MODEL-DRIVEN RENEWAL OF LAN-BASED BUSINESS INFORMATION SYSTEMS TOWARDS WEB-BASED SYSTEMS

S. K. Petsios
Unit of Medical Technology and Intelligent Information Systems
Dept. of Computer Science, University of Ioannina, GR 451 10 Ioannina, Greece
stefanos@cs.uoi.gr

D. I. Fotiadis
Unit of Medical Technology and Intelligent Information Systems
Dept. of Computer Science, University of Ioannina, GR 451 10 Ioannina, Greece
fotiadis@cs.uoi.gr

A. Zarras
Dept. of Computer Science, University of Ioannina, GR 451 10 Ioannina, Greece
zarras@cs.uoi.gr

ABSTRACT

A problem that we face today with existing LAN-based business information systems (BISs) is making them accessible over the WEB. In this paper, we propose a methodology and adequate tools that automate the renewal of LAN-based BISs into WEB-based ones. The proposed methodology focuses in cases with lack of documentation related to the architecture of the LAN-based BISs. Such cases are very often in practice, given that the availability of middleware and COTS related CASE tools highly tempts passing from the requirements acquisition phase, directly to the BIS implementation phase.

KEYWORDS

CASE tools, COTS, Model-Driven Architecture Development, Reengineering.

1. INTRODUCTION

A problem that we frequently face today with existing business information systems (BISs) is to make them accessible over the WEB. Legacy BISs, were built using COTS like Borland Delphi, Borland C-Builder, Borland J-Builder, Microsoft Visual Basic, etc. and LAN-based middleware platforms like CORBA[1], DCOM[2], etc (hereafter, we use the term *technology platforms* to refer to both COTS and middleware platforms used). The aforementioned technology platforms offer reusable software components that can be employed for building LAN-based BISs. However, they do not provide adequate support for making those BISs available over the WEB.

The development of a BIS involves the collection and analysis of corresponding business requirements. The business requirements are divided into functional ones, describing the specific business processes that should be incarnated by the BIS, and non-functional ones, referring to more general issues like performance, reliability, availability, etc. The requirements acquisition and analysis workflow is followed by the design of a model, which describes the BIS architecture. In a first step, this model is platform-independent, *i.e.* it does not include information related to the specific technology platforms used for building the BIS. The selection

[1] http://www.omg.org /technology/documents/formal/corba_iiop.htm
[2] http://msdn.microsoft.com/library/default.asp?url=/library/enus/cossdk/htm/ pgservices_events_5x4j.asp

of such platforms actually follows the specification of the Platform Independent Model (PIM). The step following the selection of specific technology platforms consists of refining the PIM of the BIS accordingly. Refining the PIM amounts in mapping the different types of elements that constitute it into corresponding platform-specific types of elements. The result of this step is called a Platform Specific Model (PSM). The overall BIS development process concludes with implementing the system based on the PSM resulting from the refinement of the PIM. The implementation workflow relies on the use of platform-specific CASE tools, *i.e.* graphical front-end tools that facilitate the selection and combination of appropriate COTS and middleware components, in a plug-and-play fashion. The use of platform-specific CASE tools greatly facilitates the BIS development process. It promotes software reuse and allows balancing the trade-off between the quality of the BIS and the time-to-market. However, the provided ability to reduce the time-to-develop the BIS is too tempting for the project managers, the designers, and the developers, involved in the BIS development process. More specifically, the aforementioned ability provokes passing from the requirements acquisition and analysis workflow directly to the implementation workflow in favor of minimizing the development time and the required resources. The negative impact of such a simplification is immediately visible when we face the problem of renewing a LAN-based BIS towards a corresponding WEB-based BIS. The PSM prescribing the architecture of the LAN-based BIS must be extracted from the source code of the system, which in most cases is quite large and complex to perceive. Then, we have to abstract away from the PSM details related to the technology platforms used for building the LAN-based BIS. The aforementioned task results in the PIM of the BIS, which further serves for producing a new PSM describing the realization of the BIS, based on functionality provided by newly selected COTS and middleware platforms, which allow accessing the BIS over the WEB.

In this paper, we propose a methodology and adequate tools that automate the aforementioned renewal process. In Section 2, we discuss work related to the general problem of reengineering systems in the lack of documentation that details their architecture. Based on this discussion we derive a number of requirements for tools that automate the reengineering process in the particular case that we face here. Section 3 details the methodology and tools we propose for the renewal of LAN-based BISs, towards corresponding WEB-based BISs. Section 4 presents experimental results from a real world case study where we applied the proposed methodology. Finally, Section 5 summarizes the major contributions of this work and points-out the future directions.

2. BACKGROUND AND RELATED WORK

The renewal of a LAN-based BIS towards a WEB-based system is a particular case of legacy system migration. Reengineering a legacy system is a challenging task raised long time ago. The basic tasks that constitute this process comprise [Jacobson, I. and Lindstrom, F., 1991]: *(1) Reverse engineering* the legacy system architecture, *i.e.* create a high-level model describing the structure and the behavior of the legacy BIS. *(2) Reasoning* about a change in the legacy BIS architecture. *(3) Forward engineering* the new architecture, *i.e. perform* the change by re-implementing the whole, or parts of the legacy BIS.

According to [Jacobson, I. and Lindstrom, F., 1991], there are three different categories of reengineering scenarios frequently appearing in practice. In the first category, we aim at completely changing the implementation technique that was used in the legacy system. More specifically, we have cases of legacy systems built using old programming paradigms (e.g. procedural) and languages (C, COBOL, FORTRAN). The ultimate goal is to re-implement the legacies according to the object-oriented paradigm and programming languages (e.g. C++, Java, etc.) that conform to this paradigm. Several different techniques have been proposed to deal with the aforementioned cases. Most of them concentrate on finding classes in legacy code by grouping together procedures and global variables. The grouping typically relies on matching the types of the global variables with the types of the parameters used in the procedures [Liu, S., S. and Wilde, L., 1990], [Schwanke, R., W., 1991], [Ong, C. and Tsai, W., T., 1993], [Canfora, G. et al., 1996]. The previous approaches are mainly targeted to systems built using strongly-typed languages (e.g. PASCAL, C and FORTRAN). However, these approaches do not support the migration of systems built using languages like COBOL [Cimitile, A. et al., 1999]. An approach that deals with the previous is detailed in [Newcomb, P. and Kotik, G., 1995]. This approach aims at grouping COBOL records and portions of source code which manipulate these records. A group that consists of a record and corresponding portions of source code

constitutes a class. A similar approach is presented in [De Lucia, A. et al., 1997] for the case of RPG programs. Finally, in [Van Deursen, A. and Kuipers, T., 1999] the authors propose a methodology for migrating COBOL systems to object-oriented ones. The novelty in this case is that the authors try to split up COBOL records into smaller pieces to improve the grouping of the COBOL source code. The authors further compare two alternative grouping techniques, clustering and concept analysis.

The scenarios falling in the first category are the ones that were the most frequent so far. However, in the near future legacy BISs built using languages like COBOL, C and FORTRAN are going to be rare. Most of the BISs built nowadays rely on the object-oriented paradigm and technology platforms that promote this paradigm. Hence, the future cases of reengineering fall in the second of the categories identified by [Jacobson, I. and Lindstrom, F., 1991]. In this category, we aim at partially changing the implementation technique used in the legacy system. More specifically, the ultimate goal nowadays and in the near future is to migrate legacy systems relying on the object-oriented paradigm and conventional technology platforms like CORBA, DCOM, etc., into systems still relying on the same paradigm and are built based on more powerful and flexible technologies like CCM[3], .NET[4], J2EE[5], JSP[6], PHP[7], etc. The renewal of LAN-based BISs is exactly such a case. Currently, there is not much work or experience on reengineering scenarios, falling in the second category. In particular, in [Cordy, R., J., et al.] the authors present an approach for migrating PERL systems to JSP/Java systems. Similarly, in [Ping, Y., et al, 2003] a technique is proposed for reengineering systems relying on the IBM Net.Data COTS to systems based on JSP. In both such cases, the authors rely on COTS-specific tools to reverse-engineer legacy systems. Moreover, they forward-engineer the new systems, using JSP-specific code generation tools.

As opposed to the aforementioned approaches, our methodology and tools are general purpose. More specifically, our methodology and tools are as independent as possible from the technology platforms used in both the legacy and the target systems. To achieve it we rely on platform-specific reverse-engineering patterns serving as input to our platform-independent reverse-engineering tools. The tools seek for those patterns in the legacy code and extract the overall architecture of the legacy. Moreover, we use platform-specific refinement and code generation patterns as input to our platform-independent forward-engineering tools. Based on these patterns, the tools generate platform-specific models of the target system and platform-dependent source code.

3. METHODOLOGY AND TOOLS

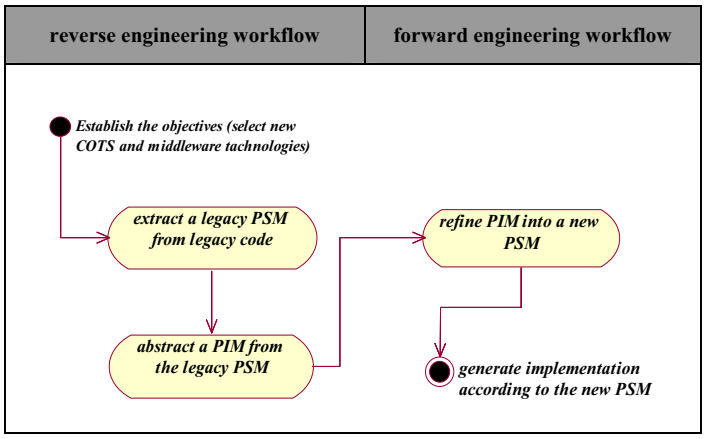
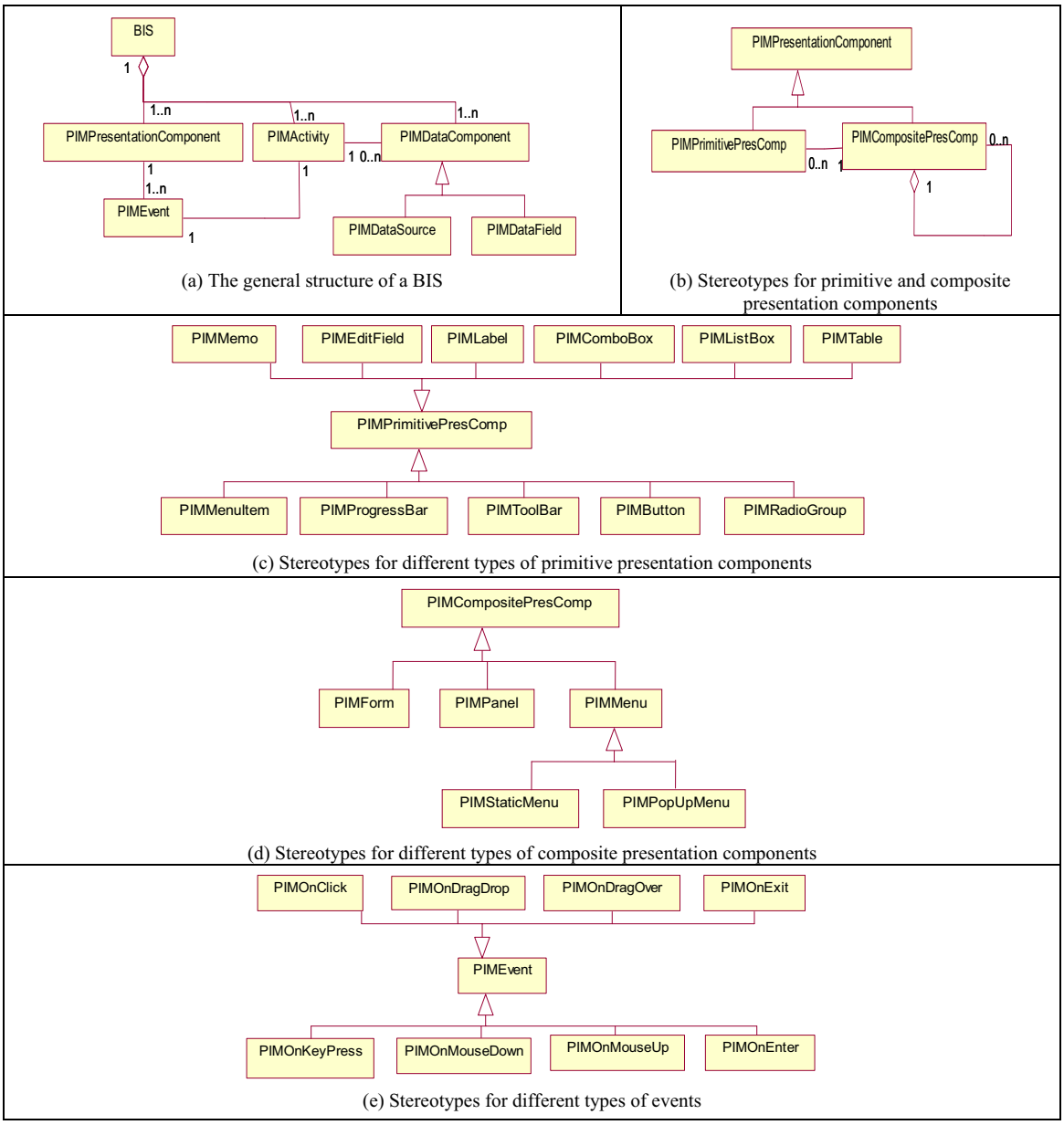


Figure 1. The renewal process of LAN-based BISs into WEB-based BISs

[3] http://www.omg.org/technology/documents /formal/components.htm
[4] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/coss dk/htm/pgcontexts_1p0z.asp
[5] http://java.sun.com/j2ee/
[6] http://java.sun.com/products/jsp/
[7] http://www.php.net/

The overall process we propose for renewing a LAN-based BIS into a corresponding WEB-based BIS relies on a general-purpose Model-Driven Architecture (MDA) development process, which was recently proposed by OMG [Soley R. and the OMG Staff Strategy Group, 2000]. Our renewal process is an instance of the MDA process. It consists of a *reverse engineering* and a *forward engineering* workflow (Figure 1). The reverse engineering workflow starts with the establishment of the objectives of the overall renewal process. Roughly, during this initial activity, the new technology platforms that are going to be used are selected. Then, we extract from the source code of the LAN-based BIS a PSM, which details the basic elements that constitute the BIS and the way those elements are implemented using functionality provided by the technology platforms that we have to change. Next, we abstract away from the PSM details that refer to the old technology platforms. The outcome of the previous activity is the PIM of the BIS. The aforementioned PIM serves as input to the forward engineering workflow, which comprises refining the PIM into a PSM describing the realization of the BIS, using the new technology platforms. Finally, the new PSM is used to generate a partial implementation of the BIS.



(a) The general structure of a BIS

(b) Stereotypes for primitive and composite presentation components

(c) Stereotypes for different types of primitive presentation components

(d) Stereotypes for different types of composite presentation components

(e) Stereotypes for different types of events

Figure 2. A UML Profile for the specification of BIS PIMs

In order to specify the main artifacts (*i.e.* the legacy PSM, the PIM and the new PSM), which result from the activities constituting the reverse and the forward engineering workflows, we use UML[8]. UML is an emerging standard modeling notation providing basic modeling constructs, which allow the specification of the structure and the behavior of software. However, the semantics of the basic UML modeling constructs are quite generic. The previous is reasonable since UML is becoming a base for the development of a family of notations, called UML profiles, which serve different modeling purposes[8]. A UML profile comprises the definition of a number of *stereotypes*. A stereotype consists of a set of *constraints* and a set of *properties* that enhance the definition of a standard class of UML model elements (a UML meta-model element). Applying the stereotype on a particular UML model element of the class (an instance of the UML meta-model element) implies that the element conforms to the enhanced definition instead of the standard one.

Hence, to enable the specification of PIMs we define a platform-independent representation (PIR)[9], consisting of stereotypes, which correspond to different types of elements typically used to build a BIS. More specifically, a BIS consists of a *presentation tier*, a *business tier*, and a *data tier*. The *presentation tier* comprises *presentation components* (specified using the PIMPresentationComponent stereotype), allowing the users to interact with the BIS. A presentation component may be *primitive* (e.g. button, text-box, etc.), or *composite* (e.g. form, panel, menu, etc.), specified using the PIMPrimitivePresComp and the PIMCompositePresComp stereotypes, respectively. Each particular type of presentation components is associated with different types of *events* (specified using the PIMEvent stereotype) generated by the users. Upon the occurrence of an event, a corresponding *event-handler* performs a particular *activity* (specified using the PIMActivity stereotype), which manipulates data previously provided by the user as input to the presentation component. The activities executing upon the occurrence of events constitute the *business tier* of the BIS. An activity may possibly access a database using components (specified using specializations of the PIMDataComponent stereotype) directly associated with certain database elements. The aforementioned components form the *data tier* of the BIS. Figure 2 gives an overview of the PIR[10].

Similarly, to enable the specification of PSMs, we define platform-specific representations (PSRs), consisting of stereotypes, which correspond to the different types of platform-specific elements that can be used for the realization of corresponding types of PIM elements. A BIS built using Borland Delphi and DCOM, for instance, consists of primitive presentation components (e.g. DelphiButton, DelphiTextBox, etc.) and containers (e.g. DelphiForm, DelphiPanel, DelphiGroupBox, etc.) that are associated with different types of DelphiEvent elements (e.g. by pushing an instance of a DelphiButton component the user generates an instance of the DelphiOnClick event). Each type of DelphiEvent elements is associated with a Delphi EventHandler element, which executes on the occurrence of the corresponding runtime event and performs a Delphi procedure. The procedure manipulates the data provided by the user of the BIS and may further access a database, using ADOTable, ADOQuery, ADODataSet components etc.

The basic concepts of the tools proposed for accomplishing the activities of the BIS renewal process are detailed in the sequel. As we have discussed, the main requirement for these tools is to keep them *as independent as possible* from both the legacy and the new technology platforms used.

3.1 Reverse Engineering Workflow

Extracting the legacy PSM from the source code of the LAN-based BIS involves using a tool accepting as input a description of the particular PSR, defined based on the technology platform used for the BIS. Each PSR stereotype X is associated with a *reverse engineering pattern*. This pattern specifies syntactical source code conventions, used in the implementation of X stereotyped architectural elements. The tool parses the source code of the LAN-based BIS and tracks-down presentation components, events, activities and data components, following the provided patterns. For every such component, a corresponding UML element is created in the legacy PSM. The element is characterized by the stereotype associated with the pattern used to discover the element. The tool further creates UML associations between presentation components and

[8] e.g. Real-time systems, Enterprise Distributed Computing (EDOC) systems, etc.
http://www.omg.org/technology/ documents/modeling_spec _catalog. htm

[9] We use the term "representation" in place of the term "profile", whenever it is necessary to distinguish between proposed UML extensions and extensions that are actually adopted by OMG.

[10] A detailed specification of the stereotypes (*i.e.* base classes, properties, and constraints) is avoided due to space limitations.

related events and activities. Moreover, the tool creates UML aggregation relations between the composite presentation components of the PSM and their constituents.

Abstracting a PIM from the resulting legacy PSM involves using a tool, which accepts as input the legacy PSM and an *abstraction pattern*, consisting of a set of *abstraction dependencies* (Figure 3). Each one of them maps a PSR stereotype X into a PIR stereotype Y. An abstraction dependency states that a particular type of X stereotyped PSM elements was used for the realization of Y stereotyped PIM elements (Figure 4(a)). Furthermore, an abstraction dependency may be associated with a set of constraints, which state that the realization of Y stereotyped elements also involved associating the corresponding X stereotyped PSM elements with other PSM elements (e.g. see the dl2pimDBGrid and the dl2pimDBCombo dependencies in Figure 4(a)). Based on such a refinement dependency, the tool seeks X stereotyped elements in the PSM. For every one of them it creates the corresponding Y stereotyped element in the PIM of the BIS.

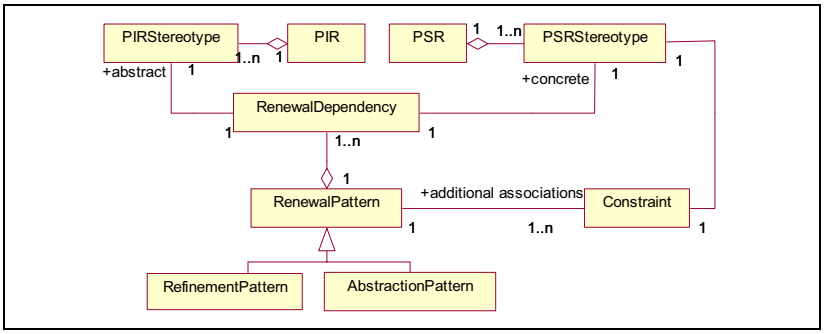
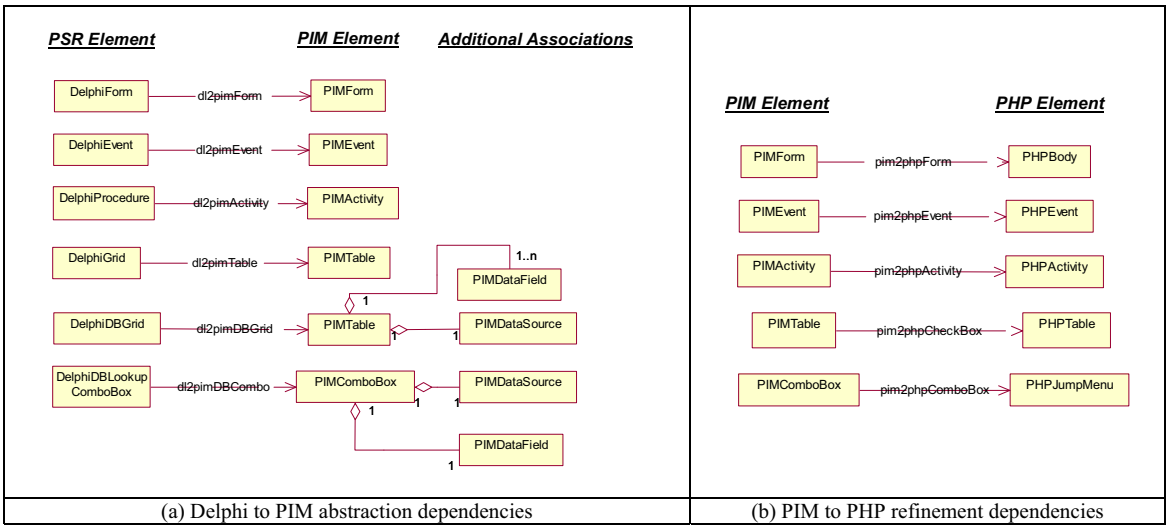


Figure 3. The structure of abstraction and refinement patterns

3.2 Forward Engineering Workflow

In order to refine the PIM resulting from the reverse engineering workflow into a new PSM relying on the new technology platform we have selected at the beginning of the renewal process, we use a tool that is similar with the one that performs the abstraction activity (see Section 3.1).



(a) Delphi to PIM abstraction dependencies

(b) PIM to PHP refinement dependencies

Figure 4. Examples of abstraction and refinement dependencies

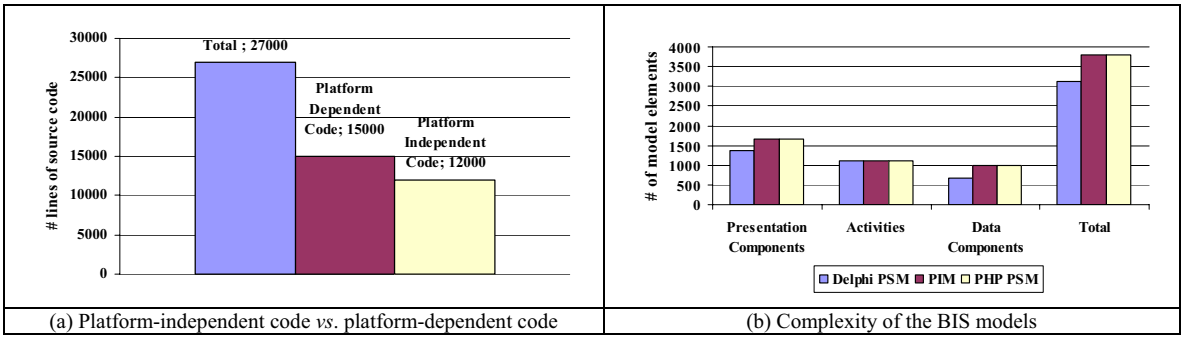
More specifically, the tool accepts as input the PIM and a *refinement pattern*, which comprises a set of *refinement dependencies* (Figure 3). Each dependency corresponds to a particular type of stereotyped PIM elements and describes how to realize this type of elements using platform-specific elements (Figure 4(b)). Hence, a refinement dependency maps a PIR stereotype X into a PSR stereotype Y. Moreover, the refinement

dependency may be associated with additional *constraints*, which state that the Y stereotyped elements must be further associated with other platform-specific elements. Given such a dependency, the tool seeks in the PIM of the BIS for X stereotyped elements. For each one of them it creates a corresponding Y stereotyped element in the new PSM. Based on the additional constraints that may be associated with the refinement dependency, the tool creates further PSM elements and associates them with the Y stereotyped element as prescribed by the constraints.

Generating parts of the new BIS implementation involves using a tool, which accepts as input the new PSM of the BIS and a *code generation pattern*. The code generation pattern is divided into different parts. Each part corresponds to a particular type of Y stereotyped PSM elements and describes the skeleton code that must be generated for this type of elements. The code generation tool parses the PSM model and for every Y stereotyped PSM element it generates a specific skeleton code, based on the aforementioned part of the code generation pattern.

4. CASE STUDY

The methodology and the early prototypes of the tools were used towards the renewal of a real-world LAN-based BIS, which is currently used by a well-known Greek bank. The business purpose of the BIS is to support the loan applications of the bank's customers and to evaluate the customers' projects. The legacy BIS has been developed by the Unit of Medical Technology & Intelligent Information Systems[11] at the University of Ioannina. Regarding the architecture of the legacy BIS we had almost no documentation and few information was gathered by personal communication with some of the remaining members of the development team that still work in the Unit. The development team actually proceeded directly to the implementation of the legacy BIS, which relies on Borland Delphi, DCOM and a MySQL database[12]. The main objective of the renewal process was to migrate the LAN-based BIS into a WEB-based BIS that relies on PHP and JavaScript. The overall size of the LAN-based BIS is 27.000 lines of source code.



(a) Platform-independent code *vs*. platform-dependent code

(b) Complexity of the BIS models

Figure 5. Experimental results from the renewal of the case study BIS

During the reverse engineering workflow, we extracted the Dephi PSM of the BIS, which includes 3.138 elements. 35% of the elements are business activities, 21% of the elements are data components and 44% of the elements are presentation components. Figure 5(a) gives the percentage of the platform-independent source code realizing the actual business process of the legacy BIS. During the extraction of the PSM we further isolate the aforementioned source code from the platform-dependent source code which is generated automatically by Borland Delphi. The platform-independent code can be reused in the Web-based BIS with small modifications. After the extraction of the Delphi PSM, we abstracted the PIM of the legacy BIS. In Figure 5(b), we compare the complexity, the number of presentation components, activities and data components, of the PIM with the complexity of the Delphi PSM. An interesting remark is that the PIM model is more complex as it contains more presentation and data components. This is because Borland Delphi provides ready-to-use presentation components (e.g. the DelphiDBLookupComboBox and DelphiDBGrid, elements in Figure 4(a)), which allow the users to insert (resp. retrieve) data to (resp. from) the database.

[11] http://medlab.cs.uoi.gr/
[12] http://www.mysql.com/

Such types of composite presentation components are not available in other COTS like PHP, JSP, etc. Consequently the PIR, we assume for specifying PIMs, does not provide such types of components. The Delphi components providing direct access to the database consist of primitive Delphi presentation and data components. For this reason, we map in the PIM these Delphi components into corresponding combinations of PIM presentation and data components (see the dl2pimDBGrid and dl2pimDBCombo dependencies in Figure 4(a)). In Figure 5(b), we further compare the complexity of the PHP PSM we generate, during the forward engineering workflow, with the complexity of the PIM and the complexity of the Delphi PSM. The PHP and the PIM are of the same complexity since the mapping of the PIM elements into corresponding PHP elements is one-to-one (see the refinement dependencies in Figure 4(b)).

5. CONCLUSIONS

In this paper, we examined the general problem of migrating legacy BISs that were built using technology platforms not permitting the use of the BISs over the WEB, into WEB-based BISs. The major contributions of this paper are a methodology and adequate tools for the aforementioned migration. The proposed methodology comprises a reverse engineering workflow that aims at the automated extraction of a platform-independent architectural model of the LAN-based BIS from the source code of the BIS. The architectural model is automatically refined according to a new technology platform, which we select at the very beginning of the renewal process. The resulting platform-specific architectural model serves for generating a partial implementation of the WEB-based BIS. The methodology and the tools proposed in this paper are independent from both the legacy and the new technology platforms. This is a distinctive feature of our approach compared to other works proposed in the literature.

REFERENCES

Canfora, G. et al., 1996. An Improved Algorithm for Identifying Objects in Code. *In Software Practice and Experience,* Vol. 26, No. 1, pp. 25-48.

Cimitile, A. et al., 1999. Identifying Objects in Legacy Systems Using Design Metrics. *In Journal of Systems and Software,* Vol. 44, No. 3, pp. 199-211.

Cordy, R., J. et al. The Whole Website Understanding Project. http://www.cs.queensu.ca/~stl/stg/

De Lucia, A. et al., 1997. Migrating Legacy Systems Towards Object-Oriented Platforms. *In Proceedings if the IEEE Conference on Software Maintenance (ICSM'97),* pp. 222-229.

Jacobson, J. and Lindstrom, F., 1991. Reengineering of Old Systems to an Object-Oriented Architecture. *In Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages, and Applications(OOPSLA'91),* pp. 340-350.

Liu, S., S. and Wilde, N., 1990. Identifying Objects in a Conventional Procedural Language: An Example of Data Design Recovery. *In Proceedings if the IEEE Conference on Software Maintenance (ICSM'90),* pp. 266-271.

Newcomb, P. and Kotik, G., 1995. Reengineering Procedural into Object-Oriented Systems. *In Proceedings of the 2nd IEEE Working Conference on Reverse Engineering (WCRE'95),* pp. 237-249.

Ong, C. and Tsai, W., T., 1993. Class and Object Extraction from Imperative Code. *In Journal of Object Oriented Programming,* Vol. 6, No. 1, pp. 58-68.

Ping, Y. et al., 2003. Migration of Legacy Web Applications to Enterprise Java Environments – Net.Data to JSP Transformation. *In Proceedings of the ACM Conference Centre for Advanced Studies Conference on Collaborative Research,* pp. 223-237.

Soley R. and the OMG Staff Strategy Group, 2000, Model-Driven Architecture. *White Paper, Object Management Group*.

Schwanke, R., W., 1991. An Intelligent Tool for Reengineering Software Modularity. *In Proceedings of the 13th IEEE-ACM-SIGSOFT International Conference on Software Engineering (ICSE'91),* pp. 83-92.

Van Deursen, A. and Kuipers, T., 1999. Identifying Objects Using Cluster and Concept Analysis. *In Proceedings of the 21st IEEE-ACM-SIGSOFT International Conference on Software Engineering (ICSE'99),* pp. 246-255.