Reverse Engineering of a Generic Relational Database Schema Into a Domain-Specific Data Model

Slavica Kordić, Sonja Ristić
University of Novi Sad,
Faculty of Technical Sciences,
Trg D. Obradovića 6,
21000 Novi Sad, Serbia
{slavica, sdristic}@uns.ac.rs

Milan Čeliković, Vladimir Dimitrieski, Ivan Luković
University of Novi Sad,
Faculty of Technical Sciences,
Trg D. Obradovića 6,
21000 Novi Sad, Serbia
{milancel, dimitrieski, ivan}@uns.ac.rs

Abstract. *Information system* (*IS*) *reengineering process comprises reverse engineering process of an existing IS followed by some form of forward engineering or restructuring. An important phase of a data-oriented software system reengineering is a database reengineering process and, in particular, its sub-process – database reverse engineering process. In this paper we present one of the model-to-model transformations from a chain of transformations aimed at transformation of a generic relational database schema into a domain-specific data model based on form types. The transformation is a step of the data structure conceptualization phase of a model-driven database reverse engineering process that is implemented in IIS*Studio development environment.*

Keywords. database reengineering, reverse engineering, relational data model, form type data model, model-driven approach

1 Introduction

The emergence of large and complex information systems (IS) increases the interest in Model-Driven System Engineering (MDSE) and its appliance in information system (re)engineering process. The model-driven approach to information system and software (re)engineering addresses complexity through abstraction. A complex system consists of several interrelated models organized through different levels of abstraction and platform specificity. A model-driven information system engineering process should cover: modeling that produces description models from existing enterprise systems; forward engineering that produces specification, prescription and implementation models; and reverse engineering that produces description models from engineered software systems.

Through a forward engineering process models need to be refined and integrated and used to produce code and therefore they would undergo a series of transformations. Each transformation adds levels of specificity and detail starting from an initial model at the highest level of abstraction (Platform Independent Model, PIM), through the less abstract models, with different levels of platform specificity (Platform Specific Models, PSMs), and resulting in an executable program code that represents a model at the lowest level of abstraction (fully PSM). Conversely, in a reverse engineering process, the abstraction level of models and degree of platform independency are increasing throughout the chain of transformations. PIMs use the domain-specific concepts that are understandable to the users and in that way users get an opportunity to participate directly in the design, implementation, integration and evolution of an information system. Once a domain-specific framework has been established, a new version of the system does not take a long time to generate and deploy using model-driven techniques. All together it enables easier uncovering of the problems and provides the opportunity to correct mistakes early in the IS (re)engineering process.

Through a number of research projects on model-driven intelligent systems for information system development, maintenance and evolution, we have developed the IIS*Studio tool. It is aimed to provide the multi-paradigm approach to IS design (Dimitrieski et al., 2015), generating executable application prototypes (Aleksić et al., 2007; Aleksić et al., 2013; Popović et al., 2015) and IS reverse engineering. Our approach is mainly based on the model-driven information system and software engineering (Bézivin, 2006; Favre, 2005) and domain specific language (DSL) paradigms (Kosar et al., 2010; Dejanović et al., 2010). In IIS*Studio we use form type data model as domain-specific data model (DM). The explanation of basic concepts of this model is presented in Section 2.

Meta-modeling is of great importance in the context of database reverse engineering. In (Ristić et al., 2014) are presented different database meta-

models (MM) in support of IIS*Studio reengineering process. The database reverse engineering (DBRE) is, according to (Hainaut et al., 2009), the process of recovering the conceptual schema of a database and it is divided in two main phases: data structure extraction and data structure conceptualization. In (Ristić et al., 2015) we have proposed a model-driven approach to data structure conceptualization phase of database reverse engineering process. The presented data structure conceptualization is conducted through a chain of model-to-model (M2M) transformations. Each of them is based on two meta-models: source and target meta-model where the source meta-model of a model transformation is at lower abstraction level than the target meta-model of the transformation.

In (Ristić et al., 2016) we have given the blueprint of the final step of the conceptualization phase—the M2M transformation of a generic relational database schema into a form-type model. The transformation is illustrated by an example of simple transformation of a basic relation scheme into a basic form type.

In practice, we face not only with basic relation schemes and form types, but with more complex data structures in both data models (relational DM and form type DM). In this paper we address the problem of transformation of weak and all key relation schemas into complex form types (in the paper they are denoted as $\mathcal{F}\_Tree^2$ and $\mathcal{F}\_Tree^n$ form types).

Apart from Introduction and Conclusion the paper has six sections. Form type concept of IIS*Studio is elaborated in Section 2. The reverse engineering process in IIS*Studio is described in Section 3. The classification of form types and terms of $\mathcal{F}\_Basic$, $\mathcal{F}\_Tree^2$ and $\mathcal{F}\_Tree^n$ form types are presented in Section 4. Terms of basic, weak and all key relation schemas are introduced and explained in Section 5. The transformation of generic relational database schema into form type data model is presented in Section 6. Related work is given in Section 7.

2 The Form Type Concept

According to (Brinkkemper et al., 2000) three main functions of an IS are: (i) to maintain a consistent representation of the state of a domain (F1); (ii) to provide information about the state of a domain (F2); and (iii) to perform actions that change the state of a domain (F3). A model-driven (MD) approach to IS engineering would support several viewpoints and modeling languages to enable creation of different models that capture the knowledge required to support main functions of an IS. IIS*Studio is based on an MD approach that comply to these requirements.

In order to provide design of various platform independent models by IIS*Studio, a number of modeling, meta-level concepts and formal rules that are used in the design process are created.

A form type is central IIS*Studio PIM concept, used to model the structure and constraints of various business forms. Business forms (documents) are broadly used in organizations to conduct daily operations and to communicate with their affiliated entities (e.g. staff, superior managers, customers, suppliers, etc.). The adjective *business* is used to emphasize that a business form need not to be a part of computer graphical user interface (GUI), but can be a paper form that is filled in manually or a printed report, too. They may provide an important input source for database (db) schema design, since the most widely used data are gathered or reported in them. Forms are objects, easy to read and understand, well-structured and, consequently, easy to formalize. Therefore, business forms are a source for eliciting user information requirements and also for designing and developing user-oriented information systems. Initially, each form type (FT) is an abstraction of a business form. In that way FT concept supports IS function F1. However, it may be enriched by additional specifications that are not included in the entry business form, like specifications of: key and unique constraints; check constraints (both additionally support function F1); allowed database CRUD (Create, Retrieve, Update and Delete) operations applied by means of screen computerized forms to manipulate data of an IS (supporting IS function F3); functionalities concerning relationships between generated screen forms, i.e. transaction programs (supporting IS functions F2 and F3), etc.

In the paper we use the case study of a simplified Airline Ticket Reservation System (ATRS). The business form *Customer Reservations* (*CR-bf*), presented in Figure 1, is used to keep track about the customers and their reservations. A customer reservation is the unit of payment and it can comprise more seats for one or more flights, that are reserved on the same date and that would be paid by single payment.

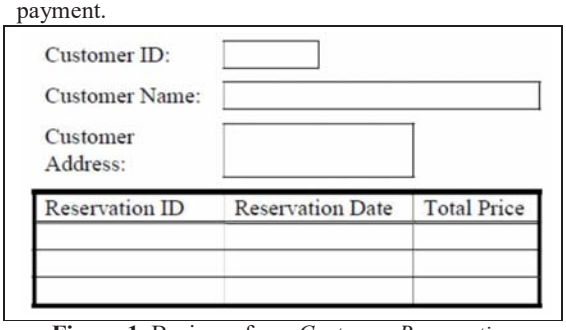


Figure 1. Business form *Customer Reservations*

The business form *Customer Reservations* may be modeled by the form type *Customer Reservations* (*CR-ft*). The simplified representation of the structure of the *CR-ft*, which generalizes the *CR-bf*, is presented in Figure 2. As can be seen, a form type is a hierarchical structure of form type components. The

form type *Customer Reservations* (Figure 2) has two component types: *CUSTOMER* and *RESERVATION*. Underlined attributes represent keys of component types. Letters placed in the rectangle on the right side of the component name stand for the allowed CRUD operations: *r* for retrieve, *i* for insert, *u* for update and *d* for delete.

In the traditional approaches to the IS design, database schema design not rarely precedes the specification of screen or report forms of transaction programs. On the contrary, in IIS*Studio a designer the first specifies screen and report forms, and indirectly, creates an initial set of attributes and constraints. The form type structuring rules provide automatic inference of relational db constraints from form types. A form type in IS design by means of IIS*Studio has a dual role. On the one hand it provides an important input data for database design (to support IS function F1), and on the other hand it is a source for the generation of a sole transaction program (to support IS function F3) and its screen or report form (to support IS function F2).

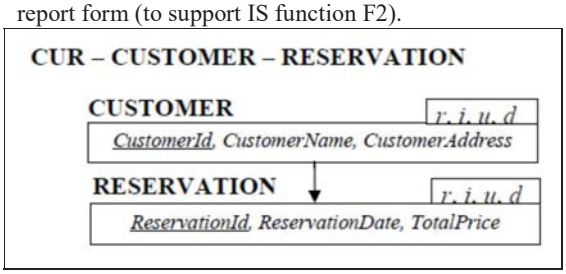


Figure 2. Form type *Customer Reservations*

IIS*Studio introduces domain-specific data model based on form type concept (Luković et al., 2007) and uses it for conceptual database design. In the paper data model based on form type concept is called *form type data model*.

The main advantage of this approach is two folded. Firstly, the FT concept is closer to the end-users' perception of data, than it would be, for example, the concepts of entity and relationship types in the Entity-Relationship (ER) data model. It is a concept that is formal enough to allow a precise expressing all of the rules significant for structuring a database schema. Secondly, a designer by means of FT specifies business, screen and report forms, and indirectly, creates an initial set of attributes and constraints. The FT specifications can be enriched with the specification of future transaction programs and business applications, too.

3 Reverse engineering process in IIS*Studio

IIS*Studio comprises three tools: IIS*Case, IIS*UIModeler and IIS*Ree. These tools communicate by means of shared repository aimed at storing project specifications. The IIS*UIModeler is an integrated part of the IIS*Studio DE, aimed at modeling of GUI static aspects. By means of IIS*UIModeler a designer specifies UI templates.

In a forward engineering process, supported by IIS*Case tool, designers start with a high-level model, abstracting from all kinds of platform issues. Through a chain of M2M transformations, ending up with a model-to-text (M2T) transformation, the initial PIM transforms iteratively to a series of models with less degree of platform independency, introducing more and more platform specific extensions. Conversely, in a reverse engineering process, supported by IIS*Ree tool, the abstraction level of models and degree of platform independency are increasing throughout the chain of transformations.

Relational databases are at the core of most company information systems, hosting critical information for the day to day operation of the company. The knowledge captured in them can serve as an important resource in a legacy information system modernization project and they are a common source of reverse engineering processes. Starting from a physical database schema, that is recorded into the relational database schema data repository, the conceptual database schema or logical database schema may be extracted. The IIS*Ree tool in our IIS*Studio extends IIS*Case tool in order to enable reverse engineering of relational databases to conceptual data models.

Several reasons motivate us to develop IIS*Ree tool. The first one is that an enterprise IS implemented to fulfill organizational information requirements, now more than ever, would adapt to emerging business models and technology changes and innovations. Legacy system replacement or reengineering can be done with significantly reduced amount of effort and cost if the conceptual models are reconstructed from them. IIS*Ree tool conceptualize a relational database into a domain-specific form type data model. That schema may be restructured and improved by means of IIS*Case tool and afterwards transformed into improved relational database schema.

The second reason is to enable database integration of different information systems. Integrated database schema can be used as a platform for ISs integration. The databases of different ISs that would be integrated can be conceptualized into several external database schemas expressed by concepts of form type data model. These external database schemas are the input of IIS*Case tool and its integration process. By means of IIS*Case tool the collisions in expressing real world constraints between different external database schema can be detected and resolved (Luković et al., 2007). In that way, PIM models of legacy ISs are restructured and consolidated. M2M and M2T transformations carried out over consolidated external database schemas are able to

generate transaction programs from form types specifications extracted from legacy databases and restructured and improved by means of IIS*Ree and IIS*Case tools.

Initial database design and further changes are poorly documented mostly thanks to the deadline pressures to which designer are exposed. Database schema design based on the experience mostly incorporates awkward constructs and non-standard design patterns that are hard to understand and communicate. A lot of knowledge about the logical and physical database design is not explicit and is hidden in the repository, program code and in physical data structures. Reverse database engineering enables data structure extraction and data structure conceptualization. In our approach form type database model is reverse engineered from a relational databases. It is very important and useful because IIS*Case tool uses a set of form types to generate, integrate and consolidate relational database schema. IIS*Case tool extracts set of functional and non-functional dependencies from the set of form types and applies modified synthesis algorithm (Beeri & Bernstein, 1979) to generate relational database schema in the 3rd normal form. The detailed description of that extraction process can be found in (Luković et al., 2007). In that way, empirically designed database model without appropriate up-to-date documentation is transformed into a database model that is designed following the disciplined database design approach.

Reverse engineering process in IIS*Ree is implemented by means of a series of *database model transformations* that are M2M transformations between database models. These transformations are based on meta-models that are conformed by the source and target database models of the transformations. In the purpose of specifying and managing meta-models in support of database model transformations implemented in IIS*Ree we use the Eclipse Modeling Framework ("EMF Eclipse Modeling Framework", 2017) Eclipse Juno 4.2.1. and OCL 3.2.1. A blueprint of a model-driven approach to database reengineering process applied in IIS*Studio is presented in (Ristić et al., 2015). Here we present one of the M2M transformations aimed at transformation of a generic relational database schema into a form type data model with focus on weak and all key relation schemes and more complex form types. Firstly, in the next section a formal specification of a form type is presented alongside with a classification of form types.

4 Classification of Form Types

A *form type* $\mathcal{F}$ is a named tree structure, whose nodes are called *component types*. Let $C(\mathcal{F})$ denotes a set of component types making up the form type $\mathcal{F}$. Each component type is identified by its name within the scope of a form type, and has nonempty sets of attributes and keys, and a possibly empty set of unique constraints. Formally, a component type is a named pair $N(Q, O)$, where N denotes name of the component type, Q is the set of component type attributes $Q=\{A_1, .., A_n\}$ and O is a set of component type constraints. O is a union of three sets: a set of key constraints, a set of unique constraints and a singleton containing a tuple constraint. The tuple constraint of a constraint type refers to a set of attribute-based constraints (attribute data type specification and not-null constraint) paired with a tuple-based constraint (constraint on tuple value). Let $C(\mathcal{F}) = \{N_i(Q_i, O_i) \mid i = 1, \ldots m\}$. $W(\mathcal{F})$ denotes a set of the form type attributes that satisfy (1) & (2).

$$\bigcup_{i=1}^{m} Q_i = W(\mathcal{F}) \quad (1)$$

$$(\forall N_i, N_j \in C(\mathcal{F}))(i \neq j \Leftrightarrow Q_i \cap Q_j = \varnothing). \quad (2)$$

A set of allowed database operations must be associated with each component type. The set of allowed operations is a subset of the CRUD operations set. Form types are classified as:

- $\mathcal{F}\_Basic$ – an elementary form type containing only one root component type (an example is presented in Figure 3);
- $\mathcal{F}\_Tree^2$ – a form type containing a root component type with only one child component type (already presented in Figure 2); and
- $\mathcal{F}\_Tree^n$ – a form type that apart from a root component type contains an arbitrary number of child component types (two examples in Figure 4).

Throughout the following text $\mathcal{F}\ T_B$ denotes a set of *F_Basic* form types, $\mathcal{F}\ T_{T2}$ denotes a set of $\mathcal{F}\_Tree^2$ form types, and $\mathcal{F}\ T_{Tn}$ denotes a set of $\mathcal{F}\_Tree^n$ form types.

5 Classification of Relation Schemes

In the context of M2M transformation aimed at transformation of a generic relational database schema into a form type data model it is necessary to classify relational schemes in the source data model.

During a reverse engineering process of a relational database schema, it is important to be aware of the limited expressiveness of the relational data model compared to other data models, like ER data model or form type data model. In (Hammer et al., 2002) a classification of relation schemes in the context of the transformation of a relational database schema into ER database schema has been proposed. Here we present a classification that is adapted according to the target data model (form type data model) that is

used in the approach presented in this paper. Identifying different categories of relation schemes is performed according to the primary key information and the inclusion dependencies of a relational database schema. We distinguish three kinds of relation schemes:

- Basic Relation Scheme (BR);
- Weak Relation Scheme (WR); and
- All Keys (AK) relation scheme.

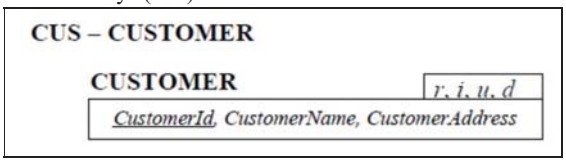


Figure 3. An example of $\mathcal{F}$ _*Basic* form type

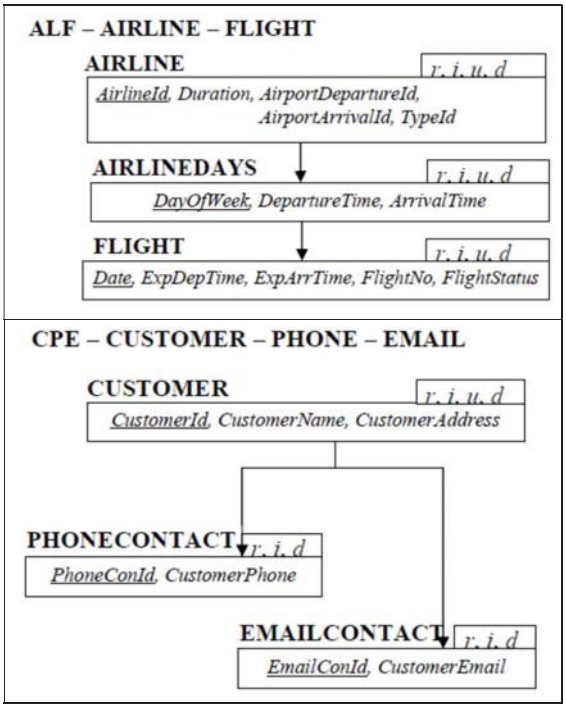


Figure 4. Two examples of $\mathcal{F}$ _*Tree*^{n} form types

A basic relation scheme is a relation scheme whose primary key (PK) does not properly contain a key attribute of any other relation.

A weak relation scheme *N* is a relation scheme that satisfies the following three conditions: i) a proper subset of its PK contains key attributes of other basic or weak relation schemas; ii) the remaining attributes of its PK do not contain key attributes of any other relation scheme; and iii) it has an identifying owner (parent) relation scheme and properly contains the PK of its parent relation scheme.

An AK relation scheme contains only key attributes of other relation schemes, and does not contain any other self-inherent attributes.

A graphic representation of ATRS relational database schema is presented in Figure 5. It contains relation schemes *Company*, *CompanyHasType*, *AirplaneType*, *Airport*, *Airlane*, *AirlineDays*, *Flights*, *Seat*, *Reservation*, *Customer*, *PhoneContact* and *EmailContact*. Underlined attributes belong to a key of a relation scheme. Relation scheme *Airline* has a key that is singleton containing *AirlineId* attribute, and relation scheme *AirlineDay* has a composite key that contains two attributes *AirlineId* and *DayOfWeek*.

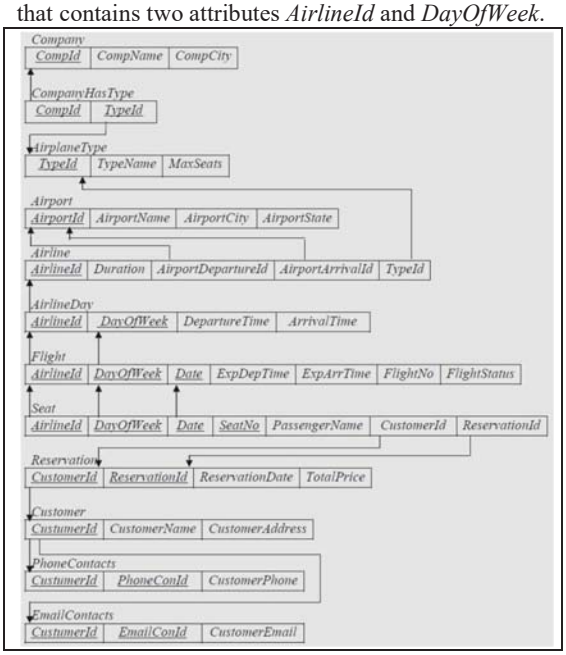


Figure 5. ATRS relational database schema

The relation schemes *Company*, *AirplaneType*, *Airport*, *Airlane*, and *Customer* are basic relation schemas. The relation schemes *AirlineDays*, *Flights*, *Seat*, *Reservation*, *PhoneContact* and *EmailContact* are weak relation schemes. The relation scheme *CompanyHasType* is an example of AK relation scheme.

6 Transformation of a Generic Database Schema into a Form Type Data Model

The input of the data structure conceptualization phase applied in IIS*Studio is XML specification of captured physical database model. This XML specification conforms to XML meta-model. The data conceptualization phase is realized as a chain of three M2M transformations: 1. XML2RDBMS, 2. RDBMS2RM, and 3. RM2IISCase.

The first transformation transforms a model conformant with XML meta-model into a model conformant with an SQL standard meta-model.

The transformation RDBMS2RM transforms a model conformant with an SQL standard meta-model into a model conformant with generic relational db meta-model. It is not possible to transform a model

conformant with an SQL standard directly into a model conformant with FT meta-model. The reason lies in the fact that FT approach to database design is based on the Universal relation schema assumption (URS assumption). Physical database meta-models and database meta-models based on SQL standard do not support URS, while generic relational database meta-models does. Both of aforementioned transformations are PSM2PSM transformations.

The third one, denoted by RM2IISCase, is a PSM2PIM transformation. It transforms a model conformant with generic relational database meta-model into a model conformant with FT meta-model. The transformation RM2IISCase is the main subject of our paper.

We have considered two approaches to implement RM2IISCase transformation. In the first approach the transformation would generate only simple form types (*F_Basic* form types). In this approach a user is free and responsible to add component types and other concepts in initial FT data model obtained by the transformation. In the second approach the transformation is able to generate all relevant combinations of form types. It is a user who chooses form types to be introduced in the form type data model. The remaining form types are deleted. RM2IISCase implementation is based on the second approach. In that way a user will invest less effort to select between already generated form types comparing to the effort that would be invested in the first approach.

The proposed transformation is carried out in three steps. They are aimed at transforming set of relation schemes from a generic relational database schema into three sets of form types, respectively: $\mathcal{F}\ T_B$, $\mathcal{F}\ T_{T2}$, and $\mathcal{F}\ T_{Tn}$. Detail description of the first step of the transformation, with parts of generic relational schema MM and FT data model MM and ATL rules to specify a mapping between the concepts of these MMs is already presented in (Ristić et al., 2016). Therefore, in subsection 6.1 just the input parameters and outputs of that transformation step are given. In subsections 6.2 and 6.3 the other two steps of the transformation are presented in detail.

6.1 Relation scheme – $\mathcal{F}$ _Basic FT transformation

In this step a $\mathcal{F}$ _Basic form type is created for each relation scheme from a relational database schema. The input parameters for this transformation are:

$$S = \{N_i(R_i, O^{RS}{}_i) \mid i = 1,.., n\}, \quad (3)$$

$$O^{RS} = K^{RS} \cup UQ^{RS} \cup CH^{RS}, \quad (4)$$

where S is a set of the relation schemes $N_i(R_i, O^{RS}{}_i)$. N_i is relation scheme name, R_i is nonempty set of its attributes and $O^{RS}{}_i$ is set of its constraints. Each constraint set is a union of three sets containing: key constraints, unique constraints; and tuple constraints, respectively (4).The transformation as output returns a set $\mathcal{F}\ T_B$ containing $\mathcal{F}$ _Basic form types.

6.2 Relation scheme – $\mathcal{F}$ _*Tree*^{2} FT transformation

The next step of the transformation is to create a set of $\mathcal{F}$ _*Tree*^{2} form types. A $\mathcal{F}$ _*Tree*^{2} form type is generated for referential integrity that has WR relation scheme on the left hand side. The set of input parameters in addition to (3) and (4) contains a set of referential integrities of a relational database schema:

$$RIC = \{ric_i: N_l[LHS] \subseteq N_r[RHS] \mid i = 1,.., m\}, \quad (5)$$

where N_l and N_r are relation schemes, *LHS* and *RHS* are subsets of attribute sets R_l and R_r of relation schemes N_l and N_r, respectively. The transformation as output returns a set $\mathcal{F}\ T_{T2}$ of $\mathcal{F}$ _*Tree*^{2} form types.

For each referential integrity $N_l[LHS] \subseteq N_r[RHS]$ from *RIC* (5), with N_l that is WR relation scheme, a $\mathcal{F}$ _*Tree*^{2} form type is created based on the pair of relation schemes (N_l, N_r). In Figure 6 are presented parts of source and target meta-models alongside with ATL rule *ChildCT* to implement transformation of a relational database schema into $\mathcal{F}\ T_{T2}$. In the lowest part of the table several helpers can be found. Four of them (*RS_AllKeys*, *AttributesOfRSs*, *Child-ComponentTypeAttributes*, and *ExistIRIC*) are called directly from *ChildCT* rule and the fifth (*AllRelatedRS*) is called from *AttributesOfRSs* helper. Corresponding calling messages are shaded within the *ChildCT* rule and *AllRelatedRS* helper.

6.3 Relation scheme – $\mathcal{F}$ _*Tree*^{n} FT transformation

The last step of the transformation is aimed at creating the third set of form types—set denoted by $\mathcal{F}\ T_{Tn}$. There are two cases to arise an $\mathcal{F}$ _*Tree*^{n} form type. An $\mathcal{F}$ _*Tree*^{n} form type will be created for each relation scheme that is referenced by at least two WR relation schemes. Besides, an $\mathcal{F}$ _*Tree*^{n} form type will be created for each relation scheme that is referenced by at list one WR relation scheme that is referenced by some WR relation scheme, too.

The input parameters are same as the input parameters of the transformation that creates $\mathcal{F}\ T_{T2}$ set. The transformation as output returns a set $\mathcal{F}\ T_{Tn}$ of $\mathcal{F}$ _*Tree*^{n} form types.

This transformation is based on the same meta-models as the transformation presented in previous section. The transformation use rules and helpers already mentioned in previous transformation steps. Rules *GetFormType* and *ChildComponentType* and helper *Children* that are specific for this step of the transformation are presented in Figure 7. The main

difference lies in the fact that the transformation problem solved by these rules is recursive because a form type tree structure may have several nodes and they can be distributed over several tree levels. The calls of rule *ChildComponentType* and helper *Children* are shaded within the rules *GetFormType* and *ChildComponentType*, respectively.

In that way the transformation process of a generic relational database schema to a form type data model made of a union of $\mathcal{F}\,T_B$, $\mathcal{F}\,T_{T2}$, and $\mathcal{F}\,T_{Tn}$ is finished. End-users are now able to reexamine and to restructure obtained conceptual model and to launch a new forward engineering process to create renewed and improved IS.

7 Related Work

Favre in (Favre, 2005) emphasizes the importance of reverse engineering and its integration with forward engineering in MDSE process to support a smooth evolution of software. Our approach is in compliance with this statement since we integrate forward (IIS*Case) and reverse (IIS*Ree) engineering tools. In (Hainaut et al., 2009) main steps of database reverse engineering are described. The creation of OO conceptual database schema from the relational data dictionary is presented in (Perez et al., 2002) and (Boronat et al., 2004). Beggar et al. (Beggar et al., 2013) propose a reverse engineering process based on MDSE that presents a solution to provide a normalized relational database which includes the integrity constraints extracted from legacy data. Vara et al. have implemented an ATL model transformation that generates an object-relational (OR) database model from a conceptual data model and an MOFScript M2T transformation that generates the SQL code for the modeled database schema (Vara et al., 2009). Our approach uses relational data dictionary, legacy data and ATL model transformations to implement reverse engineering process that conceptualize a relational database into a form type data model.

There are various research works about the use of forms (business or computerized) in different contexts. In (Tsichritzis, 1982) the concepts of form type, form template and form instance are introduced to integrate services in Office Information system, and in (Shu, 1985) they are used to specify system requirements. In the context of database schema design distinguished papers are (Batini, Demo & Di Leva, 1984) and (Choobineh & Venkatraman, 1992) that present usage of business forms as input data for the process of database schema design based on generating ER diagrams and for derivation of functional dependencies from business form, respectively. A form-based approach for reverse engineering of relational databases is proposed in (Malki, Flory & Rahmouni, 2002). A dual role of IIS*Studio form type concept described in Section 2 is what distinguishes our approach compared to other approaches based on forms.

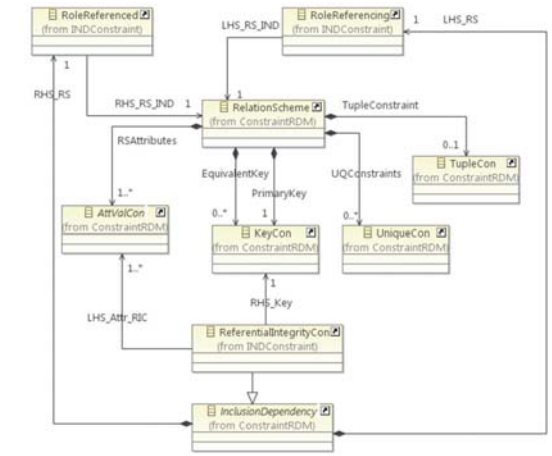
8 Conclusion

One of the main assumptions of the model-driven approach to information system and software development is that systems of large complexity can only be designed and maintained if the level of abstraction is considerably higher than that of programming languages. By means of models, semantics in an application domain can be precisely specified using terms and concepts the end-users are familiar with, such as the form types used in IIS*Studio are. Approaches to database conceptualization are mostly based just on two database meta-models. Vendor-specific physical or standard relational meta-model mainly are found on the source side of M2M transformation. On the other side, EER, class or standard/vendor-specific relational meta-models occur on the target side of M2M transformation. Most of the authors obtain relational database schema as the final result of data structure conceptualization process. According to (Hainaut et al., 2009) relational database schema cannot be seen as a pure conceptual database schema. In our approach FT database schema is obtained as the result of the data structure conceptualization process. FT specification is based on business forms, users are familiar with, and in that manner it models system as-is in a platform independent way. At the same time, the specification is platform independent prescription model of future screen and report forms and input for series of M2M transformations that ends up with model to text transformation generating application prototype.

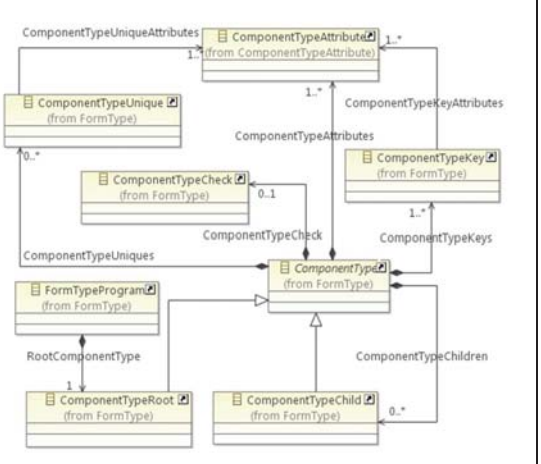
The meta-models and models that we use in our approach are intensional models. Our future research has to consider extensional database meta-models, too. Namely, we have defined the transformations of the models. Problem is what to do with the data that has been accumulated in the database conformant with a source data model. System evolution should be supported by automatic MD data migration and extensional database MM may play important role in its implementation.

Acknowledgments

Research presented in this paper was supported by Ministry of Education, Science and Technological Development of Republic of Serbia, Grant III-44010, Title: Intelligent Systems for Software Product Development and Business Support based on Models.



A part of generic relational dbS MM



A part of IIS*Case PIM (FT) meta-model

```
lazy rule ChildCT{
from
   ric: RM!ReferentialIntegrityCon
to
   ct: IISCase!ComponentTypeChild(
      Name <- 'ComponentType_' + ric.LHS_RS.LHS_RS_IND.Name,
      Title <- 'ComponentType_' + ric.LHS_RS.LHS_RS_IND.Name,
      ComponentTypeAttributes <- if thisModule.RS_AllKeys(ric.LHS_RS.LHS_RS_IND)
      then thisModule.AttributesOfRSs(ric.LHS_RS.LHS_RS_IND,
      ric.RHS_RS.RHS_RS_IND)->collect(e|
      thisModule.RSAttributes2CompTypeAttribute(e))
      else ric.ChildComponentTypeAttribute->collect(e|
         if (e.oclIsTypeOf(RM!NotNullAttr)) then
            thisModule.RSAttributes2NotNullCompTypeAttribute(e) else
            thisModule.RSAttributes2NullCompTypeAttribute(e) endif) endif,
      ComponentTypeKeys <- (ric.LHS_RS.LHS_RS_IND.EquivalentKey->collect(e|
         thisModule.EquivalentKey2CompTypeKey(e))).append(
         thisModule.EquivalentKey2CompTypeKey(ric.LHS_RS.LHS_RS_IND.PrimaryKey)),
      ComponentTypeUniques <- ric.LHS_RS.LHS_RS_IND.UQConstraints->collect(e|
         thisModule.UniqueCon2ComponentTypeUnique(e)),
      ComponentTypeCheck <- thisModule.TupleConstraints2ComponentTypeCheckCon(
         ric.LHS_RS.LHS_RS_IND.TupleConstraint),
      Query <- true,
      Delete <- false,
      Insert <- false,
      Update <- false,
      NoOfOcurrences <- if thisModule.ExistIRIC(ric) then #OneOrMany else
         #NoneOrMany endif
   )
   do{
      thisModule.attributes <- Sequence{};}}
```

Lazy rule **ChildCT**

```
helper def: RS_AllKeys(rs:RM!RelationScheme):Boolean=
   let AllRSAttributes:Sequence(RM!AttValCon)=rs.RSAttributes.asSet() in
      if AllRSAttributes-rs.PrimaryKey.KeyAttr.asSet() = Set{} then true
      else false endif;
helper def: AttributesOfRSs(rs:RM!RelationScheme,parent:RM!RelationScheme):Set(RM!AttValCon)=
let relatedRS:Sequence(RM!RelationScheme)= thisModule.AllRelatedRS(rs,parent) in
   relatedRS->iterate(rsc; rez:Set(RM!AttValCon)= Set{}|
   rez.union(rsc.RSAttributes));
helper def: AllRelatedRS(rs:RM!RelationScheme,parent:RM!RelationScheme):
                                    Sequence(RM!RelationScheme)=
let allRIC:Sequence(RM!ReferentialIntegrityCon)=
                          RM!ReferentialIntegrityCon.allInstances() in
   allRIC->iterate(ric; rez: Sequence(RM!RelationScheme) = Sequence{} |
      if ric.LHS_RS.LHS_RS_IND=rs then
         rez.append(ric.RHS_RS.RHS_RS_IND).excluding(parent) else
         rez.excluding(parent) endif);
helper context RM!ReferentialIntegrityCon
def:ChildComponentTypeAttribute:Sequence(RM!AttValCon) =
self.LHS_RS.LHS_RS_IND.RSAttributes ->
   iterate(attr; rez: Sequence (RM!AttValCon) = Sequence{}|
   if self.RHS_Key.KeyAttr->collect(e | e.AttributeName.refGetValue('Name'))
      ->asSet().includes(attr.AttributeName.Name) then rez else
      rez.append(attr) endif);
helper def:ExistIRIC(ric: RM!ReferentialIntegrityCon):Boolean =
   let allIRIC:Sequence(RM!InverseReferentialIntegrityCon)=
      RM!InverseReferentialIntegrityCon.allInstances() in
      allIRIC->iterate(iric; rez:Boolean = false |
      if iric.RIC = ric then true else false endif);
```

Helpers used (directly or indirectly via some other helper) in the lazy rule **ChildCT**

Figure 6. Relation scheme – to – $\mathcal{F}$ _*Tree*[2] Form Type transformation

```
lazy rule GetFormType{
from
   p: RM!RelationScheme
using {
   children: Sequence(RM!RelationScheme)= thisModule.Children(p);
}
to
   ft: IISCase!FormTypeProgram(
      Name <- 'FormTypeN_' + p.Name,
      Title <-'FormTypeN_' + p.Name,
      ConsideredINDBSchDesign <- true,
      Frequency <- 1,
      ResponseTime <- 1,
      RootComponentType <- ctr
   ),
   ctr: IISCase!ComponentTypeRoot(
      Name <- 'ComponentTypeRootN_' + p.Name,
      Title <- 'ComponentTypeRootN_' + p.Name,
      ComponentTypeAttributes <-p.RSAttributes -> collect(e|
         if (e.oclIsTypeOf(RM!NotNullAttr)) then
            thisModule.RSAttributes2NotNullCompTypeAttribute(e) else
            thisModule.RSAttributes2NullCompTypeAttribute(e) endif),
      ComponentTypeKeys <- (p.EquivalentKey->collect(e|
         thisModule.EquivalentKey2CompTypeKey(e))).append(
         thisModule.EquivalentKey2CompTypeKey(p.PrimaryKey)),
      ComponentTypeUniques <- p.UQConstraints->collect(e|
         thisModule.UniqueCon2ComponentTypeUnique(e)),
      ComponentTypeCheck <- if p.TupleConstraint.oclIsUndefined() then OclUndefined else
         thisModule.TupleConstraints2ComponentTypeCheckCon(p.TupleConstraint) endif,
      Query <- true,
      Delete <- false,
      Insert <- false,
      Update <- false,
      ComponentTypeChildren <- children->collect(c | thisModule.ChildComponentType(c,p))
   )
   do{
      thisModule.attributes <- Sequence{};}}
```

Lazy rule **getFormType**

```
lazy rule ChildComponentType{
from
   rs:RM!RelationScheme,
   p:RM!RelationScheme
using{
   children: Sequence(RM!RelationScheme)= thisModule.Children(rs);
}
To
   ct: IISCase!ComponentTypeChild (
      Name <- 'Child_ComponentTypeN_' + rs.Name,
      Title <- 'Child_ComponentTypeN_' + rs.Name,
      ComponentTypeAttributes <- (rs.RSAttributes.asSet()-
         thisModule.parentKeyAttributes.asSet()) ->collect(e|
         if (e.oclIsTypeOf(RM!NotNullAttr)) then
            thisModule.RSAttributes2NotNullCompTypeAttribute(e) else
            thisModule.RSAttributes2NullCompTypeAttribute(e) endif),
      ComponentTypeAttributes <- if thisModule.RS_AllKeys(rs) then
         thisModule.AttributesOfRSs(rs, p)->collect(e|
         thisModule.RSAttributes2CompTypeAttribute(e)) else rs.RSAttributes ->
         collect(e| if (e.oclIsTypeOf(RM!NotNullAttr)) then
         thisModule.RSAttributes2NotNullCompTypeAttribute(e) else
         thisModule.RSAttributes2NullCompTypeAttribute(e) endif) endif,
      ComponentTypeKeys <- (rs.EquivalentKey->collect(e|
         thisModule.EquivalentKey2CompTypeKey(e))).append(
         thisModule.EquivalentKey2CompTypeKey(rs.PrimaryKey)),
      ComponentTypeUniques <- rs.UQConstraints->collect(e|
         thisModule.UniqueCon2ComponentTypeUnique(e)),
      ComponentTypeCheck <- if rs.TupleConstraint.oclIsUndefined() then
            OclUndefined else
            thisModule.TupleConstraints2ComponentTypeCheckCon(rs.TupleConstraint)
         endif,
      Query <- true,
      Delete <- false,
      Insert <- false,
      Update <- false,
      NoOfOcurrences <- #NoneOrMany,
      ComponentTypeChildren <- children->collect(c |
         thisModule.ChildComponentType(c,rs))
   )
   do{
      thisModule.attributes <- Sequence{};}}
```

Recursive ATL lazy rule **ChildComponentType**

```
helper def: Children(rs:RM!RelationScheme): Sequence(RM!RelationScheme) =
          let allRIC:Sequence(RM!ReferentialIntegrityCon)= RM!ReferentialIntegrityCon.allInstances() in
          allRIC->iterate(ric; rez:Sequence(RM!RelationScheme) = Sequence{}|
          if ric.RHS_RS.RHS_RS_IND=rs and thisModule.isRSforFormType(ric) then rez.append(ric.LHS_RS.LHS_RS_IND) else rez
endif);
```

Helper aimed at finding direct descendents of a relation scheme

Figure 7. Relation scheme – to – $\mathcal{F}\_Tree^{n}$ Form Type transformation

References

Aleksić, S., Luković, I., Mogin, P. & Govedarica, M. (2007). A generator of SQL schema specifications. *Computer Science and Information Systems*, 4(2), 81–100.

Aleksić, S., Ristić, S. , Luković, I., & Čeliković, M. (2013) A Design Specification and a Server Implementation of the Inverse Referential Integrity Constraints. *Computer Science and Information Systems*, 10(1), 283–320.

Batini, C., Demo B., & Di Leva, A., (1984) A methodology for conceptual design of office data bases, *Information Systems* 9 (3/4), 251– 263.

Beeri, C., & Bernstein, P.A. Computational Problems Related to the Design of Normal Form Relational Schemas. *ACM Transactions on Database Systems*, 4(1), 30–59.

Beggar, O. E., Bousetta, B., & Gadi, T. (2013). Getting Relational Database from Legacy Data-MDRE Approach, *Computer Engineering and Intelligent Systems* 4(4), 10–32.

Bézivin, J. (2006). Model driven engineering: An emerging technical space", *Generative and transformational techniques in software engineering*, 36–64.

Boronat, A., Perez, J., Cars, J. A., & Ramos, J. A. (2004). Two Experiences in Software Dynamics. *Journal of Universal Computer Science*, 10(4), 428–453.

Brinkkemper, S. (2000). Method engineering with Web-enabled methods. In: S. Brinkkemper, Lindencrona E, Sølvberg A (Eds.), *Information Systems Engineering-State of the Art and Research Themes* Springer, 123–133.

Choobineh, J., & Venkatraman, S.S. (1992). A methodology and tools for derivation of functional dependencies from business form. *Information Systems* 17 (3), 269–282.

Dejanović, I., Milosavljević, G. , Perišić, B. , & Tumbas, M. (2010). A Domain-Specific Language for Defining Static Structure of Database Applications. *Computer Science and Information Systems*, 7(3), 409–440.

Dimitrieski, V., Čeliković, M., Kordić, S., Ristić, S., Alargt, A., & Luković, I.(2015) Concepts and Evaluation of the Extended Entity-Relationship Approach to Database Design in a Multi-Paradigm Information System Modeling Tool, *Computer Languages Systems and Structures*, Elsevier Inc. 44, 299–318. doi: 10.1016/j.cl.2015.08.011

EMF Eclipse Modeling Framework, (2017). Retrieved from http://www.eclipse.org/modeling/emf/.

Favre, J. M. (2005). Foundations of Model (Driven) (Reverse) Engineering: Models. *Dagstahl Seminar Proceedings*.

Hainaut, J-L. , Henrard, J., Englebert, V., Roland, D., & Hick, J-M. (2009) Database Reverse Engineering. In: L. Liu and Özsu, T. (Eds), *Encyclopedia of Database Systems*, Springer-Verlag.

Hammer, M., Schmalz, M., O'Brien, W., Shekar, S., & Haldevnekar, N. (2002). *Knowledge Extraction in the SEEK Project Part I*, Tecnical Report TR-02-008.

Kosar, T., Oliveira, N., Mernik, M., Pereira, V. J. M., Črepinšek, M., Da, C. D. & Henriques, R. P. (2010). Comparing general-purpose and domain-specific languages: An empirical study. *Computer Science and Information Systems,* 7 (2), 247–264.

Luković, I., Mogin, P. , Pavićević, J. & Ristić, S. (2007). An approach to developing complex database schemas using form types. *Software: Practice and Experience*, 37 (15), 1621–1656.

Malki, M., Flory, A, & Rahmouni, M. K. (2002). Extraction of Object-oriented Schemas from Existing Relational Databases: a Form-driven Approach. *INFORMATICA*, 13(1), 47–72.

Perez, J., Ramos, I. , & Anaya, V. (2002) Data reverse engineering of legacy databases to object oriented conceptual schemas. *Electronic Notes in Theoretical Computer Science*, 74(4), 1–13.

Popović, A., Luković, I. , Dimitrieski, V. , & Djukić, V. (2015). A DSL for modeling application-specific functionalities of business applications. *Computer Languages, Systems & Structures*, 43, 69–95.

Ristić, S., Aleksić, S., Čeliković, M., Dimitrieski, V. & Luković, I. (2014). Database reverse engineering based on meta-models. *Central European Journal on Computer Science* (*Open Computer Science*), 4(3), 150–159. doi: 10.2478/s13537-014-0218-1

Ristić, S., Kordić, S., Čeliković, M., Dimitrieski, V. , & Luković, I. (2015). A Model-driven Approach to Data Structure Conceptualization. In *Proceedings of the 2015 FEDCSIS*, 5, 977–984. doi: 10.15439/ 978-83-60810-66-8.

Ristić, S., Kordić, S., Čeliković, M., Dimitrieski, V., & Luković, I. (2016). A Model-to-Model Transformation of a Generic Relational Database Schema into a Form Type Data Model, In *Proceedings of the 2016 Federated Conference on Computer Science and Information Systems*, 1577–1580. doi: 10.15439/2016F408

Shu, N.C., (1985). FORMAL: a form-oriented, visual-directed application development system. *Computer*, 38– 49.

Tsichritzis, D., (1982). Form management. *Communications of the ACM* 25 (5), 453–478.

Vara, J., Vela, B., Bollati V.A., & Marcos, E. (2009). Supporting model-driven development of object-relational database schemas: a case study. In: R. Paige (Ed.), *Theory and Practice of Model Transformations*, 181–196.