DOI: 10.2298/CSIS121218068R

Using Reverse Engineering to Construct the Platform Independent Model of a Web Application for Student Information Systems

Igor Rožanc and Boštjan Slivnik

University of Ljubljana
Faculty of Computer and Information Science
Tržaška cesta 25, 1000 Ljubljana, Slovenia
{igor.rozanc,bostjan.slivnik}@fri.uni-lj.si

Abstract. A methodology for extracting the domain knowledge from an existing three-tier web application and subsequent formulation of the platform independent model (PIM) is described. As it was devised during a reverse engineering process of an existing web application which needed to be reimplemented on a new platform using new technology, it focuses on the domain knowledge and business functions. It produces the business model and the hypertext model leaving the presentation model aside. The methodology is semi-automated — the generation of the activity diagrams and parts of the hypertext model must be in part performed by an analyst, preferably the one with some domain knowledge. As the paper is primarily aimed at practitioners, a case study illustrating the application of the presented method is included.

Keywords: reverse engineering, web application, platform independent model, PL/SQL

1. Introduction

Reverse engineering of an existing software application can be aimed at different goals. One is to gain the insight into a competitors' product to learn how to replicate its design, the other is to discover possible patent infringements. Often it is performed to produce various kinds of documentation [5] as the documentation might be either outdated or even nonexistent. The final result of reverse engineering is a description of the application at a higher level of abstraction, but this may be understood differently by different people. If only the documentation has to be obtained, reverse engineering can result in a formal text description. However, if the result is to be used further on, i.e., for upgrading, modification or even reimplementation of the existing application, diverse design models are needed.

Although a software technical specification can be either missing or outdated simply because of a professional misconduct, the deficient software specification can result from a particular software development model used to produce the application. For instance, if the agile software development methodology is used [29], it is quite possible that no detailed software specification will ever be produced since one of the main principles of the “Manifesto for Agile Software Development” explicitly values *working software over comprehensive documentation* [3]. Furthermore, the agile methodology concentrates more on management rather than on technical aspect and documentation [25].

Later on the agile approach to software development can become an obstacle for the maintenance of the application for many reasons. First, porting an application from the existing platform to a new one might be difficult (even if the new platform is only a major new version of the existing platform). Second, business processes might change and since they are hard coded, a significant amount of the code must be changed. Third, in time people initially working on the application get replaced by new people with less insight into the application.

Hence, at certain point in the application’s life cycle the existence of a model at a higher level of abstraction is an advantage for both managers and programmers [37]. Furthermore, it can also reduce maintenance costs [18]. The appropriate model can be produced even if no domain knowledge nor the application’s architecture is known but even a limited amount of domain knowledge and insight into the application’s architecture proves to be highly beneficial.

As the model-driven development (MDD) promotes a definition of the software development through a hierarchy of defined models at different levels of abstraction, it seems a natural choice for the formulation of the results obtained by reverse engineering [20, 30, 42]. These models are defined by the model-driven architecture (MDA) which implements the MDD. An important characteristic of MDA is promotion of the automatic generation of the lower level description models - the application code in the selected technology. Additionally, MDD promotes the use UML notation which became a standard modeling approach supported by various efficient tools. Thus by selecting the platform independent model (PIM) as defined in MDA for the final result of reverse engineering we gain a clear definition and an important advantage for the subsequent reimplementation of the application.

This paper describes a methodology of reverse engineering for producing the platform independent model (PIM) in order to modernize the application. In

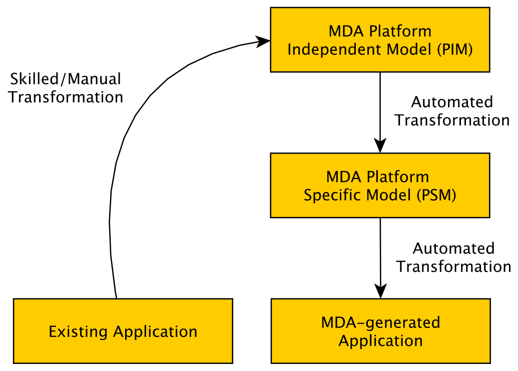
MDA Platform Independent Model (PIM)
Skilled/Manual Transformation
Automated Transformation
MDA Platform Specific Model (PSM)
Automated Transformation
Existing Application
MDA-generated Application

Fig. 1. The idea of reimplementation of an existing application using model-driven approach.

the ideal situation it would be used by the Architecture-Driven Modernization approach [41] as shown in Fig. 1. The methodology is especially targeted for situations where the team producing the PIM includes at least some members of the development team. Unlike some other papers [35], the paper includes an in-depth case study on how the methodology has been applied to a real world application. Initially the methodology advocated the generation of business model only [36]. As described here, it has been extended to include the generation of the hypertext model as well (leaving the presentation model aside as it is not needed if the application is reengineered). The generated class diagrams are now fully object oriented and by far the most critical part, i.e., the generation of the activity diagrams, has been improved significantly with (a) a new step of dead code elimination and (b) semi-automatic code simplification. Furthermore, the latest experiences with the described method are included as well.

Nowadays a large number of different languages, technologies and tools for the development of web applications is available. Hence, a completely general description of reverse engineering is almost impossible to formulate: either it is too general to be valuable as no concrete actions and procedures can be described or in trying to be comprehensive it becomes too large and imprecise. To ensure the paper has a practical value, the approach is given for the Oracle DB and Oracle Portal[1] as the selected case study is based on this technology. It is believed the Oracle technology is a good choice for presenting the new approach as it is (1) wide spread, (2) suitable for implementing the most complex web applications, and (3) a market leader and model for others. However, as PL/SQL is not object oriented (OO), reverse engineering of an existing PL/SQL application and generation of the PIM involve the shift to the OO-design.

The rest of the paper starts with Section 2 which contains an overview of the application used as a case study. Section 3 gives a short introduction into what the PIM of a web application should consist of. Section 4 describes elimination of unused parts of the application. Sections 5 and 6, the core of the paper, describe how the PIM can be extracted from an existing web application. The next two sections present the practical experience gained while producing the PIM for the actual real-world web application, and the discussion. The paper is concluded with a section on related work and conclusion. For the purpose of this paper, figures obtained from the generated diagrams have been manually translated to English.

2. A student information system as a case study

As the procedure presented in this paper sprang from practice, a case-study based on a web student information system named e-Študent (developed and initially deployed at the Faculty of Computer and Information Science of the University of Ljubljana, Slovenia) is to be introduced first [26, 27].

[1] The company, product and service names used in this paper are for identification purposes only — all trademarks and registered trademarks are the property of their respective holders.

e-Študent is a student information system developed and used at the University of Ljubljana, Slovenia. It is a three-tier web application built using the Oracle DB and Oracle Portal technology and written primarily in PL/SQL with some JavaScript. It consists of almost 800 different programming objects (dynamic pages, stored procedures and functions) with over 220.000 lines of code in total; its database contains almost 120 tables. The developer team consisted of people who were themselves developers and users at the same time, and hence the agile methodology seemed a natural choice [3, 29].

The development of e-Študent started in 2001 and by 2003 the initial release has been used by most faculties of the University of Ljubljana. Its main functions are providing electronic support for student enrollment, management of examination records and grades, and keeping the alumni records. All together it has been used by approximately 20.000 users (students, professors and staff). In 2008 the University of Ljubljana decided to replace e-Študent with at that time yet nonexistent successor. The reasons were many. First, the existing e-Študent was designed to be used by a single faculty and therefore different faculties were running their own instances instead of a single inter-faculty instance desired by the University. Second, as the number of elective courses increased dramatically by the introduction of the imminent new Bologna study programs (compared with the old pre-Bologna programs), the structure of the study programs was modified significantly and, sometimes even within the same faculty, in different directions.

The new system built after 2008 did not meet the expectations as it was focused more on implementing additional functionalities and less on suitable implementation of the existing, i.e., essential, ones. After the new system was made and evaluated it has been realized that during the development and maintenance of the original e-Študent a huge amount of the domain knowledge had been accumulated. In fact, due to the turbulent times of the Bologna reform, the source code of e-Študent is most likely the most comprehensive and the most formal specification of the student examination process. It is precisely this domain knowledge that should be extracted during reverse engineering, and it should be extracted as a model suitable for the development of e-Študent's successor after the first attempt, i.e., without eŠtudent domain knowledge, failed.

3. Platform independent model of a web application

Model-driven development is based on a notion of automatic transformations between different models describing an application on different levels. In the ideal situation, a developer would produce a platform independent model (PIM), add some platform specifications to reach the platform specific model (PSM), and finally generate the application [24]. Apart from this, the PIM provides a standard way to model an application in a technology independent way, i.e., by using UML diagrams. This approach is supported by a number of tools [42].

For web applications it has been advised [32] that the PIM should consist of

– *a business model*,

– *a hypertext model*, and
– *a presentation model*.

Apart from the (usual) business model describing the business processes, the hypertext model describes how web pages are built and linked while the presentation model contains details of the graphic appearance of a web application. The three-model web application description is a prevalent approach supported by most methods, however other model names, i.e., content, navigation and behavior model, may be used instead [7, 42].

It has been shown that by using the appropriate MD methodology, namely URDAD [39], out of 13 different kinds of UML diagrams only the following 4 kinds of diagrams are sufficient to produce the business model:

– *class diagram*,
– *use case diagram*,
– *activity diagram*, and
– *sequence diagram*.

The first three types are usually produced during the analysis phase (which is not an issue in reverse engineering) while all four types are needed during the design phase. Class diagrams are used for the service contract and for the collaboration context, use case diagrams are used for the responsibility identification and allocation, activity diagrams specify the full business process while sequence diagrams denote the user work flow and the success scenario [39].

The hypertext model is an abstract description of the composition and navigation between web application pages, page elements, and fragments of page elements [32]. This model is especially important in case of dynamic web applications with their distributed integration, user directed flow of execution and dynamic creation of HTML forms [33].

In most cases the PIM of an existing application is needed when the application must be reimplemented using the new platform. Hence, the focus is on the business model and the hypertext model; the presentation model is less important as it is to be replaced by a new one suitable for the new platform.

4. Elimination of the dead code and the dead data

During the development and especially during the maintenance developers produce a number of redundant code and data objects which are not used by the application anymore. Some of these objects might contain older or alternate solutions, some are temporary data collections, etc. Most of them are obsolete or even invalid.

The technique for dead code and dead data elimination is pretty straightforward: all objects not identified as live are considered dead. Live objects are identified using the following two steps:

1. Determining live root objects: In general, this step depends highly on the technology the application is made in. For the application that has been designed using the Oracle Portal and Oracle DB, the application's start page

	User pages	Dyn. pages	Procedures	Functions	Tables	Triggers
All objects	288	253	523	291	382	65
Live objects	236	186	388	207	117	58
Percentage	81.94	73.52	74.19	71.13	30.63	89.23

Table 1. The summary of dead code and dead data elimination.

and start pages of Portal user groups are retrieved from the Portal's Data Catalog.

2. Computing all live objects: Once the root objects are known, all objects reachable from the root objects must be identified. In other words, a transitive closure of the set containing the root objects must be computed. The connections among objects can be obtained by (a combination of) the following two methods:

   – by inspecting the metadata contained in the Portal's Data Catalog;
   – by parsing every object and extracting all links to other objects.

   Inspecting the metadata can most often produce perfectly adequate results. However, if nonstandard techniques have been used, the second method must be used also — it requires more effort since a parser for every programming language used within the application must be produced.

Note that the dead code elimination works on entire objects: if an object, i.e., a dynamic page, stored procedure or function, is found to be used, its entire contents is considered as live — dead code elimination within each particular code object is performed during the construction of activity diagrams.

Case study: e-Študent is an application made in Oracle Portal using Oracle DB. By inspecting the application's metadata in Oracle Portal the application's start page, five user groups and start pages for each user group were identified.

The core of the entry point for each user group is designed as a menu implemented in JavaScript (see also the description of use case diagrams below). By parsing JavaScript implementations of menus for all user groups the initial list of live user pages was obtained. Using Portal's Data Catalog all other live objects of the application were determined; apart from the user pages the list of live objects includes dynamic pages, stored procedures and functions, database tables and triggers.

The results of the dead code and dead data elimination are presented in Table 1. Note that more than 25 % of all code implementing the business logic, i.e., dynamic pages, stored procedures and functions, are not used. This is mainly due to the changes introduced during the Bologna reform. Likewise, almost 70 % of all tables are not used: many tables were introduced for live testing but later not removed for safety reasons.

5. Producing the business model

5.1. Class diagrams

Class diagrams represent the static structure of software systems in a graphical way [31]. Hence, it describes the structure of data and the structure of business processes.

In reverse engineering of a non-OO application based on the relational database the structure of data is obtained from the entity relationship model (ERM). The ERM might not exist or might be outdated, but it can be generated by inspecting the database using standard database tools. Once the relevant ERM is obtained, it is first transformed automatically into the conceptual class diagram, i.e., a class diagram without methods. This transformation encompasses the following rules [40]:

- Class definition: each entity is transformed into a separate class without any methods.
- Attribute definition: all table fields are transformed into attributes of appropriate classes.
- Associations definition: relations are transformed into associations and aggregations.

The names of classes and attributes are the same as the names of the corresponding entities and fields; classes corresponding to composite types get synthetic names.

The list of different tools that can carry out this transformation (at least to some degree if not entirely) includes tools like 'PowerDesigner' by SAP Sybase [4], 'UML Modeler for SQL' by Entrionics [2], and 'Altova UMODEL 2012' by Altova [1].

Although conceptional class diagram can be considered adequate [40], the proper way is to augment its classes with methods so that the resulting class diagram includes the description of behavior. During reverse engineering, the behavior is extracted from the stored procedures and functions of an existing application. To enhance the class diagram with methods, each stored procedure and function should be mapped into a method of a certain class. In cases where a stored procedure or function is associated with one table only, it is automatically transformed into a method of a class representing that table. Otherwise, a skilled analyst should manually determine the appropriate class.

Case study: The class diagram of e-Študent was made in two steps. In the first step, the ERM of e-Študent was automatically generated and transformed into the conceptional class diagram using PowerDesigner. The entire conceptional class diagram of e-Študent is shown in Fig. 2 which illustrates the overall structure of the original ER diagram and of the resulting (conceptual) class diagram at the same time.

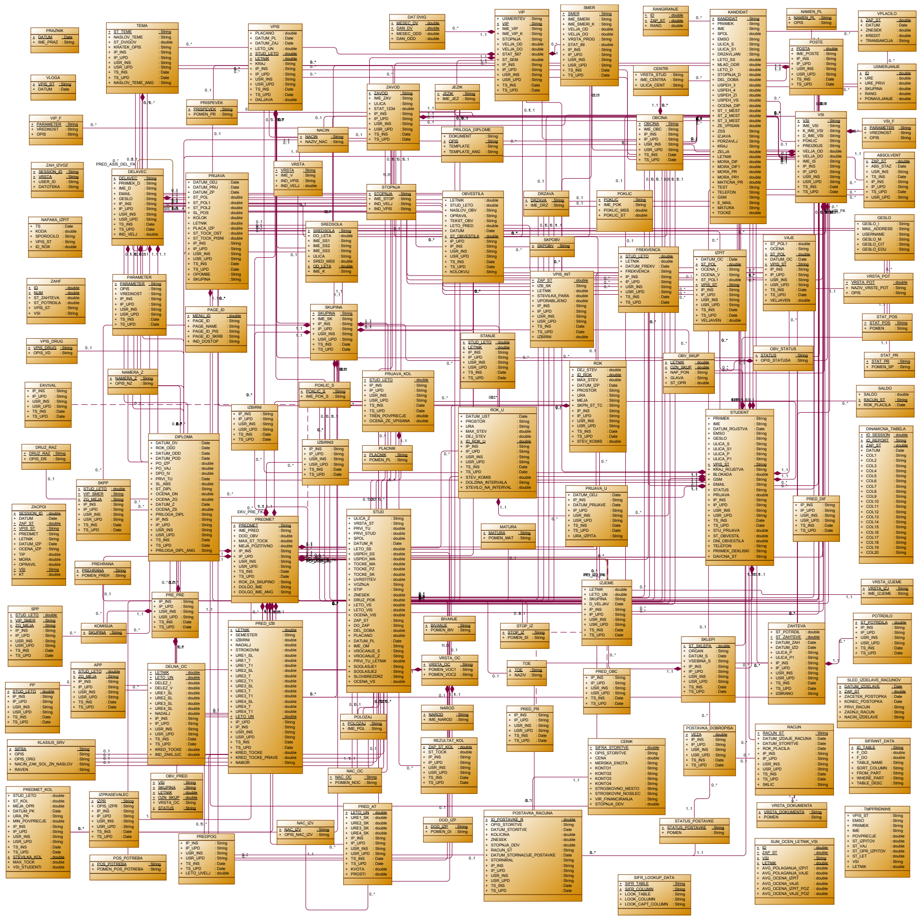

Fig. 2. The conceptual class diagram of e-Študent.

In the second step, the conceptual class diagram was transformed into the proper OO class diagram. The associations, i.e., the mapping of stored procedures and functions to tables, were first determined using the information available in Oracle DB Data Catalog and by subsequent parsing of stored procedures and functions source code[2]. In cases where a single stored procedure or function is associated with multiple tables, a heuristic was used to determine the list of most probable tables a stored procedure or function belongs to: tables with a higher number of inter table associations are placed higher on the candidate list. For each such stored procedure and function its sorted list of table candidates was presented to the analyst. He first selected the right

[2] The parser was made using the ANTLR v3.5-based PL/SQL parser by Patrick Higgins, `http://www.antlr3.org/grammar/list.html`.

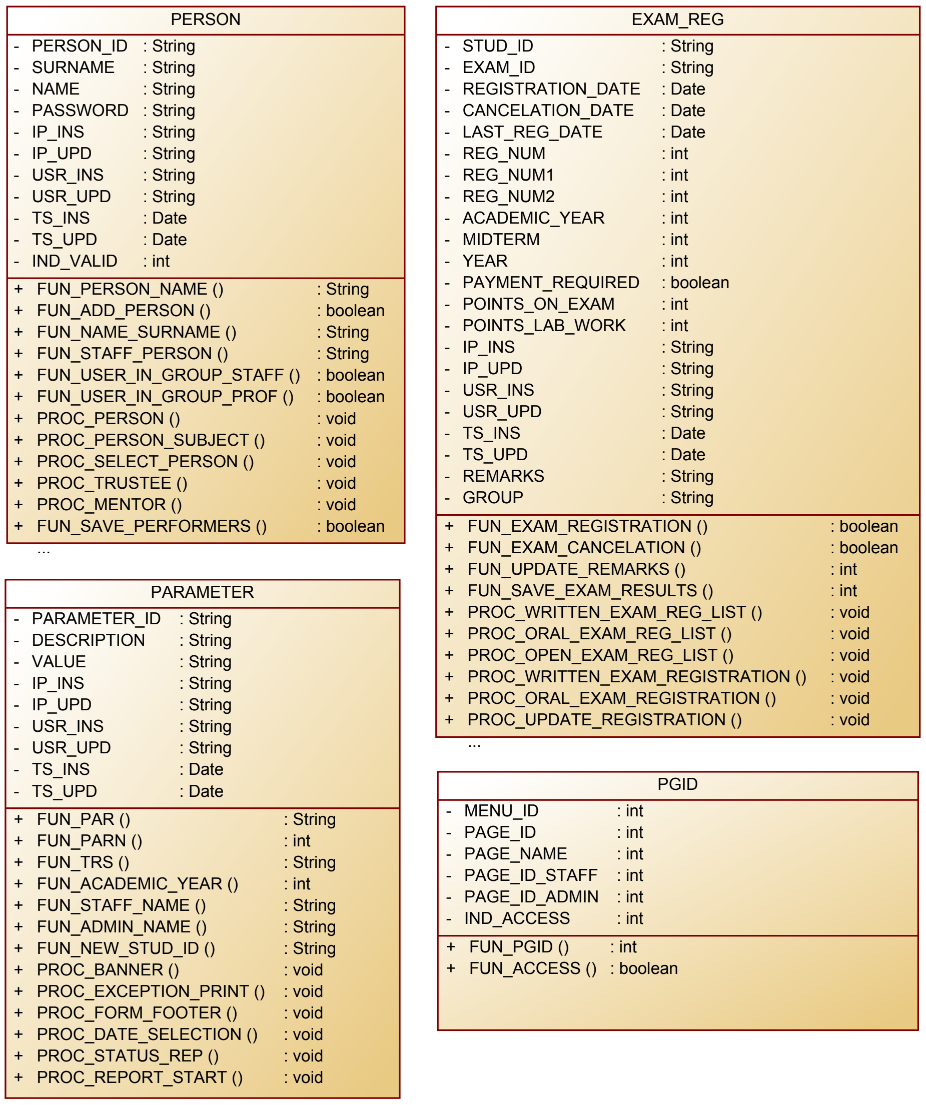
PERSON
- PERSON_ID : String
- SURNAME : String
- NAME : String
- PASSWORD : String
- IP_INS : String
- IP_UPD : String
- USR_INS : String
- USR_UPD : String
- TS_INS : Date
- TS_UPD : Date
- IND_VALID : int
+ FUN_PERSON_NAME () : String
+ FUN_ADD_PERSON () : boolean
+ FUN_NAME_SURNAME () : String
+ FUN_STAFF_PERSON () : String
+ FUN_USER_IN_GROUP_STAFF () : boolean
+ FUN_USER_IN_GROUP_PROF () : boolean
+ PROC_PERSON () : void
+ PROC_PERSON_SUBJECT () : void
+ PROC_SELECT_PERSON () : void
+ PROC_TRUSTEE () : void
+ PROC_MENTOR () : void
+ FUN_SAVE_PERFORMERS () : boolean
...
EXAM_REG
- STUD_ID : String
- EXAM_ID : String
- REGISTRATION_DATE : Date
- CANCELATION_DATE : Date
- LAST_REG_DATE : Date
- REG_NUM : int
- REG_NUM1 : int
- REG_NUM2 : int
- ACADEMIC_YEAR : int
- MIDTERM : int
- YEAR : int
- PAYMENT_REQUIRED : boolean
- POINTS_ON_EXAM : int
- POINTS_LAB_WORK : int
- IP_INS : String
- IP_UPD : String
- USR_INS : String
- USR_UPD : String
- TS_INS : Date
- TS_UPD : Date
- REMARKS : String
- GROUP : String
+ FUN_EXAM_REGISTRATION () : boolean
+ FUN_EXAM_CANCELATION () : boolean
+ FUN_UPDATE_REMARKS () : int
+ FUN_SAVE_EXAM_RESULTS () : int
+ PROC_WRITTEN_EXAM_REG_LIST () : void
+ PROC_ORAL_EXAM_REG_LIST () : void
+ PROC_OPEN_EXAM_REG_LIST () : void
+ PROC_WRITTEN_EXAM_REGISTRATION () : void
+ PROC_ORAL_EXAM_REGISTRATION () : void
+ PROC_UPDATE_REGISTRATION () : void
...
PARAMETER
- PARAMETER_ID : String
- DESCRIPTION : String
- VALUE : String
- IP_INS : String
- IP_UPD : String
- USR_INS : String
- USR_UPD : String
- TS_INS : Date
- TS_UPD : Date
+ FUN_PAR () : String
+ FUN_PARN () : int
+ FUN_TRS () : String
+ FUN_ACADEMIC_YEAR () : int
+ FUN_STAFF_NAME () : String
+ FUN_ADMIN_NAME () : String
+ FUN_NEW_STUD_ID () : String
+ PROC_BANNER () : void
+ PROC_EXCEPTION_PRINT () : void
+ PROC_FORM_FOOTER () : void
+ PROC_DATE_SELECTION () : void
+ PROC_STATUS_REP () : void
+ PROC_REPORT_START () : void
PGID
- MENU_ID : int
- PAGE_ID : int
- PAGE_NAME : int
- PAGE_ID_STAFF : int
- PAGE_ID_ADMIN : int
- IND_ACCESS : int
+ FUN_PGID () : int
+ FUN_ACCESS () : boolean

Fig. 3. Four classes of e-Študent class diagram.

table (usually the first table from the candidate list), and then, using Power Designer, declared the method (corresponding to the stored procedure or function in question) in the class based on the selected table. Four classes of the final class diagram are shown in Fig. 3. Unfortunately, the entire class diagram with all details (some methods have 20 parameters!) exceeds the available space. Method names are the same as names of the stored procedures and functions the methods are based (prefixes `PROC_` and `FUN_` are inherited from the existing eŠtudent code, not added). Keeping the same names simplifies the analysts transition from the old application to the new model.

5.2. Use case diagrams

A use case diagram represents system functionality by exhibiting the interactions between system's users and the transactions that provide value to users. They display relationship between actors and use cases [31]. In reverse engineering they are important as they offer unrivaled top-down insight of the system's functionality.

In most web applications, a use case diagram specifies a set of actions that can be performed by a certain user. In fact, as each user may play different roles, each use case diagram specifies main operations that can be performed by a particular user group. Hence a list of user groups must be retrieved from the system using one of the following two methods:

– by checking the list of user groups stored in the Portal;
– by inspecting the system from the user's point-of-view (in most cases, user roles can be determined by inspecting how each user logs into the application).

Once the list of user roles is established, one use case diagram per each user group should be produced. The generation of the use case diagram depends heavily on the technology and tools the web application is made with, and therefore no list of procedures applicable to all and every tool can be given here — for the Oracle DB/Portal case, see the case study below.

Users performing different user roles might be allowed to perform operations common to many different roles. Although a clear sign of an incautious design, two situations can nevertheless arise in practice:

– One particular function is found in different use case diagrams, either under the same name or under different names.
– Two functions found in two different use case diagrams share the same name but denote two substantially different actions.

Using the static analysis of the code it is possible to check whether two functions are carried out by the same code. However, if they are not, the resolution must be made manually by a developer/analyst.

Case study: Initially, the five user roles of e-Študent have been determined using the domain knowledge but the list of user roles kept by the Oracle Portal has been checked for verification.

The use case diagrams for e-Študent were obtained from the e-Študent menu files. The menu files were generated by HVMenu 5.41[3], a public-domain

[3] `http://www.dynamicdrive.com/dynamicindex1/hvmenu`; the latest version of HVMenu, i.e., 5.5, dates back in 2003.

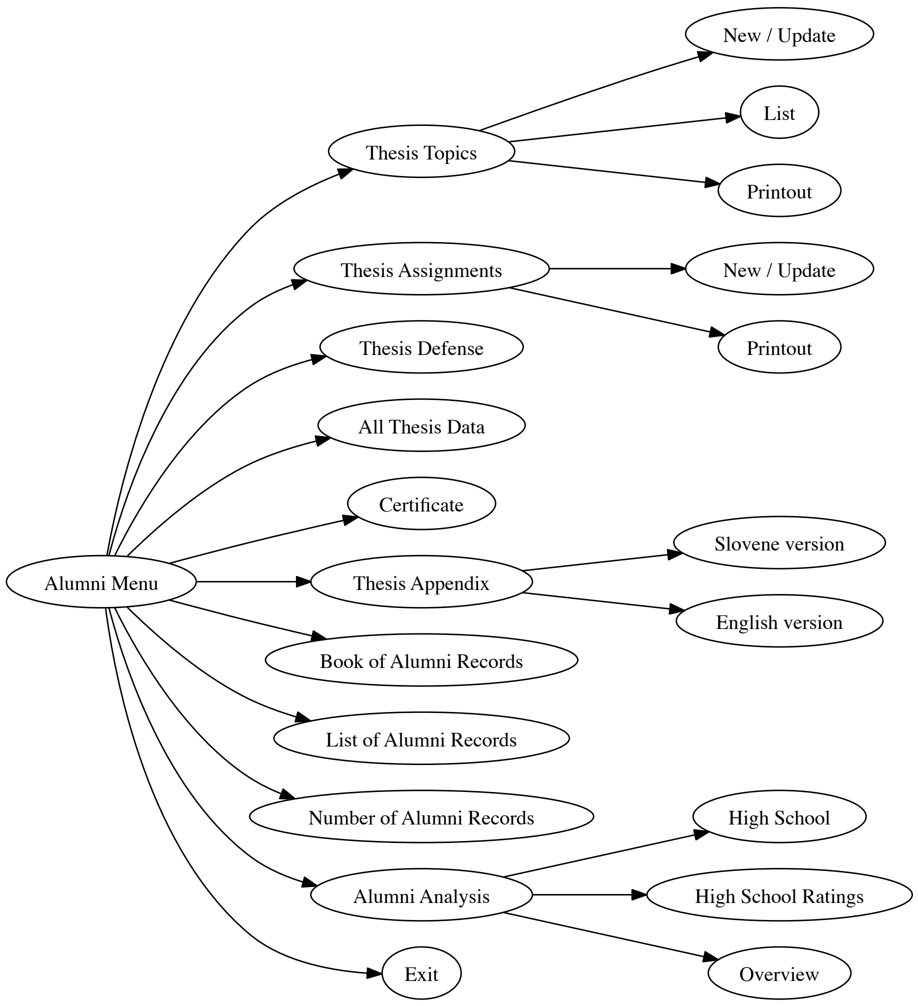

Fig. 4. The use case diagram for theses and alumni records: internal nodes represent (sub)-menus, leaves represent the main operations.

tool for producing Javascript code implementing menus to be used within web pages. Since generated, these files have a very indicative structure (one call of `Array` constructor per menu option) that allowed the entire structure of menus, submenus and options to be obtained by simple parsing. Hence, under the assumption (supported by the domain knowledge) that a single menu option reflect a single main operation a user can perform, the generation of use case diagrams was thus reduced to parsing Javascript menu files and producing the diagrams using the `dot` tool of GraphViz package by simply ranking graph nodes from left to right (`rankdir=LR`). Due to their size, only one of the resulting seven diagrams is shown; see Fig. 4. There are seven diagrams instead of five because the menu of one particular user group is split into three menus (one main menu and two submenus).

5.3. Activity diagrams

Activity diagrams are the most important artifacts in terms of the future reimplementation of the existing application as they denote the operational semantics of business processes [31].

At first glance, the activity diagrams can be produced automatically from the existing code. Certain tools are available, each specialized for a particular platform. However, no tool seems to be capable of generating activity diagrams for a real world application that would be suitable for reengineering the application. In most cases the generated activity diagrams are simply distilled code shown in a different form and thus they should not be used in the subsequent MDD for the following two reasons:

– The generated diagrams face granularity problem: understanding of the diagrams is obstructed as too many unnecessary details are included while sometimes some important details are systematically omitted [23, 43, 44].
– The generated diagrams may include some bad design elements which should not be propagated to the next versions of the application [23].

Hence, producing adequate activity diagrams cannot be fully automated but it can be machine supported. Namely, before the PL/SQL code is given to an analyst to retrieve the business logic, the code should be significantly simplified by automatically removing as much implementation details as possible. The heuristics used for code simplification is based on the classification of code fragments into the following categories:

– *object structure items*;
– *control structures* (`declare`, `begin`, `end`, `if`, `for`, etc.);
– *SQL blocks* (`cursor`, `select`, `update`, `fetch`, etc.);
– *data presentation* (`htp.` commands for data output);
– *data retrieval* (textbox, submit, button, checklist, etc.);
– *stored procedure and functions calls*;
– *other PL/SQL code*;
– *comments*;
– *other code*.

Each fragment containing an SQL block, data presentation or data retrieval is replaced by a single line. SQL block is replaced by the first SQL command augmented with the table used; the data presentation fragment and the data retrieval fragment are replaced by `print <data>` and `input <data>`, respectively, where `<data>` denotes the data either presented or read. The fragments of the last kind (other code) are removed.

The simplification introduces a risk some important details are removed. The analyst must be aware of this and when in doubt the original code of each simplified fragment must be checked.

The described method of code simplification is derived from the method used for producing workflows in IBM WebSphere Business Integration Workbench [43, 44], but it has been modified for the PL/SQL code in the Oracle platform — the list of categories has been compiled on the basis of authors' experience.

The code simplified in this way is raised to a higher level of abstraction and enhances the productivity of the analyst. Once the analyst performs the transformation of the PL/SQL code, the activity diagram can be generated from the simplified PL/SQL code automatically.

Case study: To actually produce the activity diagrams for e-Študent, the following sources needed to be reverse engineered:

- dynamic pages (HTML + PL/SQL),
- stored procedures and functions (PL/SQL),
- reports (SQL),
- triggers (PL/SQL), and
- JavaScript code.

A naive approach of using a tool to generate activity diagrams, e.g., 'UML Modeler for SQL' as

$$\begin{tikzcd}
\text{PL/SQL code: DynPages,procedures,functions} \arrow[d, "\text{UML Modeler for SQL}"] \\
\text{Activity Diagrams,}
\end{tikzcd}$$

failed exactly for the reasons outlined above. For a single stored procedure the number of elements within the activity diagram produced automatically using some tool is proportional to the number of lines of codes, and such a diagram is not readable even in case of moderate size procedures [36].

Illustrating the processing of a complete dynamic page exceeds the available space and thus only an excerpt is shown in Fig. 5. The first step consists of automatic PL/SQL code simplification. Classification of source code constructs and their subsequent removal or transformation in accordance with the rules specified above was implemented atop of another PL/SQL parser (also based on Patrick Higgins's ANTLR v3-based PL/SQL parser, see Subsection 5.1). After manual inspection and further simplification the activity diagram was generated using the reverse engineering options of 'UML Modeler for SQL'.

The entire set of activity diagrams includes 781 diagrams — one for every live dynamic page (186), stored procedure (388) and function (207). Manually

```
         ...
IF v_assistant = a THEN
  htp.p(’<tr><td width="30%" align="right">’);
  PROC_SELECT_PERSON(0);
  htp.p(’</td><td width="70%">&nbsp;’);
  PROC_SELECT_PERSON(1);
  htp.p(’</td></tr>’);
  t_person_id := FUN_STAFF_PERSON;
ELSE
  t_person_id := portal30.wwctx_api.get_user;
END IF;
htp.p(’<td width="12%" align="right" height="30"><font class="label_star">*</font><font class=
  "label_text">Date from:</font></td><td width="88%" valign="middle">&nbsp;’);
htp.formText(cname => ’f_from’, cvalue => v_from_aux, csize => ’10’, cmaxlength => ’10’, cattributes =>
  ’class="input_field" onChange="checkNull("f_from","f_from");isValidDate("f_from","f_from")"’);
PROC_DATE_SELECTION(’OpenExamRegs_form’, ’f_from’);
htp.p(’<font class="help_text">(dd.mm.llll)</font></td>’);
htp.p(’</tr><tr>’);
htp.p(’<td width="30%" align="right" height="30"><font class="label_star">*</font><font class=
  "label_text">Date from:</font></td>’);
htp.p(’<td width="70%" valign="middle">&nbsp;’);
htp.formText(cname => ’f_to’, cvalue => v_to_aux, csize => ’10’, cmaxlength => ’10’, cattributes =>
  ’class="input_field" onChange="checkNull("f_to","f_to");isValidDate("f_to","f_to")"’);
PROC_DATE_SELECTION(’OpenExamRegs_form’, ’f_to’);
         ...
```

$\Downarrow$ PL/SQL simplification parser

```
         ...
IF v_assistant = 1 THEN
  PROC_SELECT_PERSON(0);
  PROC_SELECT_PERSON(1);
  t_person_id := FUN_STAFF_PERSON;
ELSE
  t_person_id := portal30.wwctx_api.get_user;
END IF;
INPUT v_from_aux;
PROC_DATE_SELECTION(’OpenExamRegs_form’, ’f_from’);
INPUT v_do_pom;
PROC_DATE_SELECTION(’OpenExamRegs_form’, ’f_to’);
         ...
```

$\Downarrow$ manually

```
         ...
IF USER=ASSISTANT THEN
  PROFESSOR_SELECTION;
ELSE
  SET_USER_STAFF;
END IF;
INPUT_DATE1;
CHECK_DATE1;
INPUT_DATE2;
CHECK_DATE2;
         ...
```

UML Modeler for SQL $\Longrightarrow$

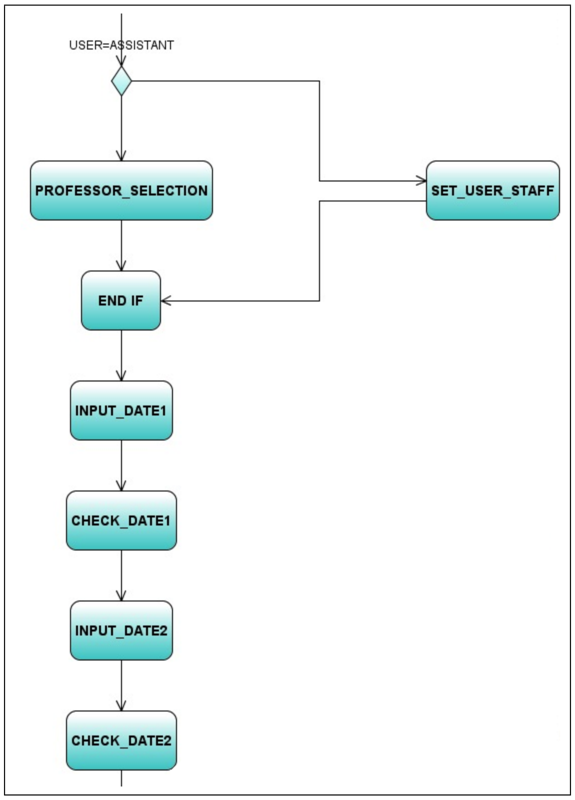

Fig. 5. Generating an activity diagram (due to its size only an excerpt is shown): the source code (top) is simplified twice (first automatically, then by an analyst) to be later transformed into to the activity diagram.

processing all this code objects is a tremendous task, but it is significantly reduced using the automatic code simplification and automatic generation of activity diagrams using UML Modeler for SQL. In many cases, e.q., maintenance or consulting, not all activity diagrams should be produced.

In e-Študent, the complexity of reports (generated using Oracle Report Builder), triggers and Javascript sources is not an issue. Therefore, they are considered as an appendix to the business model.

5.4. Sequence diagrams

Sequence diagrams express interactions and data flow between different objects within an application and thus describe a dynamic component of business processes. They may be used at different levels of abstraction to present different views of the application: usage scenarios (a description of a potential way the application is used), the logic of methods (explore the logic of a complex operation, function, or procedure) or the logic of services (a high-level method, often invoked by clients).

To generate sequence diagrams, a sequence of calls performed by every programming object must be obtained first. In PL/SQL code the programming object are dynamic pages (representing user's main operations) and stored procedures and functions (transformed to class methods).

Next, for each object a direct call tree is constructed: the root node contains the artifact itself and the leaves contain, from left to right, the artifacts called. Thus, the direct call tree is an ordered tree of height 1 if there are some calls from the programming object in the root node or 0 otherwise.

A direct call tree is transformed into a complete call tree by repetitive replacement of leaves with their direct call trees: a leaf is replaced if its direct call tree is of height 1 and no internal node on the path from the root to the leaf does not contain the programming object in the leaf. The transformation is completed once no leaf can be replaced any more.

Finally, the generated sequence diagram contains user's main operations (found in the use case diagrams) and classes (found in the class diagram) in the head sections while transitions are obtained by a preorder traversal of the complete call tree.

In this manner, a complete sequence diagram is obtained statically (disregarding PL/SQL control structures). There are at least two ways of producing the sequence diagram dynamically during reverse engineering. The first one is based on the reconstruction of clickstreams from the data obtained in the application's log files accumulated over many years [34]. The other way is to construct and apply a comprehensive set of test cases [9]. Using the dynamic approach, a number of sequence diagrams are obtained for each user's main operations while using the static method outlined above one complete sequence diagram per user's main operation, although bigger, is generated.

Case study: For all dynamic pages, stored procedures and functions of eŠtudent, sequences of inter-method calls were retrieved by a simple scanner of the source code additionally relying on information stored in Portal's Data Catalog. By another custom tool these sequences were transformed using the method described above into a GraphViz format and finally produced by `dot`.

An example of a sequence diagram is shown in Fig. 6. Due to the size of direct/complete call trees and the size of sequence diagrams, only a small excerpt of a single sequence diagram can be presented.

6. Producing the hypertext model

A suitable model is needed to adequately present the complex navigation of a dynamic web application. Several formal descriptions like FSM [10] and WebML [15] more or less meet this criterion, but the Atomic Section Model (ASM) [33] was chosen as it is capable of representing complex web applications with simple graphs. Furthermore, it can be produced using the static analysis.

ASM is a well known model primarily developed for testing web applications. Each HTML page is represented by a Component Interaction Model (CIM); CIMs of all HTML pages are combined together into an Application Transition Graph (ATG) representing ASM. Thus, the ATG of a web application is the formal representation of the hypertext model.

The CIM of a single HTML page is a quadruple $\mathrm{CIM} = \langle \mathrm{S}, \mathrm{A}, \mathrm{CE}, \mathrm{T} \rangle$ with

- a set of start pages S from which the page is referenced,
- a set of atomic sections A the page is made of,
- a component expression CE describing the page structure, and
- a set of transitions T pointing from and to (other) HTML pages.

An atomic section (AS) is a basic block of PL/SQL code producing the HTML page contents send to a client. The component expression is a regular expression denoting all possible sequences, selections and iterations of diverse ASs when dynamically constructing HTML page. All sets are fixed; in reverse engineering they can be retrieved by parsing HTML code or objects which dynamically create HTML pages, and by some manual processing by an analyst.

Originally the first component of a CIM is a single start page [33]. However, to produce the adequate hypertext model the first component of a CIM must be a set of all pages pointing to a page the CIM is made for. This is important for the construction of the ATG.

The ATG of an application is a quadruple $\mathrm{ATG} = \langle \Gamma, \Theta, \Sigma, \alpha \rangle$ with

- a set Γ of software components (CIMs),
- a set Θ containing all transitions of all CIMs,
- a set Σ of variables defining possible states of the presentation layer, and
- a set α of of all diverse starting pages (usually one).

The ATG is usually presented as a directed graph with a set of vertices Γ and a set of edges Θ.

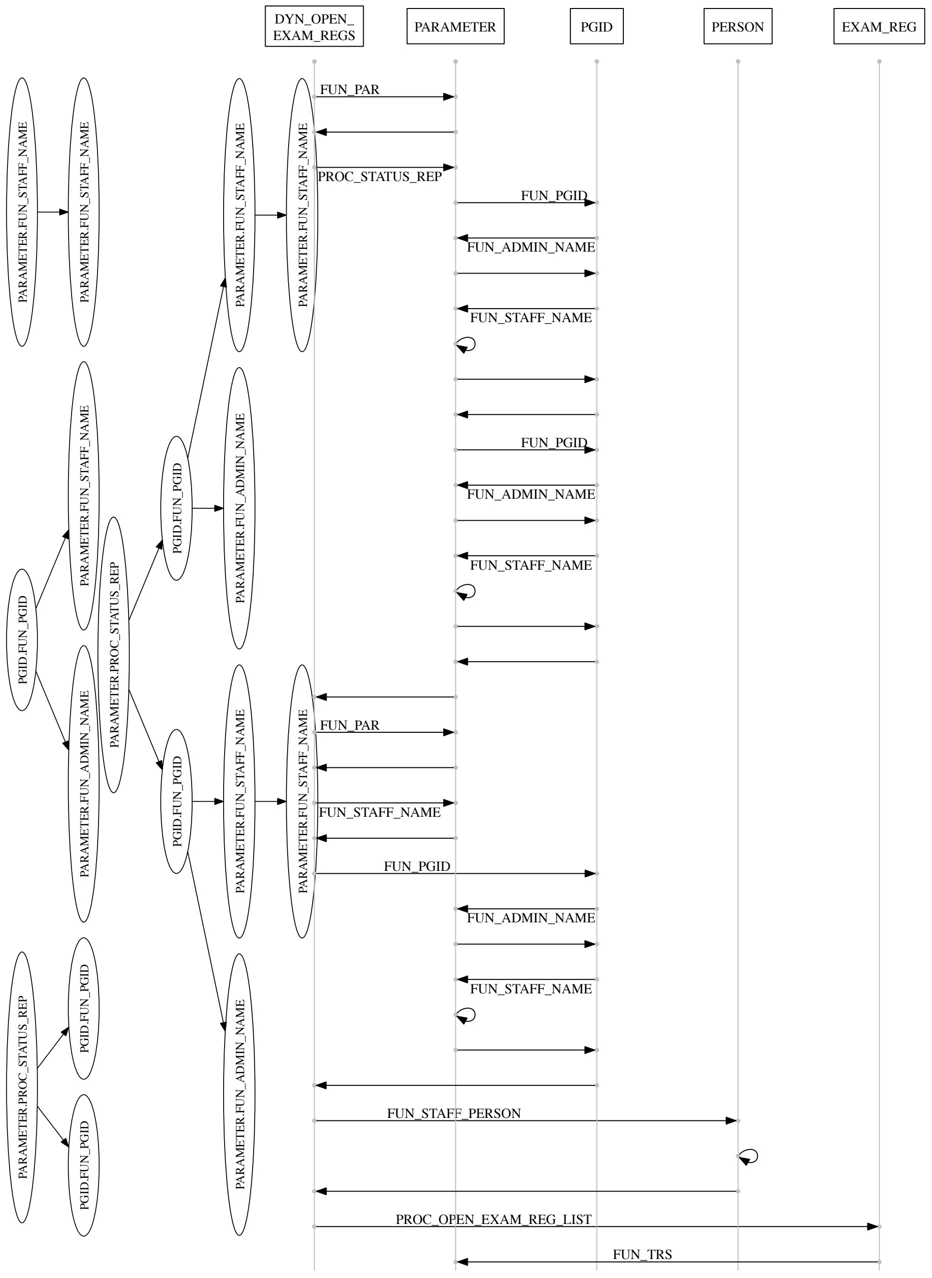

Fig. 6. An example of a sequence diagram generation (only cca 15 % is shown). The transitions corresponding to the call of `PARAMETER.PROC_STATUS_REP` from dynamic page `DYN_OPEN_EXAM_REGS` is based on the complete call graph (center) produced from the three direct call graphs (left).

A CIM is extracted from a dynamic page or a stored procedure using a method similar to extracting an activity diagram: it consists of (1) code simplification, (2) manual extraction of CIM structure and generation of graphical representation of CIM.

Code simplification is performed by the parser used for generation of activity diagrams except that the code fragments are classified into the following categories:

- *presentation elements* (code fragments which are copied to HTML page or produce the contents that is copied to HTML page);
- *links* (links to other HTML pages, SUBMIT parameter definitions, etc.);
- *control structures*;
- *procedure and function calls*;
- *other code* (SQL blocks, declarations, etc.).

All fragments classified as 'other code' are eliminated. Note that this code simplification concentrates on code fragments that were mostly left out by the code simplification used for producing activity diagrams.

Once the code is simplified, the CIM is produced manually by an analyst defining atomic sections and the control flow among them, and the transitions to and from other dynamic pages. Each atomic section represent a description of a group of presentation elements, stored procedure and function calls at a higher level of abstraction. The ATG is constructed simply by connecting CIMs using the transitions among them.

Case study: For instance, the CIM of a dynamic page for open exam registrations is shown in Fig. 7. It contains 15 atomic sections (labeled $\mathrm{P}_1 \ldots \mathrm{P}_{15}$) connected as described by the regular expression

$$\mathrm{P}_1((\mathrm{P}_2|\varepsilon)\mathrm{P}_3((\mathrm{P}_4(\mathrm{P}_5|\mathrm{P}_6)(\mathrm{P}_7|\varepsilon)\mathrm{P}_8)|(\mathrm{P}_9(\mathrm{P}_{10}|\varepsilon)\mathrm{P}_{11}(P12|\varepsilon)\mathrm{P}_{13})|\varepsilon)\mathrm{P}_{15})|\mathrm{P}_{14}$$

where P_1 is "HeadSectionAndTitle", P_2 is "StatusMissingURL", etc. Futhermore, it contains 4 transitions, namely

$$\begin{aligned}
&\mathrm{MenuProfessor} \longrightarrow \mathrm{DYN\_OPEN\_EXAM\_REGS[POST(\perp,\perp,\perp,\perp)]}\\
&\mathrm{MenuStaff} \longrightarrow \mathrm{DYN\_OPEN\_EXAM\_REGS[POST(\perp,\perp,\perp,\perp)]}\\
&\mathrm{DYN\_OPEN\_EXAM\_REGS.PrintReport} \longrightarrow \mathrm{GetReport[POST(id)]}\\
&\mathrm{DYN\_OPEN\_EXAM\_REGS.EndOfBodySection} \longrightarrow\\
&\qquad \mathrm{DYN\_OPEN\_EXAM\_REGS[POST(f\_from, f\_to, f\_button, f\_print)]}
\end{aligned}$$

The first two transitions specify where is this dynamic page reachable from while the third one specifies which dynamic page is used after the main operation performed by this dynamic page has been finished. The fourth transition specifies a return to the same dynamic page once the parameters have been refined using HTML form elements contained within the page.

To produce CIMs for eŠtudent, the source code of a dynamic page was simplified by the same parser as used for activity diagrams but using the code

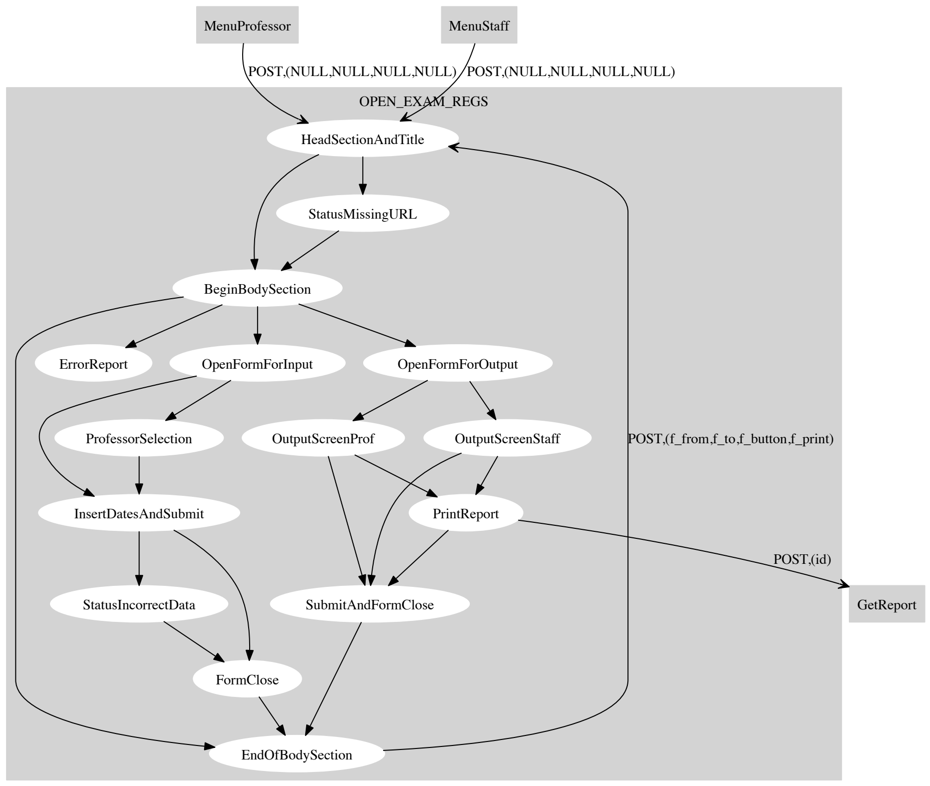

Fig. 7. The CIM for the dynamic page for processing open exam registrations.

classification as described above. Once the analyst produced the CIM description, the graphical presentation of CIMs was generated using the `dot` tool from the GraphViz package.

Apart from the four pages used during the login process, e-Študent contains five entry points, one for each user group. After a quick check it was established that one CIM must be produced for each of 186 dynamic pages 349 of 388 stored procedures and 65 of 207 stored functions. Thus, the hypertext model of e-Študent consists of almost 600 CIMs. To produce the ATG, each transition is augmented with a link to a `dot` file containing a CIM of a dynamic page reachable by the transition.

7. Lessons from the case study

Since the start of the reverse engineering of eŠtudent in 2012 [36], the focus of the reverse engineering has changed. Namely, in autumn 2012 the initial goal of producing the PIM suitable for fully automatic reengineering of eŠtudent has been replaced by producing the PIM suitable for maintenance and consulting.

The change of the goal was due to the decision of the new development team not to implement the new eŠtudent from the described models because of the complexity of the application, major modifications of the existing requirements and a number of new requirements.

The initial eŠtudent development team consisted of 9 developers but only one developer (the first author) actively maintains the application nowadays. The construction of the models proved to be a great advantage during maintenance for a number of reasons. First, it provided the maintainer a consistent top-down understanding of various aspects of the entire application. Second, it enabled the maintainer to understand the entire source code even though most of the past developers are no longer of any help. It turned out that code modifications (due to an urgent requirement changes) or bug fixes were much easier to perform once the corresponding activity diagrams were constructed. Before the diagrams were produced, finding and fixing the part of the code to be changed and testing the fix demanded significantly more work.

Once the new development team started on the next generation of eŠtudent in autumn 2012, the models were used intensively for the domain knowledge transfer during the design phase of the new application. The class diagram was transferred into the new application and later modified according to new requirements, and the use case diagrams were user for establishment of the compulsory tasks.

The consultant (the first author) found the activity diagrams made at a higher level of abstraction as described in this paper invaluable since the new development team needed information on operations at a higher level of granularity. More precisely, once the main elements of the PIM were available, the consulting became far more comprehensive and compact. The need for consulting services decreased sharply after approximately 3 months during which the consultant-analyst was producing and conveying the model (especially the activity diagrams). Hence, the method proved its importance in practice. Whenever low-level details were required, the original source code was preferred to the automatically generated activity diagrams, i.e., as described in [36].

As the construction of models requires a considerable amount of analyst's work, the models were constructed to the extent requested by maintenance needs and consulting work only. Additionally, the construction of the models was interleaved with the development of the necessary custom tools and thus no clear separation of time used for each of these two tasks is available.

The construction of class diagram and use case diagrams took 2 man-weeks to complete. The most of this time was spent for assigning methods based on stored procedures and functions to classes of the conceptual class diagram.

As expected, the most time consuming part of the reverse engineering proved to be the construction of activity diagrams. After the initial attempt described in [36], the dead code elimination and source code simplification were introduced. Now, an analyst roughly familiar with eŠtudent needs approximately 1 hour to produce the activity diagram for a relatively simple dynamic page shown in Fig. 5. Based on all activity diagrams constructed (51 in total so far) during maintenance and consulting, we estimate that 2 hours per activity diagram are needed on average and thus for the entire set of 781 activity diagrams 40 man-weeks suffices.

As the generation of sequence diagrams is fully automated, the time needed for the generation is not an issue. The CIMs are usually produced by the same analyst as activity diagrams and thus approximately 60 % less time is needed as for the activity diagrams (but since less CIMs than activity diagrams has been produced so far, this estimation is less reliable).

Many papers agree that a large part of the reverse engineering must be performed manually, but the paper by Di Lucca et al. is one of the few that provide at least some metrics on actual reverse engineering of a web application. Their conclusions are the same as ours: “the most expensive steps are those requiring human intervention” [17, p. 96] (even though there are some inconsistencies regarding Table V on the same page).

8. Discussion

An important issue in reverse engineering is to properly define the focus:

1. If the produced model can be used by the MDD tools to generate the new version of the application, a lot is gained since the automated code generation is less error prone and can be done quickly for different platforms.
   However, to produce the adequate model for automated MDD, the proper understanding of the existing web application must be gained first. This understanding usually requires a shift to a higher level of abstraction which, as all authors agree, cannot be fully automated. Only after the model on the higher level of abstraction is obtained, the usual forward engineering involving “understanding $\rightarrow$ model $\rightarrow$ code” can be applied [32].
2. Otherwise, the produced model can serve as a well formalized documentation and therefore it enables the switch from agile software development to other software development methodologies.
   Furthermore, a proper formal documentation provides a foundation for efficient maintenance and consulting. Like above, the models must be shifted on the higher level of abstraction in order to yield an insight rather than simply machine readable description.

In either case, the shift to a higher level of abstraction involves primarily processing of the existing source code to produce proper activity diagrams since these diagrams represent the main formalization of business logic. The next in line are CIMs since they specify the navigation aspect. Generation of both kinds of diagrams requires analyst support.

PL/SQL proved to be far too low-level formalism any automatic transformation into the PIM could be successful [13]. Even if activity diagrams are generated automatically, they are too detailed and include too many past design elements (including bad solutions that should not be propagated to newer versions).

There are certain tools available that can ease reverse engineering significantly (Visual Paradigm, UModel, PowerDesigner, . . .), but for two reasons the task cannot be fully automated using any of them. First, even if they produce a formally correct model or a part of it, it is usually too large and too detailed, and thus inadequate for later use as shown in [36]. Second, obviously no tool can gain a proper insight into the logic behind the application. This proves to be the major obstacle in reverse engineering of an application based on the entity relationship diagrams and PL/SQL code as producing the PIM involves a shift to the OO design. Hence, a number of custom tools must be made during reverse engineering to support but not replace the manual construction of the PIM.

As it turns out reverse engineering of PL/SQL code must be done by someone who is at least to some degree familiar with the design of the system being reverse engineered. There are at least three operations that must be human assisted: transformation of the entity relationship diagrams to class diagrams, transformation of the PL/SQL code to activity diagrams, and transformation of the presentation elements of the PL/SQL code into the hypertext model.

Furthermore, a lesson learnt the hard way is that if a reverse engineering of a PL/SQL-based web application that was produced by agile development is to be economically viable, reverse engineering must be performed by at least some members with the domain knowledge who participated in the development of the application.

Finally, even though PL/SQL is not a good starting point for a reverse engineering towards the PIM, sometimes it simply must be done. And although it might encompass a lot of manual processing, the resulting model is worth the effort [18].

9. Related work

Recently, a number of authors reported a different approaches to reverse engineering of dynamic web or similar applications. Favre described a MDA-based framework of platform-independent and platform-dependent models that are to be produced by reverse engineering of object-oriented code [20]. It is “propose(d) to apply static and dynamic analysis to generate models” [20] but no actual procedures for the generation of these models are given and no test based on a real-world application is included.

Di Lucca et al. [17] presented a reverse engineering process for dynamic web application supported by the WARE tool. Both static and dynamic analysis were performed to describe business model using class diagrams, use-case diagrams and sequence diagrams. Although similar to our work, their approach does not include activity diagrams — but only activity diagrams enable a comprehensive understanding of business logic, i.e., not just what tasks are performed but also how they are actually performed.

Concentrating on the actual procedures of static reverse engineering, Zou et al. describe how a business model definition can be constructed after undocumented changes to the source code were made [44, 45]. Like ours their technique involves source code simplification using a heuristic based on categorizing programming language constructs. However, they only describe the generation of workflows which roughly correspond to activity diagrams while our method covers other models of PIM and the hypertext model as well.

Bellucci et al. described the MARIA tool to perform static reverse engineering of user interfaces of dynamic web applications [12]. The static transformation from the concrete language (HTML, CSS, Ajax, JavaScript) to the abstract (platform independent) one is described — again it is based on categorizing source code constructs. The tool produces the description of the user interface at two different abstract levels but it cannot be used to generate a complete business and hypertext model.

The dynamic approach to reverse engineering has also been considered. Like Zou et al. [44, 45] Di Francescomarino et al. presented a reverse engineering process leading to the web application's business model only using the dynamic analysis of GUI-forms [16]. Amalfitano et al. focused on dynamic analysis of Rich Internet Applications (RIA) user interface to produce a proper description using Finite State Machines (FSMs) [8]. Similarly, Marcheto et al. presented a ReAjax tool to perform on Ajax web applications [28]. It is focused on Ajax specifics and produces GUI-based state model. Alalfi et al. [6] perform reverse engineering to obtain UML sequence diagrams for PHP web applications using instrumentation and analysis of execution traces. Likewise, Briand et al. described reverse engineering of distributed Java applications to produce sequential diagrams as well [14].

To summarize, certain papers include no or very little concrete procedures for reverse engineering [19, 20, 35] while we include the generation of these models as well. Other papers focus on a single aspect, i.e, workflow based business model [44, 45], reverse engineering of UIs [12, 16, 28], sequence diagrams [6, 14], or leave out some important aspect [17]. However, our approach tends to come as close as possible to the PIM of a web application which should consist of a business model, a presentation model and a hypertext model as many authors working on a standard (forward) modeling agree, i.e., UWE [21, 22], WebML [15], OOHDM [38], Netsilon [32], W2000 [11].

10. Conclusion

Instead of yet another paper describing a methodology of a reverse engineering for producing different models, we concentrated on one particular kind of web applications, namely those written primarily in PL/SQL and based on Oracle Portal/DB. Our methodology produces the business and the hypertext model, both at the level of abstraction suitable for human insight into the application. In presenting it, we focused on procedures rather than on a tool that might implement them.

We tried to avoid the approach of (1) some sources [35] which describe what models are to be made without giving any hints about how this models could be made, (2) of some sources [44, 45] which do not provide the entire PIM, or (3) of some sources [17] that leave out the most important components, i.e., the activity diagrams. We find these approaches inadequate as the problems are not in the selection of the appropriate models but in gaining the insight into the existing application and the subsequent applicability of the produced models.

The methodology has been tested on the real-world application intensively used in practice, i.e., a student information system eŠtudent. The models retrieved by reverse engineering have been used successfully for maintenance and consulting. We believe that the produced models represent a viable starting point for the design of a model suitable for automated MDA.

References

1. Altova UModel 2012, http://www.altova.com/umodel.html (retrieved June 4th, 2013)
2. Entrionics UML modelling for SQL, http://www.entrionics.com (retrieved June 4th, 2013)
3. Manifesto for agile software development, http://agilemanifesto.org (retrieved: June 4th, 2013)
4. SAP Sybase PowerDesigner, http://www.sybase.com/products/modelingdevelopment/powerdesigner (retrieved November 5th, 2012)
5. Akkiraju, R., Mitra, T., Thulasiram, U.: Reverse engineering platform independent models from business software applications. In: Telea, A.C. (ed.) Reverse Engineering - Recent Advances and Applications, chap. 4, pp. 83–94. InTech Press (2012)
6. Alalfi, M.H., Cordy, J.R., Dean, T.R.: Automated reverse engineering of UML sequence diagrams for dynamic web applications. In: Proceedings of the 2nd International Conference on Software Testing, Verification and Validation (ICST'09). pp. 287–294. IEEE Computer Society Press, Denver, CO, USA (2009)
7. Alalfi, M.H., Cordy, J.R., Dean, T.R.: Modelling methods for web application verification and testing: state of the art. Software Testing, Verification and Reliability 19, 265–296 (2009)
8. Amalfitano, D., Fasolino, R., Tramontana, P.: Experimenting a reverse engineering technique for modelling the behaviour of rich internet applications. In: Proceddings of the IEEE International Conference on Software Maintenance (ICSM'09). pp. 571–574. IEEE Computer Society Press, Edmonton, AL, Canada (2009)
9. Ammann, P., Offut, J.: Introduction to Software Testing. Cambridge University Press, New York, NY, USA (2008)
10. Andrews, A.A., Offut, J., Alexander, R.T.: Testing web applications by modeling with fsms. Software & Systems Modeling 4(3), 326–345 (2005)
11. Baresi, L., Garzotto, F., Paolini, P.: Extending UML for modeling web applications. In: Proceedings of the 34th Annual Hawaii International Conference on System Sciences. pp. 1–10. Honolulu, HI, USA (2001)
12. Bellucci, F., Ghiani, G., Paternò, F., Porta, C.: Automatic reverse engineering of interactive dynamic web applications to support adaptation across platforms. In: Proceedings of the 2012 ACM International Conference on Intelligent User Interfaces (IUI'12). pp. 217–226. Lisbon, Portugal (2012)
13. Billig, A., Busse, S., Leicher, A., Süss, J.G.: Platform independent model transformation based on triple. In: Proceddings of the 5th ACM/IFIP/USENIX International Conference on Middleware (Middleware'04). pp. 493–511. Springer-Verlag, New York, NY, USA (2004)

14. Briand, L.C., Labiche, Y., Leduc, J.: Toward the reverse engineering of UML sequence diagrams for distributed Java software. IEEE Transactions on Software Engineering 32(9), 642–663 (2006)
15. Ceri, S., Piero, F., Bongio, A.: Web modeling language (WebML): a modeling language for designing Web sites. Computer Networks 33(1–6), 137–157 (2000)
16. Di Francescomarino, C., Marchetto, A., Tonella, P.: Reverse engineering of business processes exposed as web applications. In: Proceedings of the 13th European Conference on Software Maintenance and Reengineering. pp. 139–148. IEEE Computer Society Press, Kaiserlautern, Germany (2009)
17. Di Lucca, G.A., Fasolino, A.R., Tramontana, P.: Reverse engineering web applications: the WARE approach. Journal of Software Maintenance and Evolution: Research and Practice 16(1-2), 71–101 (2004)
18. Dzidek, W.J., Arisholm, E., Briand, L.C.: A realistic empirical evaluation of the costs and benefits of UML in software maintenance. IEEE Transactions on Software Engineering 34(3), 407–432 (2008)
19. Escalona, M.J., Gutiérrez, J.J., Rodríguez-Catalán, L., Guevara, A.: Model-driven in reverse: the practical experience of the AQUA project. In: Proceedings of the 2009 Euro American Conference on Telematics and Information Systems: New Opportunities to increase Digital Citizenship (EATIS'09). pp. 17:1–17:6. ACM, Prague, Czech Republic (2009)
20. Favre, L.: Formalizing MDA-based reverse engineering processes. In: Proceedings of the 6th International Conference on Software Engineering Research, Management and Applications (SERA'08). pp. 153–160. IEEE Computer Society Press, Prague, Czech Republic (2008)
21. Knapp, A., Koch, N., Zhang, G.: ArgoUWE: A CASE tool for web applications. In: Proceedings of the 1st International Workshop on Engineering Methods to Support Information Systems Evolution. Geneva, Switzerland (2003)
22. Koch, N., Kraus, A.: The expressive power of UML-based web engineering. In: Proceedings of the 2nd International Workshop on Web-Oriented Software Technology (IWWOST'02). pp. 105–119. Malaga, Spain (2002)
23. Liebarman, B.: UML activity diagrams: Versatile roadmaps for understanding system behavior. The Rational Edge p. 12 (2001)
24. Luković, I., Varanda Pereira, M.J., Oliveira, N., da Cruz, D., Henriques, P.R.: A DSL for PIM specifications: Design and attribute grammar based implementation. Computer Science and Information Systems 8(2), 379–403 (2011)
25. Mahnič, V.: A case study on agile estimating and planning using scrum. Electronics and Electrical Engineering 5(111), 123–128 (2011)
26. Mahnič, V., Drnovšček, S.: Introducing agile methods in the development of university information systems. In: Proceedings of the 12th International Conference on European University Information Systems (EUNIS 2006). pp. 61–68. Tartu, Estonia (2006)
27. Mahnič, V., Rožanc, I., Poženel, M.: Using e-business technology in a student records information system. In: Proceedings of the 7th WSEAS International Conference on E-Activities (E-Activities'08). pp. 589–594. Cairo, Egipt (2008)
28. Marchetto, A., Tonello, P., Ricca, F.: Reajax: a reverse engineering tool for ajax web applications. IET Software 6(1), 33–49 (2012)
29. Martin, R.C.: Agile Software Development, Principles, Patterns, and Practices. Prentice Hall, Englewood Cliffs, NJ, USA (2002)
30. Mellor, S.J., Scott, K., Uhl, A., Weise, D.: MDA distilled - principles of model-driven architecture. Addison-Wesley, Boston, MA, USA (2004)

31. Miles, R., Hamilton, K.: Learning UML 2.0. O'Reilly Media, Sebastopol, CA, USA (2006)
32. Muller, P.A., Studer, P., Fondement, F., Bezivin, J.: Platform independent Web application modeling and development with Netsilon. Software & System Modeling 4(4), 424–442 (2005)
33. Offutt, J., Wu, Y.: Modeling presentation layers of web applications for testing. Software & Systems Modeling 9(2), 257–280 (2010)
34. Poženel, M., Mahnič, V., Kukar, M.: Separation of interleaved web sessions with heuristic search. In: Proceedings of the IEEE International Conference on Data Mining (ICDM). pp. 411–420. IEEE Computer Society Press (2010)
35. Raghupathi, W., Umar, A.: Exploring a model-driven architecture (MDA) approach to health care information systems development. International Journal of Medical Informatics 77(5), 305–314 (2008)
36. Rožanc, I., Slivnik, B.: Producing the platform independent model of an existing web application. In: Proceedings of the Federated Conference on Computer Science and Information Systems. pp. 1385–1392. IEEE Computer Society Press, Wroclaw, Poland (2012)
37. Rugaber, S., Stirewalt, K.: Model-driven reverse engineering. IEEE Software 21(4), 45–53 (2004)
38. Schwabe, D., Rossi, G., Barbosa, S.D.J.: Systematic hypermedia application design with OOHDM. In: Proceedings of the 7th ACM Conference on Hypertext. pp. 116–128. ACM, Bethesda, Maryland, USA (1996)
39. Solms, F., Loubser, D.: Generating MDA's platform independent model using URDAD. Journal of Knowledge-Based Systems 22(3), 174–185 (2009)
40. Sparks, G.: Database modelling in UML. Methods & Tools 9(1), 10–23 (2001)
41. Ulrich, W.M., Newcomb, P.: Information Systems Transformation: Architecture-Driven Modernization Case Studies. Morgan Kaufmann, Burlington, Mass., USA (2010)
42. Valderas, P., Pelechano, V.: A survey of requirements specification in model-driven development of web applications. ACM Transactions on the Web 5(2), article(10) (2011)
43. Zou, Y., Guo, J., Foo, K.C., Hung, M.: Recovering business processes from business applications. Journal of Software Maintenance and Evolution: Research and Practice 21(5), 315–348 (2009)
44. Zou, Y., Lau, T.C., Kontogiannis, K., Tong, T., McKegney, R.: Model-driven business process recovery. In: Proceedings of the 11th Working Conference on Reverse Engineering (WCRE'04). pp. 224–233. IEEE Computer Society Press, Delft, The Netherlands (2004)
45. Zou, Y., Zhang, Q., Zhao, X.: Improving the usability of e-commerce applications using business processes. IEEE Transactions on Software Engineering 33(12), 837–855 (2007)

Igor Rožanc received the M.Sc. and Ph.D. degrees in computer science from the University of Ljubljana in 1995 and 2003, respectively. He is currently at the University of Ljubljana, Faculty of Computer and Information Science. Throughout his career he has been actively involved in the design and development of student information systems. His research interests include distance learning and programming methodologies.

Boštjan Slivnik is an Assistant Professor at the University of Ljubljana, Faculty of Computer and Information Science where he received the M.Sc. and Ph.D. degrees in computer science in 1996 and 2003, respectively. His research interests include parsing algorithms, compilers, formal languages, scheduling and distributed algorithms. He has been a member of the ACM since 1996.

Received: December 18, 2012; Accepted: August 23, 2013.