FORM-DRIVEN APPLICATION DEVELOPMENT

Sonja RISTIC*, Slavica ALEKSIC*, Ivan LUKOVIC*, Jelena BANOVIC**

*University of Novi Sad, Faculty of Technical Sciences Novi Sad, Trg Dositeja Obradovica 6, 21000 Novi Sad, Serbia, tel.: +381214852171, e-mail: {sdristic, slavica, ivan}@ums.ac.rs
**Crnogorski telekom, Podgorica, Montenegro, e-mail: jelenap@t-com.me

ABSTRACT

In the paper a form-driven approach to the application development is presented. Our development environment IIS*Studio supports presented form-driven approach. It is aimed to provide the information system design and generating executable application prototypes. A form type is central IIS*Studio concept, used to model the structure and constraints of various business documents. On the one hand, a set of specified form types represents a platform independent model (PIM) of a real system. On the other hand, it is a PIM of the future software application. Starting from such a platform independent specification, through the chain of transformations, IIS*Studio generates a set of relational database subschemas in the 3rd normal form and also a global relational database schema by integration of the subschemas. It enables a full implementation of database schemas under different target database management systems. IIS*Studio comprises a tool for the formal specification of business applications, and a generator of the executable application prototypes. The case study presented in the paper illustrates main features of IIS*Studio application generator tool and the methodological aspects of its usage. We consider the chain of transformations from a PIM, through the series of platform specific models with different degree of platform specificity, towards executable program code, as a crucial in our approach.

Keywords: form type, transformation chain, automated application generation, Model-driven Software Development (MDSD)

1. INTRODUCTION

Informally, a form is a way of organizing and presenting data. In the context of organization and its information system one can distinguish between business forms and computerized 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.). They may provide an important input source for database (db) schema design, since the most widely used data are gathered or reported in them ([1], [2], [3], [4]). There are tools (CASE or model-driven) aimed at automated application generation using set of forms as input source, such as DeKlarit [5]. On the other hand, users in organizations are experienced in handling and manipulation of databases through screen (computerized) forms which are the most natural interface between user and data. Database systems have been extended to deal with forms and other types of documents used to facilitate the manipulation of data via computerized form (e.g., Oracle SQL Forms, Informix-SQL, Microsoft Access Forms, different reports tools, etc.) [6].

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. There are various research works about the use of forms (business and/or computerized) in different contexts. In order to integrate services in Office Information system, Tsichritzis in [7] introduces the concepts of form type, form template and form instance. In [8] and [9] Shu et al. proposed using forms to specify system requirements. Batini et al. in [10] and Choobineh et al. in [1] and [2] used business forms as input data for the process of database schema design based on generating entity-relationship (ER) diagrams. Choobineh and Venkatraman in [11] presented a methodology and tools for derivation of functional dependencies from business form. A form-based approach for reverse engineering of relational databases is proposed by Malki, Flory, and Rahmouni [6]. This methodology uses the information extracted from both form structure and instances as a database reverse engineering input using an interaction with a user. This approach inspired later research on extracting personalized ontology from data-intensive web application [12]. Namely, S. M. Bensliman, Malki, Rahmouni and Dj. Bensliman based their approach on the idea that semantics can be extracted by applying a reverse engineering technique on the structures and the instances of HTML-forms which are the most convenient interface to communicate with relational databases on the current data-intensive web application. This semantics is exploited to produce a personalized ontology. In a similar manner, Kreutzová, Porubán, and Václavík in [13], claim that a graphical user interface (GUI) is a partial description of the application domain. They argue that it’s possible to automatically analyze existing user interface and search for domain terms in the set of GUI forms. Tailoring some ideas from [14] and [15], Wu et al. in [16] presents a methodology that uses factoring and synthesis to process knowledge involved in forms for designing form-based decision support systems.

The objective of our research is to propose a form-driven approach to application generating. Through a number of research projects lasting for several years, we developed the IIS*Studio development environment (IIS*Studio DE, current version 7.1). It is aimed to provide the information system (IS) design and generating executable application prototypes. Input data for our tool is a set of platform independent specifications. Among these specifications, particularly important is the set of specified form types. A form type (Fig. 1, Fig. 2 and Section 3) is central IIS*Studio concept, used to model the structure and constraints of various business forms.
Through a chain of transformations IIS*Studio generates a set of subschemas in the 3rd normal form and a global relational database schema by integration of the subschemas. It also provides a full implementation of database schemas under different target database management systems, by using its own SQL Generator. IIS*Studio also comprises a tool for the formal specification of business applications, and a generator of the executable application prototypes.

The concept of a form type in our approach mainly corresponds to the concept of a business component that is the main modeling concept DeKlarit tool [5] relies on. DeKlarit, like the IIS*Studio, can generate relational database schema and SQL commands for various DBMSs. Unlike the DeKlarit, IIS*Studio further provides specific concepts and tools for the specification of the transaction programs and business applications ([17] and [18]). Our approach has some similarities with the approaches presented in [1], [2], [10], [11] and [19]. But, besides the set of functional dependencies \( F \) (like in [11]) the initial set of constraints, inferred from a form type, withal consists of: a set of non-functional dependencies \( NF \), a set of special functional dependencies \( F_s \), and a set of null value constraints \( Nv \). Their detailed explanation with examples may be found in [3]. While in approaches presented in [1], [2] and [10], just ER diagrams are generated, IIS*Studio generates relational database schemas and executes an efficient transformation of design specifications into error free SQL specifications of relational database (db) schemas for different DBMSs ([20], [21] and [22]). Although our research does not tackle the same problems as it is in [6] and [12], a common value is a reported necessity of the integration of independently developed database subschemas. Integration of db subschemas in [6] and [12] is done at the conceptual level. In our approach it is done at the implementation instead of the conceptual level [23]. A db schema at the implementation level is expressed by the relational data model.

In the paper the case study is presented that illustrates main features of IIS*Studio application prototypes generator tool, as well as the methodological aspects of its usage. The paper is organized as follows. In Section 2 it is presented a real system of Safe House Center (SHC), whose information system is designed by IIS*Studio. Section 3 explains main concepts needed to understand generation of applications in IIS*Studio. Methodological aspects of the usage of IIS*Studio Application Generator are given in Section 4 through an example of Donations subsystem of the SHC information system. The last section concludes the paper.

2. CASE STUDY: THE SAFE HOUSE CENTER

The Safe House Center (SHC) provides support for those children impacted by domestic violence. It offers a safe and nurturing environment where qualified staff assists youth with their basic needs, assessment, advocacy and stabilization. Besides, it organizes foster care and assists and supervises foster families. The SHC is on a great extent based on donations. In the paper we will present the Donation subsystem of the SHC information system. It is simple compared with the whole SHC information system, but sufficiently complex to allow an illustration of main features of IIS*Studio application generator tool, and the methodological aspects of its usage.

![Fig. 1 The business form Donation Agreement and its form type](image)

![Fig. 2 The IIS*Studio form for specification of the form type Donation Agreement](image)
A donor may be a natural person, a legal entity or a group of citizens. A contribution to SHC may be made by donating: cash, check, goods, remedies, food, bequests, insurance, stocks, bonds or mutual funds, etc. All donations must be recorded. A donation agreement is made between a donor and SHC. In the agreement all donated items are specified. A donor may require staying anonymous, meaning that all external documents should not contain data about donor.

The business forms used in SHC important for Donation subsystem beyond others are: Donor Card (containing basic data about donor: name, type, contact data, etc.) and Donation Agreement (presented in Fig. 1). In the following sections we present the process of form-driven application generating by means of IIS*Studio and using the SHC case study.

3. IIS*STUDIO BASIC CONCEPTS

All the designers' specifications of an IS model created by IIS*Studio belong to an IIS*Studio project. Each project is organized as a tree structure of application systems, where each application system may contain an arbitrary number of form types (see Fig. 5).

A form type is the main modeling concept in IIS*Studio. Initially, each form type is an abstraction of a business form. 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; allowed database CRUD (Create, Retrieve, Update and Delete) operations applied by means of screen computerized forms to manipulate data of an IS; functionalities concerning relationships between generated screen forms, i.e. transaction programs, etc. The business form Donation Agreement (DA-bf) is presented on the left-hand side of Fig. 1. It may be modeled by the form type Donation Agreement (DA-ft). The simplified representation of the structure of the DA-ft, which generalizes the DA-bf, is presented on the right-hand side of Fig. 1.

A form type is a hierarchical structure of form type components. Each component type is identified by its name within the scope of a form type, and has non-empty sets of attributes and keys, a possibly empty set of unique constraints, and a specification of the check constraint. A set of allowed database operations must be associated with each component type. If the update operation is associated with a component type, the set of updatable attributes of the component type must be specified as well. In addition, each attribute of a component type may be marked as mandatory or optional. The form type Donation Agreement (Fig. 1) has two component types: Agreement Heading and Donated Items. For both of them allowed operations are: read, insert, update and delete. An IIS*Studio form for the specification of component type Agreement Heading is presented in Fig. 2.

Each attribute of a component type is selected from a global set of all IS attributes. According to the universal relation scheme assumption present in the relational data model, the attributes are globally identified only by their names. IIS*Studio imposes strict rules for specifying attributes and their domains, and for specifying component type attributes.

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 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. These constraints are processed by the modified synthesis algorithm later in the process of a relational db schema design, as it is explained in details in [3]. The correspondence between a form type and a relation scheme is rarely one-to-one. In our simplified example it is one-to-two since the db schema generated just from the form type Donation Agreement contains two relation schemes: Agreement_Heading and Donation_Item. It is also possible for a relation scheme to contain attributes from different form types, making the aforementioned correspondence to be many-to-one [3]. By creating form types, a designer at the same time specifies: (i) a future database schema, (ii) functional properties of future transaction programs, (iii) and a look of the end-user interface.

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, and on the other hand it is a source for the generation of a sole transaction program and its screen or report form. In order to enable formal specification of functionalities concerning relationships (so called "calls") between generated screen forms, i.e. transaction programs, a concept named business application (BA) is introduced.

The form for BA specification is given in Fig. 3. The rectangles in Fig. 3 represent form types, while the arrows represent calls between them. The form type Catalog of Donations is the entry point of business application since there is no arrow sinking in it. The form type Donation Agreement is called from the form type Catalog of Donations and the form type Donator and it may call the form type Foster Family. In the following section the further explanation of business application specification is given.

![Fig. 3 The IIS*Studio diagram of a business application Donation](image-url)
generation of a program code that covers calls between generated transaction programs, i.e. their forms, and a synchronization of their behavior. The union of the sets of form types of a selected application system and all its application subsystems, alongside with the set of its business applications' specifications represents a technology (platform) independent model (PIM) of the real system being observed.

4. AN EXAMPLE OF APPLICATION GENERATION BY MEANS OF IIS*STUDIO

IIS*Studio relies on the approach that conforms to the principles of model-driven approach. By means of IIS*Studio, a designer specifies only PIM models, because they are free of any implementation details. By the chain of consecutive transformations a set of different semi or fully platform specific models (PSMs) is generated ([3], [17], [24] and [23]). The chain of transformations in IIS*Studio can be divided into three subsets: (i) transformations aimed to generate a formal and an implementation DB schemas (DB schema generator); (ii) UI (User Interface) prototype generators; and (iii) application generators. The first and the third subset of transformations are mandatory in order to generate application prototype (see Fig. 4). The second subset is optional, and helps designer in the process of selecting the best fitting UI. Therefore, its presentation is omitted here. A case study illustrating the first subset of transformations may be found in [3] and [23]. In the following text a case study illustrating the third subset of transformations is presented.

The first step of the process of business application specification and generation by means of IIS*Studio is Project Specification (Fig. 4). Fig. 5 presents the Safe House project tree, containing one application system Donation with form types Catalog of Donations, City, Donation Agreement, Donator and Foster Family that are modeling corresponding business forms. The form types are specified through IIS*Studio screen forms, like the one in Fig. 2. After selecting the tab Component type, designer may insert/edit/delete component type, or/and insert/edit/delete the attributes (Fig. 6) of selected component type, or/and specify component type constraints (key, unique or check constraints).

During the process of attribute specifications one would specify the attribute title, behavior (is it modifiable or not), allowed operations, obligingness and default value through tab Definition (see the left-hand side of Fig. 6). This part of specification is important for the database schema generation process. The tab Display is important for the application generator, because it enables specifying the display characteristics of an attribute. The selected display option for the attribute TypeD is ComboBox (Fig. 6), and possible values are a natural person, a legal entity or a group of citizens.

The example of Donation Agreement form type illustrates the multiple role of form type. At the same time it is: a model of Donation Agreement business form (Fig. 1); a data model – one can see it as a conceptual database schema (Fig. 2); a model of computerized (screen) form (Donation Agreement screen form in Fig. 8); and a model of transaction program, since the functional properties of future transaction program are specified. The form type structuring rules provide an automatic inference of the set of attributes and relational database constraints.

The next step is business application specification (Fig. 3). Specifying the form types must precede the design of a business application, because the specification
of a business application in IIS*Studio comprises a structure of the selected form types. The first step in specifying the business application structure is to select necessary form types.

One of the selected form types has to be designated as an entry-point form type. The screen form generated from the entry-point form type is the first one accessible to end-users, when they initiate the business application, and by means of they can access the other (subordinated) forms in the application. In Fig. 3, the form type Catalog of Donations is declared as the entry-point. This form type is specific since it is menu form type. The application generator will transform it into menu screen, aimed at selecting possible further choices (see Fig. 9).

After the selection of form types, it is necessary to specify their mutual relationships ("calls") so as to specify the business application structure. In the business application diagram in Fig. 3 the arrow between the Donator and Donator Agreement form types indicates a call specification within a business application. In this case, Donator is the calling form type, and Donator Agreement is the called form type. By selecting a form type and Edit option the form types that are to be called from a selected form type are specified (Fig. 8). The consequence of such a specification is that the screen form generated from the calling form type has to support calls of the screen form generated from the called form type. For example, clicking the button Donation Agreement on the screen form Donator causes calling the corresponding screen form (Fig. 9). Besides, each call specification in IIS*Studio has the following properties (Fig. 8): Passed values – the list of passing values from the calling to the called form types; Calling mode – the rules for data selection and transferring data from the calling to the called form type; Calling method – a behavior of the calling and the called form type; and UI positioning – positioning properties of the UI control item for executing the call.
A business application specification can be regarded as the dynamic part of a GUI (Graphic User Interface). In order to generate appropriate transaction programs, the static aspects of GUI may be specified as well. All visual (displayable) elements are a part of GUI static aspects. The IIS*UIModeler is an integrated part of the IIS*Studio DE, aimed at modelling of GUI static aspects. By means of IIS*UIModeler a designer specifies UI templates. UI template specification contains attribute values that describe common UI characteristics, such as: screen size, main application window position, background/foreground colour, etc. The set of UI template attributes can be classified into six groups: global attributes (Fig. 10), screen form attributes (Fig. 11), table attributes (Fig. 12), panel attributes (Fig. 13), display/update attributes (Fig. 14) and button attributes (Fig. 15). The specification of the UI template is stored in the IIS*Studio repository.
UI template specifications are independent from any specific IS project specification, generated by means of IIS*Studio tool. The same UI template may be used for application prototype generation of different ISs, as well as the same IS project specification may be transformed in different ways by means of different UI templates. The specification of UI template may be seen as a fully platform independent UI model. The detailed description of the IIS*UIModeler and UI template attributes may be found in [25].

A business application specification together with the specifications of selected UI templates is a source for the generation of a program code that covers calls between generated transaction programs, i.e. their forms, and a synchronization of their behavior. The process of transaction program generating has three steps: (i) generating of the subshemas of the form types incorporated in a business application; (ii) generating of the UIML (User Interface Markup Language) prototype specification alongside with generating of the executable user interface Java code from the UIML specification; and (iii) generating of the executable business application specification. Some of the generated screen forms of Donation application that supports Donation subsystem of Safe House IS are presented in Fig. 9.

5. CONCLUSIONS

The form-based approaches in the software engineering are present for more than two decades. Some of them are for: database analysis and design; extracting object-oriented db schemas from relational databases; extracting personalised ontology from data-intensive web applications; designing form-based decision support systems; etc. In our approach we are using the notion form-driven, instead of form-based, inspired with the different meaning of the model-based and model-driven approaches. A form type is the central concept of our IIS*Studio tool aimed at automated IS design and application generation. By creating form types, a designer specifies at the same time: (i) a future database schema, (ii) functional properties of future transaction programs, and (iii) a look of the end-user interface. One of the main assumptions of the model-driven approach to software system development is that software 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 business forms. The focus of software development is shifted from the technology domain toward the problem domain. In our future work, in order to justify the correctness of the form-driven adjective we aim to investigate the reverse influence of the changes in the database schema to the screen and business forms. Category theory ([26], [27]) provides a kind of common language and tool, in which a sketch is a specification based on graphs as the formal structure. We plan to investigate a possible usage of category theory: i) in order to improve the performance of generated code, on the one hand ([26]), and ii) as a common language and tool for software engineering task instrumentation on the other hand ([27]).

ACKNOWLEDGMENTS

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

REFERENCES


Received January 3, 2012, accepted March 23, 2012

**BIOGRAPHIES**

**Sonja Ristic** works as an associate professor at the University of Novi Sad (UNS), Faculty of Technical Sciences (FTS), Serbia. She received two bachelor degrees with honors from UNS, one in Mathematics, Faculty of Science in 1983, and the other in Economics from Faculty of Economics, in 1989. She received her Mr (2 year) and Ph.D. degrees in Informatics, both from Faculty of Economics (UNS), in 1994 and 2003. From 1984 till 1990 she worked with the Novi Sad Cable Company NOVKABEL–Factory of Electronic Computers. From 1990 till 2006 she was with High School of Business Studies -Novi Sad, and since 2006 she has been with the FTS (UNS). Her research interests are related to Database Systems and Software Engineering.

**Slavica Aleksic** received her M.Sc. degree from FTS (UNS). She completed her Mr (2 year) degree at the FTS (UNS). Currently, she works as a teaching assistant at the FTS (UNS), where she assists in teaching several Computer Science and Informatics courses. Her research interests are related to Information Systems, Database Systems and Software Engineering.

**Ivan Lukovic** received his M.Sc. degree in Informatics from the Faculty of Military and Technical Sciences in Zagreb in 1990. He completed his Mr (2 year) degree at the University of Belgrade, Faculty of Electrical Engineering in 1993, and his Ph.D. at the University of Novi Sad, Faculty of Technical Sciences in 1996. Currently, he works as a Full Professor at the Faculty of Technical Sciences at the University of Novi Sad, where he lectures in several Computer Science and Informatics courses. His research interests are related to Database Systems and Software Engineering. He is the author or co-author of over 80 papers, 4 books, and 30 industry projects and software solutions in the area.

**Jelena Banovic** received her M.Sc. degree from the Faculty of Natural Sciences and Mathematics in Podgorica. She completed her Mr degree at the University of Montenegro, Faculty of Natural Sciences and Mathematics, and her Ph.D. at the University of Novi Sad, Faculty of Technical Sciences in 2010. Currently, she works as system engineer at the Internet Crna Gora d.o.o. Her research interests are related to Information Systems, Database Systems and Software Engineering.