

DATABASE SCHEMA INTEGRATION PROCESS – A METHODOLOGY AND ASPECTS OF ITS APPLYING

Ivan Luković¹, Sonja Ristić², Pavle Mogin³, Jelena Pavićević⁴

Abstract. The paper considers an original approach to the process of integration of complex database schemas. Apart from the proposed theoretical model of the approach, we also developed a CASE tool, named IIS*Case (Integrated Information Systems*Case, R.6.0), which enables its practical application.

The basis of the approach aimed at designing and integration of database schemas and the ways of using IIS*Case are outlined. The main functionalities for database schema integration embedded in a new version of IIS*Case, which is developed in Java, are described.

IIS*Case supports conceptual modelling of a database schema, generating subschemas and their integration in a relational database schema in 3NF. It is based on a methodology of gradual integration of independently designed subschemas into a database schema. The process of independent subschema design may lead to collisions in expressing the real world constraints. IIS*Case uses specialized algorithms for checking the consistency of constraints embedded in the database schema and subschemas. The paper outlines the process of detecting collisions and possible designer's actions aimed at their resolving.

AMS Mathematics Subject Classification (2000):

Key words and phrases: Database Schema Design and Integration; Subschema, Form Type, CASE tool, Formal consistency, IIS*Case

1. Introduction

The conceptual modelling of a database (db) schema is mainly based on the Entity-Relationship (ER) model or Unified Modelling Language (UML) class diagrams. Today, many software tools, which rely on these techniques, support: (i) conceptual design of db schemas, (ii) transformation of conceptual db schemas into implementation (mainly relational) db schemas, and (iii) their implementing under different DBMSs.

¹Ph.D., Full Professor, University of Novi Sad, Faculty of Technical Sciences, Trg D. Obradovića 6, 21000 Novi Sad, Serbia, e-mail: ivan@uns.ns.ac.yu, Office: +38121 4852445

²Ph.D., Assistant Professor, University of Novi Sad, Faculty of Technical Sciences, Trg D. Obradovića 6, 21000 Novi Sad, Serbia, e-mail: sdristic@uns.ns.ac.yu

³Ph.D., Senior Lecturer, Victoria University of Wellington. P.O. Box 600, Wellington New Zealand, e-mail: pmogin@mcs.vuw.ac.nz

⁴M.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

As it concerns the process of db schema design, generally, there are two basic approaches: (a) the direct approach, and (b) the approach of a gradual integration of external schemas [18].

In the direct approach, user requests are processed all at once and the whole db schema is created directly. This approach may be appropriate in the cases of designing small db schemas, but it is inappropriate in cases when a complex db schema should be designed.

The second approach is used when the number and complexity of user requests are beyond the designer's power of perception. 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 user's view on a db schema. Each transaction program that supports a user request is based on an external schema, which is associated to it. After being created, external schemas are integrated into a conceptual db schema.

The use of the design methodologies based on the second approach and techniques like ER modelling or UML, and also relational data model and the appropriate CASE tools, requires an advanced designer's knowledge and high perception power. It is not an easy job, and sometimes with uncertain outcomes, to find and educate the appropriate number of designers that possess these skills. It may raise the risk of designing a db schema of poor quality.

Besides, these methods and techniques are often incomprehensible to end-users. In practice, this may sometimes lead to problems in communication and misunderstandings between designers and end-users. As a rule, misunderstandings cause poor quality of the designed db schema because the ability to support all the specified user requests will not be ensured. Usually, both designers and end-users become aware of that too late, when the db schema is already implemented.

Therefore, it is a challenge to provide an alternative approach and a CASE tool, which may support an automated db schema design, so that it is based on the concepts end-users are familiar with. A designer who understands and follows the rules for creating design specifications imposed by such tool, would be able to design db schemas quicker and easier, even if their complexity extends beyond the limits of human perception.

IIS*Case (Integrated Information Systems*Case, R.6.0) is a tool that we developed to support such an alternative approach. The main characteristics of IIS*Case and the proposed approach have been already presented in [20, 22, 14]. We believe that the approach and IIS*Case may help in resolving or just alleviating the aforementioned problems. The approach is tested on practical examples and applied in some projects, in which the authors took part as project team members.

This paper considers the approach to the process of integration of complex db schemas that is supported by IIS*Case. It outlines the basis of the approach methodology that is based on the so-called form type concept. The main functionalities for db schema integration embedded in the new version of IIS*Case (developed under Java environment) are presented. Despite that some origins, ideas and similarities could be recognized in references of the other authors,

for example [3, 6, 8], by our current knowledge, the approach and CASE tool presented here cannot be found in the same form elsewhere.

IIS*Case is designed to provide complete support for developing db schemas, which are complex with regard to the number of concepts used, and to give an intelligent support in the course of that process. IIS*Case supports:

- Conceptual modelling of external schemas,
- Automated design of the so-called relational database subschemas in the 3rd normal form,
- Automated integration of relational db schema from designed subschemas, and
- Detecting and resolving the constraint collisions between a db schema and a set of subschemas.

By integrating independently designed subschemas, IIS*Case produces the first version of the db schema. The process of independent design of external schemas 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 potential db schema. Programs made over an inconsistent subschema do not guarantee safe database updates.

IIS*Case uses specialized algorithms to check the consistency of constraints embedded in a db schema and the corresponding subschemas. Consistency checking is performed for each constraint type separately. IIS*Case generates two types of reports: reports on db schema design process and reports on detected collisions. Resolving collisions may lead to producing a new version of a db schema. The paper discusses the application of IIS*Case [19] in detecting collisions, including the analysis of related reports and possible designer's actions. Examples given in the paper illustrate how IIS*Case detects constraint collisions.

Apart from Introduction, the paper consists of seven sections. In Section 2, we discuss related work. In Section 3, a survey of our approach is given. In Section 4, a concept of form type, subschema and db schema are introduced. Section 5 is devoted to the constraint collisions. Section 6 gives an example of applying IIS*Case in the process of detecting and resolving collisions between designed subschemas. In Section 7 we give conclusions, while in Section 8 we discuss further research and development.

2. Related Work

In our approach we assume that the form type concept may be used for conceptual db schema design instead of ER data model. Thus, we concentrate on creating procedures for generating relational db schema using the specifications of form types, in order to widely utilize powerful mathematical formalisms relational data model is based on. Our main idea origins from the late 80's. Some

of the references that reflect its developing and implementing are [9, 10, 13, 15, 16, 17, 18, 19, 23].

In [8], the authors present methods for the analysis and design of cooperative object-oriented information systems. A model of information system is split into several subsystems that can be handled more easily. Interrelationships existing between such subsystems require that the development methods are applied cooperatively. Cooperative tools, based on a powerful and user-friendly graphic interface and working over a so-called Cooperative Data Dictionary (CDD), are introduced. The concept of form in [8] is very similar to our concept of form type. However, IIS*Case form type may be structured in a way to provide more information about various types of relational db schema constraints. Besides, it carries additional information about the embedded functionality of future transaction programs made over such a form type. A set of subschemas generated by IIS*Case also contains information about database constraints and allowable database operations. Therefore, a subschema becomes a component of a program specification. It contributes to the goal that such specifications may be implemented in many different ways, under many, even technologically different, programming systems. It will enable implementing a code generator for applying prototypes that are executable in different programming environments.

Since different users may create designs, conflicts may arise. The authors in [8] distinguish structural and semantic incompatibilities between designs. Structural conflicts arise when the same attribute is included in different schemas with different type and format definitions. Semantic conflicts are caused by homonyms and synonyms. Detecting the conflicts is possible by a conflict management system in CDD and it takes place at the level of the designs created. On the other hand, IIS*Case provides more powerful consistency control at the level of generated relational subschemas that should be integrated into a unique relational db schema. Apart from detecting homonyms, the process of consistency checking identifies collisions for various constraint types. It is even able to automatically resolve some of them.

In [24], the authors consider the problem of integrating heterogeneous legacy databases. They introduce a formalism which provides a way for logical integration of heterogeneous databases, called EITH. It supports translation of db schemas expressed in various data models (particularly ER) into a unified abstract representation. The main idea of a unified representation of db schema and application code appearing in [24] is also utilized in our methodological approach, based on the concept of form type. However, unlike [24], our main concern is the process of designing and integration of a unified db schema of an information system.

3. A Survey of the Approach

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 request is based on an external schema, which is associated to it. The modelling of the external schemas is mainly based on the ER data model or UML class diagrams. In contrast to some other approaches, in this approach that is supported by IIS*Case, external schemas are expressed by sets of the *form types*. Form type generalizes document types, i.e. screen forms that users utilize to communicate with an information system. IIS*Case imposes strict structuring rules for form types. Using this tool, a designer specifies screen forms of transaction programs and, indirectly, creates an initial set of attributes and constraints. The main motivation of introducing form type is using a concept that is more familiar to end-users' perception of information system, than it would be, for example, the concepts of entity and relationship types in ER data model. On the other hand, form type is a concept that is formal enough to precisely express all the rules significant for structuring future db schema.

After being created, external schemas are integrated into a conceptual db schema. Unfortunately, it is difficult and sometimes even impossible, to formalize the process of integration of the external schemas. The quality of the resulting db schema highly depends on the designer's knowledge and skillfulness. In contrast to other data models, relational data model has much wider possibilities to formalize and automate the process of integration of external schemas in a single db schema. Database schema integration in IIS*Case is done at the implementation level, where the db schema is expressed by relational data model. A db schema is the result of the gradual integration of *subschemas*. A *subschema* is obtained by expressing an external schema by concepts of the relational data model. The integration of relational subschemas can be supported by applying the synthesis algorithm [1, 18].

The process of independent design of external schemas may lead to collisions in expressing the real world constraints and business rules. The integration process, which is supported by our approach and IIS*Case, is not a simple unifying the generated subschemas. By integrating independently designed subschemas, IIS*Case produces the first version of the db schema. This db schema we call the *potential database schema*. If collisions are detected, at least one subschema is formally inconsistent with the potential database schema. The programs made over inconsistent subschemas do not guarantee safe database updates. Two subschemas may be integrated only if they contain compatible sets of constraints. Thus, the appropriate procedures for resolving possible collisions, which arise as a result of independent modelling of subschemas, must be applied. 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. The process of detecting and resolving constraint collisions we call consolidation of the db schema and subschemas.

Therefore, the db schema design of a complex information system in IIS*Case is organized by decomposing it into application systems [19]. *Application system* is a specification of one subsystem of the future information system and initially contains a set of form types. During the automated process of design,

an implementation db schema is generated on the basis of the set of form types. Therefore, the input specification of the application system is the set of form types. The design process produces new specifications that become additional components of the application system.

The set of all application systems of an information system is organized as a tree structure. It is a *project tree* of the information system. Thus, each application system may include one or more child application systems. All child application systems of an application system are called its *application subsystems*. Figure 1 depicts two different project trees in IIS*Case: "Factory" and "Web Hosting".

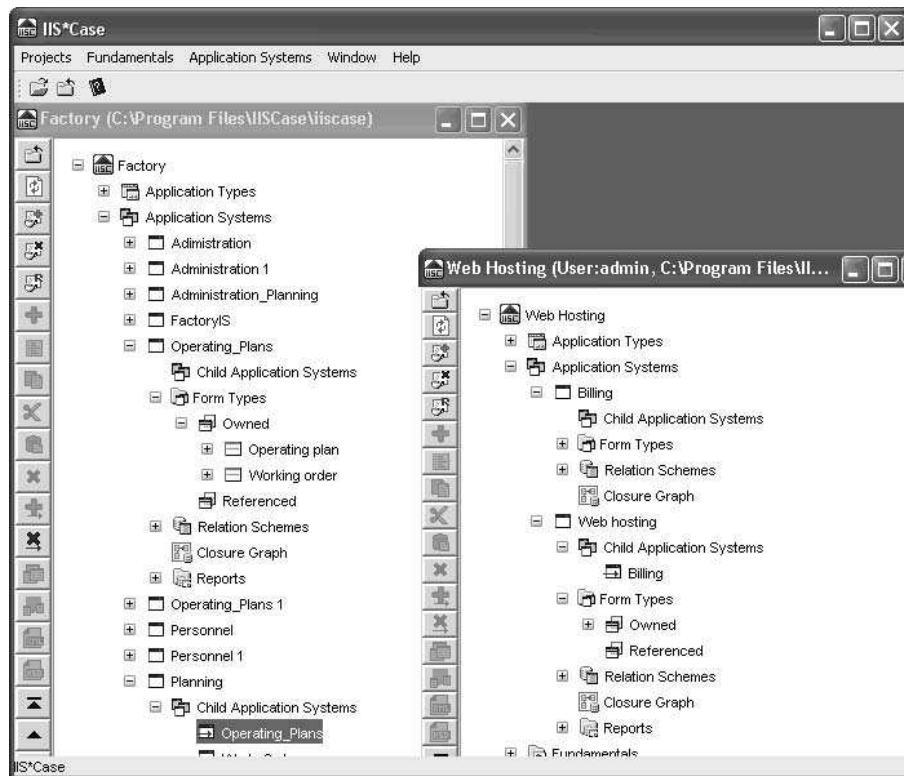


Figure 1: Project trees in IIS*Case

The db schema design supported by IIS*Case is an iterative process that includes the following phases:

- Conceptual modelling of a set of form types (i.e. an external schema) for each group of similar end-user business tasks,
- Generating a relational db schema of an application system with sub-schemas that belong to its application subsystems, and

- Consolidation of the integrated schema and subschemas.

Figure 2 depicts phases of the db schema design process.

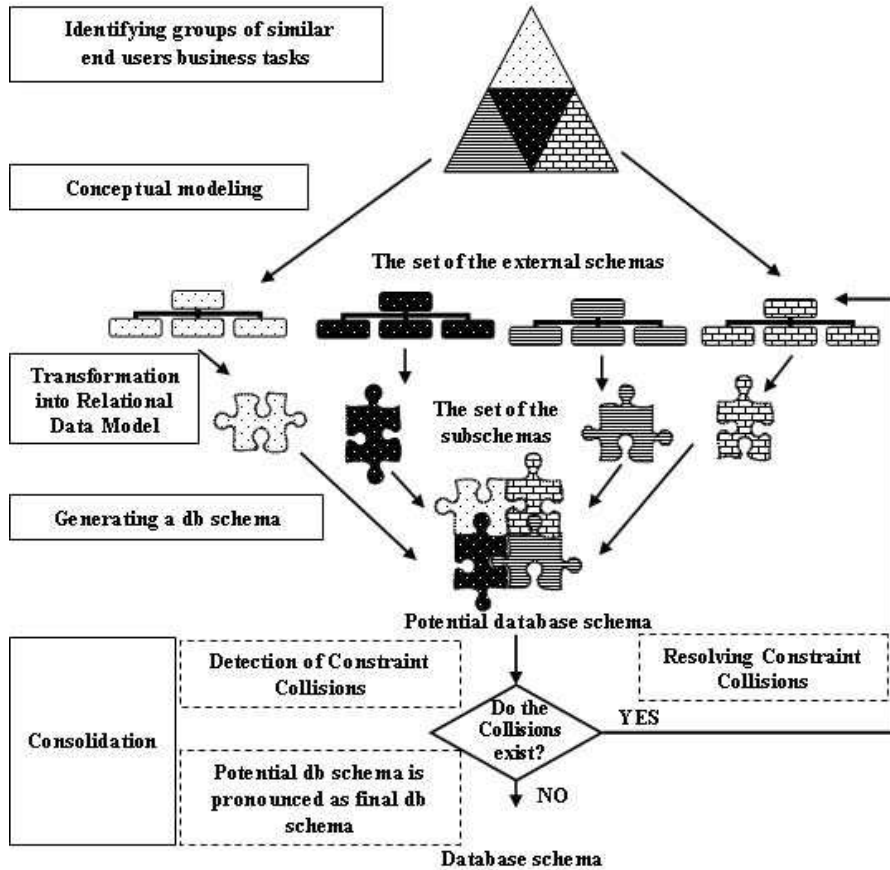


Figure 2: Phases of db schema design process

A designer is supported by IIS*Case to review or validate obtained results after each step during the design. 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 back one or more steps, make changes on form types and repeat the process.

4. Subschema Design

The *input specification* of the process of generating db schema of one application system, supported by IIS*Case, is the union of sets of form types of a chosen application system and all its application subsystems.

A *form type* is a named tree structure, whose nodes are called *component types*. Each component type is identified by its name in the scope of the form type, and has nonempty sets of attributes and keys, and a set of unique constraints that may be empty. Besides, to each component type must be associated a set of allowed database operations. It must be a nonempty subset of the set of "standard" operations $\{retrieve, insert, update, delete\}$. Each attribute of a component type is chosen from the set of all information system attributes. The attributes are globally identified only by their names. IIS*Case imposes strict rules for specifying attributes and their domains.

Example 1 Figure 4 shows the structure of a form type F , which generalizes a screen form from Figure 3. The form type consists of one component type: OP , which is graphically represented by rectangle. The set of component type attributes are also shown. Attributes underlined with solid lines indicate component type keys. A form for specifying form types in IIS*Case is presented in Figure 5.

Operating Plans		
Identification	From Date	Until Date

Figure 3: An example of a screen form

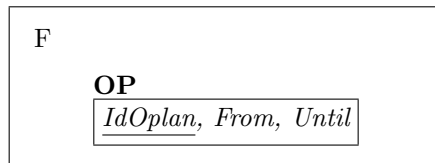


Figure 4: Form type

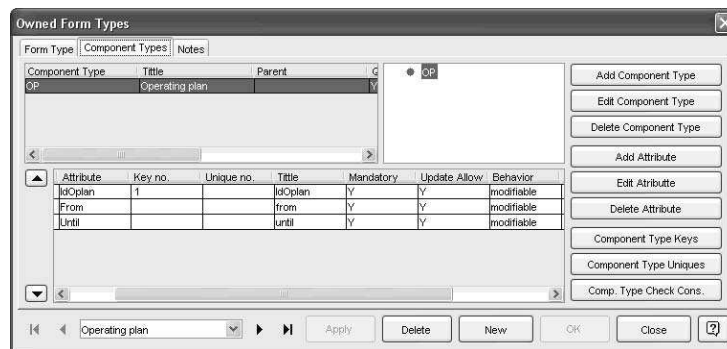


Figure 5: IIS*Case specification of a form type OP (Operating plan)

The process of generating db schema of one application system, supported by IIS*Case, consists of the following five steps:

- Generating a set of relation schemes by the synthesis algorithm [1, 18, 19],
- Generating the closure graph [19],
- Selecting candidates for primary keys [18, 19],
- Propagating primary keys [18, 19], and
- Generating interrelation constraints [18, 19].

All the steps are executed in the specified order, on a designer's demand. Figure 6 represents the screen form of IIS*Case by means of which the designer controls the process of db schema generation for a chosen application system.

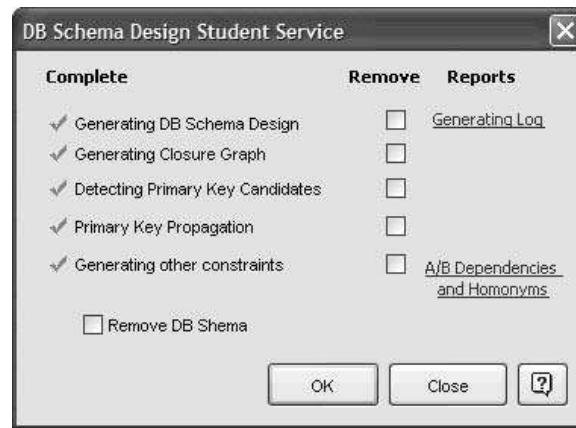


Figure 6: IIS*Case form for controlling the process of generating a db schema

In this way, a db schema of a chosen application system is generated. If the application system is at the same time a subsystem of another application system, then the generated db schema becomes a *subschema*. Such subschema has to be subjected to the integration process with other subschemas of the same parent application system.

Formally, a *subschema* is a named pair $P_k(S_k, I_k)$, where P_k is a subschema name, S_k is a nonempty set of relation schemes, and I_k is a set of interrelation constraints. The set of relation schemes of a subschema P_k is:

$$S_k = \{N_i^k(R_i^k, C_i^k, K_p(R_i^k), Role(P_k, N_i^k), Mod(P_k, N_i^k), \\ Sr(P_k, N_i^k), LK_i^k) \mid i \in \{1, \dots, n\}\},$$

where N_i^k is a scheme name, R_i^k is an attribute set, C_i^k is a specification of relation constraints of the form $(K_i^k, \tau(N_i^k), Uniq(N_i^k))$, where K_i^k is a set of

keys, $\tau(N_i^k)$ will be called *tuple integrity constraint*, and $Uniq(N_i^k)$ is a (possible empty) set of *uniqueness constraints* $Unique(N_i^k, X_i^{km})$, where X_i^{km} is a proper subset of R_i^k , which does not contain any key from K_i^k . A uniqueness constraint $Unique(N_i^k, X_i^{km})$ means that each non null value of X_i^{km} must be unique in a relation over N_i^k . The tuple integrity constraint $\tau(N_i^k)$ is a pair $\tau(N_i^k) = (\{\tau(N_i^k, A) | A \in R_i^k\}, Con(N_i^k))$, whose first component contains attribute domain constraints $\tau(N_i^k, A)$ of each attribute $A \in R_i^k$. Each $\tau(N_i^k, A)$ is of the form $\tau(N_i^k, A) = (Dom(N_i^k, A), Null(N_i^k, A))$, where $Dom(N_i^k, A)$ is a domain constraint of attribute $A \in R_i^k$ and $Null(N_i^k, A) \in \{\perp, T\}$ is a null-value constraint of $A \in R_i^k$. The second component of $\tau(N_i^k)$, $Con(N_i^k)$ is a logical expression defined over the attributes from R_i^k and their domain values. It must be satisfied by each tuple from an instance over N_i^k , whose first component contains attribute domain constraints of each attribute $A \in R_i^k$. $K_p(R_i^k)$ is a primary key. The detailed explanation of the relation constraint specifications is beyond the scope of the paper, and can be found in [15].

$Role(P_k, N_i^k)$ is a set of relation scheme *roles* and it defines the operations that may be performed on an instance of the relation scheme N_i^k . Only these operations may be built into a transaction program made using the concepts of a subschema P_k . A set of relation scheme roles is a nonempty set, for which $Role(P_k, N_i^k) \subseteq \{r, i, m, d\}$ holds, where: r stands for data reading, i.e. referencing, i for insert, m for modification and d for data deleting.

A subschema P_k is intended for database *querying* only if $(\forall N_i^k \in S_k) (Role(P_k, N_i^k) = \{r\})$ holds. Otherwise, it is intended for *updating*, and *querying*.

The set $Mod(P_k, N_i^k)$ contains those attributes of the relation scheme N_i^k that may be modified. If $m \in Role(P_k, N_i^k)$, then $Mod(P_k, N_i^k)$ must not equal \emptyset .

For each $N_i^k \in S_k$ the db schema must have at least one relation scheme that contains the attribute set of N_i^k . One of such relation schemes is pronounced as the *corresponding relation scheme* for N_i^k . Often, there is only one relation scheme with the given property. However, if there are more, IIS*Case will choose the one with the maximum set of attributes. $Sr(P_k, N_i^k)$ denotes the corresponding relation scheme for N_i^k .

Some of the relation scheme keys may be pronounced as local keys. If $X \in LK_i^k$ then each value of X must be unique and not null in a relation over N_i^k , belonging to the instance of the subschema P_k . At the same time, in a relation over $N = Sr(P_k, N_i^k)$, belonging to the db instance, a value of X may be null, but each non-null value of X must be unique.

Named 7-tuple $N_i^k(R_i^k, C_i^k, K_p(R_i^k), Role(P_k, N_i^k), Mod(P_k, N_i^k), Sr(P_k, N_i^k), LK_i^k)$ is the specification of the subschema relation scheme. In the following text an abbreviated notation $N_i^k(R_i^k, K_i^k)$, where R_i^k is the attribute set and K_i^k is the set of keys, will be used when it is appropriate.

The set of interrelation constraints I_i^k may contain various types of constraints such as: basic and extended referential integrity constraints, referential integrity constraints based on non-trivial inclusion dependencies, inverse referential integrity constraints, basic and those based on non-trivial inclusion dependencies, etc.

A subschema is a formal and abstract definition of data, constraints, and database update activities that are needed to make a transaction program. A transaction program issues query and update commands that are executed by a DBMS against a database instance. That is the main reason why $Role(P_k, N_i^k)$, $Mod(P_k, N_i^k)$, $Sr(P_k, N_i^k)$ and LK_i^k are parts of the relation scheme specification in the subschema.

On the other hand, the relation scheme specification in a db schema does not contain $Role(P_k, N_i^k)$, $Mod(P_k, N_i^k)$, $Sr(P_k, N_i^k)$ and LK_i^k . Thus, a relational db schema is a pair (S, I) , where S is a set of relation schemes and I is a set of interrelation constraints. I may contain the same types of constraints as I_i^k does. Each relation scheme from S is a named triple: $N(R, C, K_p(R))$, where components R , C and $K_p(R)$ have the same meaning as R_i^k , C_i^k and $K_p(R_i^k)$, respectively. Analogously, the abbreviation $N(R, K)$ will also be used whenever it is appropriate.

5. Collisions of Constraints

The process of simultaneous and independent design of each subschema may lead to the collisions in expressing the real system constraints and business rules, in the different subschemas. If the collisions between the different subschemas exist, then some of the subschemas are not *consistent* with the potential database schema in a formal sense. Consequently, the programs made over the inconsistent subschemas do not allow *safe database updates*, i.e. their using may lead to logically incorrect database updates. Accordingly, such a potential schema must not be considered as a resulting database schema, after the integration of the set of subschemas. The problem of safe database updates is discussed in [21]. In the following text, we give only an explanation of the notion of a safe database update.

A subschema is a description of the data of a relatively small part of the database. Each relation scheme of a subschema may be considered as a view on a single database relation scheme. Subschema instances are not materialized. A subschema instance may be obtained as a result of the applying appropriate join, select and project operations on a database instance.

A transaction program issues queries and updates that are executed by a database management system (DBMS). Let T_k be a transaction program based on subschema concepts, and let T be a transaction program that is equivalent to T_k , but based on database schema concepts. To consider database updates initiated by T_k as *safe updates*, the subschema and the database schema should satisfy the following two conditions at the abstraction level of instances.

1. A unique (hypothetical) subschema instance, named the *corresponding* subschema instance, may be produced by applying the appropriate relational join, project and select operations on a database schema instance; and
2. If an update of a hypothetical subschema instance issued by T_k would be successful, then T must be committed by DBMS.

If a subschema is intended for queries only, it has to satisfy only Condition 1.

In the paper, the aforementioned conditions are called the principles of a *database update using subschema concept*. A subschema that satisfies these conditions is said to be *consistent* with the corresponding database schema. Let P be the set of subschemas and let the potential database schema be the result of the integration of subschemas from P . The potential database schema, which is consistent with all of the subschemas from P , may be declared as a *database schema*.

A subschema and a database schema are formally consistent if:

- C1. The set of attributes of each subschema relation scheme is a subset of the attribute set of a corresponding database relation scheme;
- C2. Each set of attributes X with a unique value property (as defined in [15] and [23]) in a subschema relation scheme has the same property in the corresponding database relation scheme; and
- C3. All the constraints that can be inferred from the database schema and that are relevant for the subschema are embedded into it.

A constraint o is embedded into a subschema P_k if it can be inferred from the subschema set of constraints O_k , which is denoted by the implication $O_k \models o$.

The first and the second condition are expressed in a formal way in [23]. Satisfying these two conditions is a prerequisite for the validation of the third condition, which is expressed by the logical implication:

$$(1) \quad O_k \models O_{P_k}^r,$$

where O_k is the set of all constraints of the subschema P_k , and $O_{P_k}^r$ is the set of all database schema constraints that are *relevant* for P_k .

A database schema constraint o should belong to the set of relevant constraints for the subschema P_k , if the operation that might violate o is allowed in P_k . The operations that might violate some constraint are called *critical operations*.

There are two kinds of relevant constraints:

- The *includible* constraints, denoted by $Ini(O, P_k)$; and
- The *extending* constraints, denoted by $Exi(O, P_k)$,

where O is the set of all constraints of the database schema.

Suppose a database schema constraint o is relevant for the subschema P_k that satisfies conditions C1 and C2.

The constraint o belongs to $Ini(O, P_k)$ if it can be expressed using the concepts of subschema P_k .

A constraint o belongs to $Exi(O, P_k)$ if and only if it is relevant for P_k , and $o \notin Ini(O, P_k)$ holds.

A database constraint o is potentially inconsistent if it is relevant for at least one subschema P_k and:

- it cannot be inferred from the set of subschema constraints O_k , i.e. $\neg(O_k \models o)$ holds, or
- it cannot be expressed using the concepts of the subschema P_k , i.e. $o \in Exi(O, P_k)$ holds.

Described constraint inconsistencies are called *constraint collisions*. They must be resolved since programs made over an inconsistent subschema do not guarantee safe database updates [21].

For each potentially inconsistent constraint, the designer has to decide whether it should be embedded into the db schema.

If the decision is positive, the potentially inconsistent constraint must be embedded into all the subschemas for which it is relevant.

Otherwise, a potentially inconsistent constraint must not be embedded into the set of database constraints. It must be emphasized that subschema constraints may be stronger, but not weaker than the corresponding database constraints. 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 it must not be embedded into the database schema. Subschema constraints that are embedded into a database schema are considered as *globally valid*.

Let us consider a potentially inconsistent constraint and the subschema into which it has already been embedded as a relevant one. During the process of collision resolving, a designer may decide between the following two alternatives:

- A potentially inconsistent constraint may be excluded from the subschema; or
- It may be pronounced as a *locally valid constraint* of the subschema.

In the first step of the integration process, all constraints of the subschema may be pronounced as globally valid. Some of them may be pronounced as locally valid in the subsequent iterations.

There are three possible relationships between a subschema P_k and a potentially inconsistent database constraint o .

- A potentially inconsistent constraint o is not relevant for P_k , and consequently o does not induce inconsistency between P_k and the database schema. The designer does not need to redesign the subschema P_k , but probably needs to redesign another subschema.
- A potentially inconsistent constraint o is relevant for P_k , but it is not embedded into the set of constraints of P_k . P_k is potentially inconsistent and the designer may redesign it by embedding o into its set of constraints or by excluding critical operations from P_k .
- A potentially inconsistent constraint o is relevant for P_k and it is embedded into the set of constraints of P_k , but there is another subschema P_l , for which o is also relevant, but not embedded into it. P_k has "introduced"

o into the set of database constraints. Accordingly, P_k is potentially inconsistent. The designer may redesign it by excluding o from its set of constraints or by pronouncing o as a locally valid constraint for the subschema P_k .

IIS*Case uses specialized algorithms to check the consistency of constraints embedded in a database schema and the corresponding subschemas. Consistency checking is performed for each constraint type separately. IIS*Case detects collisions of: attribute sets, 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 are going to demonstrate the application of IIS*Case in detecting collisions, together with an analysis of related reports and possible designer's actions.

6. Detecting and Resolving Collisions of Constraints In IIS*Case

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

We identified four groups of similar user requests:

- Personnel – insert/update/delete details of staff members;
- Working Units (WU) – insert/update/delete details of working units and update some details of staff members working in the particular WU;
- Work Orders (WO) – insert/update/delete the details of working orders and display some details of staff member who launched WOs; and
- Operating Plans (OP) – insert/update/delete the details of operating plans containing the work orders that should be accomplished during a time period.

For each of these groups, a set of form types is designed. Figures 7–9 represents some of the form types, designed in IIS*Case. Figure 5 contains the form type *OP* from the application system *OPERATING PLANS*.

For each group of user requests, IIS*Case maps form types into a relational subschema by inferring attributes and constraints from form types and embedding them into the relational subschema. It also generates the appropriate reports of db schema design progress. In this way, we get four subschemas: *PERSONNEL*, *WORKING_UNIT*, *WORK_ORDERS*, and *OPERATING_PLANS*. In the following text, an abbreviation $N(R, K, \text{Uniq}(N))$ is used in their representation.

PERSONNEL

- $\text{Staff}\{\{IdWU, SurN, DatB, Addr, JMBG, Name\}, \{JMBG\}, \{\}\}$,
- $\text{WU}\{\{WRoom, IdWU, NamWU, ManagWU\}, \{IdWU\}, \{\}\}$,

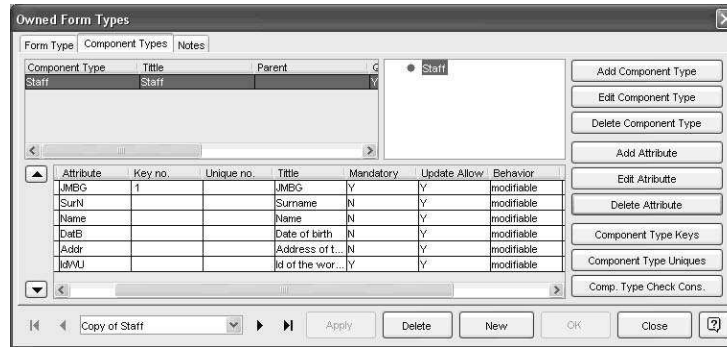


Figure 7: Form type Staff from the application system PERSONNEL

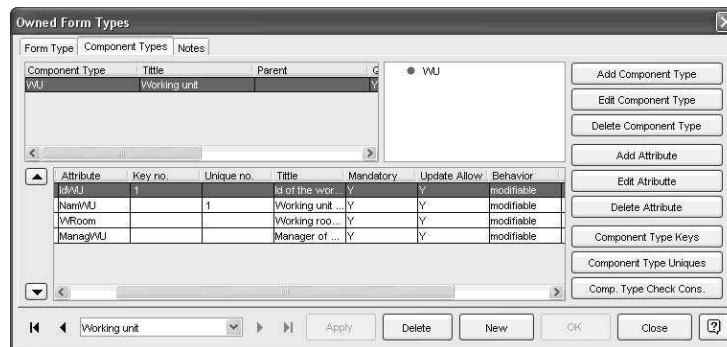


Figure 8: Form type WU from the application system WORKING UNIT

- $WU[ManagWU] \subseteq Staff[JMBG]$,
- $Staff[IdWU] \subseteq WU[IdWU]$.

WORKING_UNIT

- $WU\{WRoom, IdWU, NamWU, ManagWU\}, \{IdWU\}, \{NamWU\}$,
- $Staff\{IdWU, SurN, DatB, Addr, JMBG, Name, School, IdS, Manag, CelTel\}, \{IdWU+IdS, JMBG\}, \{\}$,
- $WU[ManagWU] \subseteq Staff[JMBG]$,
- $Staff[IdWU] \subseteq WU[IdWU]$,
- $Staff[Manag] \subseteq Staff[JMBG]$.

WORK_ORDERS

- $WO\{IdPr, DatWO, Amount, IdWO, JMBG\}, \{IdWO\}, \{\}$,

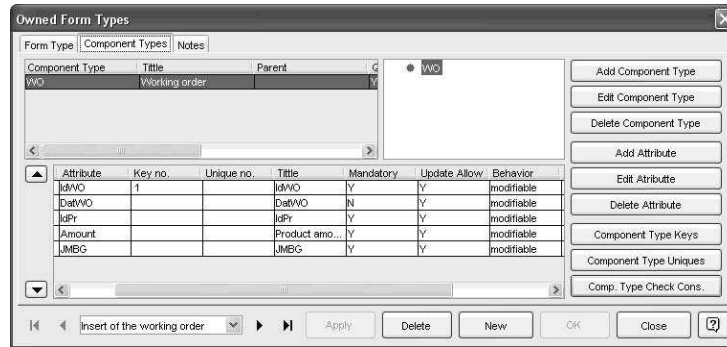


Figure 9: Form type WO from the application system WORKING ORDERS

- $Staff\{\{SurN, JMBG, Name\}, \{JMBG\}, \{\}\}$,
- $WO[JMBG] \subseteq Staff[JMBG]$.

OPERATING_PLANS

- $WO\{\{SurN, IdWO, Name, IdOplan\}, \{IdWO\}, \{\}\}$,
- $OP\{\{From, Until, IdOplan\}, \{IdOplan\}, \{\}\}$,
- $WO[IdOplan] \subseteq OP[IdOplan]$.

IIS*Case produces the first version of a db schema by integrating independently designed subschemas. The order of integration is irrelevant. It is even possible to integrate all the subschemas at once. We believe that the order of integration, described in the paper, enables gradual and clear presentation of the detection and resolving of constraint collisions.

IIS*Case uses specialized algorithms for checking the consistency of constraints embedded in the db schema and the subschemas. Figure 9 represents the screen form of IIS*Case by means of the designer controls the process of consistency checking for a chosen application system. The consistency checking is performed for each constraint type separately. The order of consistency checking with respect to the constraint type is relevant. Consistency of a set of constraints of a given type may be a prerequisite for the consistency validation of another constraint type. IIS*Case performs the consistency checking by imposing the following order: the attribute sets, the key and unique constraints, the null value constraints, and finally the referential integrity constraints. The red cross in Figure 9 indicates that the algorithm detected collisions during the consistency checking of the attribute sets. The consistency checking for the rest of the constraints can not be performed, while the detected collisions are not resolved. The hyperlink on the form leads to the appropriate report, containing the detail description of all collisions. The reports also contain the explanations, how to interpret them. The structure of these reports for different constraint types will be presented in the following subsections.

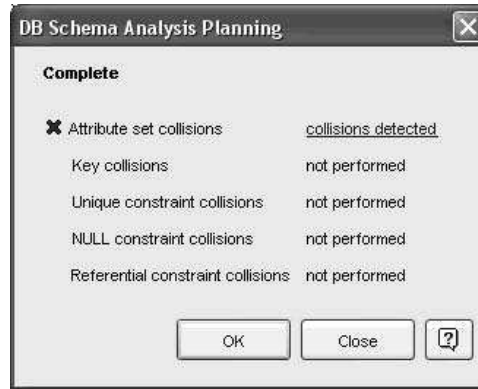


Figure 10: IIS*Case form for controlling the process of consistency checking

6.1. Collision of the Sets of Attributes

Let P_k be a subschema integrated into a database schema, and let N_k be one of relation schemes in P_k . The database schema must have at least one relation schema that contains the attribute set of N_k . One of such relation schemes is pronounced as the *corresponding relation scheme* for N_k . Often, there is only one relation scheme with the given property. However, if there are more of them, IIS*Case will choose the one with the maximum set of attributes.

Suppose we decided to integrate subschemas *WORKORDERS* and *OPERATING_PLANS* first. Using IIS*Case, we make a new application system *PLANING*, with two child application systems containing the starting subschemas. Here is the new database schema *PLANING*:

- $OP\{\{From, Until, IdOplan\}, \{IdOplan\}, \{\}\}$
- $WO\{\{IdPr, DatWO, Amount, IdWO, JMBG, IdOPlan\}, \{IdWO\}, \{\}\}$
- $Staff\{\{SurN, JMBG, Name\}, \{JMBG\}, \{\}\}$
- $WO[IdOplan] \subseteq OP[IdOplan]$
- $WO[JMBG] \subseteq Staff[JMBG]$.

After the integration, the analysis of collisions in the obtained db schema is performed. In the first step, the attribute set collisions are detected. The first part of the appropriate report is shown in Figure 10. Apart from this, the Database Schema Collision Report has two other parts: Rule and Example.

The rule for the attribute set collision:

Each relation scheme from child application system must have a corresponding relation scheme in the parent application system, such that its attribute set is a subset of the attribute set of the corresponding scheme.

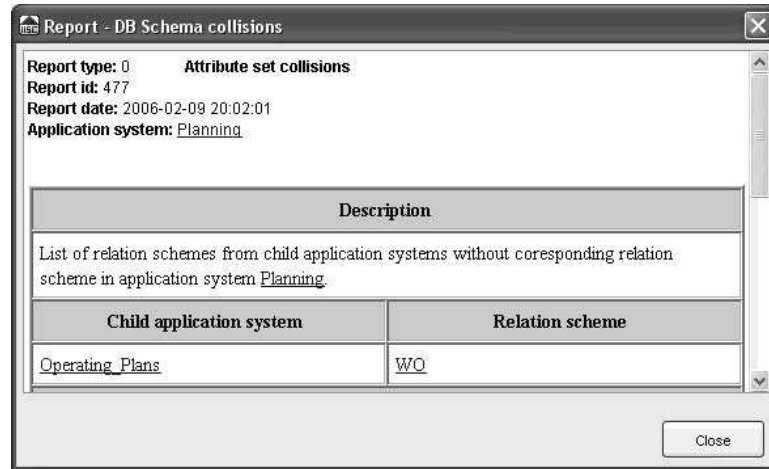


Figure 11: Report on attribute set collisions

Example:**Notation remarks:**

$N(R, K)$ denotes the relation scheme N with the set of attributes R and the set of keys K

S_1 is a child application system of an application system S .

$S_1 : N_1(\{A, C\}, \{A\})$

$S : N_2(\{A, B\}, \{A\}), N_3(\{B, C\}, \{B\})$

Collision:

N_1 from S_1 has no corresponding relation scheme in S because $\{A, C\}$ is not a subset of $\{A, B\}$ or $\{B, C\}$.

Relation scheme WO from the subschema $OPERATING_PLANS$ does not have a corresponding relation scheme in the database schema $PLANING$. In order to resolve the collision, the designer may replace attributes $SurN$ and $Name$, with the attribute $JMBG$. The replacement should be made on the form type WO (*Working order*), which was used to generate the subschema $OPERATING_PLANS$.

6.2. Collisions of Key, Unique and Null-Value Constraints

Now, we integrate the subschemas $WORKING_UNIT$ and $PERSONNEL$, and obtain a potential database schema $ADMINISTRATION$:

- $WU\{\{WRoom, IdWU, NamWU, ManagWU\}, \{IdWU\}, \{NamWU\}\},$
- $Staff\{\{IdWU, SurN, DatB, Addr, JMBG, Name, School, IdS, Manag, Cel-Tel\}, \{IdWU+IdS, JMBG\}, \{\}\},$
- $WU[ManagWU] \subseteq Staff[JMBG],$

- $Staff[IdWU] \subseteq WU[IdWU]$,
- $Staff[Manag] \subseteq Staff[JMBG]$

The analysis of attribute set collisions finishes successfully. However, collisions of key constraints are detected. The first part of the appropriate report is shown in Figure 11. The other two parts have the form:

The rule for the key collision:

Suppose there is a relation scheme N_1 in the child application system S_1 , for which N_2 is the corresponding relation scheme in the parent application system S . A key X from N_2 must be included in the relation scheme N_1 if some of the attributes from X belong to the attribute set of N_1 and insert or modify these attributes, is allowed in N_1 .

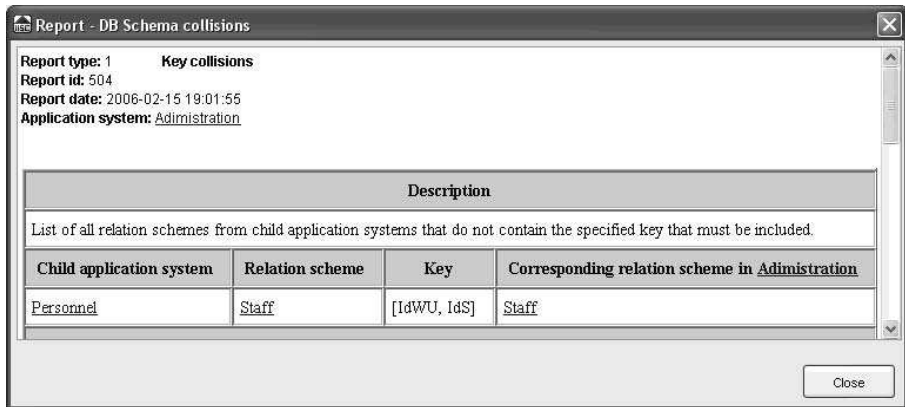


Figure 12: Report on key collisions

Example:

Notation remarks:

- $N(R, K)$ denotes the relation scheme N with the set of attributes R and the set of keys K .
- $Key(N, X)$ denotes that X is a key of N .
- $Role(N)$ denotes a set of database operations that are allowed in a relation instance over N .
- $Mod(N)$ denotes a set of attributes from R that are modifiable in a relation instance over N .

S_1, S_2 are the child application systems of the application system S .

$S_1: N_1(\{A, B, C, E\}, \{A\})$
 $Role(N_1) = \{r, i, m\}$, // Allowed operations: read, insert and modify
 $Mod(N_1) = \{C\}$ // Modifiable attribute: C

$S_2: N_2(\{A, B, C, D, E\}, \{A, CD\})$
 $S: N_3(\{A, B, C, D, E\}, \{A, CD\})$

Collision:

N_3 is the corresponding relation scheme for N_1 and N_2 . The constraint $Key(N_1, CD)$ is not included in N_1 but it must be, because:

- The attribute C is included in both N_1 and N_3 ,
- Insert or modify of C is allowed in N_1 , and
- There is the constraint $Key(N_3, CD)$.

Accordingly, the attribute D must be included in N_1 .

We change the subschema *PERSONNEL* by adding the attribute *IdS* and an additional key *IdWU+IdS* into the form type *Staff*. After these changes, the analysis of key collisions finishes successfully.

In the next step, unique constraint collisions are detected. The first part of the appropriate report is shown in Figure 12. Because the third part (example) is analogous to the example for the key collision, it is omitted here. The second part of the report is of the following form.

Description			
List of all relation schemes from child application systems that do not contain the specified unique constraint that must be included.			
Child application system	Relation scheme	Unique constraint	Corresponding relation scheme in Administration
Personnel	WU	[NamWU]	WU

Figure 13: Report on unique constraint collisions

The rule for the unique constraint collision

Suppose there is a relation scheme N_1 in a child application system S_1 , for which N_2 is the corresponding relation scheme in the parent application system S . A unique constraint X from N_2 must be included in the relation scheme N_1 if:

- some of the attributes from X belong to the attribute set of N_1 , insert/modify of these attributes in N_1 is allowed, and
- X is not a key of N_1 .

In this case, the collision is resolved by embedding a unique constraint for the attribute *NamWU* in the form type *WU* from the application system *PERSONNEL*.

In the fourth step, we analyze null-value constraints. All detected collisions of null-value constraints are automatically resolved. The appropriate report contains the list of the changes made (Figure 14). Like the others collision

reports, it contains a rule and an example parts.

Description					
List of relation scheme attributes from the application system <u>Administration</u> , for wich NULL constraint has been changed.					
Relation scheme	Attribute	Before	After	Relation scheme	Child application system
<u>Staff</u>	SurN	NOT NULL	NULL	<u>Staff</u>	<u>Personnel</u>
<u>Staff</u>	Name	NOT NULL	NULL	<u>Staff</u>	<u>Personnel</u>

Figure 14: Report on NULL constraint collisions

The rule for the null value collision:

A not-null attribute A of a relation scheme N_1 will be changed into a nullable one, if there exists a relation scheme N_2 in a child application system for wich N_1 is the corresponding scheme and A in N_2 is nullable.

Example:

Notation remarks:

$N(R, K,)$ denotes the relation scheme with the set of attributes R and the set of keys K

S_1 is a child application system of the application system S .

$S_1 : N_2(\{A, B\}, \{B\})$,

$Null(N_2, A) = True$ // Nulls for A in N_2 are allowed

$S : N_1(\{A, B\}, \{B\})$

$Null(N_1, A) = False$ // Nulls for A in N_1 are not allowed

Collision:

The attribute A must be nullable in N_1 because it is nullable in N_2 and N_1 is the corresponding scheme for N_2 .

Automatic Correction Method:

A in N_1 will be changed into a nullable attribute ($Null(N_1, A) = True$).

6.3. Collisions of Referential Integrity Constraints

The final step is the consistency analysis of the referential integrity constraints. After detecting collisions, IIS*Case produces an appropriate report (Figure 15). The rule and examples for this type of constraint are rather complex.

Description			
The list of referential constraints from Administration, which are missed from the child application systems but that must be embedded.			
Message type	Referential constraint	Child application system	Missing relation scheme
X collision	RI_IC_Staff_Staff	Personnel	

Figure 15: Report on referential integrity collisions

The rule for the referential integrity collision:

Suppose there is a referential constraint $RC : N_1[X] \subseteq N_2[Y]$ in the application system S and there is a relation scheme N_3 in a child application system S_1 , for which N_1 or N_2 is the corresponding relation scheme. RC must be included in S_1 if:

- N_1 is the corresponding scheme for N_3 and insert or modify operations on X are allowed in N_3 ; or
- N_2 is the corresponding for N_3 and delete is allowed in N_3 .

Example 1:

$S_1 : N_1(\{A, B\}, \{A\})$
 $Role(N_1) = \{r, i, m\}$
 $Mod(N_1) = \{B\}$
 $S : N_2(\{A, B\}, \{A\}), N_3(\{B, C, D\}, \{B\})$
 $RC : N_2[B] \subseteq N_3[B]$

Collision:

Referential constraint RC must be included in S_1 because N_2 is the corresponding relation scheme for N_1 and insert or modify of B is allowed in N_1 . N_3 is a missing relation scheme. It is not included in S_1 but it should be.

Example 2:

$S_1 : N_4(\{B, C\}, \{B\})$
 $Role(N_4) = \{r, d\}$
 $S : N_2(\{A, B\}, \{A\}), N_3(\{B, C, D\}, \{B\})$
 $RC : N_2[B] \subseteq N_3[B]$

Collision:

Referential constraint RC must be included in S_1 because N_3 is the corresponding relation scheme for N_4 and delete is allowed in N_4 . N_2 is a missing relation scheme. It is not included in S_1 but it should be.

More examples of collisions may be found in [19].

In this case the collision is resolved by adding the attribute *Manag* in the form type *Staff* of the application *PERSONNEL*. After this change, referential integrity collision analysis is successfully finished.

Finally, the potential database schema can be pronounced as a db schema *ADMINISTRATION*. We should emphasize that during the consolidation process the designers are changing the structure of application systems and the sets of form types (i.e. external schemas). Afterwards, IIS*Case generates subschemas and integrates them into a db schema. Therefore, after the consolidation process is successfully finished, we obtain the consistent set of subschemas and the consistent sets of form types. 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.

7. Conclusion

The form type concept is semantically rich enough to enable specifying such an initial set of constraints, which makes it possible to generate an implementation database schema automatically.

Design of external schemas, relying on high-level abstract data models, facilitates significantly identification of attribute and constraint sets. The presented approach is based on the form type data model. From the designer's point of view, the form type data model offers a simple way for defining the initial set of attributes and constraints. By the knowledge of the authors, this is an original approach, which cannot be found in the same form in similar tools.

IIS*Case is developed on the basis of the results of the theoretical researches presented in [9, 16, 19, 23]. The principles of a *database update using subschema concepts* are introduced in [23] at the abstraction level of instances to express the fact that a subschema and the corresponding database schema must satisfy certain conditions to allow safe database updates using a program made in accordance with subschema concepts. The conditions of the *formal* subschema and database schema *consistency* are introduced at the schema abstraction level, as well. It is shown that formal consistency implies database update principles, which leads to the conclusion that a db schema design by integrating independently designed subschemas should adhere to formal consistency conditions. Therefore, detecting and resolving subschema collisions is an important part of IIS*Case. Our approach is specific because the collisions are not detected between different subschemas, but between a db schema and a set of subschemas, since the integration process is not just a process of unifying the subschemas. It is not rare case that the process of detecting and resolving collisions helps the designer to recognize new database constraints, which have not been previously identified.

The process of the detection of collisions is fully automated by IIS*Case for the most important constraint types. It makes considerably easier the process

of collision resolving. Some of the collisions can even be resolved automatically, whereas the most of them the designers must resolve themselves at the conceptual level. Despite that, IIS*Case enables designers to work together and cooperate so as to reach the most appropriate solutions.

A designer may devote his or her time and power to the analysis and modelling business processes and rules. The database design of even complex information systems may be an easier task if it was based on this approach and IIS*Case, because the process of modelling is raised to the level which is closer to a user without an advanced knowledge of the database design.

8. Further Research and Development

At the time being, IIS*Case R.6.0 produces a formal specification of an implementation database schema as its final result. Further research and development efforts will be oriented towards extending functionality of IIS*Case to support complete development of an information system. Accordingly, we are planning or already working on:

- Implementation of a SQL generator,
- Visualisation of form types,
- Implementation of an application generator,
- Further improving integration and consolidation algorithms, etc.

SQL generator will enable generating SQL specifications of a database schema for different DBMSs.

One of the goals is to provide visual design and specification of form types by using a graphical editor, flexible enough for modelling user forms of varying functionality.

Form types, apart from constraints that make a basis for the database schema design, carry additional information about transaction programs and their screen forms. This enables one to implement a code generator within IIS*Case, which will be able to generate transaction programs. Such processes already exist and are specified in [9], where the problem of formalizing and generating programming specifications and applications of an information system based on XML technology, is discussed. These processes should be improved and implemented in IIS*Case R.6.0.

References

- [1] Beerl 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.
- [2] Casanova M. A., Fagin R., Papadimitriou C. H., Inclusion Dependencies and Their Interaction with Functional Dependencies. *Journal of Computer and System Sciences*, Vol. 28 No. 1 Feb. 1984, pp. 29 - 59.

- [3] 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.
- [4] 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.
- [5] Date C. J., Darwen H., Foundation for Object/Relational Databases: The Third Manifesto. Addison-Wesley Professional, 1998.
- [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] Diedrich I., Milton J., New Methods and Fast Algorithms for Database Normalization. *ACM Transactions on Database Systems* Vol. 13 No. 3 Sept. 1988, pp. 339-365.
- [8] Gálvez S., Guevara A., Caro J. L., Gómez I., Aguayo A., Collaboration Techniques to Design a Database. Universidad de Málaga, Spain, 2004.
- [9] 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.
- [10] 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.
- [11] Honeyman P., Scoire, E., New Characterization of Independence, Proceedings of ACM SIGMOD. San Jose, California, USA, 1983, pp. 92-96.
- [12] Kambayashi Y., Tanaka K., Yajima S., Problems of Relational Database Design. In: *Data Base Design Techniques I*, Edited by Yao S, B, et al., Lecture Notes in Computer Science, Springer Verlag, Berlin, 1982, pp. 172-218.
- [13] Luković I., Govedarica M., Mogin P., Ristić S., The Structure of A Subschema and Its XML Specification. *Journal of Information and Organizational Sciences (JIOS)*, Varazdin, Croatia, Vol. 26, No. 1-2, 2002, pp. 69-85.
- [14] Luković I, Mogin P, Pavićević J, Ristić S, An Automated Design and Integration of Database Schemas. Conference devoted to 25th anniversary of Faculty of Science at University of Montenegro Contemporary Mathematics, Physics and Biology, September 8-9, 2005, Podgorica, Serbia and Montenegro, Invited paper
- [15] Luković I., Ristić S., Mogin P., On The Formal Specification of Database Schema Constraints. *1st Serbian-Hungarian Joint Symposium on Intelligent System SISY 2003*, September 19-20, 2003, Subotica, Serbia and Montenegro, Proceedings, pp. 125-136.
- [16] Luković I., Ristić 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, Proceedings in CD ROM.
- [17] Mogin P., Luković I., A Prototyping CASE Tool. *XXVIII International Symposium on Automotive Technology and Automation*, Stuttgart, Germany, September 18-22, 1995, Proceedings for the Dedicated Conference on Rapid Prototyping in the Automotive Industries, pp. 261-268.

- [18] Mogin P., Luković I., Govedarica M., Database Design Principles, 2nd Edition. University of Novi Sad, Faculty of Technical Sciences, Novi Sad, Serbia and Montenegro, 2004.
- [19] Pavićević 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.
- [20] Pavićević J., Luković I., Mogin P., Ristić S., IIS*Case – A tool For Automated Design and Integration of Database Schemas. 13th Scientific Conference on Industrial Systems IS'05, Herceg Novi, September 07 - 09, 2005, Proceedings pp. 321-330.
- [21] 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, ISBN 977-246-048-3.
- [22] Ristić S., Luković I., Mogin P., Pavićević J., Integrating a Database Schema Using IIS*Case Tool. 13th Scientific Conference on Industrial Systems IS'05, Herceg Novi, September 07 - 09, 2005, Proceedings pp. 331-340.
- [23] Ristić S., Research of Subschema Consolidation Problem. PhD Thesis, University of Novi Sad, Faculty of Economics, Subotica, Serbia and Montenegro, 2003.
- [24] 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.

Received by the editors February 21, 2006