ReingIS: A Toolset for Rapid Development and Reengineering of Business Information Systems

Renata Vaderna, Željko Vuković, Gordana Milosavljević, Igor Dejanović
University of Novi Sad, Faculty of Technical Sciences, Chair of Informatics, Serbia
{vrenata, zeljkov, grist, igord}@uns.ac.rs

***Abstract*—ReingIS is a set of tools for rapid development of client desktop applications in Java. While it can be used to develop new applications, it is primary intended use is for reengineering existing legacy systems. Database schema is extracted from the database itself and used to generate code for the client side. This remedies the fact that most existing systems do not have valid documentation.**

I. INTRODUCTION

Enterprises often use their information system (IS) for a long time. Maintaining such systems can become hard and expensive. They may use libraries, frameworks or even languages that are no longer actively maintained. This can also lead to security threats. Developers who are fluent in technologies used to develop the IS may become scarce. These are some of the reasons why reengineering of existing legacy systems may be necessary. The new information system must take into account existing data structures, persisted data, business processes and flows modeled by the legacy system.

An ideal starting point for developing a new IS would be the technical documentation of the old one. However, such documentation can often be incomplete, not properly maintained (describing the initial version of the legacy system without reflecting changes that have been performed over time) or even non-existent. Therefore, some reverse engineering is usually needed first. Developers may also seek information from the user documentation if it is available. Users themselves can also participate in the process. The goal of these efforts is to replicate functionality of the legacy system while preserving or migrating its contained data.

ReingIS toolset consists of a database schema analyzer, a code generator and a framework. The database analyzer extracts the schema information (tables, columns, types, constraints, etc.) from the database itself. A graphic user interface then allows the developer to define information that could not be read from the schema, like labels and menu structure. Afterwards, the code generator generates components on top of the generic enterprise application framework. The result is an application the users can run straight away in order to inspect it and note necessary changes. These changes are then made using the generator GUI and the process is repeated until satisfactory results are achieved. The toolset also provisions inserting manually coded components and modifications in such a way that subsequent code generation will not overwrite manual changes.

ReingIS toolset facilitates quick introduction of new team members or in-house programmers who maintained the legacy system. The problem with object-oriented technologies is that they are too sophisticated – successful designing and programming using objects takes well-educated and mentored developers, not novices. Classes in class libraries serving as building blocks are too small so the novice has no support. With coarse-grained components and tools built upon the knowledge and experience of senior team members, a novice gets enough support to almost immediately be productive, with the opportunity to gradually master the secrets of modern technologies.

II. RELATED WORK

JGuiGen [1] is a Java application whose main purpose is generation of forms which can be used to perform CRUD (Create, Read, Update, Delete) operation on records of relational database tables. Similarly to our application, JGuiGen can work with a large number of different databases. On top of that, code generated by JGuiGen handles usage by multiple users. Information regarding the database tables is not entered manually. The database is analyzed and descriptions of its tables and their columns are stored in a dictionary, optionally accompanied by comments added by the user with the intention of describing certain elements in more detail, as well as the database schema change history.

When defining a form, it is possible to choose a table contained by the previously mentioned dictionary which will be associated with it. One graphical user interface component is generated for each column of the table and its properties can be customized. Furthermore, JGuiGen also puts emphasis on localization, input validation, accessibility standard [2], and ease of generation of documentation. On top of that, it enables creation of simple reports, which are usually quite significant to business applications.

However, unlike our solution, it does not provide a way in which a user would be able to specify positions of user interface components. They are simply placed one below the other. Additionally, the number of components which can be contained by one tab cannot be higher than 3, while our solution does not enforce this limitation. Associations between two forms cannot be specified using JGuiGen, which means that this feature would have to be implemented after all forms are generated. Similarly, there is no support for calling business transactions.

In [3] the authors use a domain specific language (DSL) to describe tables and columns of a relational database and how they are mapped to user interface components, such as textual fields, lists, combo boxes etc. Description of columns should also contain instructions on how to lay these components out – their vertical and horizontal positions and lengths for components which have it. The generator then uses this information to generate fully functional forms which can perform various operations on records of previously specified tables. The authors prefer textual notation to a visual one stating better support for big systems as the reason. However, it can be noticed that this solution, just like previously described one, does not support associations between forms, although it is a quite important concept for all, and especially more complex business application. Furthermore, it is not possible to describe and generate activation of business reports and transactions. Finally, as mentioned, this solution demands manual description of tables and columns instead of analyzing the database meta-schema, making the whole process more time consuming and error-prone.

Module for generating application prototypes of the IIS* Case tool [4] is another interesting project which generates fully functional applications which satisfy previously defined visual and functional requirements, allowing records of database tables to be viewed and edited. The process of generation of these applications includes generation of UML (User Interface Markup Language) documents which specify visual and functional aspects of the applicative system and their interpretation which uses Java Renderer. This interpreter transforms UML specifications into Java AWT/Swing components. Furthermore, the module contains Java classes which provide the ability to communicate with the database and pass parameters and their values between forms. Visual properties of the applicative system can be defined by the user by choosing one of the available user interface templates and specifying visual attributes and groups of fields. This is done using another module of the tool. Generation of subschemas of types of forms, whose results provide information which can be used to create SQL queries, is done before the application is generated.

The main difference between this module and our solution lays in the fact that the IIS* Case module is supposed to be used when developing new systems, while ReingIS is optimized to be used when reengineering existing projects with the desire of keeping already existing database schema.

III. IMPLEMENTATION

ReingIS was developed using Java programming language and enables generation of a fully functional client side of a business application based on the meta-schema of an existing database. The architecture of the system is shown in Fig. 1. The application for specifying user roles and permissions is referenced as “security”. Each component of ReingIS will be described in more detail in the upcoming sections.

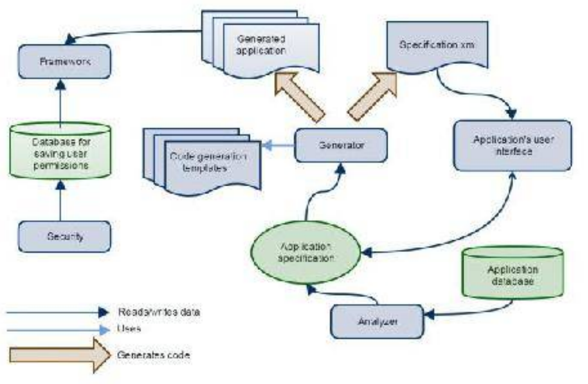


Figure 1. Architecture of the system’s framework

The framework provides a generic implementation of all basic concepts of business applications: standard and parent-child forms, data operations (viewing, adding, editing, deleting and searching data), user interface components which enable input validation, activation of reports and stored procedures. For this reason, the framework makes development of business applications easier and quicker. Since all of the important elements were already implemented and tested, that does not need to be done when creating each specific form. Implementation of application elements within the framework follows our standard for user interface specification [5].

1) User interface standard of business applications

The most important elements of our standard, supported by the framework are: standard and parent-child forms and form navigation. The complete description can be found in [5, 6].

Standard form was designed with the intention of making all data and operations which can be performed on them visible within the same screen. Standard operations (common for all entities) are available through icons located in the upper part of the form (toolbar), while specific operations (reports and transactions) are represented as labeled buttons and located on the right side of the form.

Navigation among forms includes zoom and next mechanisms. Zoom mechanism enables invocation of the form associated with the entity connected with the current one by association, where the user can pick a value and transfer it back to the form where zoom was invoked. On the other hand, next mechanism provides the transition from the form associated with the parent entity to the form associated with the child entity in a way that the child form shows only data which was filtered according to the selected parent.

Parent-child form is used to show data which has a hierarchical structure, where every hierarchy element is modeled as an entity in the database and is shown within its standard panel. Panel on the n^{th} level of the hierarchy filters its content based on the chosen parent on the $(n-1)^{th}$ level.

2) Implementation of generic standard and parent-child forms

The core component of the framework is the generic implementation of the standard form (Fig. 2), which allows creation of fully functional specific forms by simply passing description of the table associated with the form in question, its columns and links with other tables, as well as the components which will be used to input data.

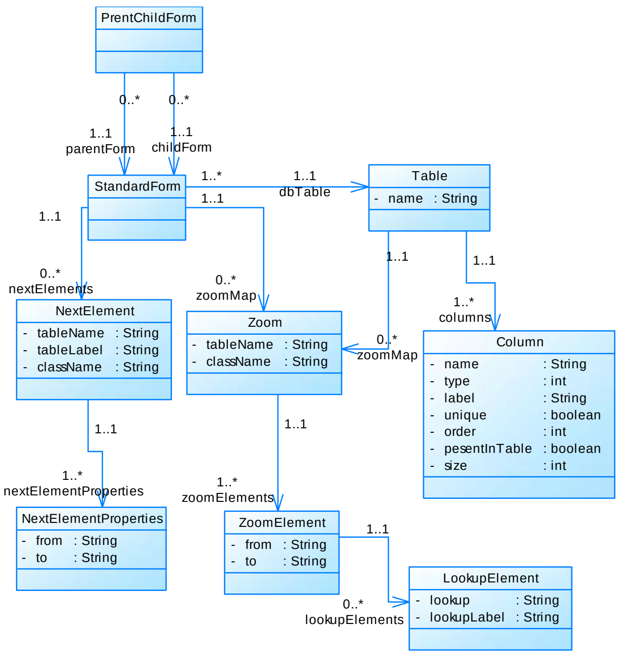


Figure 2. Class diagram of standard and parent-child forms, the framework's core components

The information about the tables and its columns which are passed to the generic forms are represented with classes `Column` and `Table`. Attributes of these classes are used during the GUI construction phase, as well as for dynamic creation of database queries and retrieving their results. This eliminates the need to write any additional code for communicating with the database.

The description of the table's links with other tables is necessary for generic zoom and next mechanisms and is represented by the following classes: `Zoom`, `ZoomElement`, `LookupElement`, `NextElement` and lastly `NextElementProperties`. Classes `NextElement` and `Zoom` contain data related to the tables connected with the current one, as well as names of Java classes which correspond to forms associated with those tables (attribute `className`). The name of a class is all that is needed to instantiate it using reflection. Classes `ZoomElement` and `NextElementProperties` contain information regarding the way in which the columns of one table are mapped to the columns of the other one. This is important for automatic retrieval of the chosen data when zoom mechanism is activated and for automatic filtering when a form is opened using next mechanism. If additional columns of tables connected through the zoom mechanism with the current one should be shown (for example, name and not just id of an entity), their names and labels should be specified as well. Class `LookupElement` is used for this reason.

Validation of the entered data can be enforced on the form itself by using specially developed graphical user interface components. The query is not sent to the database unless all validation criteria is met, which reduces its workload. These components are:

- `ValidationTextField` – represents a textual field which can be supplied with validation data, such as the minimal and maximal possible length of the input, indicator if the field can only contain digits or other characters as well, the minimal and maximal value for numerical input, indicator if the field is required, and, finally, patterns, i.e. regular expressions that the input value is checked against (for example, this can be used to validate an e-mail address).
- `DecimalTextField` –field which is used to enter decimal values. The input is aligned to the right side and the thousands separator is automatically shown when needed. Maximal length of the number and the number of decimals can be specified.
- `TimeValidationTextField` – field used to input time as hours, minutes and seconds.
- `ValidationDatePicker` – component which extends `Jcalendar` component, which is licensed under *GNU Lesser General Public* license.

The generic parent-child form was implemented as two joined standard forms linked through the next mechanism. Creation of a specific form of this type only requires two previously constructed standard forms to be passed.

3) Reports and transactions

Calling previously created Jasper [7] reports and stored procedures can be done through a menu item of the application's main menu, as well as inside standard forms. It is necessary to define parameters needed by the procedure or a report, if there are any. Everything else is done automatically. Framework also provides a generic parameters input dialog, which needs to be extended and supplied with specific input components.

B. *Analyzer*

The analyzer uses the appropriate Java Database Connectivity (JDBC) driver and establishes connection with the database which needs to be analyzed and, using `java.sql.DatabaseMetaData` class, finds the information regarding its tables and their columns, relations and primary and foreign keys. The end user doesn't need to know which JDBC driver and Database Management System (DBMS) are used, which means that a large number of different databases can be analyzed.

Based on the retrieved information, the analyzer creates an initial, in-memory, specification of the business application (i.e. instances of the `StandarForm` class are created), using the following transformation rules:

- Every table is mapped to a standard form
- Every column of the database tables is mapped to an input component contained by the form
- Names of the columns and tables are used as labels of components and forms, in that order
- Types of the input components corresponding to the columns are determined based on the types of those columns. A textual field is added when the column is of a textual type (char, varchar), a date input component is added when the column is of a date type, a textual field which automatically enforced validation which only enables numbers to be entered is added when the column is of a numerical type

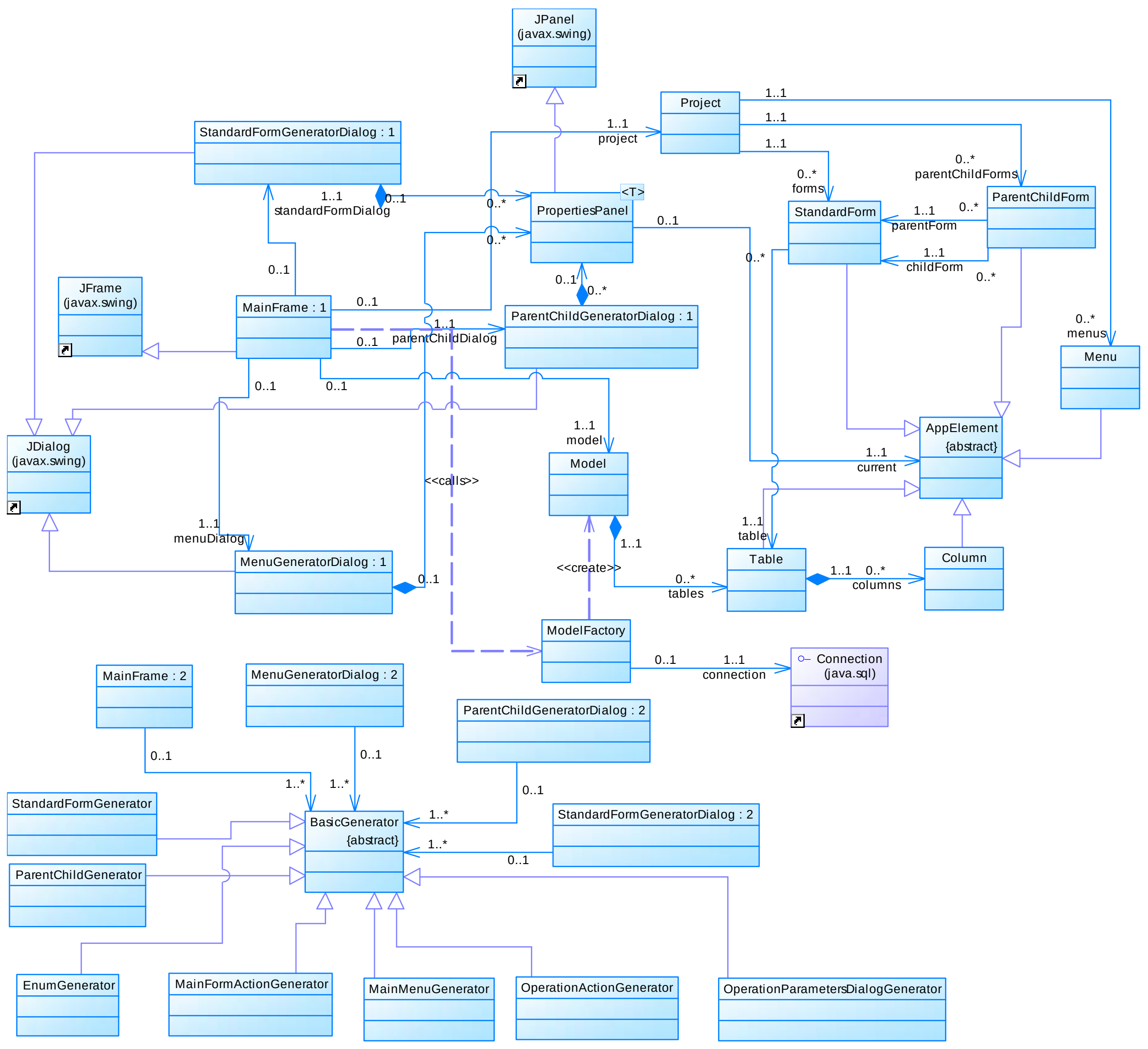


Figure 3. Class diagram of the analyzer, generator and its user interface

- Length of a textual field, as well as width of the corresponding column in the form's table are determined based on the length of the database column

Analysis of the database meta-schema is done when the application is run for the first time or when it is connected to a different database. The acquired data, regarding found database tables and their columns, is saved to a XML file, which is then loaded when the application is started again. Therefore, the time consuming database analysis process is avoided unless it is necessary. If needed, the user can activate the analysis from the application at any time.

The class which provides the mentioned functionality is `ModelFactory`, while the class `Model` represents the database meta-schema, as shown in Fig. 3. This class contains a list of tables discovered during the analysis. Each of them is described by class `Table`, which contains a list of columns represented by class `Column`.

This default specification created by the analyzer can later be changed and enhanced through the generator's components, grouping of fields into tabs and panels, creation of zoom, next and lookup elements and parent-user interface by the users. The mentioned application enables additional specification of labels of forms and child forms etc.

C. *Code Generator User Interface*

The user interface of the generator application consists of three dialogs represented by classes `StandardFormGeneratorDialog` (for specifying standard forms), `ParentChildGeneratorDialog` (for specifying parent-child forms) and `MenuGeneratorDialog` (for specifying menus) – Fig. 3. These dialogs are activated through the main form of the generator application.

The mentioned dialogs rely on instances of classes `StandardForm`, `ParentChildForm` and `Menu` to store data needed to generate forms and menus, such as sizes, titles and input components of forms and names and structures of menus.

When a form is first created, default settings are set (for standard forms, they are based on the analysis results). Therefore, the generator can generate usable program code straight away. These settings can me modified by the users through certain panels contained by the mentioned dialogs. These panels are instances of class `PropertiesPanel` – a parametrized class which enables various properties of an element associated with it to be changed (class `AppElement`). Class `AppElement` is extended by all classes which represent an element of a business application or one of its parts. In order to enable work to be saved and edited on a later occasion, the term project is introduced. It is represented by class `Project`, which contains lists of defined standard and parent-child forms and menus.

1) Specifying standard forms

Dialog for specifying standard forms enables modification of default form settings set by the analyzer i.e. based on the database meta-schema. The following properties of a form can be adjusted: label, size, associated database table, initial mode (add, search, view/edit), allowed data operations, grouping of contained components into tabs and panels, links with other forms through zoom and next mechanism, specific operations activated from the form – reports and transactions. One database table can be associated with multiple forms (Fig. 4). Additionally, the following properties of form components can be set: label, indicator if its content can be edited, position – specified using MigLayout [10] constants, validation rules etc.

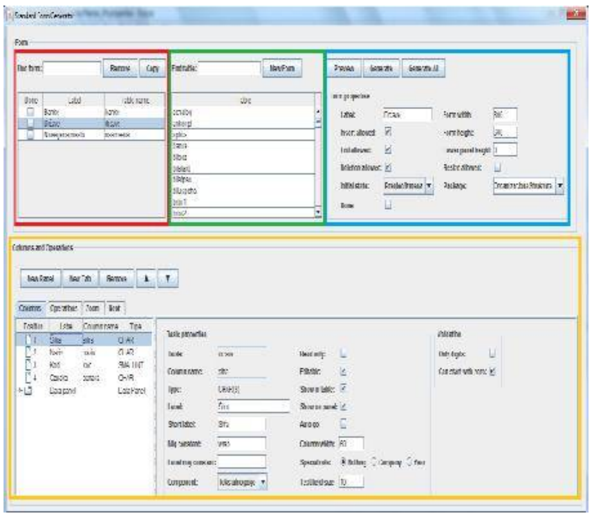


Figure 4. Dialog for specifying standard forms

The dialog consists of:

- A part for choosing associated database table – within green borders
- A part for selecting and searching previously created forms – within red borders
- A part for setting basic form properties – within blue borders
- A part for specifying links with other forms, component groups and component properties – within orange border

Fig. 7 shows an example of a generated standard form.

2) Specifying parent-child forms

Dialog for specifying parent-child forms (Fig. 6) enables users to choose two standard forms, one of which will be the parent, while the other one will be the child. After this step is performed, a new parent-child form is created and default initial values of its properties, such as title and size, are set. Therefore, the corresponding Java class can be generated at any moment and the form's current appearance can be previewed if so desired.

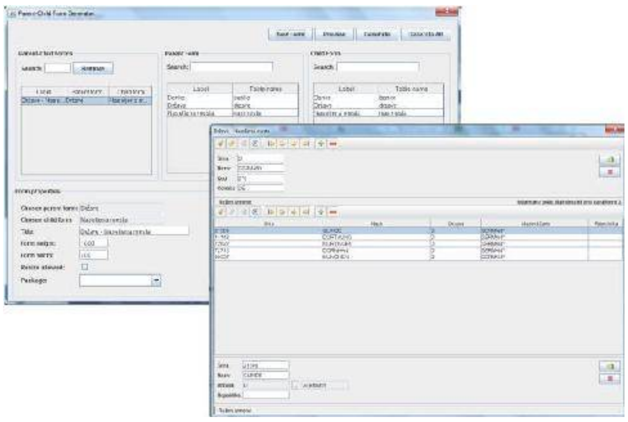
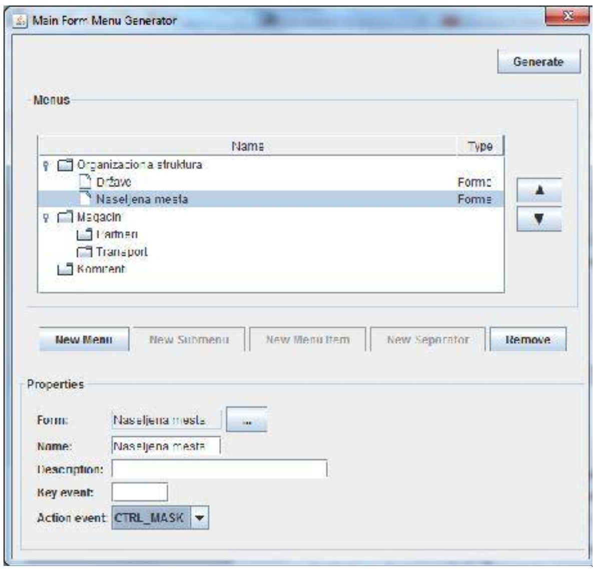


Figure 6. Dialog for specifying parent-child forms and the resulting form

3) Specifying menus

Menus of business applications can be specified using another dialog of the generator application. It is possible to create menus with submenus, which can contain additional submenus as well. The following properties can be defined for menu items: name, shortcut, description (mapped to tool tip text), form or report which will be activated by clicking on the item. If the menu item activates a report, it is necessary to specify parameters which will be passed to it. This dialog is shown in Fig. 5.



Main Form Menu Generator

Menus

Name	Type
Organizacioná struktura	
Države	Forma
Naseljena mesta	Forma
Magacin	
Partneri	
Transport	
Komitent	

Properties

Form: Naseljena mesta

Name: Naseljena mesta

Description:

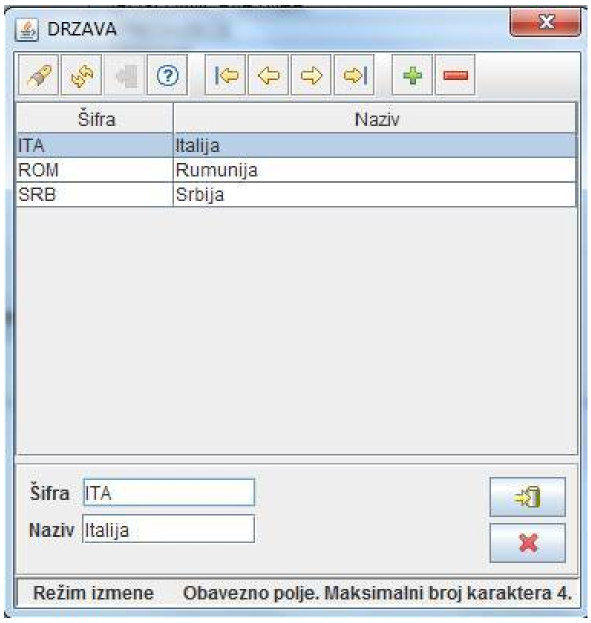
Key event:

Action event: CTRL_MASK

Figure 5. Dialog for specifying application menus

D. *Code Generator*

Using the database metadata and additional information given by the developers, code is generated (using Freemarker [9] template engine) for the framework components. This includes the main form and its menus, standard forms and parent-child forms. Generated form classes extend generic classes that are a part of the framework. It can be noted that the generator supports synchronization with changes made to the database after the specification of the forms was started. If some columns were added or removed from a database table, the corresponding components are automatically added to or removed from the appropriate form. Similarly, when a database table is deleted, the form associated with it is also deleted.



Šifra	Naziv
ITA	Italija
ROM	Rumunija
SRB	Srbija

Šifra ITA
Naziv Italija

Režim izmene Obavezno polje. Maksimalni broj karaktera 4.

Figure 7. A generated standard form

E. Security

Handling of security concerns for the generated application is based on the Apache Shiro framework [8]. Shiro is a Java framework that supports authentication, authorization, cryptography and session management. Within the framework each operation is given an identity string. Based on the currently active user and the operation identity it can be determined if the operation is allowed. A user interface was implemented in order to allow managing users and groups and assigning them access rights. It is also possible to import existing users from the legacy system.

In order to facilitate construction of the user interface for the end application, a set of classes was implemented. They extend the standard Swing classes and make them aware of the security context. When these components are used, user interface elements are automatically disabled or hidden if the current user is not allowed to perform the action that they are associated with. `SecureAction` is an abstract class which extends Swing's `AbstractAction`. The `actionPerformed`() method is redefined and made final. This method uses Shiro to authorize the action and then calls the `action`() method. This method is to be implemented by classes extending the `SecureAction` class. The identity string of the action is returned by `getActionIdentity`() method, which is also to be implemented by subclasses. Classes `SecureJButton`, `SecureJMenu` and `SecureJMenuItem` extend classes `JButton`, `JMenu` and `JMenuItem` respectively and are to be used with `SecureAction`. All classes register themselves as an `AuthenticationListener` in Shiro. This enables them to react interactively when the current user is switched.

IV. Conclusion

The toolset presented here enables reengineering of legacy enterprise information systems. It uses existing database structure as a basis for replicating functionality and preserving data contained in the original system. After adding user interface details that could not be extracted from the schema (labels, menus, etc.) developers can run the code generator which produces generated code on top of the generic framework, resulting in a runnable application. This application can then be presented to the users who can verify its functionality. Since it is possible to repeat the code generation while maintaining all settings and customization, the process also supports forward engineering and incremental development.

The process could be further improved if we were able to extract user interface elements in addition to the database structure. In order for the approach to remain applicable for a wide variety of applications, this requires development of a generic UI element extractor. Each plugin could provide extraction facilities for one technology, e.g.: COBOL screens, .NET forms, Swing frames, web pages, etc.

References

[1] JguiGen, http://jguigen.sourceforge.net

[2] A11y standard, http://a11y.me

[3] Lolong S, Kistijantoro A, Domain Specific Language (DSL) Development for Desktop-based Database Application Generator, International conference on Electrical Engineering and Informatics, Bandung, Indonesia, 17-19 July 2011.

[4] Ristic S., Aleksic S., Lukovic I., Banovic J., Form-Driven Application Development, Acta electrotechnica et Informatica, Vol. 12, No. 1, 2012, pp. 9–16, DOI: 10.2478/v10198-012-0002-x

[5] Milosavljevic G., Ivanovic D., Surla D., Milosavljevic B., Automated construction of the user interface for a CERIF-compliant research management system, The Electronic Library, Vol. 29 Iss: 5, pp.565 - 588

[6] Perišić, B., Milosavljević, G., Dejanović, I., Milosavljević, B., UML Profile for Specifying User Interfaces of Business Applications, Computer Science and Information Systems, Vol. 8, No. 2, pp. 405-426, DOI 10.2298/CSIS110112010P, 2011.

[7] Jasper Reports, http://jaspersoft.com/

[8] Apache Shiro framework, http://shiro.apache.org/

[9] Freemarker Java Template Engine, http://freemarker.incubator.apache.org/

[10] MigLayout - Java Layout Manager for Swing, SWT and JavaFX2, http://www.miglayout.com