WU*[*IdWU*]
- ***Staff*[*Manag*] ⊆ *Staff*[*SSN*]**

WORKING_UNIT

- *Staff*({*IdWU*, *SurN*, *DatB*, *Addr*, *SSN*, *Name*, *School*, *IdS*, *Manag, CelTel*}, {*IdWU+IdS*, *SSN*}, {})
- NULL values allowed for the attributes: *Addr*
- Operations allowed in the relation scheme *Staff*: *read*, *insert*, *update*, *delete*
- *WU*({*WRoom*, *IdWU*, *NamWU*, *ManagWU*} {*IdWU*}, **{ }**),
- Operations allowed in the relation scheme *WU*: *read*, *insert*, *update*, *delete*
- *WU*[*ManagWU*] ⊆ *Staff*[*SSN*]
- *Staff*[*IdWU*] ⊆ *WU*[*IdWU*]
- *Staff*[*Manag*] ⊆ *Staff*[*SSN*]

WORKING_ORDER

- *WO*({*IdWO*, *DatWO*, *Amount*, *IdPR*, *Sign*}, {*IdWO*}, {})
- NULL values allowed for the attributes: *Sign*
- Operations allowed in the relation scheme *WO*: *read, delete*

ADMINISTRATION

- *Staff*({*IdWU*, *SurN*, *DatB*, *Addr*, *SSN*, *Name*, *School*, *IdS*, *Manag, CelTel*}, {*IdWU+IdS*, *SSN*}, {})
- NULL values allowed for the attributes: *SurN*, *Addr*, *Name*
- Operations allowed in the relation scheme *Staff*: *read*, *insert*, *update*, *delete*
- *WU*({*WRoom*, *IdWU*, *NamWU*, *ManagWU*}, {*IdWU*}, {})
- NULL values allowed for the attributes: ***ManagWU***
- Operations allowed in the relation scheme *WU*: *read*, *insert*, *update*, *delete*

- *WO*({*IdWO*, *DatWO*, *Amount*, *IdPR*, *Sign*}, {*IdWO*}, {})
- NULL values allowed for the attributes: *Sign*
- Operations allowed in the relation scheme *WO*: *read*, *delete*
- *WU*[*ManagWU*] ⊆ *Staff*[*SSN*]
- *Staff*[*IdWU*] ⊆ *WU*[*IdWU*]
- *Staff*[*Manag*] ⊆ *Staff*[*SSN*]
- *WO*[*Sign*] ⊆ *Staff*[*SSN*]. □

During the consolidation process, designers may also change the structure of application systems, i.e. the sets of form types (i.e. external schemas). Afterwards, IIS*Case generates subschemas and integrates them into a db schema. Therefore, when the consolidation process successfully finishes, a consistent set of subschemas and consistent sets of form types are obtained. IIS*Case consolidates not only the attribute sets and the constraint sets, but also the sets of allowed operations and modifiable attributes. Form types carry additional information about transaction programs and their screen forms. Consequently, transaction programs generated over such form types will be in accordance with the designed db schema.

5. Conclusion

IIS*Case supports collaborative work of designers so as to reach the most appropriate solutions through their cooperation. A designer may devote his or her time and power to analysis and modelling business processes and rules. The db design of even complex information systems may be an easier task if it would be based on this approach and IIS*Case, because the process of modelling is raised to the level, which is closer to the users without an advanced knowledge of the database design.

IIS*Case is developed on the basis of the results of a theoretical research presented in [2], [3], [4], [5], [8], [10] and [18]. The principles of database updates using subschemas are introduced in [9], and we argue that a subschema and the corresponding db schema must satisfy certain formal conditions to allow safe database updates using a program utilizing the concepts of a subschema. Such conditions are formulated at the abstraction level of instances. Using them, we were able to formulate the conditions of formal consistency, and develop the algorithm for checking the formal consistency of db schema constraints. The algorithm is embedded into IIS*Case. Therefore, detecting and resolving collisions is an important activity in the db schema design process supported by IIS*Case. The specificity of our approach is in that the collisions are not detected between different subschemas, but between a db schema and a set of subschemas, since the integration process is not mere unifying of subschemas. The process of detecting and resolving collisions may also help designers to recognize new database constraints, which have not been previously identified.

The collisions are resolved by interrupting the process of a db schema integration and making changes in the subschemas, i.e. application subsystems. Therefore, the integration process must be restarted from the point of origin of a collision. Sometimes, it must be restarted from the very beginning, and this is a side effect of our approach. Also, the resolving of some collisions may cause new collisions. Even more, such new collisions may concern constrains of the different types that were already successfully passed consistency checking. However, our primary goal of proposing the approach presented here was not only to make the design process of complex db schemas easy. Instead, we intended to make such a tool and the approach that would considerably improve the quality of resulting db schemas, in contrast to applying an intuitive approach, and make the design process faster and easier, at the same time.

At present, IIS*Case R.6.21 produces a formal specification of an implementation database schema. It also has an SQL generator that supports generating SQL specifications of a database schema for different DBMSs. Further research and development efforts are oriented towards extending current functionality. In the scope of the approach presented in the paper, we are planning to make further improvements of the algorithms for consistency checking and db schema integration. Those improvements should cover consistency checking for the following constraint types: check constraints, extended referential integrity constraints, and inverse referential integrity constraints [3].

References

MONOGRAPHS

[1] Date C. J., Darwen H., *Foundation for Object/Relational Databases: The Third Manifesto*, Addison-Wesley Professional, 1998.

[2] Govedarica M., *An Automated Development of Information System Application Prototypes*, PhD Thesis, University of Novi Sad, Faculty of Technical Sciences, Novi Sad, Serbia and Montenegro, 2002.

[3] Mogin P., Lukovic I., Govedarica M., *Database Design Principles*, 2nd Edition, University of Novi Sad, Faculty of Technical Sciences, Novi Sad, Serbia and Montenegro, 2004.

[4] Pavicevic J., *Development of A CASE Tool for Automated Design and Integration of Database Schemas*, M.Sc. Dissertation, University of Montenegro, Faculty of Science, Podgorica, Serbia and Montenegro, 2005.

[5] Ristic S., *Research of Subschema Consolidation Problem*, PhD Thesis, University of Novi Sad, Faculty of Economics, Subotica, Serbia and Montenegro, 2003.

COLLECTIONS

[6] Diet J., Lochovsky F., Interactive Specification and Integration of User Views Using Forms, *Proceedings of the Eight International Conference on Entity-Relationship Approach Toronto*, Canada 18-20. October, 1989, pp.171-185.

[7] Gálvez S., Guevara A., Caro J. L., Gómez I., Aguayo A., Collaboration Techniques to Design a Database, *Universidad de Málaga*, Spain, 2004.

[8] Lukovic I., Ristic S., Mogin P., A Methodology of A Database Schema Design Using The Subschemas, *IEEE International Conference on Computational Cybernetics,* Siofok, Hungary, August 29-31, 2003

[9] Ristić S., Luković I., Mogin P., Specifying Database Updates Using a Subschema 7th *IEEE International Conference on Intelligent Engineering Systems INES 2003*, Assiut-Luxor, Egypt, 4–6 March, 2003, Proceedings Vol. 1, pp. 203–212.

[10] Ristic S., Lukovic I., Mogin P., Pavicevic J., Integrating a Database Schema Using IIS*Case Tool, *13th Scientific Conference on Industrial Systems IS'05*, Herceg Novi, September 07 – 09, 2005.

[11] Schmalz M. S., Hammer J., Wu M., Topsakal O., EITH – A Unifying Representation for Database Schema and Application Code in Enterprise Knowledge Extraction, Proceedings of the *22nd International Conference on Conceptual Modeling*, Chicago, IL, November 2003.

JOURNAL ARTICLES

[12] Beeri C., Bernstein P. A., Computational Problems Related to the Design of Normal Form Relational Schemas, *ACM Transactions on Database Systems*, Vol.4, No.1, March, 1979, pp. 30-59.

[13] Choobinch J., Mannio V. M., Nunamaker F. J., Konsynski R. B., An Expert Database Design System Based on Analysis of Forms, *IEEE Transactions on Software Engineering*, Vol.14, No 2, Feb. 1988, pp. 242-253.

[14] Date C. J., Composite Foreign Keys and Nulls, *In C.J. Date and H. Darwen Relational Database Writings* 1989-1991, Addison-Wesley Publishing Company, Reading, Massachusetts, 1992.

[15] Govedarica M., Lukovic I., Mogin P., Generating XML Based Specifications of Information Systems, *Computer Science and Information Systems* (*ComSIS*), Belgrade, Serbia and Montenegro, Vol. 1, No. 1, 2004, pp. 117-140.

[16] Lukovic I., Govedarica M., Mogin P., Ristic S., The Structure of A Subschema and Its XML Specification, *Journal of Information and Organizational Sciences*, Varazdin, Croatia, Vol. 26, No. 1-2, 2002, pp. 69-85.

[17] Lukovic I., Ristic S., Mogin P., Pavicevic J. Database Schema Integration Process – A Methodology and Aspects of Its Applying, *Novi Sad Journal of Mathematics* (*Formerly Review of Research, Faculty of Science, Mathematic Series*), Novi Sad, 2006, pp. 115 – 140.

[18] Lukovic I, Mogin P, Pavicevic J, Ristic S, An Approach to Developing Complex Database Schemas Using Form Types, *Software: Practice and Experience, John Wiley & Sons Inc*, Hoboken, USA, ISSN: 0038-0644, Accepted for publication, February 11, 2007.

WEB SOURCES

[19] ARTech, *DeKlarit™* (*The Model-Driven Tool for Microsoft Visual Studio 2005*), Chicago, USA, Available: http://www.deklarit.com, current: December 2006.