Software Tools for Rapid Development and Customization of Medical Information Systems

Petar Rajković, Dragan Janković, Tatjana Stanković
University of Niš, Niš, Serbia
{rajkovicp,dragan,jankovic,tanja}@elfak.ni.ac.rs

Vladimir Tošić
NICTA & University of New South Wales, Sydney, Australia
vladat@computer.org

Abstract — We present data modeling and code generation tools for easier and faster development and customization of electronic health record (EHR) based medical information systems (MIS). In development of MIS, it is usually necessary to create a large number of different, but somewhat similar, graphical user interface (GUI) forms and database tables corresponding to different medical data. This process can be inefficient and time consuming. Our software tools enable power users to define meta data models, from which the tools automatically generate database tables, data object model classes and several common components (input forms, selection components, components for tracking measured values, reporting profiles, access control configurations) for EHR-based MIS. While we first developed these tools for clinical and ambulatory MIS in Serbia, they can be used in other countries, due to the built-in flexibility and multi-language support.

Keywords—electronic health record (EHR), medical information system (MIS), code generation, rapid application development

I. INTRODUCTION AND MOTIVATION

In medical information systems (MIS), collected medical data has greater scientific significance than any other data (e.g., financial payment data). Apart from being used for medical (and, more generally, healthcare) decisions, the collected medical data can be used in different ways for subsequent research and in education of future medical staff.

To enable entry of various relevant medical data and generation of appropriate reports, MIS should support diverse healthcare procedures/processes. The diversity of medical processes causes the need for a large number of different graphical user interface (GUI) forms. For example, there are many medical specializations and they involve different medical procedures, each one of which could be associated with many different medical examinations, lab analyses, therapeutic treatment reports and many other different documents [1]. Usually, each data entry form should have a mapping to a table in the MIS database. For different forms, it often means that different tables in the MIS database have to be created. However, in addition to differences, the required forms and thus data tables usually have many similarities. Creating many similar data tables and supporting forms can be an annoying and long process. Therefore, we decided to develop software tools that make this development faster and more reliable. These software tools are the focus of this paper.

Closely related to the need for rapid development of MIS is the need for customization of MIS, which is also associated with generation of various forms and data tables. Several ways of customization can occur. One of them is customization of MIS developed in one country to the healthcare system and local circumstances in another country. This involves translation of forms to different languages, but oftentimes also adjustment of the structure of various forms (and database tables). We analyzed several existing MIS solutions and realized that administrative and insurance-related parts of MIS are usually country specific, while the MIS parts for collection of medical data (e.g., from medical examinations) often have significant similarities (along with some variations that require customization). The need for customization also exists within a single healthcare system, when MIS is tailored to specifics and processes in particular healthcare organization (e.g., hospital or ambulant). MIS have much greater chance for practical acceptance when they are customized to the existing healthcare processes and work habits of medical personnel than when medical personnel has to modify their work to comply with the MIS [3]. Thus, there is a need for software tools that not only enable rapid development, but also easy customization of MIS.

Since 2002, our research group has worked on a number of projects [2, 3, 4, 5, 6] for developing MIS in Serbia, in which the needs for software tools for rapid development and customization of MIS were very prominent. For example, our 2002 MIS for medical data acquisition and reporting in cardio, neurological and pediatric clinics in Niš [5] supported specialist doctors with around 150 different examinations and analysis reports kept within a general patient record of hospitalization. Then, in 2008 the Serbian Ministry of Science started a new project [2] to extend our 2002 MIS by implementing specialist support modules for other clinics, including also support for the ambulatory part of the Serbian public healthcare system [3] and introducing effective collaboration and data exchange procedures between various clinics and ambulatory institutions. While this new project built upon the previously developed MIS, it required many new forms and reports with appropriate mapping database tables in the background. While there are off-the-shelf form generators, such as Microsoft’s Data Form Wizard and OnyakTech’s H2O, they can only generate forms with all fields from related database tables with basic

The presented medical information systems and tools are results of projects supported and funded by the Ministry of Science and Technological Development of the Republic of Serbia.

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.
add/update/delete/navigate options. We needed much more in our MIS: powerful custom form functionalities to fit into our system, more flexible ways to access data, filtered access data from multiple tables, version tracking, specialized selection components, reporting profiles, generation of both Windows and Web forms at the same time, and defining and incorporating objects in the data object model. Since no off-the-shelf tools or routines from the used development environment satisfied our needs, we decided to follow the modeling methodologies from [7, 8] and develop data modeling and code generation tools that speed up MIS development in our projects and are suitable for developing similar MIS. In particular, due to the similarities in healthcare systems in countries of former Yugoslavia [3], the applicability of our solutions in various countries was our goal from the beginning. Another significant characteristic of our new MIS is high level of customization through different configurations.

In the next section of the paper, we present an overview of our solution. In Section III, we summarize the main aspects of the data modeling tool. In Section IV, we focus on the code generation tools that automatically create data object model classes and several common components (input forms, selection components, components for tracking measured values, reporting profiles, and access control configuration options) in MIS. In Section V, we describe how we evaluated (verified and validated) our data modeling and code generation tools. The last section summarizes conclusions.

II. SYSTEM OVERVIEW

Since we have been involved in development of several MIS in the past, our general strategy in development of new MIS systems is to support partial reusability and extensibility of past solutions (e.g., the base EHR and administrative parts of the MIS described in [3]), where appropriate. The general architecture of our MIS is organized into several layers [3]: database (Microsoft SQL Server 2005), data object model (Microsoft Entity Framework), virtual EHR, synchronization services and end user (e.g., doctors’, nurses’) applications and modules.

To make our MIS useful in complex healthcare systems with many different specialists, we enabled plug-in addition of specialists’ supporting modules (SSM), but these modules require many different Windows forms and other GUI components. Our generator tool helps in rapid development and customization of various GUI components and database tables for these modules. In this way, the development of new components for our new MIS is more efficient than starting development from the beginning. It also results in fewer development errors (e.g., in mappings between database tables and code that manipulates this data) and a uniform end user interface.

Before we give a system overview, it is important to explain who is actually involved in the development and customization of our MIS. We distinguish between power users who apply the data modeling and generator tools and end users who simply work with (e.g., enter data into) the resulting MIS. The end users are medical professionals, such as general purpose (GP) doctors, specialist doctors, nurses, laboratory technicians, etc. They need to know medical processes/procedures and other healthcare domain knowledge. Our MIS does not assume high computer literacy of these end users (as explained in [3], in developing countries such as Serbia computer literacy among medical professionals is low) and we adapt our MIS to their existing work practices instead of requiring from them extensive training in using our MIS. On the other hand, the power users of data modeling and generator tools must have some knowledge and understanding of both the medical domain and IT systems (particularly MIS). IT knowledge is particularly necessary for the use of the generator tool, because it creates different software components. Contrary, the results of the data modeling tool (e.g., used terminology, modeled fields, and value ranges) must be developed under supervision and validated by medical experts because these models represent domain-specific knowledge from some field of medicine. In many cases, these power users are IT professionals working in medical/healthcare facilities (e.g., as system administrators or developers of custom MIS software). Medical professionals with good understanding of IT can also be power users. Oftentimes, no single individual has the right combination of medical and IT knowledge and then a “power user” is actually a small team of medical and IT people with complementary expertise.

Figure 1. Developed tools and generated components

The data modeling tool and the generator tool are two Microsoft .NET libraries combined in one Windows forms application. Relations between these tools and the generated components are shown in Figure 1. The development process is:

- A power user defines a meta data model using the data modeling tool and save it is as an XML (Extensible Markup Language) file or tables in a model storage database.
- The power user starts the generator tool to create the actual tables in the main database, based on the saved meta data model. In case of minor changes in the meta model, the main database tables are only updated.
- The generator tool automatically generates (or, in case of minor changes, updates) the data object model (entity framework model classes) and updates the data access and manipulating library.
- The generator tool automatically generates input value selection components, Windows and ASP.NET Web forms, components for tracking changes of measured values, and translation resources for multi-language support.
- These generated components are incorporated into the component library and can be attached to the Microsoft .NET project containing code (including all custom-made controls) of the target MIS application.
- The generator tool produces a reporting profile that can be used for report generator by our reporting tool [4]. This re-
Our software has advantages over both the commercial form generators (e.g., Microsoft's Data Form Wizard, Onyak-Tech's H2O) and previously published academic work. In particular, [9] described a way to automatically generate visual components based on a predefined model. Their generation process results in Microsoft Access Application objects that have customizable forms. Similarly, [10] presented a system based on meta-data driven automatic generation of user interfaces and discussed differences between static and dynamic table-based mapping, pointing advantages and disadvantage of both solutions. In static table mapping, one table is generated for each entity defined in the meta-model (in our case, medical records, medical record items, and medical documents). In dynamic table-based mapping, there are few general-purpose and reusable tables with numerous fields, each mapping to attributes from the meta-model (in our case, medical record item elements). Due to the higher speed (important for big MIS), we chose to use static table-based mapping. While our software is based on ideas similar to [9, 10], it offers more flexibility, due to generation of both Windows and Web forms, usage of the data object model, generation of additional components (e.g., for data selection, tracking of measured values), reusability of the generated components in various independent (possibly unforeseen) target applications, compatibility with multiple databases, and multi-language support.

III. THE DATA MODELING TOOL

Our data modeling is based on the electronic health record (EHR), for which we use the term "medical record". One patient can have more than one medical record, while one medical record is by default linked to only one patient. (This is consistent with the organization of the public healthcare system in Serbia and neighboring countries. However, to increase applicability of our data models in other countries, we also enabled that more than one patient can be associated with a single medical record, e.g., for family medical records.) Usually, the main medical record is the one kept by the patient’s general purpose (GP) doctor. It contains full history of any noted illness, prescribed medications, the list of immunizations, the list of allergies, notes about hospitalizations, and other general medical data. Beside the main medical record, some departments (such as dental service, gynecology, and pediatrics) can keep additional medical records related only to their scope.

In our data models, all medical records are represented with classes derived from the base medical record class (BaseMedicalRecord) that contains general properties and is inherited by specific medical records (e.g., GynecologyRecord). A medical record is a collection of data from different medical procedures and each such data item is a medical record item (MRI). The base class for medical record items (BaseMedicalRecordItem) is inherited by specific classes (e.g., CTInfarctusCerebriExamination). Thus, all medical treatments, examinations, analyses, procedures, and processes result in a creation of a medical record item. Both medical records and medical record items can contain medical documents. The base class for medical documents (BaseMedicalDocument) can be inherited by specific documents (e.g., LabAnalysisRequest). These hierarchies are illustrated in a figure posted on our project’s Web page [2].

![Figure 2. EHR meta model structure](image-url)
ing different medical documents associated with medical record items. The 3 mentioned classes inherit the MedicalDataContainer class. Each MedicalDataContainer object (instance) has a list of properties stored as instances of the class MedicalRecordItemElement. Each of medical record item elements has a name, type, description and measurement unit, and is associated a list of boundaries of normal values, stored in instances of the MedicalRecordItemElementRange class. Our modeling approach supports two different types of ranges – regular and ENUM ranges. Regular ranges are defined as ordered pairs of lowest and highest regular value associated with some target group (e.g., infants, children under 6 years, adults). ENUM ranges (not shown in Figure 2) are used to define catalogs of allowed values for related medical record item element. There is also multi-language support (see IV.E) through instances of the EHRTranslation class, each keeping name of the field to be translated and the term in the destination language.

![Figure 3. Data modeling tool – definition of a MedicalRecordItem instance](image)

The data modeling tool is the application used for defining instances of the presented EHR meta model. Figure 3 shows how this tool is used for defining an instance of the MedicalRecordItem class. On our project’s Web page [2] we posted an additional figure that illustrates how the data modeling tool can be used for defining instances on different hierarchical levels (EHRMetaModel, MedicalRecordItem, MedicalRecordItemElement, and MedicalRecordItemElementRange) of the meta data model. Such meta modeling offers the possibility to perform various extensions in a uniform and consistent way. It also enables our generator tool (explained in the next section) to automatically create a separate object model class, database table, and GUI form for each MedicalDataContainer object. Since this generated code is stored in a separate library, it can be reused in different MIS.

An additional feature of the data modeling tool enables creation of an EHR meta model from a specific database. (This is opposite from what the generator tool does.) After a power user specifies Object Linking and Embedding (OLE) connection settings and relevant database table names, our modeling tool examines the database structure and automatically creates or updates the corresponding EHR meta model. The tool can automatically resolve situations when changes are detected only on one side (in the database or in the EHR meta model), but when changes are found on both sides, it presents the list of changes to the power user and ask her/him for guidance in merging the two versions.

IV. THE GENERATOR TOOL

As already mentioned, the data modeling tool prepares input for the generator tool that generates tables in the database and related classes and forms used in the further development process. The purpose of the generator tool is to enable automatic generation of some parts of MIS modules and, thus, speed up the development and customization process. Before the generation of any application module starts, the power user can choose the programming language for the generated .NET project, from the languages supported by the Microsoft’s CodeDom library. The generator tool connects to the database using an OLE DB provider, examines and verifies its structure, and then generates all requested elements. Currently, the generator tool supports only databases supported by the Microsoft Entity Framework but it can be easily adapted for use with any database supporting OLE. In that case, the power user should define parts of its data object model using external tools. Before the table generation process starts, the power user has to enter a few necessary parameters (e.g., OLE connection settings) and to select tables that are related to the base medical record and the base medical document. After this, the power user can start specifying the structure of the new medical records and documents, including the corresponding elements/fields and data ranges. When this definition process is finished, the power user can run the table generation process. This process generates new tables in the database and saves the model in an XML file. Once the data model is mapped into the database, it can be used by a MIS. Generator tool’s customization screen is shown in Figure 4. The following sub-sections explain how the customization is performed.

![Figure 4. Generator tool - customization form](image)

A. Creation of data object models

A data object model is a set of classes generated from database tables and enabling communication with the database. Usually, each class represents one database table and class attributes map table’s columns, while class methods allow execution of basic SQL (Structured Query Language) operations (select/insert/update/delete) over table’s records. In our software, database tables are created from an EHR meta data mod-
el and then the data object model is generated (or updated) from these tables. Power users can define data object model generation options applicable either to all database tables or only to a particular database table. By default, the generated data object model classes have the same names as database tables, but power users can define different names. The result of the data object model generation is a single .NET class library that can be included into .NET projects for target MIS.

Our generator tool supports two different data object model generators. The default is to generate data object models based on the Microsoft Entity Framework (we use the EdmGen2 tool for this). However, the Entity Framework was introduced in .NET version 3.5 and cannot be used in applications using older versions of .NET, which are used in some of the MIS relevant for our work. Thus, we also developed our own data object model generator working in older .NET applications but using the same naming conventions and providing functionalities and façade interface analogous to the Entity Framework. Due to this compatibility, it is possible to easily substitute between data object models generated using our data object model generator and the Entity Framework, based on the used .NET version. On our project’s Web page [2], we have posted a UML (Unified Modeling Language) diagram showing how our generator uses the Factory pattern to create classes that work with Microsoft SQL Server and Oracle databases. This can be easily extended to any other database (e.g., PostgreSQL) supported by Microsoft’s OLE DB drivers or conforming to the ODBC (Open Database Connectivity) interface.

For each database table a selection component can be defined. This is a GUI component that applies filters defined in search fields and when a MIS end user (e.g., doctor, nurse) types some text in these search fields, the selection component automatically displays a drop-down list of filtered results from which the end user can choose one. Figure 5 illustrates how entering the first 3 digits of a medicine code (label “Sifra” in Serbian; this is the search field) produces a drop-down list of corresponding medicine names (label “Naziv” in Serbian). For a selection component for one database table, up to three search fields can be defined. This is shown in the middle of Figure 4, in the section “Selection component”. Selection components are particularly useful for large catalog tables (such are catalogs of medicines, diagnoses, physicians, and similar), but can be defined for any database table. When a selection component is generated as a Windows (end) user component, it can be included into any Windows form. Each generated selection component inherits the GeneralSelectionComponent class containing base features (which can be overridden in the code of derived classes, if needed). One of these base features is minimal length of the text typed in a search field. By default, filtering is started after the third character entered in a search field (because catalogs have many items). Another base feature is the maximum number of items displayed in the drop-down list.

C. Generation of forms

Windows forms are the main GUI elements in our MIS applications, so our generator tool allows power users to (if they wish) automatically generate these forms and save them in a selected destination folder. The generated forms can be reused in various end user application modules. Currently we support generating Windows forms for .NET versions 1.0 to 3.5, using Microsoft’s CodeDom library. We recently made progress on automatic generation of Web and XAML (Extensible Application Markup Language) forms, which will be used in our forthcoming software.

Different forms can be generated based on the data stored in database tables, data object model related classes, as well as multi-language translation resources. However, all generated forms use the same conventions allowing creation and update of underlying data objects (which are managed within the central application component). For example, each element from a medical record item object (and, thus, each field from the corresponding database table) is represented with a label and an appropriate field allowing editing. Different data types correspond to different visual components: shorttext is represented with a regular textbox, description with a multiline textbox, numeric value with a light orange background, image with an image boxe, Boolean value with a checkbox, while collections of ENUM range objects with combo boxes (allowing an end user to choose from a list of offered values). Measurement unit is represented as a label on the right of the editing component, while all ranges are combined in a label placed right of the measurement unit. Figure 6 shows a Windows form generated for the temperature_list entity that represents a set of values from the basic medical examination of a patient: body temperature (label “temperatura”), systolic and diastolic blood pressure (“sistolni pritisak” and “dijastolni pritisak”), and heart beats per minute (“puls”) along with general remarks (“napomena”).

D. Generation of components for tracking measured values

Another type of automatically generated component shows (tracks) how a particular measured value changes over time. It is illustrated in Figure 7, for the same example as the form in Figure 6. This type of component is generated based on the generated forms – each form can have a corresponding measured values tracking component. After a form is generated, a power user can specify which numeric fields (measured values) from the form should be displayed in the corresponding meas-

Figure 5. Selection component for a medicine (“Lek” in Serbian)

Figure 6. Form for entering and editing values on the temperature_list record
E. Building of translation resources

Multi-language support is another feature of our software, necessary for its applicability in different countries. As mentioned in Section III and shown in Figure 2, multi-language support is an integral part of the EHR meta data model, which enables translation on every definition level. Currently, we support two ways of defining translation resources: translation incorporated into a model and translation defined in a separate external file. In the first approach, a power user defines translation values for different languages in the class EHRMetaModel. Since this multi-language support is fixed during design-time, it is not very efficient. Therefore, the second approach, based on external XML or Microsoft Excel XLS files, is more flexible. In this approach, our generator tool creates a translation XML or XLS file that lists all strings (e.g., “{.#Echo.”) unique label identifier) that can be translated and power users then enter translation terms in different languages (e.g., “Eho” in Serbian Cyrillic script, “Echo” in Serbian Latin script, “Echo” in English) into this file. We have posted on our project’s Web page [2] a figure showing an XLS translation file where names of strings properties and translation terms for 3 languages are in different columns and each row corresponds to translation of a single string. If there is no translation term in a particular language, the name of the string property is used by default.

These translation resources are used during form generation. Each form can be defined either as a single language form or as a form supporting labels in multiple languages (one of which is chosen during run-time). For a single language form, labels are fixed during the generation process to terms (in the chosen language) defined within the translation resource. For forms supporting multiple languages at run-time, the generation process represents each label with a unique identifier (in the first column of a translation resource file) and embeds all available translations for these labels into a resource added into the .NET library containing the generated form. At run-time, one language is selected and unique label identifiers are replaced with corresponding terms from this language.

F. Generation and customization of reports

In MIS, there is a frequent need to generate various reports. During the code generation process, our generator tool can define a new reporting profile containing all fields from a single database table for which a Windows form is already generated. After all forms have been generated, the end user will have a list of predefined reporting profiles containing all fields from all tables. Each profile contains a list of medical record item elements that can be grouped to achieve more efficient organization of item elements. Groups are particularly useful when one medical record item contains a many elements (30 and more). At run-time, the reporting profiles are used by our report generation software tool [4], the main parts of which are: the Profile Manager, the Query Editor, the Query Parser, the Data Module and the Result Providing Module.

The main window of the report generation tool is illustrated in Figure 8. On the left side is a tree structure of medical record item elements (groups and item elements that are not members of any group) within the active profile. In the upper part is a list of sub-groups or medical record item elements within the selected medical record item element group. On the right side is a set of buttons for easier query writing. In the central part (with the blue background) is the query editor that enables end users to create query statements by combining medical record item elements (but not groups or sub-groups) and available logical operators (NOT, AND, and OR). The resulting query is closer to spoken language than SQL and, thus, more understandable to medical professionals.
all corresponding visual elements in the generated forms are disabled. Figure 9 shows a window of the access control configuration tool extension. In the upper-left part a user class is selected, while in the lower-left part a form to be configured is selected. The right part shows a list of access privileged to actions and database table fields ("Dozvoli" means "Allow" in Serbian). This possibility of fine-grained definition of end user access control profiles is needed when working with sensitive data, and most medical data are sensitive. Configuration profiles also enable defining groups of related medical record items, as well as end user classes. The defined access control configurations are stored in the main database of the target MIS system, so that they can be applied at run-time. Run-time changes in end user access control profiles are applied without the need to restart the running MIS software.

Figure 9. Access control configuration tool

H. Building applications with the generated components

All generated components are used for building MIS applications, primarily to support doctors’ work. Based on a medical record item in the meta-model, our generator tool creates a database table, a corresponding data object, a Windows form, as well as some other components. The generated form interacts with the database using the data object. Every data object is controlled by the data object model of the central MIS application. In this way, the generated forms only need to implement medical record items, while doctor's license and name are received from the forms, and saves further coding is needed to make them available to the MIS. In several iterations of tool improvement and re-testing. This testing (performed in our university lab by members of our development team) was done for the code of our tools, but also for the code of the generated modules. For several forms we compared in detail the code generated by our tools and the code manually written by programmers. During this testing, we corrected found errors, but also got several insights about possible improvements of our tools. This resulted in several iterations of tool improvement and re-testing.

To verify functional correctness of our data modeling and code generation tools, we conducted both black-box functional testing and white-box structural (control flow and data flow) testing. This testing (performed in our university lab by members of our development team) was done for the code of our tools, but also for the code of the generated modules. For several forms we compared in detail the code generated by our tools and the code manually written by programmers. During this testing, we corrected found errors, but also got several insights about possible improvements of our tools. This resulted in several iterations of tool improvement and re-testing.

To validate our data modeling and code generation tools and to estimate their impact on efficiency of customizing our MIS for different healthcare organizations, we interviewed all current power users of our tools and solicited their impressions and improvement suggestions. They were satisfied with the tools, but provided several suggestions, particularly about increasing their work efficiency. The main measure of this efficiency is the speed of development of a set of various modules (particularly, forms) with our tools compared to manual coding. Since there are many similar forms in our MIS, the power users were able to give a subjective opinion about the increased efficiency. They agreed that the overall development time was significantly shorter (less than 50% of the time needed with manual coding), particularly to get an initial functional version.

Figure 10. An example of a medical data collection application

V. VERIFICATION AND VALIDATION

We evaluated (verified and validated) our data modeling and code generation tools in several ways, discussed in this section. Note that this evaluation is different from the evaluation of the resulting MIS, summarized in our previous publications (e.g., [3]). While there are many end users of our MIS, there are only a few (currently: 6) power users who use the data modeling and code generation tools. They are IT professionals and specialist doctors from the Central Ambulatory Facility in Nis and the Clinic for Neurology of the Clinical Centre Nis.

To verify functional correctness of our data modeling and code generation tools, we conducted both black-box functional testing and white-box structural (control flow and data flow) testing. This testing (performed in our university lab by members of our development team) was done for the code of our tools, but also for the code of the generated modules. For several forms we compared in detail the code generated by our tools and the code manually written by programmers. During this testing, we corrected found errors, but also got several insights about possible improvements of our tools. This resulted in several iterations of tool improvement and re-testing.

To validate our data modeling and code generation tools and to estimate its impact on efficiency of customizing our MIS for different healthcare organizations, we interviewed all current power users of our tools and solicited their impressions and improvement suggestions. They were satisfied with the tools, but provided several suggestions, particularly about increasing their work efficiency. The main measure of this efficiency is the speed of development of a set of various modules (particularly, forms) with our tools compared to manual coding. Since there are many similar forms in our MIS, the power users were able to give a subjective opinion about the increased efficiency. They agreed that the overall development time was significantly shorter (less than 50% of the time needed with manual coding), particularly to get an initial functional version.
of the generated modules (only 10% to 30% of the time needed before). However, since we used only 1 layout template for form generation, there was often a need for some additional layout customization of the generated initial functional version. While this layout customization is somewhat quicker using our tools (80% of the time needed before), the power users noted that the need for customization should be reduced. To achieve this, we are working on developing additional layout templates, as well as software for easier creation of such layout templates. The power users also remarked that it would be good if several people could work on the same model simultaneously. Therefore, we allowed storing the meta-data model in the Model Storage Database, to which a number of modeling tools (used by different power users) can connect simultaneously and access the same models. Another suggestion by the power users was introduction of a translation table, which we addressed with the translation resources explained in Subsection IV.E. Additional power user requests that are still not incorporated into the modeling tool are: enabling the generator tool to support other popular databases in addition to Microsoft SQL Server, enabling merging different models. We are currently working on support for several additional databases (PostgreSQL, Oracle, DB2), while merging models will be done next.

To validate our data modeling and code generation tools and to determine their effects on the work of end users, we also performed interviews with several end users of our MIS. Their comments were positive. For example, they commented that the use of these tools resulted in consistency of various characteristics (e.g., layout, style, size, fonts, and colors) of the generated forms, which helped these end users shorten the time needed to learn how to use our MIS.

VI. CONCLUSIONS

MIS are very complex information systems. An important part of this complexity is the need to create a large number of different, but somewhat similar, forms and database tables for different medical data. The past commercial form generation tools provided at best coarse-grained and semi-automatic implementation of these forms. This can be inefficient and time consuming. A much more powerful approach to implementation of various MIS components (not only forms) related to entry, storage, and reporting of diverse medical data was needed.

In this paper we presented our answer to this need – the data modeling and code generation tools for development of EHR-based MIS. The data modeling tool enables a power user to define meta-data models, from which the generator tool automatically creates database tables, data object model classes, Windows and Web forms for input of medical data, selection components, components for tracking measured values, translation resources, reporting profiles, end user access control configuration profiles, and dispatcher forms for central MIS applications. Over the last several years, we have successfully used early versions of these tools for development of several MIS for public healthcare institutions in Serbia [2, 3, 4, 5, 6]. The recent solutions presented in this paper systematized our software engineering approach to rapid development of MIS and significantly increased the power of our tools. These tools can be very helpful in development of MIS (and other information systems) modules that share similar functionality, but work with different data formatted in different ways. They enable significantly easier, faster, and more efficient development of MIS. Their use also resulted in fewer development errors, more compact code, and uniform end user interfaces (an important feature for medical staff with low computing literacy – a frequent case in developing countries). These tools also facilitate reusability and customization of MIS in several ways. First, they enable that modules developed for one type of MIS can be more easily adapted to a different type of MIS. For example, these tools allowed us to reuse and extend modules developed for past MIS for ambulatory facilities [3] for development of new hospital (clinical) MIS in Serbia. Second, the built-in flexibility and support for use of additional configuration tools facilitate that developed MIS can be easily customized for different work practices in seemingly similar healthcare organizations (e.g., two hospitals in the same healthcare system). Last but not the least, these tools and the resulting MIS require only minor customizations for use in public healthcare systems in countries neighboring Serbia, which have similar organization of public healthcare systems. Due to the inherent adaptability and multi-language support, these tools can also be adjusted for use in other (particularly developing) countries worldwide, but the level of required changes depends on differences between the EHR meta-model built into our system and characteristics of health records used in the country for which our software tools and MIS are to be customized.

REFERENCES