UDC 65.01

# A Method and a Tool for Rapid Prototyping of Large-Scale Business Information Systems

Gordana Milosavljević, Branko Perišić

Faculty of Engineering
University of Novi Sad
Trg Dositeja Obradovića 5,
21000 Novi Sad, Serbia and Montenegro
{grist, perisic}@uns.ns.ac.yu

**Abstract.** This paper presents a method and concepts of a supporting tool for rapid prototyping of large-scale business information systems. Our method is based on the following guidelines: (1) small team of highly skilled members with combined skills, (2) prototype-based development of subsystems and the system as a whole, (3) brainstorming sessions always involving system analysts, database and application designers, and user representatives (if needed), and (4) application generator providing for efficient prototype development by maximum automation of all design phases. The also presented application generator (AppGen) is based on standardization of functional and visual characteristics of an application, a library of high-level, coarse-grained components, and a set of rules for model-to-application mapping enabling automatic application reconfiguration in case of changes in the data model.

## 1. Introduction

Despite the large number of methodologies, standards, and tools, development of large-scale information systems remains a challenging task. The percentage of unsuccessful development projects in terms of exceeding time and/or budget is constantly between 50% and 70%, from the early 80's [1] to the late 90's [2]. Thirty percent of all projects never reaches deployment.

Projects developed using classical software life-cycle models (e.g., the waterfall model) typically fail for the following reasons: output-driven orientation of design process, application backlog due to the document maintenance workload, user dissatisfaction due to inability to "see" the system before it is operational, etc. [3].

Prototype-based methods, intended to correct these shortcomings and to bring a software project closer to its users, have other problems:

- It is hard to identify aspects of the large system to be prototyped and to set boundaries [5]. In order to get an overview of the overall system, a detailed understanding of the different tasks is mandatory, while one cannot understand the different tasks without an overview of the system as a whole (the "abstraction" paradox [6]).
- It is hard to determine the level of fidelity for a prototype capable of capturing significant user feedback while being cheap enough for development. Low-fidelity prototypes have low development cost, but because they are often demonstrated to, rather than exercised by the user, it is more difficult for user to identify design inconsistencies and shortcomings [4]. On the other hand, high-fidelity prototypes make greater end-user acceptance, but usually their development becomes a development effort in itself, sometimes requiring many weeks of programming support [4].

Project teams using the prototype approach, respecting large-scale systems more than it is actually needed and in fear of making errors, repeat the same mistakes as teams using classical methods. Only in this case, users' time and patience are spent on evaluation of different prototypes of low usability. In our opinion, both approaches share the same problems concerning the belief that the software construction (or a high-fidelity prototype construction) is expensive, takes a long time, and requires many developers. Hence, construction phase is delayed as much as possible, while communication with users utilizes "cheaper means". Even if an initial specification obtained this way is good enough, the lack of capability of quick adaptation to changes will raise problems for the project, since business rules are changing during project development and users' requirements grow while gaining experience in using the system.

Since users are "extremely capable of criticizing an existing system but not too good at articulating or anticipating their needs" [5], we argue that the solution is to present them a fully functional prototype as fast as is possible. Such an approach to development does not necessarily need to be expensive and time-consuming provided that the appropriate methods, tools, and team organization are used.

The aim of this paper is to present our method for rapid prototyping, including our standards and tools, as well as experiences gained in practicing it. This method enables a small development team to develop and deploy large-scale information systems in a relatively short period of time. It is based on appropriate team organization, brainstorming techniques, and a simple and highly efficient application generator, AppGen. AppGen is based on our HCI (Human-Computer Interaction) and programming standards, a library of high-level, coarse-grained components, and a set of rules for model-to-application mapping with embedded expert knowledge.

The rest of the paper is structured as follows. Section 2 presents our method for large-scale information system development. Section 3 describes high-level, coarse-grained components used by AppGen and gives a short overview of the HCI standard. Essential features of AppGen are described in Section 4. Section 5 presents our experiences in applying the method and the tool to several real-life projects. Section 6 reviews the related work. Finally, Section 7 concludes the paper.

## 2. The Method for Information Systems Development

Our method for large-scale information system development is characterized by the following four main principles.

1. The development is undertaken by a small team, so there is less effort needed to organize it and for members to communicate. Since the whole team can be present in the same room, or can go to the users' location, all participants immediately have information on how requirements might have changed or, conversely, enables them to question or adapt to changing requirements immediately [10].

2. The development process comprises brainstorming sessions on system or software aspects. The core of these sessions always includes system analysts, database designers and application designers, while domain experts and end-users attend sessions as needed. Domain experts and end users need to attend sessions on requirements elicitation, specification of user interface functionality, system (subsystem) characteristics, and system verification. On the other hand, they do not have to be present in database generation and software construction sessions (see Section 2.1). As these specialists observe the system from different aspects, their united work significantly decreases the number of iterations needed to reach an acceptable prototype. In contrast to the Rational Unified Process (RUP) [18], we do not take the serial approach of completely developing and accepting one type of a model before moving on to the next one, but develop multiple types of models[1] in parallel allowing rapid switching of modeling sessions. This way of conducting the development process assumes that each expert is acquainted with the areas of expertise of others, while all of them must have the knowledge of the modeling tools and the standards of the application being built.

3. In order to gain team credibility and significant user feedback it is necessary to show "something that works" as soon as possible. The system as a whole is not suitable for prototype-based development because of its initial vagueness and inherent complexity. Therefore, the first task is to decompose the system to a set of well-defined subsystems, each corresponding to a group of related jobs, and to create a plan of their imple-

[1] UML (Unified Modeling Languange) models, physical database models, user interface models, etc.

mentation, integration, and deployment. Each subsystem can then be implemented by an evolutive prototype method. Really rapid and effective system decomposition requires, besides cooperative users, an analyst familiar with the application domain.

4. Development team translates elicited requirements for subsystems into UML diagrams, using an appropriate modeling tool. Class diagrams are used as a basis for generating the physical data model and the database itself (a feature present in most CASE tools). The application generator uses the CASE tool's repository containing information on class diagrams and the physical model to generate a fully functional application prototype. The generating process is based on a set of rules and application templates supplied with library components. The application appearance is based on our HCI standard developed to provide easy learning, robustness, and comfortable environment for users.

### 2.1. Brainstorming Sessions

Information system development takes place in brainstorming sessions starting from system identification and decomposition phase, to verification and deployment phase. A single brainstorming session usually lasts several hours. A single development phase may take from several to several dozen sessions.

The system identification and decomposition sessions are lead by application domain experts (users), end-users and system analysts. Other participants take part in terms of understanding the system structure, application domain and system constraints. The primary source of "information leaking" due to indirect access to information about user requirements by means of various specification documents, is avoided in this approach. The aim is to constrain this phase to last not more than a month.

After the system as a whole is decomposed into subsystems, each subsystem is developed with an evolutive prototype method. Activity diagram in Figure 1 presents development phases for a single subsystem.

The goal of the initial phase of subsystem development, lead by a system analyst and end users, is identifying the requirements. The next phase, involving design and construction of a prototype, is lead by application and database designers. Prototype evaluation and refinement is the goal of the third phase, with all necessary team members and relevant users taking part in the process.

As soon as the initial (requirements specification) session is finished, it is *immediately* followed by a prototype design session where class diagrams (and other types of diagrams, as needed) are formulated. The core of the class diagram is designed by the system analyst and is based on entities of the given subsystem and documents obtained from users. Application designer closely follows the development of the model[2] and can suggest adding supplemental attributes, methods, or classes needed for construction. Database designer, involved in translating the class diagrams to the physical model, can suggest alternative ways of associating particular classes and/or modeling some situations in order to ensure adequate performance of database operations. Shortcomings in user requirements are usually detected very quickly during this session. These shortcomings can be corrected in the next requirements session taking place the next day (or even on the same day, if the work is conducted at the users' site). This way, only a few days are needed to formulate a quality subsystem model acting as a basis for prototype construction.

[2] A model of a subsystem is actually a submodel, ie. a package in the model of the whole project.

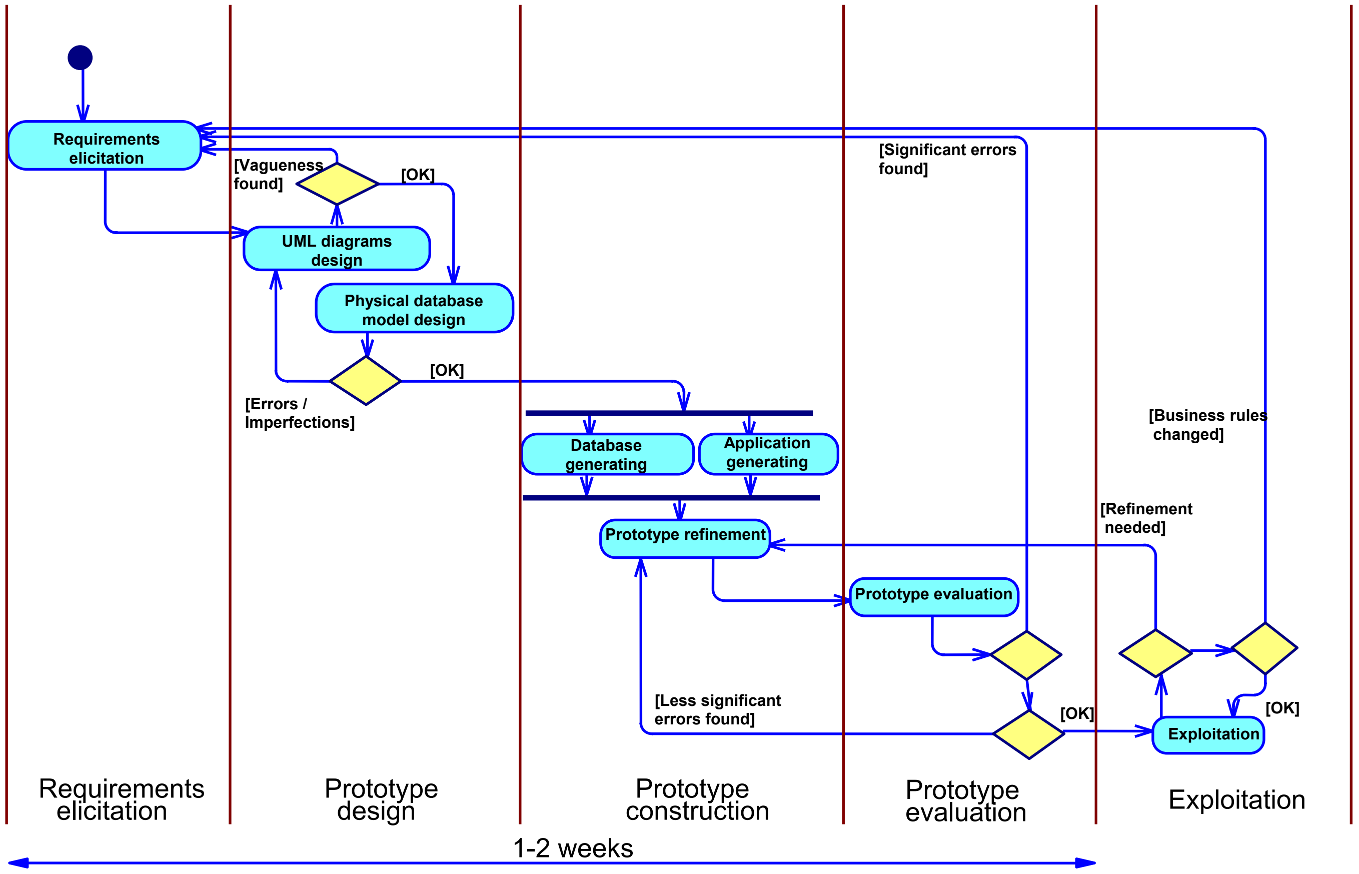


**Figure 1.** Development phases of a single subsystem.

The prototype construction phase is used by a database designer making further customizations, defining constraints, implementing rules, managing indexes, and performing feasibility analyses for data processing operations about to be implemented. A database designer can initiate a new cycle of subsystem model refinement, returning the model to the previous phase for further improvements. When all team members agree on the model, the database schema and the working prototype are generated (see Section 4 for the
description of the application generating process). At this point, the prototype has a fully functional user interface conforming to our HCI standard. However,
instead of presenting the prototype to users at this point, we choose to spend one or two extra days on construction of complex data manipulation procedures (by the database designer), customization of forms layout, and construction of the most important reports (by the application designer). In our experience, the time spent on initial prototype refinement pays back in later development phases, because quality user feedback is gained in a far less number of sessions.

The prototype evaluation phase begins with a prototype presentation. Users then perform the first interactions with the subsystem, assisted by team members, who answer the questions and gather new suggestions. Users should use their real documents during this test, so that data manipulation procedures and reports can be evaluated as well. In the case that significant errors are detected, the process is returned to the requirements elicitation phase. Less significant errors and imperfections can be corrected on the spot allowing the evaluation to continue. The correction of "less significant errors" consists of activities that do not require model changes and can be performed fast enough, without significant user engagement (label and form layout customization, etc.). This way, a user is directly involved in the creation of his or her future working environment, resulting in reduced users' resistance to system deployment.

When users and the development team agree on prototype functionality, the team undertakes burn-in tests (on system stability and scalability), and integrates the subsystem with the rest of the system. Users can then proceed to verify the subsystem on their own entering the subsystem exploitation phase.

The final phase includes possible refinements as users gain more experience in using the subsystem. The subsystem then moves on to its stable phase that lasts until the business environment or rules are changed.

In optimal conditions (users motivated for cooperation, the work taking place at the users' site), an average subsystem (30-50 tables, 10-15 complex data manipulation procedures, 10-20 reports) takes at most two weeks to be deployed.

The description given above assumes the subsystem is completely new to the project team. In case a similar subsystem exists in previous projects, it is integrated in the current project (using the CASE modeling tool and our construction tool) and then evaluated and reconfigured until it satisfies user requirements. This way a subsystem can be deployed in a matter of days.

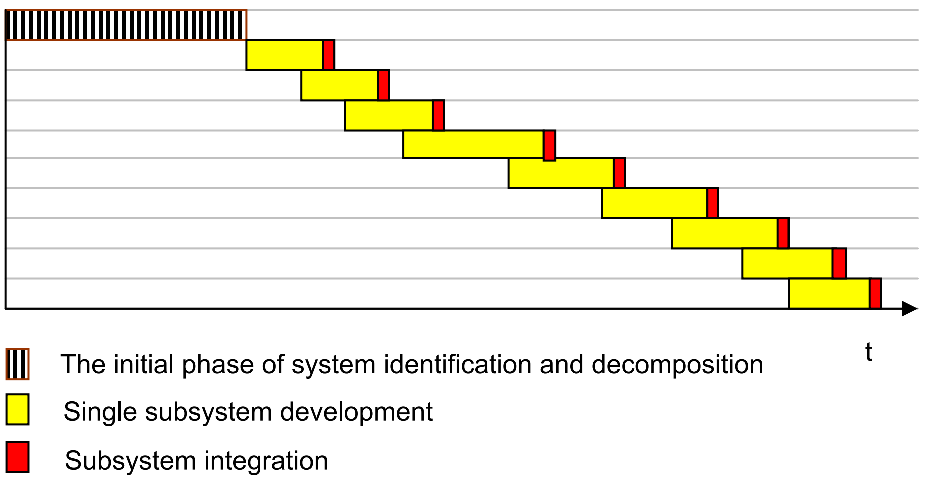


**Figure 2.** An information system development "pipeline"

Final phases of subsystem development (users' independent subsystem verification, fine tuning) witnesses reduced team activity, so the team may start initial phases of the next subsystem (see Figure 2). This way a period of team idleness is avoided and the process reaches a "pipeline", thus futher accelerating development. The pipeline may also facilitate development progress in cases when users are not ready for an intensive subsystem verification. By moving on to the next subsystem, development blocking is avoided.

In order to enable the incremental and iterative development, it is necessary to possess a strategy of integrating new subsystems with the ones already deployed. The strategy includes the following:

– Careful planning of a sequence of developing subsystems, based on inter-subsystem dependencies, and
– The use of tools supporting iterative and incremental development.

The CASE tool used for system modeling must support incremental development and generating partial scripts for modifying database schema. The application generator must support repetitive code generation while preserving manual code customizations. The development team must follow programming standards providing an efficient integration of a new subsystem into the rest of the project. The use of a version control software facilitates system integration and keeping different versions of the project source code.

## 3. Application Structure

The concept of model-based automated code generating is not new [13]. Usually, tool design aims to reach the largest possible amount of generated code. On the other hand, we strive to build a fully functional prototype with as little generated code as possible. A large body of program code complicates management and reconfiguration. The strategy of reducing the program code size is carried out at two different levels.

1. At the project level, the required functionality should be implemented with keeping the number of coarse-grained components as small as possible. The concept of a *generic application* is introduced.
2. At the level of coarse-grained components, the concept of a generic component is introduced.

A generic application is an application comprising a union of all coarse-grained components. Information on these components is stored in an application repository. The basic coarse-grained components (application building elements) are forms, reports, and data manipulation procedures (DMPs). Higher-level building elements are subsystems corresponding to particular tasks in an enterprise (see Figure 3). Subsystems are created using the basic building elements by an application administrator. This way, a single basic element can appear in multiple subsystems. Each subsystem appears as a pull-down menu in the generic application's main menu.

Subsystems represent elements for building *user roles*. A user role bears information about self-contained software unit that models a group of tasks associated to a specific job or user. Each role behaves as an independent application that has the appearance and functionality of the generic one, with the main menu reduced to the necessary and sufficient group of items.

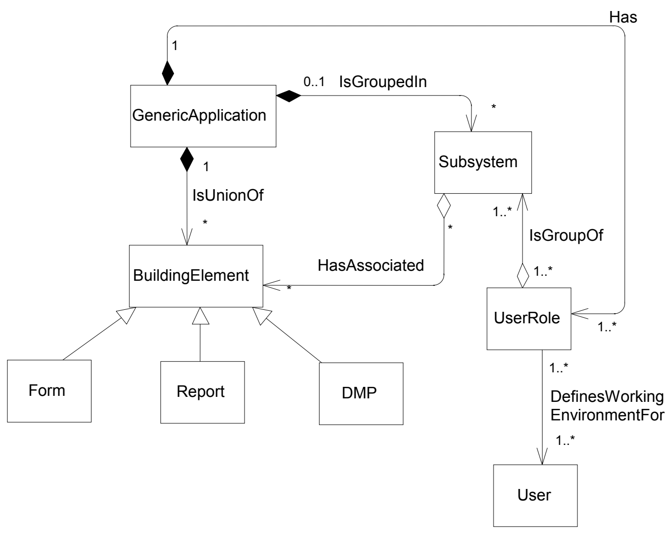


**Figure 3.** Generic application structure

The rest of the section discusses features of the basic building elements.

## 3.1. Forms

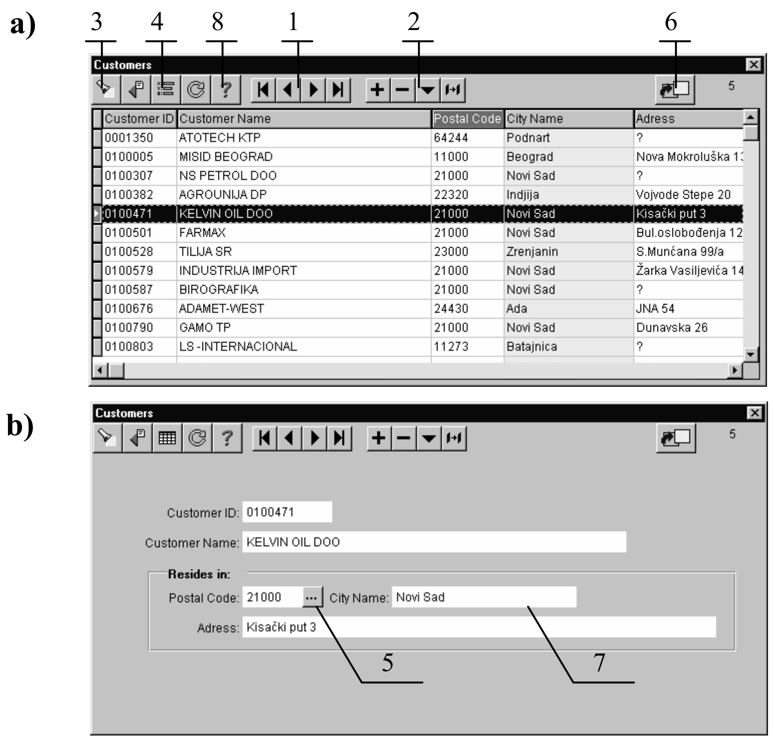
We have distinguished several standardized types of forms within business applications as follows:

- Simple form – providing basic operations (browse, add, update, copy, delete, query-by-form) on the rows of one database table (every table has exactly one associated simple form).
- Complex or “Many-to-Many” form – intended for intensive row inserting into database tables with the primary key composed of two or more primary keys from referenced tables (see Figure 10).
- Report parameterization form – entering data-filtering parameters for the report.
- Form displaying the set of available reports within a subsystem, where the user can invoke a report.
- The main user role form, comprising the main menu with options conforming to current user rights.

Data manipulation forms (both simple and “Many to Many”) are descendants of the *generic form* that provides (see Figure 4):

- Navigation through current set of rows.
- Row operations (if permitted): add, update, delete, and copy.

- Query by form.
- Possibility to change a display mode (row browse or single record view).
- Zoom (lookup) buttons (see Figure 8).
- “Next form” functionality. “Next form” function provides access to simple forms associated to child tables of the current one (see Figure 9).
- Calculated, aggregated and lookup values, when appropriate.
- Online help.



| Customer ID | Customer Name | Postal Code | City Name | Adress |
|---|---|---|---|---|
| 0001350 | ATOTECH KTP | 64244 | Podnart | ? |
| 0100005 | MISID BEOGRAD | 11000 | Beograd | Nova Mokroluška 1 |
| 0100307 | NS PETROL DOO | 21000 | Novi Sad | ? |
| 0100382 | AGROUNIJA DP | 22320 | Indjija | Vojvode Stepe 20 |
| 0100471 | KELVIN OIL DOO | 21000 | Novi Sad | Kisački put 3 |
| 0100501 | FARMAX | 21000 | Novi Sad | Bul.oslobođenja 12 |
| 0100528 | TILIJA SR | 23000 | Zrenjanin | S.Munćana 99/a |
| 0100579 | INDUSTRIJA IMPORT | 21000 | Novi Sad | Žarka Vasiljevića 14 |
| 0100587 | BIROGRAFIKA | 21000 | Novi Sad | ? |
| 0100676 | ADAMET-WEST | 24430 | Ada | JNA 54 |
| 0100790 | GAMO TP | 21000 | Novi Sad | Dunavska 26 |
| 0100803 | LS -INTERNACIONAL | 11273 | Batajnica | ? |

Customer ID: 0100471
Customer Name: KELVIN OIL DOO
Resides in:
Postal Code: 21000 City Name: Novi Sad
Adress: Kisački put 3

**Figure 4.** Simple form a) Browse view b) Single record view

A *generic form* is represented by a generic superclass encapsulating the full functionality defined by the HCI standard. Concrete data manipulation forms in a project are descendants of this superclass. A concrete form can extend or redefine the behavior of its ancestor. Conversely, a change in behavior of all forms can be made by changing the implementation of the superclass. In order to reduce the number of descendants defined, the generic form is supplied with the possibility of dynamic adjustment of its appearance and the presented set of data according to current application context, data from the application repository, access rights of the current user, and a set of parameters. In most cases, a single database table has a single corresponding concrete form that can be invoked within different subsystems.

In the prototype evaluation mode, the form loads initialization parameters from the application repository (see Figure 5) and adjusts its appearance and behavior on-the-fly. This way it is possible to tune form appearance and behavior without changing and compilation of the program code.

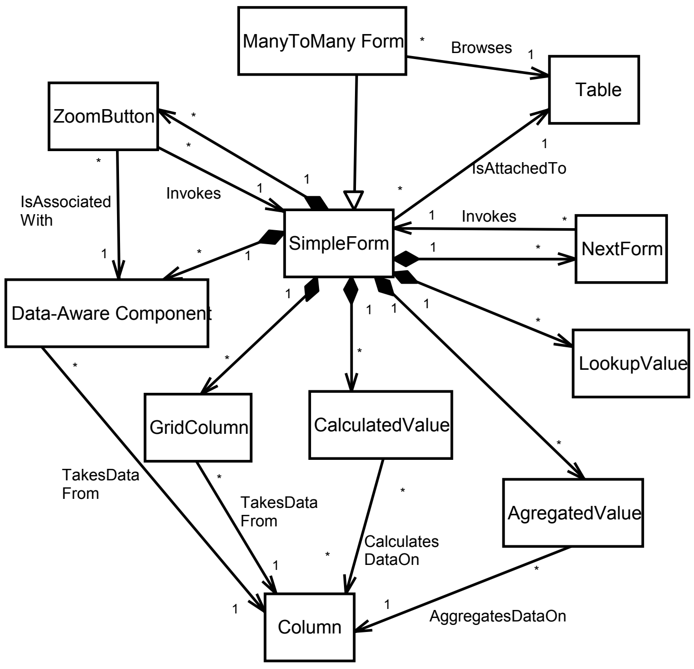


**Figure 5**. The segment of the AppGen repository for specifying forms

## 3.2. Data Manipulation Procedures

The notion of a data manipulation procedure (DMP) stands for the programmatic procedure operating on the database data. There are two types of DMPs in our model:

- Basic DMPs, performing elementary operations on a single database table (insertion, update, and removal of a single row).
- Complex DMPs, implementing whole or part of business transactions (stock entry shipping, payroll calculation, etc.), that generally may modify contents of multiple database tables.

DMPs are executed within a transaction control mechanism provided by the application server or database server and may be implemented in different ways. There are two versions of the application generator tool; the former uses Enterprise JavaBeans [21] while the latter uses Transact-SQL stored procedures [22]. Each standard data manipulation form invokes three basic DMPs and, if needed, particular complex DMPs (see Figure 6).

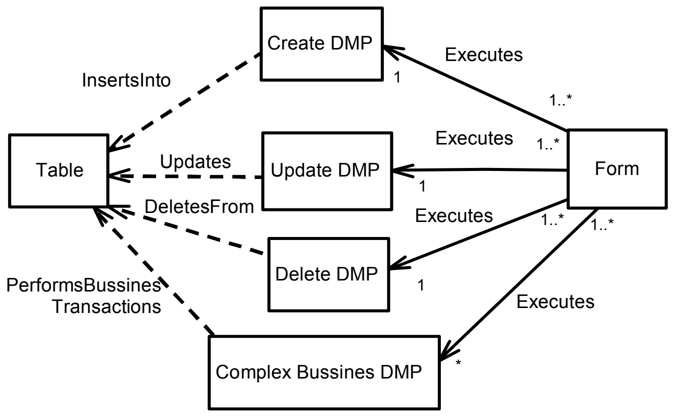


**Figure 6.** Standard data manipulation form structure

Removal of DMPs from client applications achieves the following benefits:

– Reduced client application complexity and increased development efficiency (higher-level languages used for developing DMPs increase productivity up to ten times compared to general purpose languages [15, 17]).
– Faster execution (DMPs are precompiled and optimized before they are used, making them faster than other ways of executing operations in the database server [15, 16]).

## 3.3. Reports

We have distinguished two types of reports in the generic application as follows:

– Hand-coded reports implemented within the development tool used, intended for efficient printing on dot-matrix printers.
– Graphical reports implemented using a report generating tool (i.e., Seagate Crystal Reports, Progress Report Builder etc.), intended for printing on laser printers.

Each report, independently of its type, comprises the following:

– Parameterization dialog.
– Destination choice dialog (screen, printer, and file).
– Reports are grouped within subsystems and can be invoked using the appropriate standard form.

## 4. Application Generator Tool

The application generator tool comprises several elements presented in Figure 7.

**Model Analyzer** imports models from the CASE modeling tool's repository into the application generator repository. The import process also includes generation of concrete form specifications according to given rules. The rules contain expert knowledge about the HCI standard and the model-to-application mapping. This process analyzes the type and cardinality of associations, the structure of primary and foreign keys, special comments embedded in code, etc. The details of the process are presented in more details in [19, 20, 21].

During this process, all necessary data is imported from the CASE tool repository into "entrance" tables of the application generator repository. Further implementation of the given subsystem is carried out depending on data from these tables, while designers can independently work on the other subsystems.

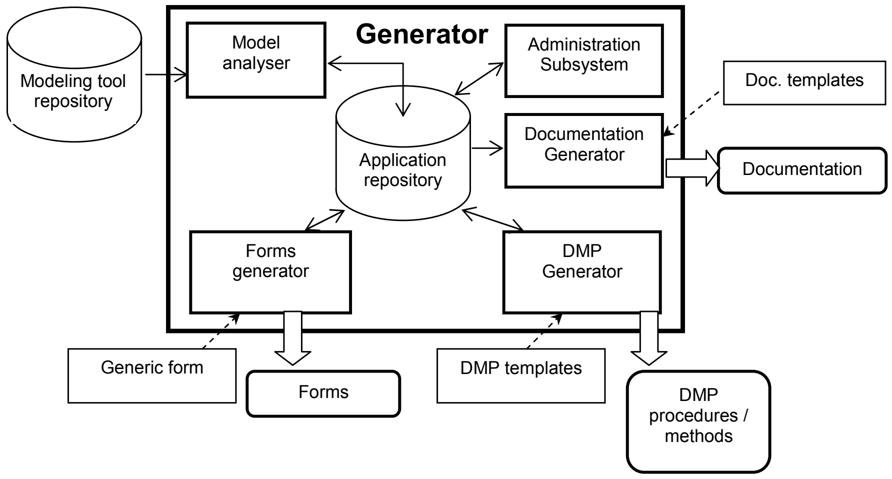
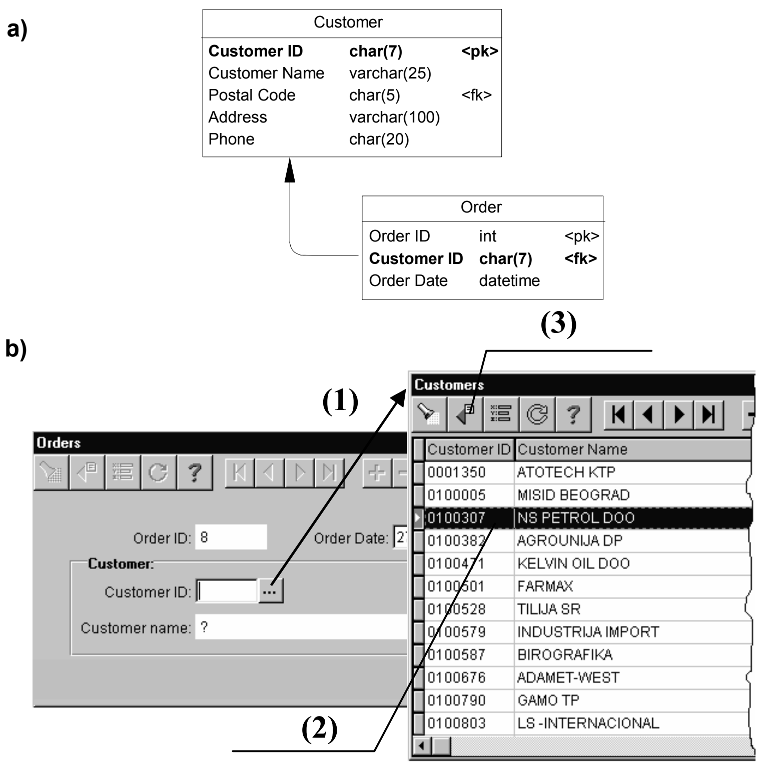


**Figure 7**. Structure of the application generator AppGen

Data in "entrance" tables and a set of rules (see Table 1 and Figures 8, 9, and 10 for an example) are used by AppGen to automatically generate an initial specification of standard forms. This initial set is stored in the AppGen repository and can be reviewed by a Forms Generator visual tool (see Figure 12). Changing the initial specification usually includes addition of calculated, aggregated, or lookup fields, correcting spelling errors in field labels, removal of columns not intended for display, etc.

**Table 1**. Set of rules for model-to-application mapping

| From model | To application |
|---|---|
| Diagram | Subsystem |
| Table | Simple form |
| Table name | Form caption, main menu item caption |
| Table code name | Associated form name ("frm" + table code name), basic DMP names ("c_", "d_", "u_" + table code name) |
| Column | Data-entry component, grid column |
| Column name | Data-entry component / grid column label |
| Column code name | Associated component/ column name, DMP parameter and variable names |
| Column comment | Component /column help |
| Column type and format | Component/column type and format |
| Relations | Zoom buttons, Next menus, Lookup values |
| Relations names | Next menu captions |
| Constraints | Component/column constraints, custom error messages |



| Customer | | |
|---|---|---|
| **Customer ID** | **char(7)** | **<pk>** |
| Customer Name | varchar(25) | |
| Postal Code | char(5) | <fk> |
| Address | varchar(100) | |
| Phone | char(20) | |

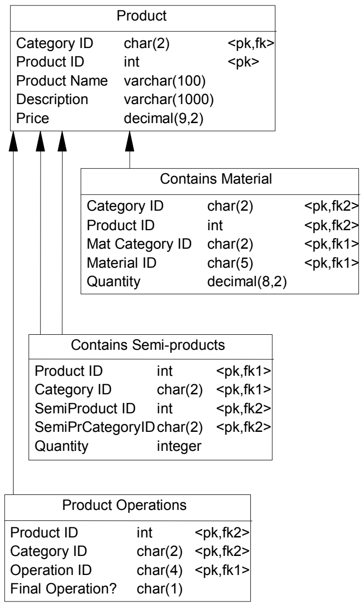
| Order | | |
|---|---|---|
| Order ID | int | <pk> |
| **Customer ID** | **char(7)** | **<fk>** |
| Order Date | datetime | |

Order ID: 8
Order Date: 2
Customer:
Customer ID:
Customer name: ?

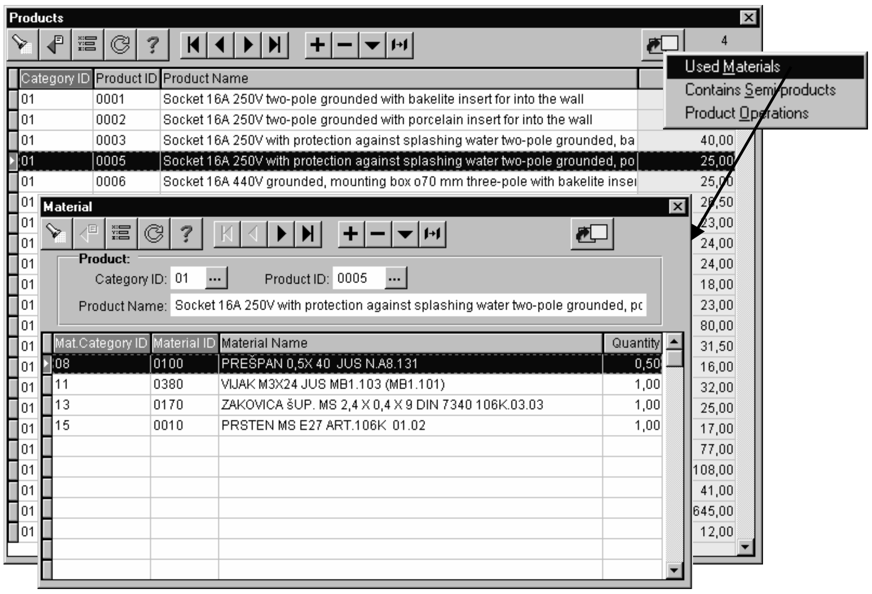
| Customer ID | Customer Name |
|---|---|
| 0001350 | ATOTECH KTP |
| 0100005 | MISID BEOGRAD |
| 0100307 | NS PETROL DOO |
| 0100382 | AGROUNIJA DP |
| 0100471 | KELVIN OIL DOO |
| 0100501 | FARMAX |
| 0100528 | TILIJA SR |
| 0100579 | INDUSTRIJA IMPORT |
| 0100587 | BIROGRAFIKA |
| 0100676 | ADAMET-WEST |
| 0100790 | GAMO TP |
| 0100803 | LS -INTERNACIONAL |

**Figure 8**. An illustration of zoom buttons. a) Physical model segment. b) Corresponding application segment

a)


Product
Category ID char(2) <pk,fk>
Product ID int <pk>
Product Name varchar(100)
Description varchar(1000)
Price decimal(9,2)
Contains Material
Category ID char(2) <pk,fk2>
Product ID int <pk,fk2>
Mat Category ID char(2) <pk,fk1>
Material ID char(5) <pk,fk1>
Quantity decimal(8,2)
Contains Semi-products
Product ID int <pk,fk1>
Category ID char(2) <pk,fk1>
SemiProduct ID int <pk,fk2>
SemiPrCategoryID char(2) <pk,fk2>
Quantity integer
Product Operations
Product ID int <pk,fk2>
Category ID char(2) <pk,fk2>
Operation ID char(4) <pk,fk1>
Final Operation? char(1)

b)



| Category ID | Product ID | Product Name | |
|---|---|---|---|
| 01 | 0001 | Socket 16A 250V two-pole grounded with bakelite insert for into the wall | |
| 01 | 0002 | Socket 16A 250V two-pole grounded with porcelain insert for into the wall | |
| 01 | 0003 | Socket 16A 250V with protection against splashing water two-pole grounded, ba | 40,00 |
| 01 | 0005 | Socket 16A 250V with protection against splashing water two-pole grounded, po | 25,00 |
| 01 | 0006 | Socket 16A 440V grounded, mounting box o70 mm three-pole with bakelite inse | 25,00 |
| 01 | | | 26,50 |
| 01 | | | 23,00 |
| 01 | | | 24,00 |
| 01 | | | 24,00 |
| 01 | | | 18,00 |
| 01 | | | 23,00 |
| 01 | | | 80,00 |
| 01 | | | 31,50 |
| 01 | | | 16,00 |
| 01 | | | 32,00 |
| 01 | | | 25,00 |
| 01 | | | 17,00 |
| 01 | | | 77,00 |
| 01 | | | 108,00 |
| 01 | | | 41,00 |
| 01 | | | 645,00 |
| 01 | | | 12,00 |

**Product:**

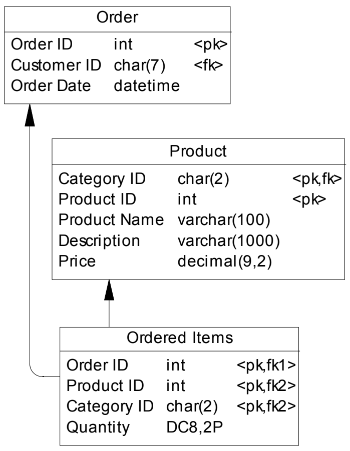
Category ID: 01 Product ID: 0005

Product Name: Socket 16A 250V with protection against splashing water two-pole grounded, pc

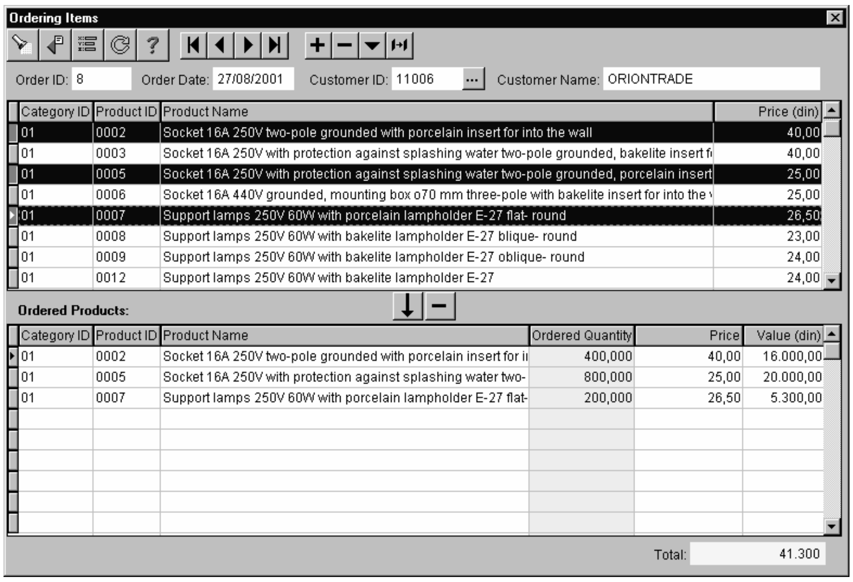
| Mat.Category ID | Material ID | Material Name | Quantity |
|---|---|---|---|
| 08 | 0100 | PREŠPAN 0,5X 40 JUS N.A8.131 | 0,50 |
| 11 | 0380 | VIJAK M3X24 JUS MB1.103 (MB1.101) | 1,00 |
| 13 | 0170 | ZAKOVICA ŠUP. MS 2,4 X 0,4 X 9 DIN 7340 106K.03.03 | 1,00 |
| 15 | 0010 | PRSTEN MS E27 ART.106K 01.02 | 1,00 |

**Figure 9.** An illustration of next forms. a) Physical model segment. b) Corresponding application segment

a)



b)



Order ID: 8 Order Date: 27/08/2001 Customer ID: 11006 Customer Name: ORIONTRADE

| Category ID | Product ID | Product Name | Price (din) |
|---|---|---|---|
| 01 | 0002 | Socket 16A 250V two-pole grounded with porcelain insert for into the wall | 40,00 |
| 01 | 0003 | Socket 16A 250V with protection against splashing water two-pole grounded, bakelite insert f | 40,00 |
| 01 | 0005 | Socket 16A 250V with protection against splashing water two-pole grounded, porcelain insert | 25,00 |
| 01 | 0006 | Socket 16A 440V grounded, mounting box o70 mm three-pole with bakelite insert for into the | 25,00 |
| 01 | 0007 | Support lamps 250V 60W with porcelain lampholder E-27 flat- round | 26,50 |
| 01 | 0008 | Support lamps 250V 60W with bakelite lampholder E-27 blique- round | 23,00 |
| 01 | 0009 | Support lamps 250V 60W with bakelite lampholder E-27 oblique- round | 24,00 |
| 01 | 0012 | Support lamps 250V 60W with bakelite lampholder E-27 | 24,00 |

Ordered Products:

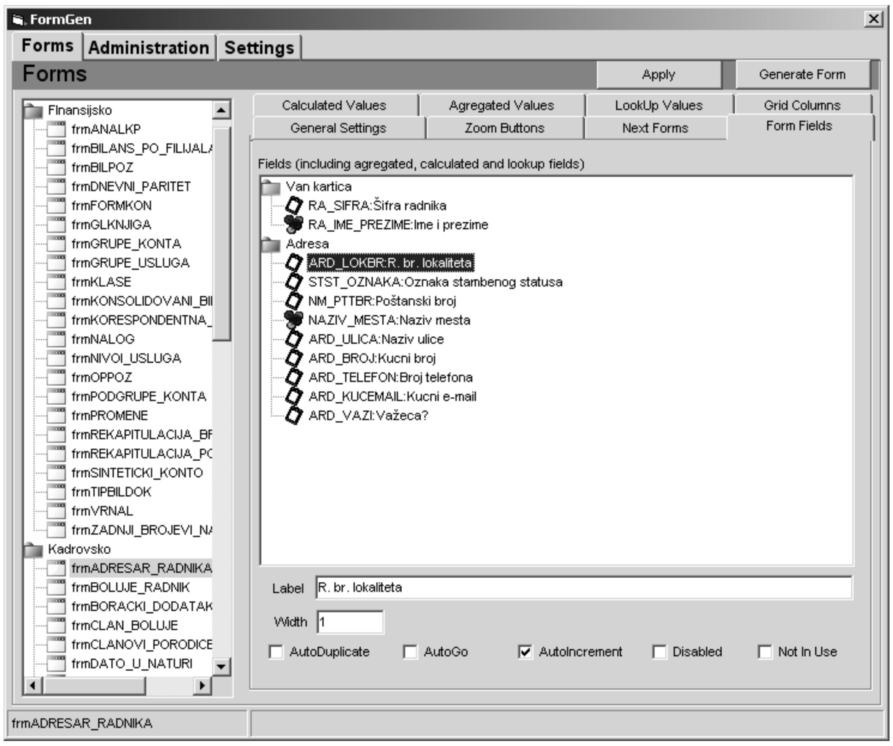
| Category ID | Product ID | Product Name | Ordered Quantity | Price | Value (din) |
|---|---|---|---|---|---|
| 01 | 0002 | Socket 16A 250V two-pole grounded with porcelain insert for i | 400,000 | 40,00 | 16.000,00 |
| 01 | 0005 | Socket 16A 250V with protection against splashing water two- | 800,000 | 25,00 | 20.000,00 |
| 01 | 0007 | Support lamps 250V 60W with porcelain lampholder E-27 flat- | 200,000 | 26,50 | 5.300,00 |

Total: 41.300

**Figure 10.** An illustration of a "Many to Many" form. a) Physical model segment b) Corresponding application segment

**Forms Generator** implements a user interface for previewing and editing the application specification produced by Model Analyzer. This element implements the code generating process as well. Any of Model Analyzer's "design decisions" can be overridden, with the changes being stored in the application repository thus enabling repetitive form generating. Forms Generator provides for rapid user interface development because of the use of high-level components and a set of functions minimizing the need for manual form customization. However, if a need to customize a generated form arise (in terms of changing the form layout or extending/overriding functionality), it can be done with a general-purpose development tool for the chosen platform. By parsing the program code, Forms Generator gathers information on manual changes in the code and stores it in the repository, making it available for the next generator iteration. Hence, each generated form is completely described in the repository, enabling both application recovery after changes to the database schema and documenting the application.

Changes in the model require reconfiguration of the generated forms. The first, Model Analyzer applies changes to the application specification (adds new forms, or adds and removes form fields). Then, Forms Generator forwards these changes to generated forms and presents a list of changes made.



FInansijsko
- frmANALKP
- frmBILANS_PO_FILIJAL
- frmBILPOZ
- frmDNEVNI_PARITET
- frmFORMKON
- frmGLKNJIGA
- frmGRUPE_KONTA
- frmGRUPE_USLUGA
- frmKLASE
- frmKONSOLIDOVANI_BI
- frmKORESPONDENTNA_
- frmNALOG
- frmNIVOI_USLUGA
- frmOPPOZ
- frmPODGRUPE_KONTA
- frmPROMENE
- frmREKAPITULACIJA_BF
- frmREKAPITULACIJA_PC
- frmSINTETICKI_KONTO
- frmTIPBILDOK
- frmVRNAL
- frmZADNJI_BROJEVI_NA

Kadrovsko
- frmADRESAR_RADNIKA
- frmBOLUJE_RADNIK
- frmBORACKI_DODATAK
- frmCLAN_BOLUJE
- frmCLANOVI_PORODICE
- frmDATO_U_NATURI

Fields (including agregated, calculated and lookup fields)

Van kartica
- RA_SIFRA:Šifra radnika
- RA_IME_PREZIME:Ime i prezime

Adresa
- ARD_LOKBR:R. br. lokaliteta
- STST_OZNAKA:Oznaka stambenog statusa
- NM_PTTBR:Poštanski broj
- NAZIV_MESTA:Naziv mesta
- ARD_ULICA:Naziv ulice
- ARD_BROJ:Kucni broj
- ARD_TELEFON:Broj telefona
- ARD_KUCEMAIL:Kucni e-mail
- ARD_VAZI:Važeca?

Label: R. br. lokaliteta

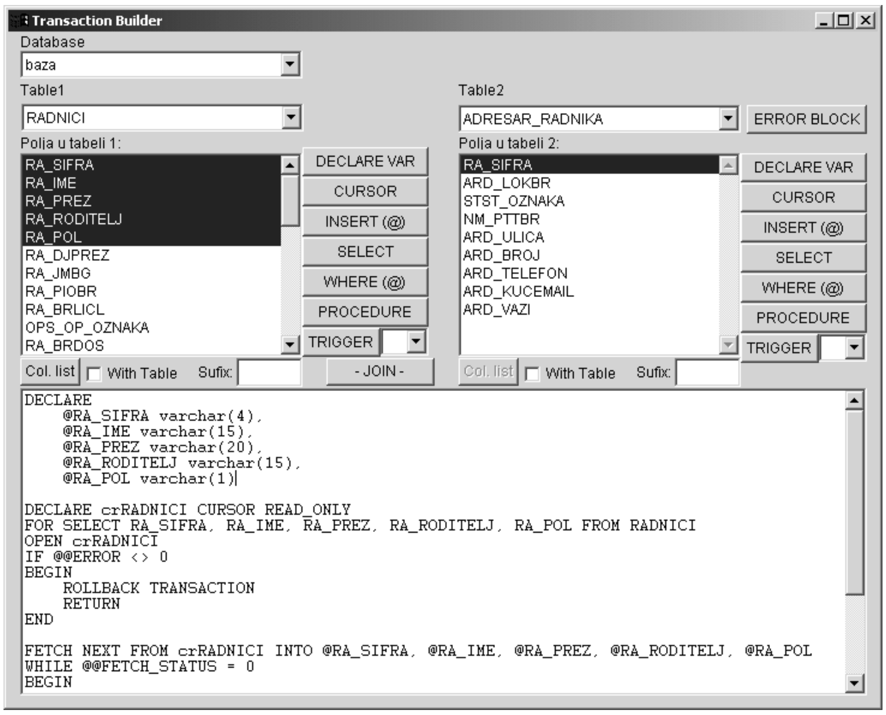
Width: 1

☐ AutoDuplicate ☐ AutoGo ☑ AutoIncrement ☐ Disabled ☐ Not In Use

frmADRESAR_RADNIKA

**Figure 11.** Forms Generator

**DMP Generator** is a tool for efficient construction of complex data manipulation procedures. DMP Generator comprises a set of templates for the most common operations in business transactions and a user interface enabling a database designer to apply these templates to selected tables and columns (see Figure 12). A skilled database designer, well acquainted with the database structure and the business logic, is able to implement all DMPs of a subsystem in a day.



```
DECLARE
    @RA_SIFRA varchar(4),
    @RA_IME varchar(15),
    @RA_PREZ varchar(20),
    @RA_RODITELJ varchar(15),
    @RA_POL varchar(1)

DECLARE crRADNICI CURSOR READ_ONLY
FOR SELECT RA_SIFRA, RA_IME, RA_PREZ, RA_RODITELJ, RA_POL FROM RADNICI
OPEN crRADNICI
IF @@ERROR <> 0
BEGIN
    ROLLBACK TRANSACTION
    RETURN
END

FETCH NEXT FROM crRADNICI INTO @RA_SIFRA, @RA_IME, @RA_PREZ, @RA_RODITELJ, @RA_POL
WHILE @@FETCH_STATUS = 0
BEGIN
```

**Figure 12.** DMP Generator

**Administration Subsystem.** After forms, reports, and DMPs are finalized, the application administrator can integrate them in a subsystem as its basic building elements. The AppGen application administration subsystem serves this purpose. The integration consists of defining the subsystem's vertical menu layout, associating menu items to application building elements, and defining user rights on these elements (see Figure 13).

Upon finalizing subsystem definitions, the user roles are formed and associated to application users. User rights on application elements are defined for user roles, with the possibility of expressing restrictions for particular users. The main application menu is dynamically created depending on the structure of subsystems associated with the current user during application startup. Hence, each application user can have a customized version of the application, while maintenance is carried out centrally on the generic application.

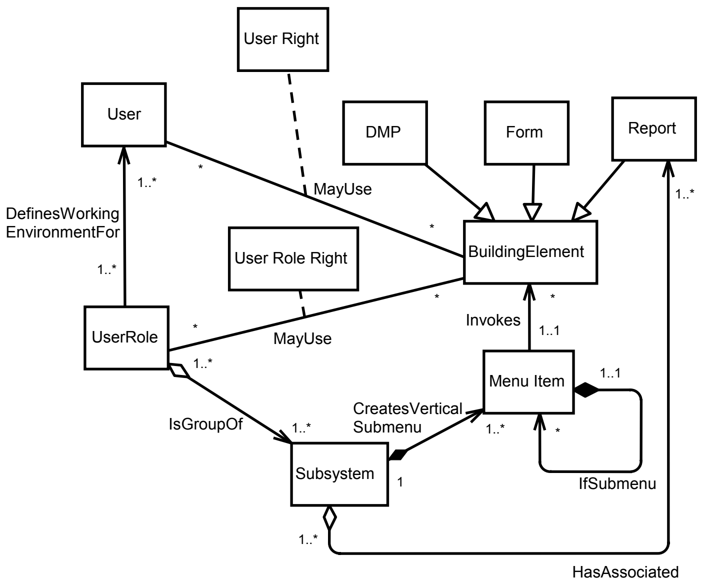


**Figure 13.** The AppGen application administration repository segment

**Documentation Generator** uses the data from the repository to generate documentation in HTML and RTF formats.

## 5. Experiences

This section describes experiences gained by applying our approach and the tool, from 1997 until today. It presents a brief overview of realized projects, as well as analysis of percentage of generated code, tool performance, and end users' attitude.

### 5.1. Realized Projects

The development approach and the tool presented here have been verified by intensive application on several real-life engineering/reengineering projects. Those projects had a number of tables in the database ranging from 320 to 430, and the number of subsystems ranging from 11 to 14, respectively.

The first project, for a large trade company, took seven months from the initial decomposition to the deployment of the last subsystem. The team comprised one system analyst, two application designers, and one database designer. The users were very cooperative, and the team was able to work at the users' location.

The second project, for a trade company with a different business profile, was deployed in two months. This project was able to reuse about 50% of the results of the previous project. The users and their management were extremely cooperative, since the old system had a problem with Y2K compliance. The team comprised one analyst, two application designers and one database designer.

The third project, for a production-based company, took more than a year. Users were satisfied with the old system, and unmotivated for cooperation and verifying the prototype; the development of the new system was forced by the company management because the old one was developed using an obsolete platform. The team comprised one analyst, two database designers, and two application designers.

### 5.2. Generated Code Percentage

The performance of our tool cannot be expressed with commonly used measures, because it is not a commonly used application generator. Instead, it uses concepts of generic forms, generic applications, and application specifications in the repository. Our estimate is that the tool would, in the case of classical code generating, provide as much as 90 to 95 percent of program code, depending on the application.

### 5.3. Tool Performance

Tool performance from the standpoint of time savings can be expressed with the results of the following experiment. Three specialists for the chosen platform were given the task of building a small subsystem with 8 forms, according to our HCI standard. The first specialist started from scratch, the second one used generic classes as a basis for development, and the third one used our tool.

The first expert needed 3.2 hours on average to build a form that conforms to the standard (query by form, navigation, basic data manipulation in a multi-user, transactional environment, printing).

The second expert had a task of inheriting the generic form superclass, defining the form layout. This task took 20 minutes per form on average.

The third expert that used our tool needed 4 minutes to analyze the suggested form specification and start the automatic form generating.

If it is estimated according to the results of this experiment, and assuming the linear life cycle model, to complete the project comprising 500 forms, it would take about 200 working days for the first expert, 20.8 days for the second expert, and 4.3 days for the third.

### 5.4. End-users' Attitude

In order to validate end-user attitudes we have conducted a poll at one of the large-scale reengineered systems site. The results of the poll with 46 participants have shown that the application, standardized in this way, suits end-user particular needs. The time for a novice user to start a self-reliant usage of the associated user role (job) varied from 30 minutes to 2 hours. Since none of the polled users suggested any change in the domain of HCI standard, we have concluded that the standard was transparent to them, i.e. it allowed them to concentrate on their job, and not on the application software that served as a mediator. It is worth mentioning that all of those polled were individually trained in their own working environment.

## 6. Related Work

Work related to the topics discussed in this paper includes research in the areas of team organization strategies, rapid prototyping and tools construction.
Recommendations for the use of small highly-skilled teams in rapid software development are given in [7], [8], and [14].

Positive experiences in team working with the whole teams present in the same physical environment are the subject of a number of papers dealing with "war rooms". A review on this subject is given in [10].

Successful application of brainstorming sessions, within the software storming method that deals with rapid software development for military applications can be found in [9]. Although authors claim that such an intensive way of work is not suitable for broadly focused problems, we have showed that it is applicable to business information systems.

In the area of development tools research, there is a large number of papers dealing with a model-based code generating or descriptive languages, some of these intended to support rapid prototyping. The closest research to our work, in terms of concepts presented and results achieved, are the CAPS tool for building real-time systems using an abstract language as its foundation (see [11], [12]), and the Quava tool [13] for synthesizing distributed, object-oriented servers for the enterprise from object models. Both these tools use a repository as a knowledge base about the previously developed systems and to store a model of the application being built. However, they do not provide on-the-fly adjustment of the generated application based on changes in the repository data during testing.

## 7. Conclusions

This paper presents a method and a supporting tool for rapid development of large-scale information systems. The method is based on an optimal organization of a small highly-skilled development team, brainstorming techniques and a simple to use, highly efficient tool. The tool efficiency comes from the existence of a HCI standard, a library of high-level, coarse-grained components, a set of rules for model-to-application mapping with expert knowledge embedded and the ability of on-the-fly modification during testing.

The presented approach provided the following benefits: (1) fully working business subsystem prototype can be finalized in a matter of hours, (2) minimal number of team coordination documents is needed, hence the most of the effort can be focused on the product itself, (3) the possibility of introducing errors in early development phases is minimized, and (4) high user satisfaction and cooperation due to rapidly achieved results.

The limits of the presented approach can arise from the following requirements: (1) the approach assumes a team with motivated, highly-skilled members, and (2) optimum results require a continuous contact with the users motivated for cooperation.

## 8. References

1. Kalle Lyytinen. Different Perspectives on Information Systems: Problems and Solutions. ACM Computing Surveys, Volume 19, Issue 1, March 1987.
2. E. H. Conrow and P. S. Shishido. Implementing Risk Management on Software Intensive Projects. IEEE Software, May / June 1997.
3. David E. Avison and Guy Fitzgerald. Where Now for Development Methodologies? Communications of the ACM, January 2003. Vol. 46. No. 1.
4. Jim Rudd, Ken Stern, and Scott Isensee. Low vs. High-Fidelity Prototyping Debate. ACM Interactions, Volume 3, Issue 1, January 1996.
5. M. Alavi. An Assessment of the Prototyping Approach to Information Systems Development. Communications of the ACM, June 1984/Vol. 27, Issue 6.
6. A. Krabbel, I. Wetzel, and H. Zullighoven. On the Inevitable Intertwining of Analysis and Design: Developing Systems for Complex Cooperation. Symposium on Designing Interactive Systems, 1997, Amsterdam, The Netherlands.
7. D. Millington and J. Stapleton. Developing A RAD Standard. IEEE Software, Vol. 12, No. 5, September 1995.
8. E. Demirors, G. Sarmasik, O. Demirors, and D. Eylul: The Role of Teamwork in Software Develpment: A Microsoft Case Study. 23rd EUROMICRO Conference '97 New Frontiers of Information Technology, Budapest, Hungary
9. P. W. Jordan, K. S. Keller, R. W. Tucker, and D. Vogel. Software Storming: Combining Rapid Prototyping and Knowledge Engineering. IEEE Computer, May 1989 (Vol. 22, No. 5). pp. 39-48

10. Gloria Mark. Extreme Collaboration. Communications of the ACM, June 2002. Vol.45, No.6.
11. Luqi. Knowledge-Based Support for Rapid Software Prototyping. IEEE Intelligent Systems, 1988. Vol. 3, No. 4. pp. 9-15.
12. V. Berzins, O. Ibrahim, and Luqi. A Requirements Evolution Model for Computer-Aided Prototyping. Proceedings of the 9th International Conference on Software Engineering and Knowledge Engineering, Madrid, Spain, June 1997.
13. William J. Ray, and Andy Farrar: Object Model Driven Code Generation for the Enterprise. Proceedings of the 12th International Workshop on Rapid System Prototyping (RSP'01)
14. J. D. Blackburn, G. D. Scudder, and L. N. Van Wassenhove: Improving Speed and Productivity of Software Development: A Global Survey of Software Developers. IEEE Transactions on Software Engineering, Vol. 22, No. 12, December 1996.
15. M. Grechanik, D. Perry, D. Batory. An approach to evolving database dependent systems. Proceedings of the workshop on Principles of software evolution, Orlando, Florida, 2002
16. Microsoft SQL Server 2000 Books Online,. Microsoft Corporation, 2000
17. A. S. Fisher. CASE:Using Software Development Tools. John Wiley & Sons. Inc.1988.
18. I. Jacobson, G. Booch and J. Rumbaugh: The Unified Software Development Process. Addison-Wesley Longman, Inc. 1999.
19. G. Milosavljević, B. Perišić. An Approach to Automating Large-Scale Business Software Systems Construction Phase. 6th Conference on Operational Research, Thessaloniki, Greece, 2002.
20. B. Milosavljević, M. Vidaković and Z. Konjović. Automatic Code Generation for Database-Oriented Web Applications. In J. Power and J, Waldron, eds., Recent Advances in Java Technology: Theory, Application, Implementation. Computer Science Press, Trinity College Dublin, 2002. pp. 89-98.
21. B. Milosavljević, M. Vidaković, S. Komazec and G. Milosavljević. User Interface Code Generation for Data-Intensive Applications with EJB-Based Data Models. Software Engineering Research and Practice, Las Vegas, NV 2003.
22. G. Milosavljević, B. Perišić. Really Rapid Prototyping of Large-Scale Business Information Systems. 14th IEEE Intl. Workshop on Rapid System Prototyping, San Diego, CA, 2003. pp. 100-106.

**Gordana Milosavljević** is a teaching assistant and Ph.D. student at University of Novi Sad, Faculty of Engineering, Computer Sciences Department. She has received her B.Sc. and M.Sc. from University of Novi Sad, Faculty of Engineering, Computer Sciences Department. Her research interests focus on software engineering methodologies, rapid development tools and enterprise information systems design. Contact her at grist@uns.ns.ac.yu.

**Branko Perišić** is an associate professor of computer science and software engineering at University of Novi Sad, Faculty of Engineering, Computer Sciences Department. Professor Perišić has received his B.Sc. from University of Sarajevo, Faculty of Electrical Engineering. He has received his M.Sc. in computer system

security and Ph.D. in management information systems at University of Novi Sad, Faculty of Engineering. His research interests focus primarily on software engineering methodologies, software product standards, management information systems design and rapid software development. He is a member of PATIENT CLASSIFICATION SYSTEMS/EUROPE (PCS/E) and IEEE Computer Society. Contact him at perisic@uns.ns.ac.yu.