KnowDIME: An Infrastructure based on Architecture-Driven Modernization for Improving Legacy System

Rafael S. Durelli[1,3], **Márcio E. Delamaro**[1] **and Valter V. de Camargo**[2]

[1]Computer Systems Department University of São Paulo
São Carlos, SP, Brazil.

[2]Computing Departament
Federal University of São Carlos (UFSCAR)
São Carlos, SP, Brazil.

[3]RMoD Team, INRIA, Lille, France

{rdurelli, delamaro}@icmc.usp.br[1], valter@dc.ufscar.br[2]

Abstract. *Legacy systems mainly consist of two kinds of artifacts: source code and databases. Usually, the maintenance of those artifacts is carried out through refactoring processes in isolated manners and without following any kind of standardization. In order to provide a more effective maintenance of the whole system both artifacts should be analyzed, evolved jointly and standardized way. As is known refactoring is a very useful process for dealing with software aging problem since it can improve maintainability and reusability of these system. Recently, researches have been shifted from the typical refactoring processes to the so-called Architecture-Driven Modernization (ADM), which is an approach that follows the all the principles of Model-Driven Development (MDD). As one of the challenges of Software Engineering focuses on mechanisms to support the automation of software refactoring process in this paper we put forward KnowDIME to assist the modernization of a legacy system based on ADM, which uses the Knowledge Discovery Metamodel (KDM) standard. This infrastructure analyses Structured Query Language (SQL) queries embedded in a legacy source code in order to restructure and re-organize the system by using design patterns, such as Data Access Object (DAO) and Service-Oriented Architecture (SOA).*

1. Introduction

Companies have a vast number of operational legacy systems and these systems are not immune to software aging problems. However, they can not be discarded because they incorporate a lot of embodied knowledge due to years of maintenance and this constitutes a significant corporate asset. Furthermore, these kind of systems mainly consist of two artifacts: source code and databases. Usually, the maintenance of those artifacts is carried out through restructuring processes in isolated manners [**?**]. Nevertheless, we claim that for a more effective maintenance of the whole system both should be analyzed and evolved jointly.

Refactoring can be used in order to deal with the maintenance of these systems [**?**]. Empirical studies of refactoring have shown that it can improve maintainability [**?**] and reusability [**?**] of legacy system. Furthermore, not only does existing work suggest that

refactoring is useful, but it also suggests that refactoring is a frequent practice [**?**]. Cherubini and colleagues' survey indicates that developers rate the importance of refactoring as equal to or greater than that of understanding code and producing documentation [**?**].

In a parallel research line, researchers have been shifted from the typical refactoring process to the so-called Architecture-Driven Modernization (ADM) [**?**]. ADM has been proposed by OMG (Object Management Group) and advocates conducting reengineering processes following the principles of Model-Driven Development (MDD) [**?**], i.e., it treats all the software artifacts involved in the legacy system as models and can establish transformations among them. Firstly a reverse engineering is performed starting from the source code and a model instance (**P**lataform **S**pecific **M**odel - PSM) is created. Next successive refinements are applied to this model up to reach a good abstraction level (PSM) in model called KDM (**K**nowledge **D**iscovery **M**etamodel). Upon this model, several refactorings, optimizations and modifications can be performed to solve problems found in the legacy system. Secondly a forward engineering is carried out and the source code of the modernized target system is generated again. According to the OMG the most important artifact provided by ADM is the KDM metamodel, which is a multipurpose standard metamodel that represents all aspects of the existing information technology architectures. The KDM is divided into four layers representing both physical and logical software assets of information systems at several abstraction levels. Each layer is further organized into packages. Each package defines a set of meta-model elements whose purpose is to represent a certain independent facet of knowledge related to existing legacy systems. However, in this paper we are only interested in both Program Elements layer and Runtime Resources layer, used to represent a language-independent intermediate representation for programming languages and useful to represent information such as database, respectively.

One of the challenges of Software Engineering focuses on mechanisms to support the automation of software reengineering process, aiming to reduce time and effort spent in this process. Motivated by this challenge, we put forward an infrastructure to assist the reengineering of a legacy system based on ADM, which uses the Knowledge Discovery Metamodel (KDM) standard. This paper is organized as followed: Section 2 provides information related to the infrastructure - Section 3 the architecture of the infrastructure is depicted - in Section 4 there are related works and in Section 5 we conclude the paper with some remarks and future directions.

2. KnowDIME

In this section **Know**ledge **D**iscovery **I**ntegrated **M**odernization **E**nvironment (KnowDIME) is presented. The theory behind it is based on the horseshoe modernization model, which is depicts in Figure 1. As can be seen, this model consist of three steps, (*i*) **Reverse Engineering**, (*ii*) **Restructuring**, and (*iii*) **Forward Engineering**. The first step is represented by the left side of the horseshoe, it analyzes the legacy system in order to identify the components of the system and their interrelationships. Usually the reverse engineering step builds one or more representations of the legacy system at a higher level of abstraction (PSM). The second one is represented by the curve of the horseshoe since this steps takes the previous system's representation (PSM) and transforms it into another one (**P**lataform **I**ndependent **M**odel - PIM) at the same abstraction level. Finally, the last step is represented by the right side of the horseshoe

because it generates physical implementations (source-code) of the target system at a low abstraction level from the previously restructured representation of the system.

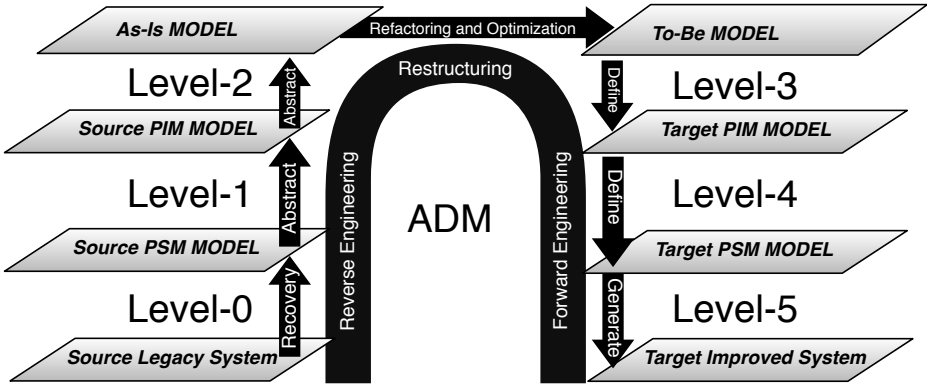

Figure 1. Horseshoe modernization model.

It worth noticing that KDM is the most essential part of our infrastructure. KDM supplies the representation and management of knowledge extracted by means of reverse engineering from all the different software artifacts of the legacy system. Therefore, the legacy knowledge obtained is then modernized into a target improved system by using the concept of MDD. As the infrastructure herein follows the three steps depicts in Figure 1 then it has to go through six abstraction levels with five transformations among them. In Figure 2 is depicted the infrastructure and also illustrates all abstract levels and the transformations (see the letters 'A'- 'F'). The Section 2.1 describes these abstract levels and the Section 2.2 explains the transformations.

2.1. Abstract Levels

The **Level-0** represents all artifacts of the legacy systems in the real world, such as source code and database. The infrastructure uses this artifacts as input to start the ADM process.

The **Level-1** consist of two PSM. The former model is a source code model, which represents all abstraction of the legacy system's source code, Figure 2(B) illustrates it. More specifically, this model represents an **A**bstract **S**yntax **T**ree (AST), which is a tree representation of the abstract syntactic structure of Java source code. The latter is a database model that illustrates informations related to database used in the legacy system, see Figure 2(A). Similarly, this model is also represented by an AST, the only difference is that it represents syntactic structure of database instead of Java source code. This database model is based on the SQL-92 Data Manipulation Language which can represent the SQL operations such as *Select*, *Delete*, *Update* and *Insert*.

The **Level-2** consist of a combination of the two PSMs (see **Level-1**) for creating a single 'As-Is' PIM which is based on the KDM, Figure 2(C) depicts it. This single model makes possible to represent the artifacts of the legacy systems in an integrated and technological-independent way. Afterwards, in the **Level-3**, by using this PIM the infrastructure allows the software engineer to realize a set of refactoring, refinement and optimizations in such model in order to meet new requirements.

Then in **Level-4** the infrastructure use the PIM redesigned to generate a target PSM, which is represented by a 'To-Be' **U**nified **M**odeling **L**anguage - UML model. This

model illustrates the target improved system. Finally, in the **Level-5** the source-code of the target improved system is generated by using the UML as input.

2.2. Transformation

The first transformation happens between the **Level-0** to the **Level-1**. As previously mentioned, this transformation is carried out by following a set of Text-To-Model (T2M) rules in order to obtains two PSM from legacy system i.e., source code model and database model. To transform the legacy system in these models it is necessary carry out a syntactical analyzer. As for the first model we used a parser that provides a complete self-describing representation of Java source code, i.e., this representation reflects the structure of the software artifact directly in the nesting of elements. Regarding the second model we devised a SQL parser. It consists of a extension of the previously parser but it intends to identify SQL embedded in the legacy system's source code. This parser focus on identifying SQL operations such as *Select*, *Delete*, *Update* and *Insert*; the tables, the columns, primary keys, etc, that appear in these SQL operations are then represented into a AST. Both models, the source code model and the data base model are represented in the XMI (**XM**L Metadata **I**nterchange)-based syntax as can be seen in Figure 2 (B) and Figure 2 (A), respectively.

The second transformation is **Level-1** to **Level-2**. This transformation is based on a set of Model-To-Model (M2M) patterns to obtain a 'As-Is' PIM, which is based on the KDM, see Figure 2 (C). This PIM is generated based on the PSMs from **Level-1**. The first PSM (source-code) is transformed to the KDM Code Package, which represents common program elements supported by various programming languages, such as data types, classes, procedures, and templates. The second PSM (database model) is transformed to the KDM Data Package that represents complex data repositories, such as relational databases.

The third transformation **Level-2** to **Level-3** realizes a set of M2M operations to transform the 'As-Is' PIM to a 'To-Be' PIM, i.e., model refactorings and model optimizations are realized in the PIM from **Level-2**. Therefore, a set of automated model refactorings are carried out against the PIM to generate a redesigned PIM. For instance, the software engineer can pick out if he would like to refactor the legacy system to meet either Data Access Object (DAO) or Service-Oriented Architecture (SOA) or both refactorings. As result, new meta-classes may be added to this PIM model, as well as new meta-attributes and meta-relationship in order to meet these requirements.

The fourth transformation **Level-3** to **Level-4** executes a M2M transformation taking as input the PIM and producing as output a model conforming to the KDM models into UML meta model, see Figure 2(D). Finally, the last transformation **Level-4** to **Level-5** realizes a set of Model-To-Code (M2C) generation based on design patterns, such as Data Access Object (DAO) or Service-Oriented Architecture (SOA), see Figure 2(E) and Figure 2(F). The generated code can be complemented by the software engineer in accordance with the more detailed specifications of the application business rules, such as the implementation of specific behaviors and features not covered by the code generation.

3. Architecture

In Figure 3 is depicted the architecture of our infrastructure. As shown in this figure, we devised it on the top of the Eclipse Platform and used both Java and Groovy as pro-

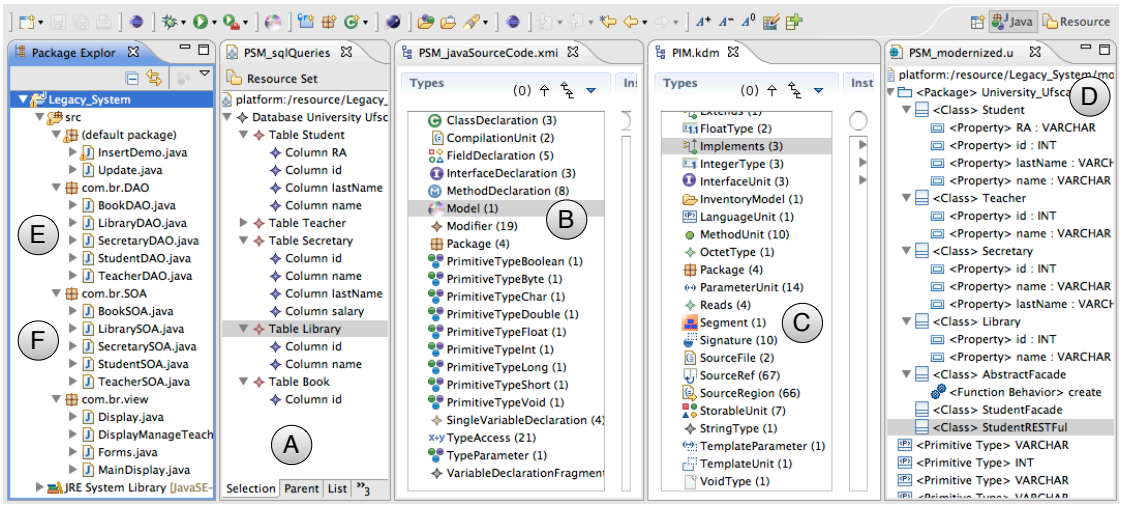

Figure 2. Screenshot of the KnowDIME

gramming language. Moreover, we used Eclipse Modeling Framework (EMF)[1] to create the SQL model and to reutilize the UML model. MoDisco is used by the infrastructure since it provides an **A**pplication **P**rogramming **I**nterface - (API) to easily access the KDM model.

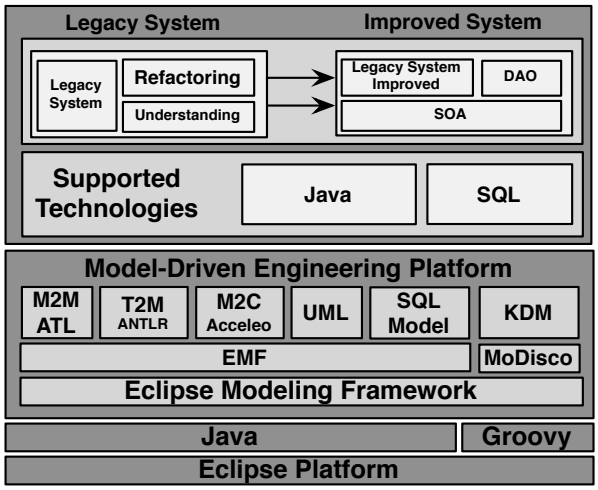

Figure 3. Architecture of KnowDIME

ANother **T**ool for **L**anguage **R**ecognition - (ANTLR) is used herein to create parsers to obtain information related to the legacy system's artifacts. Therefore, two parser were developed: (*i*) the first takes as input a Java grammar and generates as output an AST and (*ii*) the second parser is a extension of the first one to identify SQL embedded in the legacy system's source code, i.e., it takes as input a Java source-code and generates as output an AST which contains informations such as, tables, columns, primary keys, etc. Then, to transform these ASTs in PSMs we used an API provided by EMF. Afterwards,

[1]http://www.eclipse.org/modeling/emf/

all transformation M2M are done by **A**tlas **T**ransformation **L**anguage - ATL, which provides ways to produce a set of target models from a set of source models. Therefore, ATL is used to transform the PSMs to conform the KDM specification and to transform the improved KDM to an UML that represents the target systems. Then, in order to transform this last model in a set of physical artifacts (source code) Acceleo was used, which is based on textual template approach. A template can be thought of as the target text with holes for variable parts. The holes contain metacode which is run at template instantiation time to compute the variable parts. Furthermore, we have used Java Persistence API (JPA) 2.0 to deal with the way relational data is mapped to Java objects. Similarly, RESTful API have been used to implements SOA artifacts.

4. Related Work

Modernization of Legacy Web Applications into Rich Internet Applications [**?**] proposes a approach for systematic and semiautomatic modernization of legacy Web applications in rich interfaces applications. In this process, the MDD principles are applied and the use of OMG Architecture Driven Modernization (ADM) specifications for the generation of rich interfaces from the source code of navigation and presentation layers of legacy web applications. The proposed approach differs from this work by having a more general purpose and are not intended to only a single type of applications, such as Web applications.

Model-Driven Reengineering of Database [**?**] presents a process to perform relational databases reengineering. The process is conducted through of repeated model transfor- mations and divided into the stages of extraction and contextualization. In the extraction stage, a Platform Specific Model (PSM) is obtained from the database structure. In the contextualization stage, a Platform Independent Model (PIM) is generated from the PSM. As a result of the process, there are entity-relationship models and class diagrams.

5. Concluding Remarks

In this paper is presented the KnowDIME to support the refactoring of legacy systems based on ADM, which uses the KDM standard. It follows the theory of the horseshoe modernization model, which is threefold: (*i*) **Reverser Engineering**, (*ii*) **Reestructuring** and (*iii*) **Forward Engineering**.

Firstly, all the artefacts of the legacy system must be transformed into PSMs by statically analyzing the legacy source code. Still in the first step, these PSMs are integrated into a KDM model, i.e., a PIM model, through a M2M transformation implemented using ATL. Secondly, the KnowDIME applies a set of model refactorings and model optimizations in this PIM. Afterwards, KnowDIME executes a set of M2M transformation taking as input the PIM and producing as output a model conforming to the KDM models into UML meta model. Finally, an improved system is obtained from this UML by means of a set of M2C transformation; additionally, the generated code can be complemented by the software engineer in accordance with the more detailed specifications of the application business rules, such as the implementation of specific behaviors and features not covered by the code generation.

We believe that KnowDIME makes a contribution to the challenges of Software

Engineering which focuses on mechanisms to support the automation of software refactoring process. Long term future work involves conducting experiments to evaluate the level of maintenance provided by KnowDIME. It is worth highlighting that KnowDIME is open source and it can be downloaded at *www.dc.ufscar.br/∼valter/crossfire*.

6. Acknowledgements

The authors would like to thank CNPq for Processes 241028/2012-4 and FAPESP for Process 2012/05168-4.

References

Fowler, M., Beck, K., Brant, J., Opdyke, W., and Roberts, D. (2000). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley.

Griffith, I., Wahl, S., and Izurieta, C. (2011). Evolution of legacy system comprehensibility through automated refactoring. In *Proceedings of the International Workshop on Machine Learning Technologies in Software Engineering*, pages 35–42, New York, NY, USA. ACM.

Kolb, R., Muthig, D., Patzke, T., and Yamauchi, K. (2005). A case study in refactoring a legacy component for reuse in a product line. In *Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on*, pages 369–378.

M. Cherubini, G. Venolia, R. D. (2007). How and why software developers use drawings. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 557–566.

Moser, R., Sillitti, A., Abrahamsson, P., and Succi, G. (2006). Does refactoring improve reusability? In *Proceedings of the 9th international conference on Reuse of Off-the-Shelf Components*, ICSR'06, pages 287–297.

Murphy-Hill, E., Parnin, C., and Black, A. (2011). How we refactor, and how we know it. In *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*, pages 287–297.

Rodríguez-Echeverría, R., Conejero, J. M., Clemente, P. J., Preciado, J. C., and S$#225;nchez-Figueroa, F. (2012). Modernization of legacy web applications into rich internet applications. In *Proceedings of the 11th international conference on Current Trends in Web Engineering*, pages 236–250, Berlin, Heidelberg. Springer-Verlag.

Ulrich, W. M. and Newcomb, P. (2010). *Information Systems Transformation: Architecture-Driven Modernization Case Studies*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Wang, H., Shen, B., and Chen, C. (2009). Model-driven reengineering of database. In *Proceedings of the 2009 WRI World Congress on Software Engineering - Volume 03*, Washington, DC, USA. IEEE Computer Society.