View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Centro de Servicios en Gestión de Información

V International Conference on Computational Methods for Coupled Problems in Science and Engineering
COUPLED PROBLEMS 2013
S. Idelsohn, M. Papadrakakis and B. Schrefler (Eds)

SCIENCE & ENGINEERING SOFTWARE MIGRATION: MOVING FROM DESKTOP TO MOBILE APPLICATIONS

FEDERICO AMÉNDOLA AND LILIANA FAVRE

Universidad Nacional del Centro de la Provincia de Buenos Aires (UNCPBA)
Comisión de Investigaciones Científicas de la Provincia de Buenos Aires (CICPBA)
Tandil, Argentina
e-mail: famendola@alumnos.exa.unicen.edu.ar
lfavre@exa.unicen.edu.ar

Key words: Mobile Computing, Reengineering, Reverse Engineering, Model Driven Architecture, Android Platform, Eclipse Modeling Framework

1 INTRODUCTION

The proliferation of mobile devices over the last years provides opportunities and challenges for solving problems in science and engineering. Among other novel features, mobile devices contain global positioning sensors, wireless connectivity, built-in web browsers and photo/video/voice capabilities that allow providing highly localized, context aware applications. Mobile phones have become as powerful as any desktop computer in terms of applications they can run. However, the software development in mobile computing is still not as mature as it is for desktop computer and the whole potential of mobile devices is wasted [7, 8].

Although mobile technologies present new opportunities for services and business, they also present development and implementation challenges. Various authors describe challenges of mobile software development, for example, in [7] authors highlight creating user interfaces for different kinds of mobile devices, providing reusable applications across multiple mobile platforms, designing context aware applications and handling their complexity and, specifying requirements uncertainly. To ensure that the application provides sufficient performance while maximizing battery life is remarked in [18]. Some mobile applications also must determine the user location before offering the service and then track the location to adapt services and information accordingly. Besides, an additional challenge is to achieve the required level of security, reliability and quality of mobile services. Accepted rules for the design of traditional interfaces can not be fully implemented in the design of mobile interfaces [7].

A current problem in the engineering community is the modernization of legacy software. Software modernization, understood as technological and functional evolution of legacy systems, provides principles, methods, techniques and tools to support the transformation from an existing software system to a new one that satisfies different requirements. To meet new demands, existing systems must be constantly evolved. Many of the existing systems may be written for technology which is expensive to maintain and which may not be aligned with current organizational politics. However, they resume key knowledge acquired over the life of an organization and there is a high risk to replace them because they are generally business-critical systems. A number of solutions have been proposed to deal with this problem such as redevelopment, which rewrites existing applications, or migration, which moves the

existing system to a more flexible environment while retaining the original system data and functionality. A good solution should be to restore the value of the existing software, extracting knowledge and exploiting investment in order to migrate to new software that incorporates the new technologies.

On the one hand, traditional reverse engineering techniques can help in the software migration to mobile applications. They are related to the process of analyzing available software with the objective of extracting information and providing high-level views on the underlying code [5,19].

On the other hand, the rapid proliferation of different mobile platforms has forced developers to make applications tailored for each type of device. Within the mobile development, many companies have different development teams redoubling the software engineering efforts for functionally similar mobile applications. To achieve interoperability with multiple platforms the migration needs of novel technical frameworks for information integration and tool interoperability such as the initiative of the Object Management Group (OMG) called Model Driven Architecture (MDA) [11]. The outstanding ideas behind MDA are separating the specification of the system functionality from its implementation on specific platforms, managing the software evolution from abstract models to implementations, increasing the degree of automation of model transformations, and achieving interoperability with multiple platforms. Models play a major role in MDA which distinguishes at least Platform Independent Model (PIM) and Platform Specific Model (PSM). An MDA forward engineering process focuses on the creation of PIMs which are automatically transformed by tools to PSMs which are next transformed to specific code.

The essence of MDA is the Meta Object Facility Metamodel (MOF) that allows different kinds of software artifacts to be used together in a single project [14]. MOF provides two metamodels: EMOF (Essential MOF) and CMOF (Complete MOF). EMOF favors simplicity of implementation over expressiveness. CMOF is a metamodel used to specify more sophisticated metamodels. Transformations are expressed in the MOF 2.0 Query, View, Transformation (QVT) metamodel [16].

OMG is involved in the definition of standards to successfully modernize existing information systems. The OMG Architecture-Driven Modernization Task Force (ADMTF) is developing a set of specifications and promoting industry consensus on modernization of existing applications [1].

The objective of this paper is to describe a reengineering process that allow moving existing desktop applications for solving engineering problems of multidisciplinary character to mobile platforms. Our research aims to simplify the creation of applications for mobile platforms by integrating traditional reverse engineering techniques, such static and dynamic analysis, with MDA. It is worth considering that mobile applications are not different applications but are mainly intend to complement the existing desktop systems in the organization to make them mobile. We validated our approach by using the open source application platform Eclipse, EMF (Eclipse Modeling Framework), EMP (Eclipse Modeling Project) and the Android platform [2, 9].

This paper is organized as follows. Section 2 describes a reengineering process for adapting existing object-oriented software applications to mobile platforms. In Section 3 we summarizes reverse engineering techniques such as static and dynamic analysis. Section 4 describes metamodeling techniques in the context of the reenginering process. Particularly,

this section specifies how to transform models by using transformation languages aligned with the MDA standards. Also, it describes how to obtain a target application in the mobile platform. Finally, Section 5 presents conclusions and challenges in the modernization of legacy systems to mobile technologies.

2 A REENGINEERING PROCESS: FROM DESKTOP TO MOBILE APPLICATIONS

We propose a reengineering process for modernizing desktop applications to mobile platforms (Fig. 1). Reengineering process can be summarized into three steps: reverse engineering, model transformation and implementation. Reverse engineering extracts out higher level views of the system expressed by different kind of artifacts that allow creating a model of the source application called PIM in the MDA context. The transformation for one PIM to several PSMs is at the core of MDA. The objective of the model transformation step is to transform the source model (a PIM), into target models (PSMs linked to different mobile platforms). Finally, during the implementation step, target applications for different mobile platforms are generated from the PSMs.

The proposed process starts from a source application and the application of reverse engineering techniques to support the understanding of it. We consider that only the source code is the repository of information for recovering the system design. Because of this, the first stage of this process consists of applying different techniques of reverse engineering that are based on two main types of analysis: structural or static analysis, and behavioral or dynamic analysis.

Static analysis extracts static information that describes the software structure reflected in the documentation (e.g., the source code text) and is supported by CASE tools. Dynamic analysis information describes the structure of the run-behavior and can be extracted by using debuggers, event recorders and general tracer tools. Then, the first stage of the reengineering process allows extracting artifacts in a high abstraction level that describe the application being analyzed.

At this point, it is necessary to consider the dependencies that have the recovered software artifacts with the technologies applied to implement the system under analysis. These dependencies should not impact to the artifacts that describe the new system to be implemented. To avoid these situations, the integration of reverse engineering techniques with MDA is proposed. MDA aims interoperability between platforms and technology independence proposing that all devices involved in a development process are represented from MOF. MOF allows different kinds of software artifacts to be used together in a single project. The transformation between models allows representing the new system to be implemented.

There are different ways to achieve model transformations, for example by using a programming language or metamodeling techniques. There exists specific transformation languages that provides a way to specify how generate a target model that conforms to a target metamodel from a source model that conforms to a source metamodel, for example, we can mention QVT or ATL transformation languages.As a result of this step, models of target applications related to different mobile platforms are created.

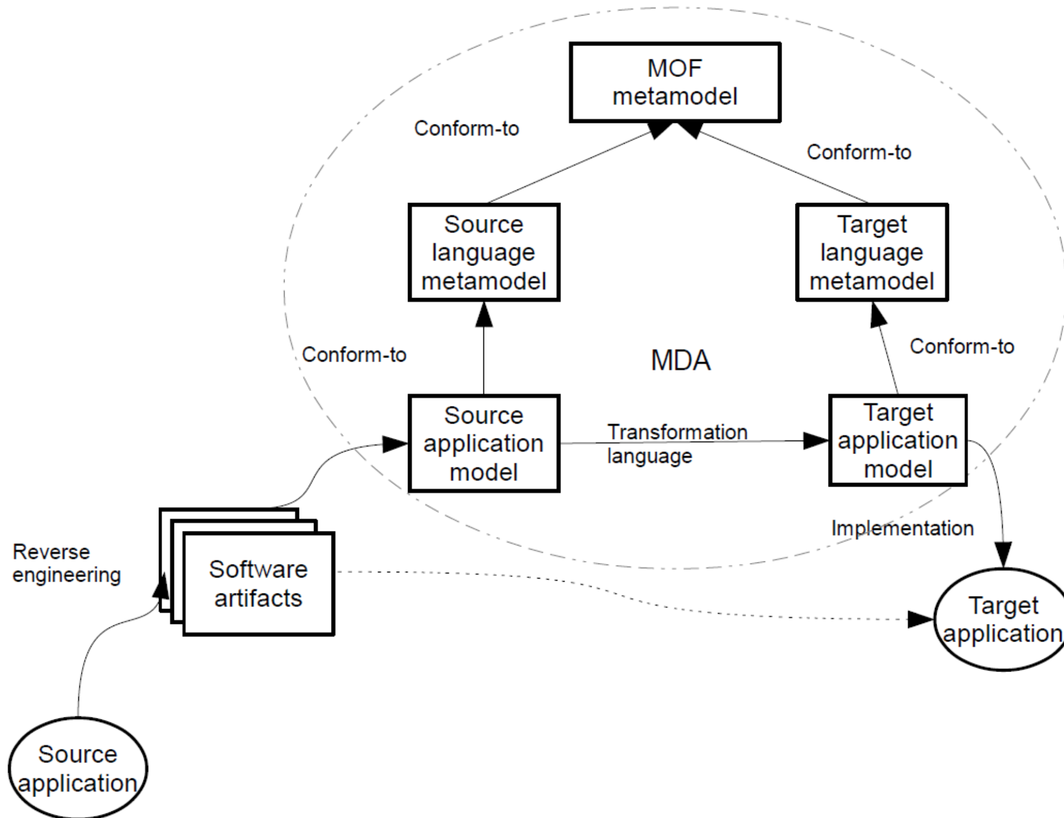

Figure 1 : A reengineering process

Next, in the implementation step, target applications are generated. To ensure the success of the steps of the reengineering process different tools known as CASE (Computer Aided Software Engineering) are needed. Each of these tools presents different features and support for the techniques involved in the reengineering process. In Section 2.1 we describe characteristics of different CASE tools.

2.1 CASE Tools

The success of MDA depends on the existence of CASE tools that make a significant impact on software processes such as forward engineering and reverse engineering processes. All of the MDA tools are partially compliant to MDA features. CASE MDA are generally extensions of CASE UML and most of them are not aligned with MOF. They provide good support for modeling and limited support for automated transformation in reverse engineering [6].

Many CASE tools support reverse engineering, however, they only use more basic notational features with a direct code representation and produce very large diagrams. Reverse engineering processes are facilitated by inserting annotations in the generated code. These annotations are the link between the model elements and the language.

The Eclipse Modeling Framework (EMF) [9] was created for facilitating system modeling and the automatic generation of Java code. EMF started as an implementation of MOF resulting Ecore, the EMF metamodel comparable to EMOF. EMF has evolved starting from the experience of the Eclipse community to implement a variety of tools and to date is highly related to Model Driven engineering (MDE). For instance, commercial tools such as IBM Rational Software Architect, Spark System Enterprise Architect or Together are integrated with Eclipse-EMF. Blu Age Reverse Modeling recovers from legacy systems some information necessary to build UML 2 models [6]

Few MDA-based CASE tools support any of the QVT languages. As an example, IBM Rational Software Architect support model-to-model and model-to-text transformations but not MOF and QVT. Spark System Enterprise Architect is based on MDA and UML 2.1 and then is compatible with MOF.

Other tools partially support QVT, for instance Together allows defining and modifying transformations model-to-model (M2M) and model-to-text (M2T) that are QVT-Operational compliant [17]. Medini QVT supports partially MOF and implements QVT. It is integrated with Eclipse and allows the execution of transformations expressed in the QVT-Relation language [13]. The MMT (Model-to-Model Transformation) Eclipse project, is a sub-project of the top-level Eclipse Modeling Project that provides a framework for model-to-model transformation languages. Transformations are executed by transformation engines that are plugged into the Eclipse Modeling infrastructure. The main transformation engines developed in the scope of that project are ATL and QVT [3,17]. ATL is a model transformation language and toolkit developed by ATLAS INRIA & LINA research group. In the MDE field, ATL provides ways to produce a set of target models from a set of source models. To date, the QVT declarative component is in its “incubation” phase and provides only editing capabilities to support the QVT language.

Currently, there are no tools supporting a complete reengineering process as proposed by this work. However, various tools are available to deal with it. In this paper we validate our approach by using the open source application platform Eclipse, EMF and EMP. In this context, we select a set of appropriate tools that will be described in the following sections. Besides, we select as a running example an adaptation of a CRM (Customer Relationships Management) desktop application to mobile platforms, Android platform in particular. Next, in Section 2.2, we introduce the running example.

2.2. A running example

In the following sections we describe in detail the proposed reengineering process. The different steps are illustrated by using a common example, a CRM (Customer Relationship Management) application. A CRM manages company interactions with current and future customers. Interactions are supported and guided by creating dynamic customer profiles that register information such as contracted services and products, frequent contact channels, and commercial transactions and their associated responses.

The CRM application that will be used to exemplify each step of the proposed process reengineering is called *SellWin* [17]. The analysis in this examples, will prioritize entities related to managing customer data. The simple client-server architecture of the application follows a component-oriented design separated in different modules: Domain, Data Base,

Server and User Interface. *SellWin* lacks adequate documentation to understand its design, which allows us to analyze the strengths and weaknesses of the application of reverse engineering techniques for understanding its functionality.

3 REVERSE ENGINEERING: FROM OBJECT-ORIENTED CODE TO MODEL

Reverse Engineering is the process of analyzing available software artifacts such as requirements, design, architectures, code or byte code, with the objective of extracting information and providing high-level views on the underlying system. Reverse engineering does not involve changing the source legacy systems, but understanding them to help reengineering processes that are concerned with their re-implementing. The main traditional techniques related to reverse engineering are static and dynamic analysis.

Static analysis extracts static information that describes the software structure reflected in the software documentation (e.g., the source code text) whereas dynamic analysis information describes the structure of the run-behavior and can be extracted by using debuggers, event recorders and general tracer tools. Static analysis is based on classical compiler techniques and abstract interpretation.

In [10], author provides a comparison of static and dynamic analysis from the point of view of their synergy and duality. He argues that static analysis is conservative and sound. Conservatism means reporting weak properties that are guaranteed to be true, preserving soundness, but not be strong enough to be useful. Soundness guarantees that static analysis provides an accurate description of the behavior, no matter on what input or in what execution environment. Dynamic analysis is precise due to it examines the actual run-time behavior of the program, however the results of executions may not generalize to other executions. Also, author argues that whereas the chief challenge of static analysis is choosing a good abstract interpretation, the chief challenge of performing good dynamic analysis is selecting a representative set of test cases. A test can help to detect properties of the program, but it can be difficult detect whether results of a test are true program properties or properties of a particular execution context. The combination of static and dynamic analysis can enrich reverse engineering process. There are different ways of combination, for instance performing first static analysis and then dynamic one or perhaps, iterating static and dynamic analysis. Likewise, the definition of appropriate heuristics may guide the search for information on the traces generated during the dynamic analysis.

3.1 Example

The first stage focuses on retrieving software artifacts that are useful to understand the design and implementation decisions for the chosen application. The aim is to detect the classes that make up the application and objects involved in the different functionality. With this information, models expressed by UML diagrams are generated [21].

Static analysis allows detecting the classes that compose the application and their relationships. Dynamic analysis is used to detect how they interact to solve the offered functionality. In this case, dynamic information is recovered using two techniques: execution trace and memory snapshot.

3.1.1 Static analysis: Class Diagrams

The initial step had to do with the recovering of class diagrams to detect relationships between the various components that make up the main modules. The explorer tool integrated with the Eclipse development environment, called UML ObjectAid [6], was used in this step. ObjectAid is a free tool for working with class diagrams but, it restricts access to sequence diagrams using a special license.

As an example, we show the class diagram of the *Customer Management* (Fig. 2). The purpose of this diagram is to visualize the relationships between the various modules. As we can see, the user interface module is unrelated to the database, and the access to data is provided by the server module, with which it maintains a direct association via a defined interface. Moreover, the user interface is the only one that has direct associations with the domain, since both the server and database, have only registered dependencies according to the methods of the interface of each class.

3.1.2. Dynamic analysis: execution traces and memory snapshots

To obtain and analyze execution traces of an application, we select the Eclipse Test and Performance Tools Platform (TPTP). It provides an open platform supplying powerful frameworks and services that allow software developers to build unique test and performance tools. TPTP allows executing instances of the application and registering the invocations. While the result is not a classic sequence diagram (for example, control statements are not detected) it is a good approximation to detect methods involved in each specific functionality and method invocation sequences. Also, the resulting diagram lets see how user interface components interact directly with domain components but not with the database components. Other dynamic analysis technique that was used in the process is *memory snapshots*. This analysis seeks to recover what is the current value of each of the attributes of the objects created during the execution of the application. This information is important not only to successfully deploy the application on the target platform, but in the modeling stage, as described in the next section. To detect the state of the memory was used a commercial tool that can be freely used for a limited evaluation time called YourKit Java Profiler. This tool allows running the application and capturing the information of the objects that were created in memory.

As a result of the application of static and dynamic analysis techniques it is possible to recover artifacts that allow reconstructing the design of the application under consideration. From this design, the source application can be implemented on the target platform, making the necessary modifications according to the mobile restrictions (memory space, screen size, usage limitations, among others).

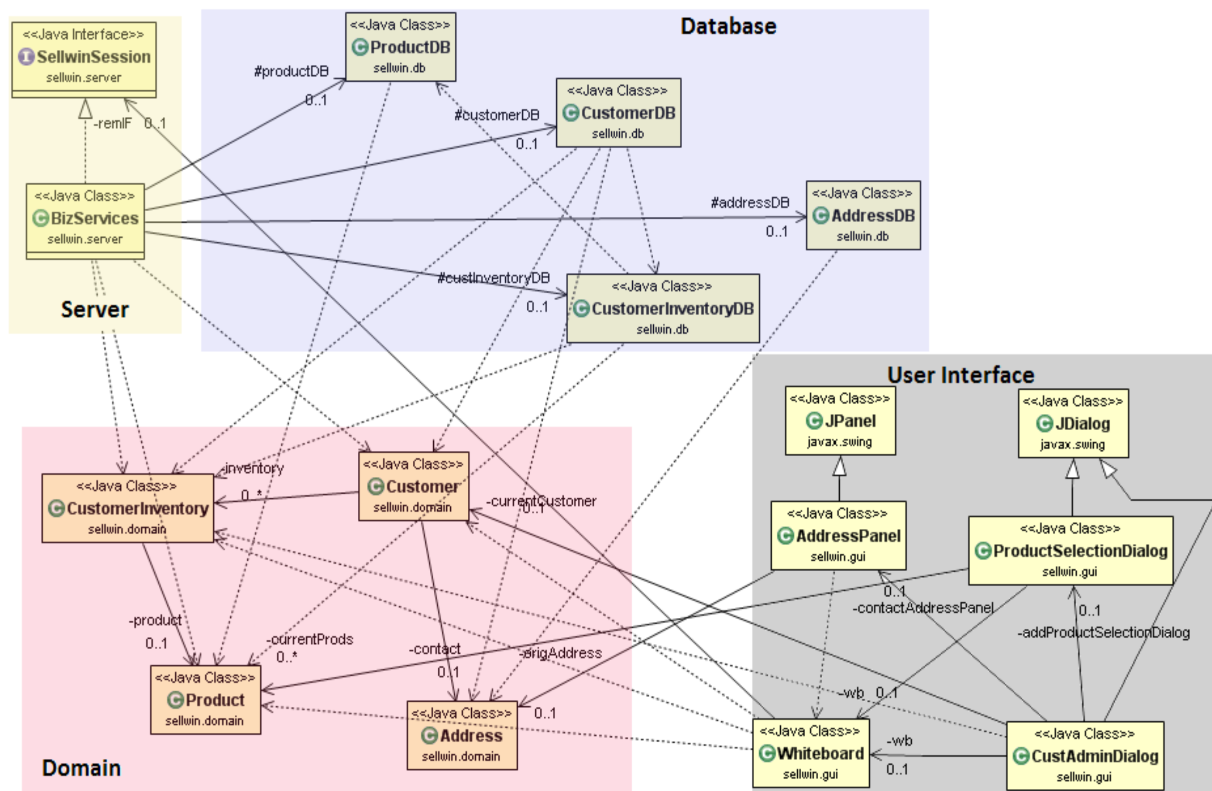

Figure 2 : Class diagram *Customer Management.*

4 MODEL TRANSFORMATION

MDA aims at the development of software systems based on the separation of business and application logic from underlying platform technologies, facilitating technology independence and interoperability between platforms. All artifacts involved in a development process are represented by means of metamodeling techniques, MOF metamodeling in particular. MOF metamodels are used to describe the transformations at model level. For each transformation, source and target metamodels are specified. A source metamodel defines the family of source models to which transformation can be applied. A target metamodel characterizes the generated models. In the MDA context, we consider that source models are PIMs and target models are PSMs. Model transformation provides a way to specify how generate a target model that conforms to a target metamodel from a source model that conforms to a source metamodel.

We validate our approach in the Eclipse Modeling Framework. Source and target metamodels conform to Ecore metamodel, which is comparable to EMOF.   In this experience, we select ATL as model transformation language. This stage of the translation process, was supported by the Eclipse Modeling Project (EMP) which provides tools for both defining metamodels and transformation rules, and executing the translation process.

4.1 Example

The Android platform provides a version of the Java language that is different to the version provided by environments of standard execution (*Java Runtime Environment*). One of main differences of this version of Java is the way of constructing graphic interfaces. It does not provide frameworks such as Swing or AWT but its own component libraries called *widgets*. Considering the above-mentioned, we present examples of translation centered on the components of the user interface module, which require substantial changes.

Fig. 3 shows a simplified *Java/JSwing* metamodel that includes classes (and attributes) used for the construction of client management screen . On the other hand, Fig. 3 also shows a simplified *Java/Android* target metamodel to implement screens of client management.

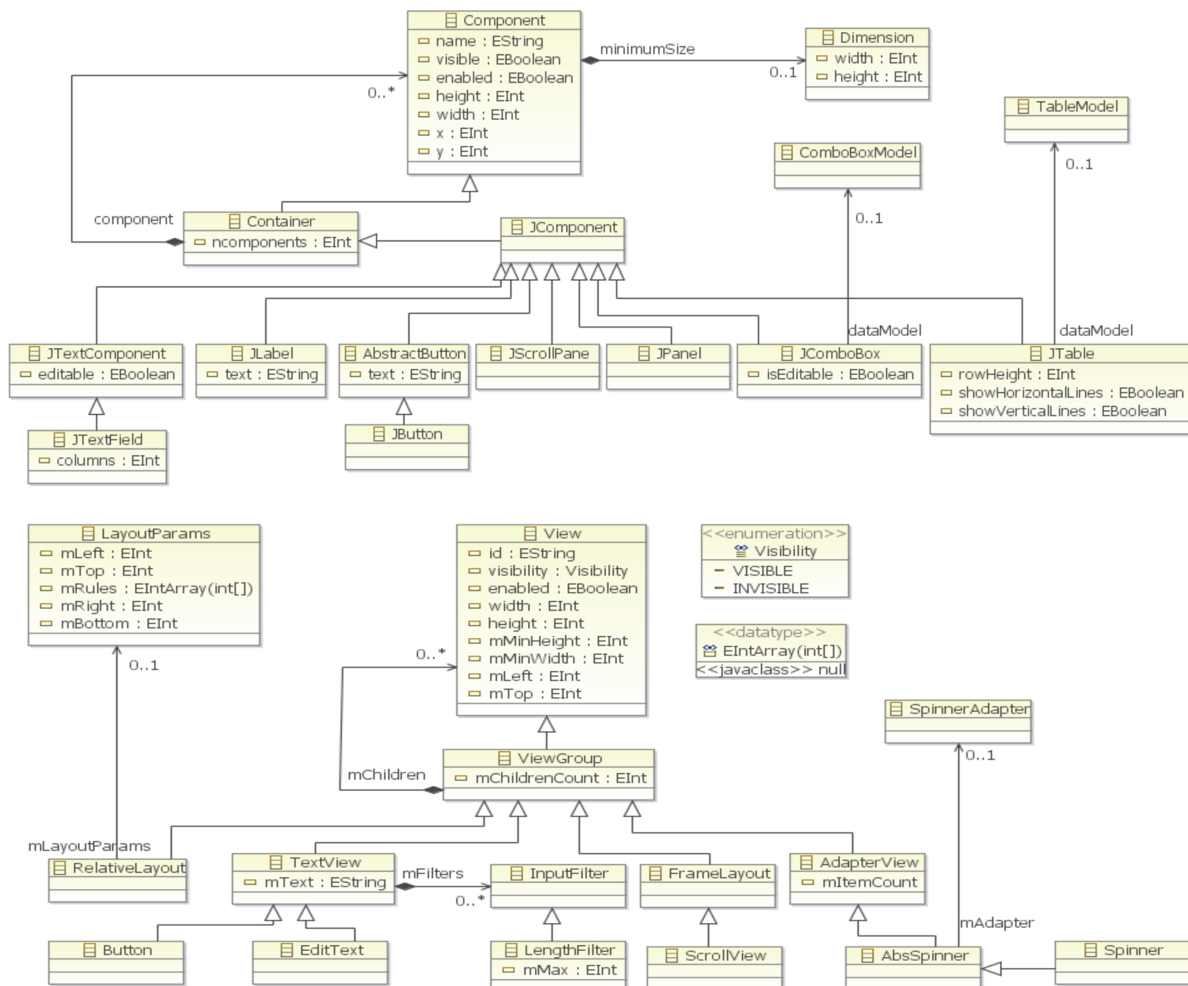

Figure 3. *Java/JSwing* source metamodels and *Java/Android* target metamodel

The main difference between the source and target metamodels is that interface controls do not provide the same functionality for all cases. In some cases, due to technological constraints and characteristics of the target platform, it is necessary create equivalent functionality using different *widgets*. One such case may be the *JTable* class, which

```
module SwingToAndroid;
create OUT : JavaAndroid from IN : JavaSwing;
helper context JavaSwing!Component def:
getVisibility(): JavaAndroid!Visibility =
if self.visible = true then
   #VISIBLE
else
   #INVISIBLE
endif;
helper context JavaSwing!Component def: getWidth(s:
JavaSwing!Dimension): Integer =
if s.oclIsUndefined() then
   0
else
   s.width
endif;
helper context JavaSwing!Component def:
getHeight(s: JavaSwing!Dimension): Integer =
if s.oclIsUndefined() then
   0
else
   s.height
endif;
rule ComponentToView {
   from
      jc: JavaSwing!Component
   to  tv: JavaAndroid!View (
         visibility <- jc.getVisibility(),
         id <- jc.name,
         enabled <- jc.enabled,
         width <- jc.width,
         height <- jc.height,
         mLeft <- jc.x,
         mTop <- jc.y,
         mMinHeight <- jc.getHeight(jc.minimumSize),
         mMinWidth <- jc.getWidth(jc.minimumSize))
}
rule ContainerToViewGroup extends
ComponentToView {
   from
         jc: JavaSwing!Container
   to
         tv: JavaAndroid!ViewGroup (
         mChildren <- jc.component,
         mChildrenCount <- jc.ncomponents )
}
rule JComponentToViewGroup extends
ContainerToViewGroup {
   from
         jc: JavaSwing!JComponent
   to
         tv: JavaAndroid!ViewGroup
}
rule JLabelToTextView extends
JComponentToViewGroup {
   from
         jc: JavaSwing!JLabel
   to
         tv: JavaAndroid!TextView(
         mText <- jc.text )
}
rule JTextFieldWithColumnsToEditText extends
JComponentToViewGroup {
   from
         jc: JavaSwing!JTextField(jc.columns > 0)
   to
         tv: JavaAndroid!EditText(
         enabled <- jc.editable,
         mFilters <- filters ),
         filters: JavaAndroid!LengthFilter (
         mMax <- jc.columns)
}
rule JTextFieldToEditText extends
JComponentToViewGroup {
   from
         jc: JavaSwing!JTextField(jc.columns = 0)
   to
         tv: JavaAndroid!EditText (
         enabled <- jc.editable )
```

Figure 4. From *Swing* to *Android*: ATL rules

implements a data table, which has no equivalent functionality in Android and will be implemented by combining other controls. In other cases, we can also see restrictions that are configured from attributes of a control, becoming associations between *widgets*. For example, to set a maximum size for the number of characters that can be entered in an edit control (for class *JTextFiels*, attribute *column*), it is represented in Android by means of the association between the class *editText* with a filter of input of lenght (class *LenghtFilter* and

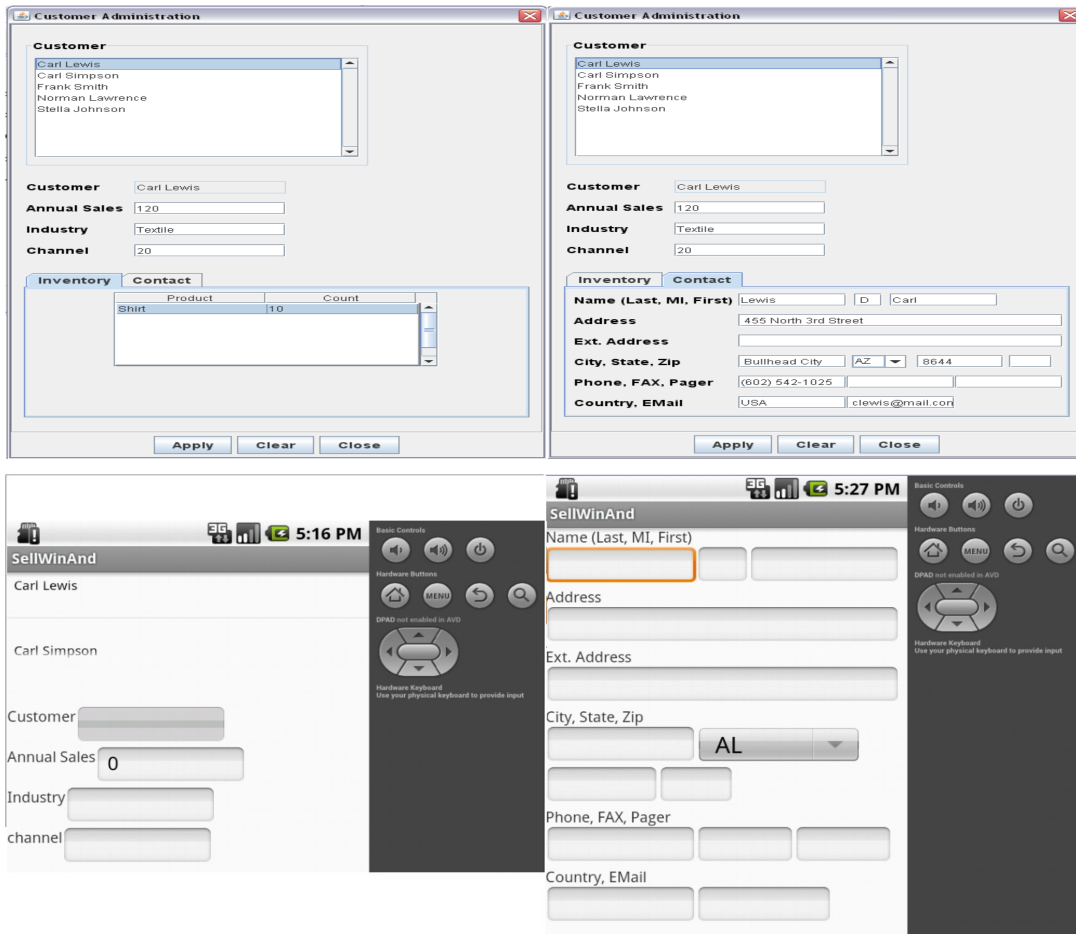

Figure 5. The original and the resulting screen of *Client Management*

the configuration of its attribute *nMax*). These considerations will be present at the moment of establishing translation rules in ATL. In Fig. 4, we present some of the ATL rules that allow the translation between the two metamodels. For example, the first rule describes how to transform the parent metaclass of the source class *Component* into the parent metaclass of the target class *View*. The transformation is performed for each attribute in an almost direct way, except for attributes that need to be invoked from the previously defined helpers. The main difficulties in the new implementation are associated with the particular features of platforms, primarily the size of the screen available to build the user interface, and methods of use of the input devices available which differ significantly from those found in a classical computer. Fig. 5 shows the original screen of client management and the resulting screen on the mobile device.

6 CONCLUSIONS

In this paper we have described a model driven reengineering process to adapt software systems for mobile platforms. Our proposal intends to improve the productivity of development teams taking into account the following strategies: working at a higher abstraction level focusing on design rather than on implementations, encapsulating mobile

platform model knowledge in a metamodel specification that will be reused in different applications, linking different CASE tools to the different activities , and particularly linking models to full code generators.

To propose a development process that considers platform-independent models is a very important practice to prevent future duplication of effort when trying to deploy the application to a new target platform. However, we detect some inconveniences. When the only information is the code, the success of the reverse engineering process depends largely on the availability of assistance and automation tools. This is one of the most important complications when attempting to migrate legacy system logic into a new application.

Beyond these difficulties, we show the acceptable feasibility of the proposed reengineering process by integrating different CASE tools and highlight the importance of having tools that assist during each stage of the proposed process. Our approach focuses on important problems in mobile development: creating user interfaces and enabling software reuse across multiple mobile platforms.

REFERENCES

[1] ADM. Architecture-Driven Modernization Task Force. http://www.omgwiki.org/admtf/doku.php (2013).

[2] Android Platform.http://www.android.com/ (2013).

[3] ATL Documentation. www.eclipse.org/m2m/atl/documentation (2013).

[4] Bruneliere, H., Cabot, J., and Dupé G. How to Deal with your IT Legacy: What isComing up in MoDisco?. *ERCIM NEWS 88*, http://ercim- news.ercim.eu/ images/stories/EN88/EN88-web.pdf (2012) 43-44

[5] Canfora, G. and Di Penta, M. New Frontiers of Reverse Engineering. Future of Software Engineering, FOSE'07, IEEE Press (2007) 326-341.

[6] CASE MDA. Committed Companies And Their Products. www.omg.org/mda/committed-products.htm (2013).

[7] Dehlinger, J. and Dixon, J. Mobile Application Software Engineering: Challenges and Research Directions. Proc. Workshop on Mobile Software Engineering. www.mobileSEworkshop.org, USA (2011).

[8] Dunkel, J. and Bruns, R.. Model-Driven Architecture for Mobile Applications. *Lecture Notes in Computer Science* 4439, Springer-Verlag (2007) 464-477.

[9] Eclipse Modeling Framework. http://www.eclipse.org/emf/ (2013).

[10] Ernst, M. . Static and Dynamic Analysis: Synergy and duality. Proceedings of ICSE Workshop on Dynamic Analysis (WODA 2003) (2003) 24-27.

[11] MDA. The Model-Driven Architecture. http://www.omg.org/mda/ (2013).

[12] Medini . Medini QVT. http://projects.ikv.de/qvt (2013).

[13] MoDisco . Model Discovery. http://www.eclipse.org/MoDisco (2013).

[14] MOF. Meta Object Facility (MOF) Core Specification Version 2.4.1, OMG Document Number: formal/2011-08-07. http://www.omg.org/spec/MOF/2.4.1 (2011).

[15] ObjectAid. http://www.objectaid.com/home (2013).

[16] QVT. QVT: MOF 2.0 Query, View, Transformation. Version 1.1, OMG Document Number : formal/2011-01-01. http://www.omg.org/spec/QVT/1.1/ (2012).

[17] SellWin, http://sellwincrm.sourceforge.net/. (2013).

[18] Thompson, C. Schmidt, D Turner, H. and White, J. Analyzing Mobile Application Software Power Consumption via Model-Driven Engineering. PECCS 2011 (2011) 101-113.

[19] Tonella,P. and Potrich, A. *Reverse Engineering of Object Oriented Code*. Monographs in Computer Science. Heidelberg: Springer-Verlag (2005).

[20] UML. Unified Modeling Language: Infrastructure. Version 2.4.1, OMG Specification formal/2011-08-05. http://www.omg.org/spec/UML/2.4.1/ (2011).