ISBN 978-83-60810-51-4

Producing the Platform Independent Model of an Existing Web Application

Igor Rožanc, Boštjan Slivnik
University of Ljubljana
Faculty of Computer and Information Science
Tržaska 25, 1000 Ljubljana, Slovenia
Email: {igor.rozanc, bostjan.slivnik}@fri.uni-lj.si

***Abstract*—A reverse engineering procedure for producing a platform independent model (PIM) of an existing Web application is presented using a case study. It focuses on extracting the domain knowledge built into the application and thus it produces the PIM, leaving the hypertext and presentation models aside. It is especially focused on reverse engineering of applications produced using agile software development methodology where documentation is scarce, and as it assumes that in large part the activity diagrams are produced and refined manually, it is particularly useful in environments where at least some developers of the original agile team are still available. Rather than being a result of a theoretical work, the method has crystallized during a lot of practical work. As such it is aimed at practitioners and following the spirit of its formulation, it is presented as a case study where it has been applied.**

I. Introduction

REVERSE engineering of a software application is most often performed for one of the following two reasons [1]. First, to gain an insight into a competitors' product and either learn how to replicate its design or to discover possible patent infringements. Second, to produce various kinds of documentation of the software application when the documentation is either outdated or even nonexistent.

Although a software specification, i.e., a technical documentation, can be either missing or outdated simply because of a professional misconduct, the deficient software specification can result from a particular software development model used to produce the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 agile software development methodology is used [2], 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].

However, if the application proves to be successful, it inevitably grows in both size and complexity, and the environment the application now lives in changes. Hence, although it can be a key factor for success during the design, the agile approach to software development can become an obstacle for the maintenance of the application later on. The reasons are many. First, porting an application from the existing platform to a new one might be difficult (sometimes even if the new platform is in fact only a major new version of the old platform). Second, business processes might change and since the existing processes are hard-wired in the code, a significant amount of the code must be changed. Third, in time people initially working on the application, either business people or developers, are replaced by new people with less insight into the application.

Therefore, at certain point in the application's life cycle the existence of a model at a higher abstraction level becomes beneficial for both managers and programmers [4]. Furthermore, if designed properly, the appropriate model can also reduce maintenance costs [5]. The generation of the appropriate model might be focused on two different issues:

- extracting the model from the existing application in the situation when no domain knowledge nor application architecture is known, or
- producing the model of the existing application using a (limited) domain knowledge and insight into the application architecture.

In both cases, reverse engineering of the application is to be performed. However, in the first case, the reverse engineering must be started from scratch while in the second case the model generation, although still complex, is much simpler (or at least feasible).

An important issue is the final result of the reverse engineering. By definition it is a description of the application at a higher level of abstraction, but this can mean very different things to different people. Usually the reverse engineering is aimed towards the most useful form of description for some latter. These forms include, among others,

- diverse design models needed for the reengineering of the existing application, or
- a formal text description in case when only the documentation has to be obtained.

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 the reverse engineering [6], [7]. 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. Thus by selecting the platform independent model (PIM) as defined in MDA for the final result of the reverse engineering we gain

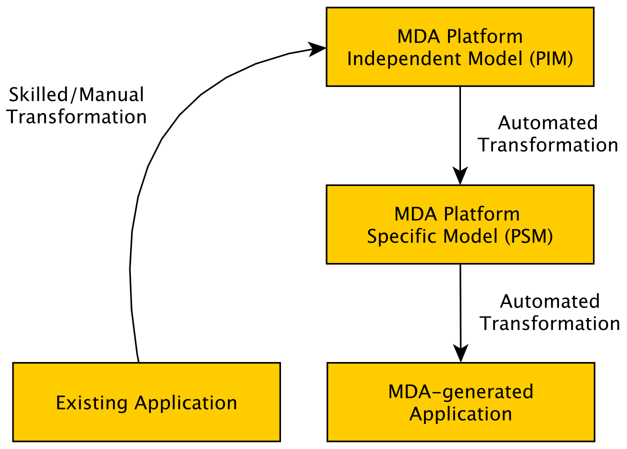
MDA Platform Independent Model (PIM)
Skilled/Manual Transformation
Automated Transformation
MDA Platform Specific Model (PSM)
Automated Transformation
Existing Application
MDA-generated Application

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

an important advantage for the following reconstruction of the application in the selected technology whatever it might be.

Note that the OMG's Architecture-Driven Modernization(ADM) proposals suggest this kind of solution: this paper presents a tipical Application / Data Architecture-Driven Modernisation through a case study of an existing application redesigned using MDA [8].

Using a case study, the paper describes an agile methodology of reverse engineering that can be applied in order to produce the platform independent model as used by the MDD as shown in Figure 1: the paper focuses on reverse engineering of the existing application to produce the platform independent model, a step performed by a mostly manual transformation since at the current state of technology it cannot be fully automated. It is especially targeted for Web applications that were produced using agile software development. It is assumed that to apply the methodology the team producing the model includes at least some members of the original development team.

Nowadays a baffling number of different 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 to general to be valuable as no concrete actions and procedures can be described or in trying to be comprehensive it becomes to 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 the implementing the most complex Web applications, and (3) a market leader and model for others. However, as PL/SQL is not object oriented (OO), the reverse engineering of an existing PL/SQL application and generation of the PIM involve the shift to the OO-design.

Hence, the paper starts with Section II containing a short overview of the Web application used as a case study and the events that triggered the introduction of major modifications that actually caused the shift from the agile to the model-driven approach. Section III gives a short introduction into what the PIM of a Web application should be made of, and Section IV, the core of the paper, describes how a PIM can be extracted from a existing Web application by using a case study. The paper concludes with a section on the lessons learnt in producing the PIM for the actual real world Web application, and conclusion.

II. 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 [9], [10].

e-Študent is a three-tier Web application built using the Oracle DB and Oracle Portal technology, and written primarily in PL/SQL with some JavaScript. Technically speaking it has over 800 different programming objects (namely dynamic pages, stored procedures and functions) with over 220 KLOC in total, 130 different reports, and its database contains more than 160 tables. As it has been designed and developed at the Faculty of Computer and Information Science, the developer team consisted of people who were themselves developers and users at the same time, and the agile methodology seemed a natural choice [2], [3].

The development of e-Študent started way back 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 enrolment, managment of examination records and grades, and keeping the alumni records. All together it has been used by approx. 20000 users (students, lecturers 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

[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.

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 processes at the University of Ljubljana. It is precisely this domain knowledge that should be extracted during the reverse engineering, and it should be extracted as a model suitable for the development of the next e-Študent successor.

III. Platform Independent Model of a Web Application

Model-driven development is based on a notion of an 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.

For Web applications in particular, it has been advised [12] 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.

Leaving the hypertext and presentation models aside and concentrating on the business model, it has been shown that by using the appropriate MD methodology, namely URDAD [13], out of 13 different kinds of UML diagrams only the following 4 kinds of diagrams are sufficient to produce the PIM:

- *class diagram*,
- *activity diagram*,
- *use case 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 for the services contract and for the collaboration context, sequence diagrams for the user work flow and for the success scenario, a use cases for the responsibility identification and allocation, and finally a set of activity diagrams for the full business process specification [13].

IV. A Procedure for Producing the PIM using Reverse Engineering

Even thought the four types of diagrams aforementioned in Section III were identified as the minimal requirements for the PIM [13], it is much easier to talk about these diagrams rather than actually produce the concrete instances of these diagrams by reverse engineering an existing (Web) application. To produce the PIM of a Web application not by analysis of a problem domain but by reverse engineering of an existing Web application, the appropriate diagrams can only be produced using the combination of both static and dynamic analysis. More precisely, there are basically two phases of constructing PIM by reverse engineering:

1) A *static phase of reverse engineering* is performed first: a class diagram, a number of use case diagrams, activity diagrams, and sequential diagrams are produced, in this particular order.
2) A *dynamic phase of reverse engineering* comes second: sequence diagrams are upgraded with additional information on the data flow.

Finally, a third phase, namely the optimization of the reverse engineered PIM, can be performed. In the following four subsections the procedure of producing each type of diagram by reverse engineering is outlined.

A. Class Diagrams

Class diagrams represent the static structure of software systems and subsystems in a graphical way [14]. In the case of the non-OO business applications database objects are typically modeled in this way.

Class diagrams are the easiest to produce: the relational database tools are capable of producing entity relationship model (ERM) and there exists a relatively simple transformation from the ERM to a class diagram. More precisely, as the behaviour information is ommited, we produce a *conceptional class diagram.* The transformation is done by applying the following rules:

- each entity is transformed into its own class with no methods and stereotype “DatabaseTable”;
- each composite data type appearing in the ERM is transformed into its own class;
- entity fields are mapped into attributes of the class corresponding to the entity the field belongs to;
- relations between tables are mapped as dependencies between classes.

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

The list of different tools that can carry out this transformation (at least to some degree if not entirely) includes among others tools like ‘UML Modeller for SQL’ by Entrionics [15] and ’Altova UMODEL 2012’ by Altova [16].

In our case the database has a quite simple structure with no composite types. Hence, the mapping between tables from ERM and UML classes is quite straighforward. As this study is focused primarily on the data layer, the presentation layer description is left for further work.

However, the database can contain “garbage”, i.e., currently unused tables that were either used in the past or were used during development. Although all such tables should have been removed, one cannot assume that they actually were. This is usually not a problem: for instance, if Oracle DB is used, one can use the data available in the Oracle DB Data Catalogue to remove all unused tables.

Case study: The transformation of the e-Študent ERM to a class diagram was straightforward — the e-Študent ERM does not even contain any composite type. A class diagram of e-Študent, immensely simplified for the presentation purposes, is shown in Figure 2.

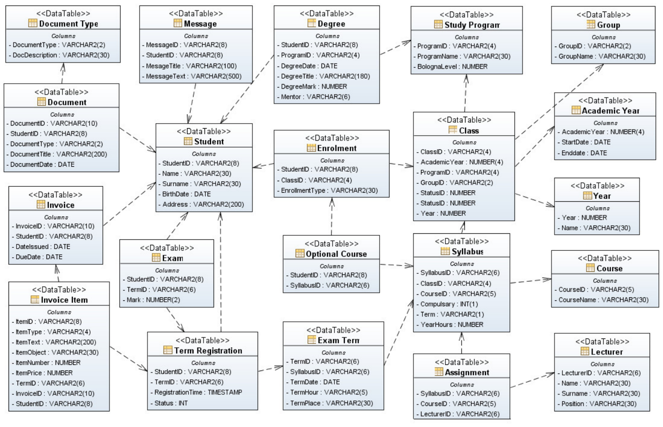

Fig. 2. An example of a class diagram obtained by reverse engineering of e-Študent (the class diagram is significantly simplified: in reality there are approx. 160 classes and some classes might contain dozens of attributes).

B. Use Case Diagrams

A use case diagram represents system functionality by exhibiting the interactions between system users and the transactions that provide value to users. They display relationship between actors and use cases and dependencies that exist between users [14].

As they describe user requirements use case diagrams should be among the first artifacts produced by both software development and reverse software engineering. In reverse engineering they are important as they offer unrivaled top-down insight of system functionality. Furthermore, as the system is being reverse engineered it is assumed the system is adequate and thus the use case diagrams produce a very clear top-down picture of the problem domain.

In most Web applications, use case diagram specifies a set of actions that can be performed by a certain actor (describing a user role for a group of users). In fact, as each user may play different roles, each use case diagram specifies functions that can be performed by a particular user role. Hence a list of user roles (actors) must be retrieved from the system using one of the following two methods:

- by checking the list of user roles stored in the database or within the application server;
- 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 or how the user can explicitly change its role afterwards).

Once the list of user roles is established, one use case diagram per each user role 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' at the end of this subsection).

Users performing different user roles might be allowed to perform functions 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.

Case study: Initially, the five user roles of e-Študent have been determined using the domain knowledge. However, checking the list of user roles kept by the Oracle Portal has been performed as well.

As the application front-end of e-Študent has a simple single-level menu structure, the use case diagrams for e-Študent were obtained from the menu files especially designed for e-Študent. These files, written in Javascript, have a very indicative structure that allowed the entire structure of menus, submenus and options to be obtained by simple parsing. Hence, under the assumption (based upon the domain knowledge) that a single menu option reflect a single function a user can perform, the generation of use case diagrams was more or less reduced to parsing Javascript menu files. The smallest of the five resulting diagrams is shown in Figure 3.

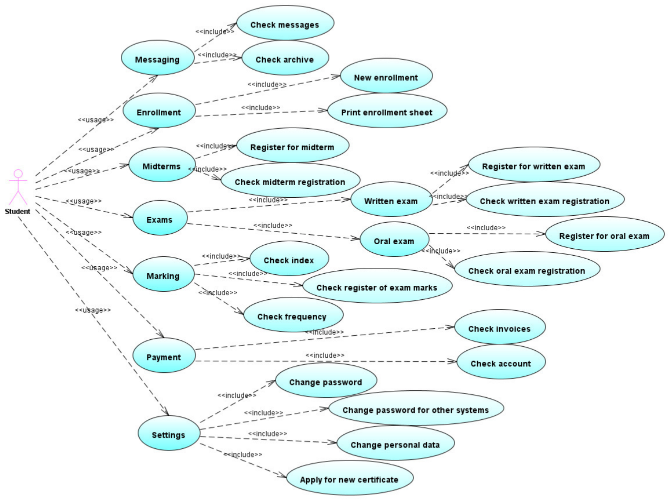

Fig. 3. A use case diagram for user role 'student' obtained by reverse engineering of e-Študent (the leaves of the tree represent the functions this particular user role can perform).

C. 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 [14].

At first glance, 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 generating adequate activity diagrams automatically for the real world applications — adequate in terms of the forward software development phase. In fact, in most cases the generated activity diagrams are simply distilled code shown in a different form. Hence, they are not to be used in the subsequent MDD for the following 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 [17], [18], [19].
- The generated diagrams include many system design elements, including bad ones which, since the diagrams are to be used in forward system design, would be propagated to the next versions of the application [17].

To conclude, as the tools cannot distinguish between the business process and its implementation, i.e., between what should be done and how it is done, producing the adequate activity diagrams is essentially a human lead process. Nevertheless, we promote the use of tools as they can quickly generate syntaticaly correct diagrams. These can be used as an additional input for further manual processing. To be precise, our activity diagram definition process is based on the generated activity diagrams if they are of moderate size. Otherwise, it is better to start from a scratch due to the granularity problem of generated diagrams.

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 Modeller for SQL' as

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

failed exactly for the reasons outlined above. For a single procedure of approx. 300 lines of PL/SQL code, the tool produced the activity diagram shown in Figure 4. The number of elements in the diagram is proportional to LOC, and such a diagram is not readable even in case of modarate size procedures.

Furthermore, tools mentioned above couldn't merge PL/SQL code that is embedded into the dynamic pages with PL/SQL code found in the stored procedures and functions (however, this is a deficiency of the tools used, not a problem per se).

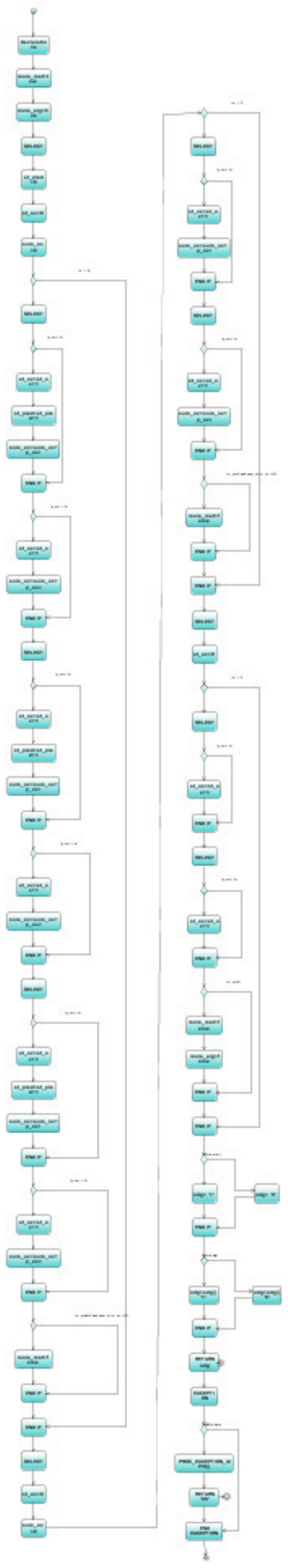

Fig. 4. An example of a generated activity diagram for a PL/SQL procedure of approx. 300 lines of code (the diagram is rotated 90 degrees leftward).

The difference between the PL/SQL code and SQL queries must be noted at this point. While the structure of the first can be clearly described in the activity diagram using sequence, selection and iteration elements, the second demands a different approach [20]. For this reason the reverse engineering tools simply skip the analysis of SQL chunks in PL/SQL code and present them as a simple activity. Thus, the analysis of SQL code should be done separately. At present the reverse engineering of the existing e-Študent is limited on collecting all SQL code chunks, forming a list of them, and establishing two-way references between items from the list of SQL chunks and the resulting activity diagrams.

a) Dynamic pages, procedures and functions: Dynamic pages are basically HTML pages with fragments of embedded PL/SQL code while procedures and functions are written in pure PL/SQL. To extract the workflow and create an activity diagram representing the action performed by a particular dynamic page, no lightweight tool was available. Thus, the reverse engineering was performed manually using the following steps:

1) **Establishing a proper list of dynamic pages:**
   A proper list of dynamic pages must be defined on the application server since a number of unused dynamic pages exists as well. To do this, parsing the menu structure can be used in a similar way as in use case diagram case. Each menu option references a particular Oracle Portal user page, each consisting of a predefined structure of portlets where main portlet points to a specific dynamic page.
2) **Obtaining the code of dynamic pages:**
   PL/SQL code of all dynamic pages from the list compiled in the previous step is to be collected. Unfortunately, the dynamic pages are stored at the Oracle Portal and thus not directly available in the Oracle DB, which would facilitate the access and analysis. Each dynamic page has to be extracted using Oracle Portal functionality which takes a lot of time. However, each dynamic page is stored in the database as a compiled package as well (dynamic page can be identified, but the code is not available) and this information can be used to cross-check whether all code has been processed.
3) **Parsing the PL/SQL code:**
   Once the PL/SQL code has been retrieved, it must be processed. This can be performed partially using the appropriate reverse engineering tool, but it must definitely be completed manually.
   At this point, specific knowledge about the programming standards used by e-Študent team has been heavily used since
   - all dynamic pages have a limited size and a similar structure, and
   - a list of strict rules for content distinction between dynamic pages and stored procedures/functions code has been applied, e.g., all SQL code is in the stored procedures/functions and thus omitted from the dynamic pages.

   Furthermore, in the initial phase of the development, strictly formalized comments have been systematically inserted into dynamic page headers; later, during the implementation of modifications needed due to the Bologna reform, this practice was abandoned.
   One activity diagram at a higher level of abstraction per dynamic page is the main objective of this step, but a separate lists of all called stored procedures/functions, reports and JavaScript code is obtained for later use. References to other dynamic pages can emerge as well - they are added to the dynamic pages list and processed in the same manner.
4) **Parsing the PL/SQL code of stored procedures and functions:**
   All PL/SQL code for all stored procedures and functions used by dynamic pages must be processed. These procedures and functions are found on the list compiled in the previous step.
   The code of the stored procedures and functions is saved in Oracle DB. The code can be retrieved from Oracle Data Catalog and thus the process is much simplified as a special step for obtaining the code is not needed.
   On the other hand, the structure of actual procedures and functions can be quite diverse in length, use of comments and content. Just like in the previous step, the use of a tool is suitable in the first place, but a careful manual analysis must be performed to complete the task. During parsing, several new procedures and functions are added to the list and those have to be processed in the same manner. Likewise, new reports can be added to the report list as well, while one of the main artifacts is a list of SQL code, which is collected and left for processing later. Of course, references to all list items are stored in the activity diagrams.

Once a member of reverse engineering team who was a member of the development team as well took care of the diagram in Figure 4, he was able to simplify it (by throwing out design elements) and obtained the diagram shown in Figure 5. Obtaining the right level of abstraction seems to be the most difficult task at this point.

b) Reports, triggers and JavaScript code: In e-Študent, reports were generated by Oracle Reports Designer tool. The basic report information includes the identity and the layout of how fields are to be displayed or printed out. However, a report is in fact generated by executing an SQL query which includes, alongside the appropriate data to be reported, data and formatting transformation as well. All reports are saved in separate files on the application server, the exact identity and location of report files is defined in a special table. There is no simpler way than checking all different reports manually, collecting SQL code and adding it on the list of SQL code with the proper references. Processing of SQL code is left for the next version of reverse engineering process as stated above.

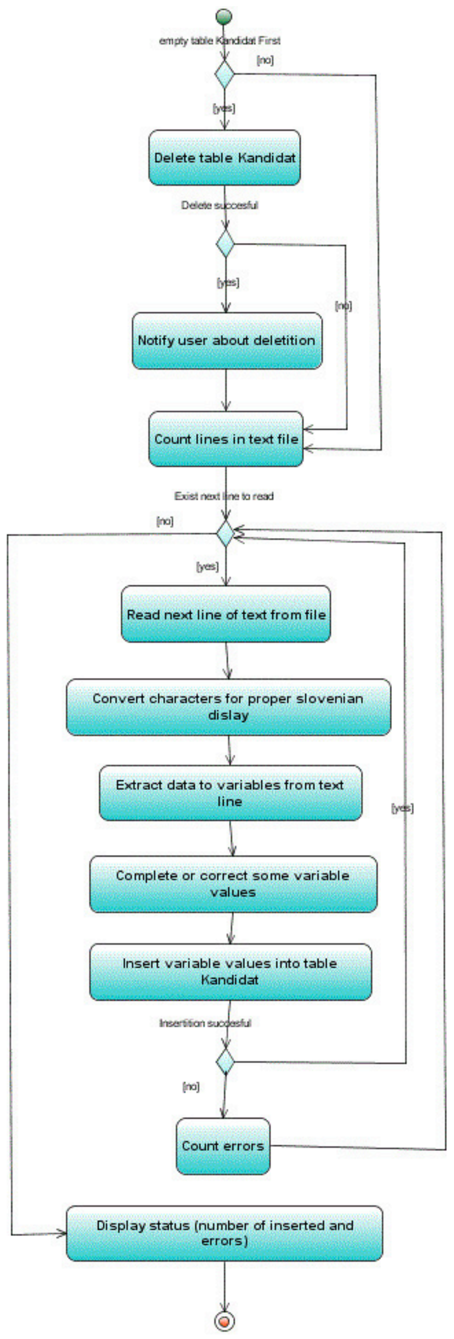
empty table Kandidat First
[no]
[yes]
Delete table Kandidat
Delete succesful
[yes]
[no]
Notify user about deletition
Count lines in text file
Exist next line to read
[no]
[yes]
Read next line of text from file
Convert characters for proper slovenian dislay
Extract data to variables from text line
[yes]
Complete or correct some variable values
Insert variable values into table Kandidat
Insertition succesful
[no]
Count errors
Display status (number of inserted and errors)

Fig. 5. An example of an activity diagram (a) based on the activity diagram in Figure 4 but (b) created by a member of a reverse engineering team.

It is important to process the triggers as well as most of them implement important data transformations and common safety policies. The code of triggers for all used tables, obtained by retrieving it from Oracle Data Catalog, is usually very simple. Hence, an appropriate tool can be used in most cases, but the results have been manually checked nevertheless.

JavaScript is a different language, but still JavaScript code can be transformed to activity diagrams in the same way as PL/SQL stored procedures and functions. The list of JavaScript files is obtained in the previous steps by parsing dynamic pages. Files are located on the application server, so they must be retrieved using Oracle Portal. In the last step they have been analyzed using the selected tool and manually post processed as in the previous steps. The amount of JavaScript code is quite small, so this part didn't require much effort.

D. Sequence Diagrams

Sequence diagrams express the interactions and the data flow between different objects within an application. In PL/SQL, these interactions are most often representations of procedure and function calls. While procedure and function calls can be extracted from the source code during parsing, the data flow, although not always needed, can only be obtained by dynamic analysis of the application. Sequence diagrams are needed to augment the activity diagrams, but they have a limited role in this kind of application.

Case study: The lists of all different sources, e.g., dynamic pages, stored procedures and functions, etc., obtained while producing the activity diagrams have been used.

Thus by analyzing this information we can produce appropriate sequence diagrams. It is important to observe that the sequential diagram usualy presents comunications between different actors or objects, but this is not the case in our situation. Furthermore, objects should be substituted with diverse PL/SQL packages for suitable presentation of the diagrams [21].

By parsing programming objects more data can be obtained (or at least cross checked): dynamic pages, stored procedures and functions have a clearly defined way of passing parameters which are the essence of this diagrams. Reports, triggers and JavaScript code can be processed similarly. In practice parsing for this part has not been done as a separate step, but it was performed together with previous parsing while generating activity diagrams. An example of a very simple sequence diagram is presented in Figure 6. Unfortunately, no automatization can be used at this point and such a sequence diagram is in fact an adaptation of parsing results performed by a developer.

The dynamic analysis for upgrading sequential diagrams should be performed by executing different scenarios on live system, collecting data and defining diagrams through analysis of those data. Careful preparation of data is needed, and specific approaches from the field of software testing are applicable here. This part is left to be done in the future.

V. Lessons Learnt

Once e-Študent application was developed and deployed, it proved to be successful. However,

1) PL/SQL proves to be far too low-level formalism any automatic transformation into the PIM could be successful [22]. Even if activity diagrams are generated automatically, they are too detailed and include too many past design elements (including all bad solutions that should not propagate to newer versions).

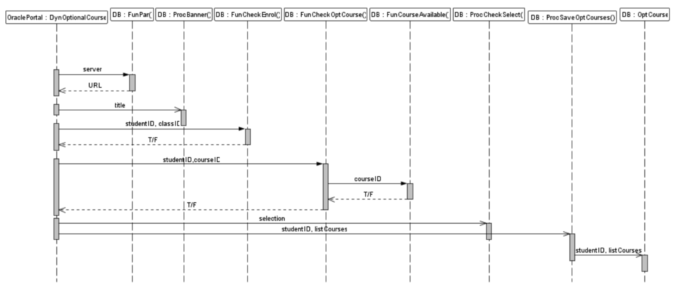

Fig. 6. An example of a simple sequence diagram depicting the sequence of calls of various stored procedures and functions from a dynamic page.

2) As it turns out, using the contemporary tools the reverse engineering of PL/SQL code must be done by someone who is pretty familiar with the design of the system being reverse engineered. At the end the reverse engineering turns into a more or less manual translation of PL/SQL code into a model on a significantly higher level of abstraction. The motivation is as follows:
    a) 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.
    b) 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 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, the 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 as reimplementing the application might end in the same situation in a few years time.

VI. Conclusion

Instead of yet another paper describing a procedure of a reverse engineering for producing different models, we concentrated on an actual case study. Furthermore, some sources, e.g., [23], describe what models are to be made without giving any hints about how this models could be made. We find these approaches inadequate as the problems are not with the automatic model generation but in understanding the generated models (which must afterwards be processed manually).

At present, the detailed eŠtudent model is still being compiled. In the future, the dynamic part of the reverse engineering procedure presented in this paper must be refined and extended further.

References

[1] R. Akkiraju, T. Mitra and U. Thulasiram, "Reverse Engineering Platform Independent Models from Business Software Applications", chapter 4 in *Reverse Engineering - Recent Advances and Applications*, InTech Press, 2012, pp. 83–94.

[2] R. C. Martin, *Agile Software Development, Principles, Patterns, and Practices*, Prentice Hall, Englewood Cliffs, NJ, USA; 2002.

[3] Manifesto for Agile Software Development, http://agilemanifesto.org (retrieved April 17th, 2012).

[4] S. Rugaber, K. Stirewalt, "Model-driven reverse engineering", *IEEE Software*, vol. 21, 2004, pp. 45–53.

[5] W. Dzidek, E. Arisholm, and L. C. Briand, "A realistic empirical evaluation of the costs and benefits of UML in software maintenance", *IEEE Transactions on Software Engineering*, vol. 34, 2008, pp. 407–432.

[6] S. J. Mellor, K. Scott, A. Uhl, D. Weise, *MDA distilled - principles of model-driven architecture*, Addison-Wesley, Boston, MA, USA, 2004.

[7] L. Favre, "Formalizing MDA-based reverse engineering processes", *Proceedings of the 6th International Conference on Software Engineering Research, Management and Applications (SERA'08)*, Prague, Czech Republic, 2008, pp. 153–160.

[8] White paper: Architecture-Driven Modernization: Transforming the Enterprise, http://www.omg.org/cgi-bin/doc?admtf/2007-12-01.pdf (retrieved June 29th, 2012).

[9] V. Mahnič, S. Drnovšček, "Introducing agile methods in the development of university information systems", *Proceedings of the 12th International Conference of European University Information Systems (EUNIS 2006)*, Tartu, Estonia, 2006, pp. 61–68.

[10] V. Mahnič, I. Rožanc, M. Poženel, "Using e-business technology in a student records information system", *Proceedings of the 7th WSEAS International Conference on E-Activities (E-Activities'08)*, Cairo, Egypt, 2008, pp. 589–594.

[11] P. Bulić et. al., "The professional study program 'Computer and Information Science"' (in Slovene), University of Ljubljana, Faculty of Computer and Information Science, Ljubljana, Slovenia, 2008.

[12] P.-A. Muller, P. Studer, F. Fondement, J. Bezivin, "Platform independent Web application modeling and development with Netsilon", *Journal of Software & System Modeling*, vol. 4, 2005, pp. 424–442.

[13] F. Solms, D. Loubser, "Generating MDA's platform independent model using URDAD", *Knowledge-Based Systems*, vol. 22, 2009, 174–185.

[14] R. Milles, K. Hamilton, *Learning UML 2.0*, O'Reilly Media, Sebastopol, CA, USA, 2006.

[15] Entrionics UML Modelling for SQL, http://www.entrionics.com (retrieved May 7th, 2012).

[16] Altova UModel 2012, http://www.altova.com/umodel.html (retrieved May 7th, 2012).

[17] B. Lieberman, "UML activity diagrams: versatile roadmaps for understanding system behaviour", *Rational Edge*, 2001, pp. 12.

[18] Y. Zou, T. C. Lau, K. Kontogiannis, T. Tong, and R. McKegney, "Model-driven business process recovery", *Proceedings of the 11th Working Conference on Reverse Engineering (WCRE'04)*, Delft, The Netherlands, 2004, pp. 224–233.

[19] Y. Zou, J. Guo, K. C. Foo, and M. Hung, "Recovering business processes from business applications", *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 21, 2009, pp. 315–348.

[20] E. Song, S. Yin, I. Ray, "Using UML to model relational database operations", *Computer Standards and Interfaces*, vol. 29, 2007, pp. 343-354.

[21] IBM, Rational Rose Forum, Sequence Diagrams for Stored Procedures, https://www.ibm.com/developerworks/forums/thread.jspa?messageID=2-907020&#2907020 (retrieved May 7th, 2012).

[22] A. Billig, S. Busse. A. Leicher, J. G. Süss, "Platform independent model transformation based on triple", *Proceddings of the 5th ACM/IFIP/USENIX International Conference on Middleware (Middleware'04)*, New York, NY, USA, 2004, pp. 493–511.

[23] W. Raghupathi and A. Umar, "Exploring a model-driven architecture (MDA) approach to health care information systems development", *Int. J. of Medical Informatics*, vol. 77, 2008, pp. 305–314.