Adapting CRM Systems for Mobile Platforms: An MDA Perspective

Abstract— **Mobile phones have become as powerful as any desktop computer in terms of applications they can run. However, the software development does not take advantage of the whole potential of mobile devices. Enterprises are now adopting mobile technologies for numerous applications. A current problem is the modernization of useful legacy systems to mobile platforms. In this context, we describe a reengineering process that integrates traditional reverse engineering techniques with Model Driven Development (MDD), MDA (Model Driven Architecture) in particular. We describe a case study that shows how to move CRM (Customer Relationship Management) applications from desktop to mobile platforms. We validated our approach by using the open source application platform Eclipse, EMF (Eclipse Modeling Framework), EMP (Eclipse Modeling Project) and the Android platform.**

Keywords*— *Mobile Computing; Reengineering; Reverse Engineering; Model Driven Development; Model Driven Architecture; Customer Relationship Management

I. Introduction

Nowadays mobile devices come with their users all the time and everywhere. 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 [8].

Enterprises are now adopting mobile technologies for numerous applications to increase their operational efficiency and meet new customer demands. Improvements in the development of business applications for mobile devices are emerging in recent years. In [23], authors present a new taxonomy of enterprise mobile applications. They classify mobile applications into five categories: mobile broadcast (applications that broadcast different kind of content to a large group of mobile users), mobile information (applications that primarily present user-request information, where the flow is usually from the application to the user), mobile transaction (applications that facilitate and execute transactions such as buy and sell goods and services, place and track orders and make electronic payments), mobile operation (applications that primarily support the operational aspects of a business without direct interaction with costumer and client) and mobile collaboration (applications that foster collaboration among employers and various functional units in an enterprise and with other enterprises of interest and business partners).-

In [6], authors express that the fundamental challenges of mobile application software engineering involve 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. Authors remark in [8] that a critical aspect of developing future applications for mobile devices will be ensuring that the application provides sufficient performance while maximizing battery life. The rapid proliferation of different mobile platforms has forced developers to make applications tailored for each type of device. To achieve interoperability with multiple platforms the software community needs to adapt development approaches. Model Driven Development (MDD) is considered a promising approach to meet these challenges.

A current problem in the software community is the adaptation of legacy systems for mobile technologies. On the one hand, legacy systems resume key knowledge acquired over the life of an enterprise and, if they are business critical, there is a high risk in replacing them. On the other hand, mobile technologies have changed the way in which enterprises do business and create value. A number of solutions can be considered 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 mobile technologies.

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 [4].

To achieve interoperability with multiple platforms the modernization needs of technical frameworks for information integration and tool interoperability such as MDD. It refers to a range of development approaches based on the use of software models as first class entities; the most well-known realization of MDD is the OMG standard Model Driven Architecture (MDA) [12]. 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). 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 [15]. 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 [17].

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. The success of the Architecture-Driven Modernization (ADM) depends on the existence of CASE tools that make a significant impact on the automation of processes involved in the modernization [1].

The objective of this paper is to describe a reengineering process that allows moving existing desktop application to mobile platforms achieving interoperability with multiple 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. We analyze a case study on modernization of desktop CRM (Customer Relationship Management). Mobile CRM applications have a rich functionality related mainly to mobile transactions but also with mobile broadcast, mobile information and mobile collaboration. 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].

The paper is organized as follows. Section II presents background. Section III describes the proposed reengineering process. Next, Section IV includes a case study related to the modernization of desktop CRM applications. Finally, conclusions are included in Section V.

II. REENGINEERING, REVERSE ENGINEERING AND MDD

Software reengineering starts from an existing implementation and requires an evaluation of every part of the system that could be transformed or implemented anew from scratch. This definition distinguishes the following main phases: "*the examination and the alteration of a subject system to reconstitute it in a new form*" and "*the subsequent implementation in a new form*" [7]. In other words, reengineering includes some form of reverse engineering followed by some form of forward engineering.

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 success of reengineering depends on the existence of CASE tools that make a significant impact on software processes such as forward engineering and reverse engineering processes. In the context of MDD, the more relevant advances are linked to the Eclipse Modeling Framework (EMF) [9]. EMF was created for facilitating system modeling and the automatic generation of Java code. It 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 MDD. For instance, Commercial tools such as IBM Rational Software Architect, Spark System Enterprise Architect or Together are integrated with Eclipse-EMF [5].

MoDisco provides an extensible framework to develop model-driven tools to support use-cases of existing software modernization [14]. It uses EMF to describe and manipulate models [9], M2M to implement transformation of models into other models, Eclipse M2T to implement generation of text and Eclipse Java Development Tools (JDT) to create models out of Java source code.

The Eclipse-MDT MoDisco open source project is considered by ADMTF as the reference provider for implementations of several of its standards. It is a reusable and extensible model-based framework that facilitates the construction of reverse engineering applications. To date, MoDisco approach only support reverse engineering of class diagrams.

The MMT (Model-to-Model Transformation) Eclipse project, is a subproject 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 [3] and QVT [17]. ATL (Atlas Transformation Language) is a model transformation language in the field of MDD that is developed on top of the Eclipse platform. It is an hybrid language that provides a mix of declarative and imperative constructs.

Few MDA-based CASE tools support any of the QVT languages. As an example, IBM Rational Software Architect and Spark System Enterprise Architect do not implement QVT. 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. Medini QVT partially implements QVT. It is integrated with Eclipse and allows the execution of transformations expressed in the QVT-Relation language [13]. To date, the QVT declarative component is in its "incubation" phase and provides only editing capabilities to support the QVT language.

III. A REENGINEERING PROCESS FOR MIGRATING LEGACY APPLICATIONS

We propose a reengineering process for modernizing desktop applications to mobile platforms (Fig. 1). This process can be summarized into three steps. First, the information is extracted out of the source desktop application. Second, this information is analyzed in order to take adequate modernization decisions and finally, the information is used to implemenent new mobile applications. These steps are supported by metamodels to describe existing systems, discoverers to automatically create models of these systems and, tools to understand and transform complex models created out of existing systems.

The proposed process starts from a source application and the application of reverse engineering. Reverse engineering techniques aim to support the understanding of a program by using source code as the main source of information about the organization and program behavior, and extracting a set of potentially useful views, expressed by models. Different techniques are involved to recover this information and generally, 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 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 of program model state that is easier to manipulate, although it loses some information. These ideas, applied in compiler optimizations, require information about program semantics and are semantics-preserving program transformations.

Dynamic analysis is based on testing and profiling on execution models. Execution tracer tools generate execution model snapshots that allow deducing complementary information. Execution models, programs and models coexist in this process. Dynamic analysis allows generating execution snapshot to collect life cycle traces of object instances and reason from tests and proofs.

[10] provides a comparison of static and dynamic analysis from the point of view of their synergy and duality. Author 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, [10] 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.

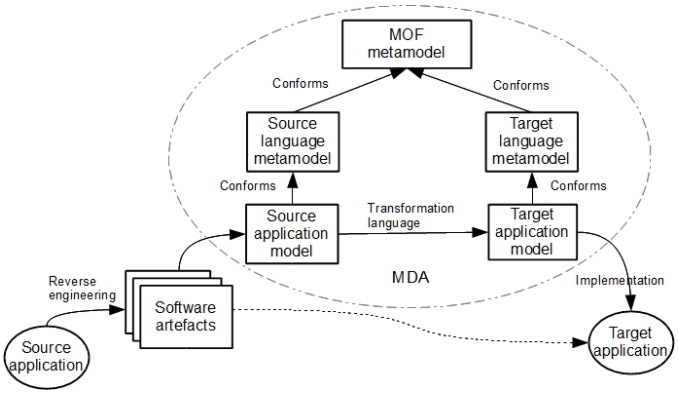

Fig.1. The reengineering process

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.

Static and dynamic analysis support reverse enginering process (first stage of the reengineering process) and allow 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 tecnologies 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 is proposed the integration of reverse engineering techniques with MDD, MDA in particular. MDD aims interoperability between platforms and technologies 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. A metamodeling technique is used in this step. MOF metamodels are used to describe the transformations at model level. We consider PIM expressed in UML [21] [22]. 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.

We validate our approach in the Eclipse Modeling Framework. Source and target metamodels conform to Ecore metamodel, which is comparable to EMOF. There are different ways to achieve transformations, for example by using a programming language or a language like QVT or ATL. In this experience, we select ATL as model transformation language (see Section II). As a result of this step, a PIM of the target application is created. Next, forward engineering processes must be applied to generate target models for different mobile platforms (PSM) and implementations.

IV. A CASE STUDY: MOVING FROM DESKTOP CRM APPLICATIONS TO MOBILE PLATFORMS

*A. Application Domain: **C**ustomer **R**elationship **M**anagement*

Future embedded and ubiquitous computing systems will operate through mobile devices. Enterprises are adopting mobile technologies for numerous applications to increase operational efficiency and meet new customer demands. The emergence of mobile devices in business applications has improved care and has streamlined the customer relationship, besides greatly simplify the exchange of information between a client and a consultant.

In this section we exemplify our approach with the modernization of a CRM software system developed to run over desktop computers.

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. Having rich customer profiles and good customer segmentation is the condition to achieve powerful CRM solutions.

With the advent of smartphones, CRMs have evolved from client-server applications to large Web applications (such as the case of Salesforce.com). Mobile CRM tries to complement the existing CRM systems in the enterprise to make them mobile. We propose to analyze how to move CRMs to a mobile platform, Android platform in particular. Developing software for mobile devices requires more large effort compared to software development for desktop computers and servers. While mobile device support advanced features like integrated databases, photo/video capabilities, voice recognition, global positioning sensors and wireless connectivity, these devices have limited resources, such as battery capacity, screen size, use of primary memory and availability of development libraries.

B. Description of the source application

The application that will be used as a case study is called *SellWin* [18]. It is a simple sales-oriented CRM that centers the data management around what it call *opportunities*. *SellWin* allows managing customer data, system users and sale opportunities. To illustrate the migration process, the analysis in this case study, will prioritize entities related to managing customer data. From the technical point of view, we can mention that *SellWin* is an open source application implemented entirely using Java. It uses Swing programming interface for the user interface and JDBC for database connections. The simple client-server architecture of the application follows a component-oriented design separated in different modules: Domain, Data Base, Server and User Interface. Choosing *SellWin* is due to its simple features and the possibility of having an open source CRM system. Furthermore, *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. Following, the steps that need to be executed to transform the CRM application to the Android platform are described.

C. Application of techniques for recovering designs

As mentioned above, we consider that only the source code is the repository of information for recovering the system design. Because of this, the first stage consists of applying different techniques of reverse engineering to generate UML 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* [16], 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.

From the recovered design and by applying other techniques to extract artifacts, the same application can be deployed on the target platform and then adapted taking into account the characteristics of the platform such as memory space, screen size and use limitations.

The next step integrates these artifacts with the ideas behind MDD to achieve the platform-independent representation involved so far.

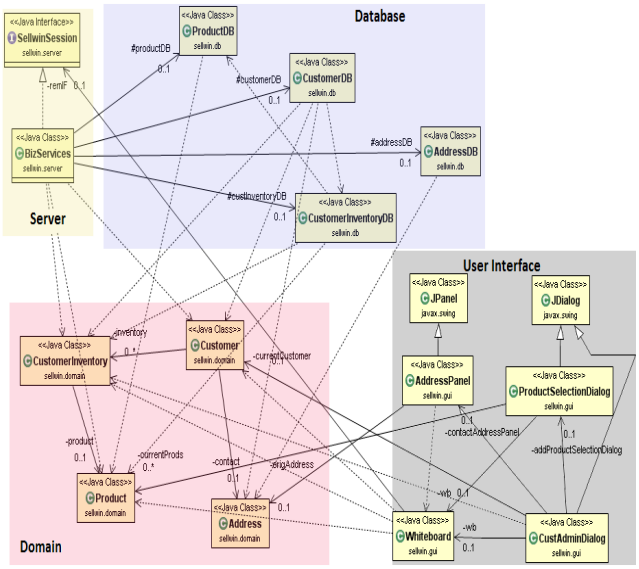

Fig. 2. Class diagram of the customer management

D. *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 independency and interoperability between platforms. All artifacts involved in a development process are represented by means of metamodeling techniques, MOF metamodeling in particular. Our goal is to generate platform independent models by using reverse engineering techniques such as static and dynamic analysis. The consistency of resulting models is evaluated by expressing the model translation in ATL.

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. ATL mainly focuses on the model-to model transformations which can be specified by means of ATL modules. An ATL module is composed of the following elements:

- a header section that defines the names of the transformation module and the variables of the source and target metamodels.
- an optional import section that enables to import some existing ATL libraries
- a set of helpers that can be used to define variables and functions.
- a set of rules that defines how source model elements are matched and navigated to create and initialize the elements of the target models.Source models and target models conforms a source metamodel and target metamodel respectively.

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. 1 shows the relation between models, source and target metamodels, and model transformations.

Following, we describe a simplified Java/JSwing metamodel that includes classes (and attributes) used for the construction of client management screen (Fig. 3). On the other hand, Fig. 4 shows a simplified Java/Android target metamodel and the concrete model of the application to implement screens of client management.

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 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 the configuration of its attribute *nMax*). These considerations will be present at the moment of establishing translation rules in ATL.

Following, we present some of the ATL rules that allow the translation between the two metamodels:

```
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 )
```

Because the main source metamodel components are related to each other by a hierarchy, the rules also have the same structure. Therefore, 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 following rule describes the transformation from *Container* to *ViewGroup* by using inheritance among rules, a feature added in the release ATL 2006. The association of components in the context of

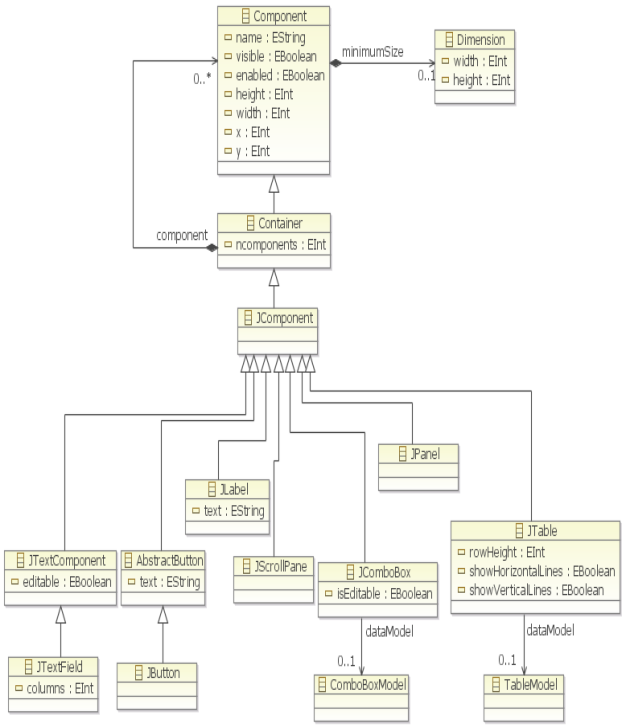

Fig. 3. Java/JSwing Metamodel

ViewGroup. This is possible due to the semantics of rule execution and the resolution algorithm of ATL. This algorithm states that, after determining the need for a link between objects of different metamodels, first it is necessary to solve each of the objects before linking objects of different metamodels.

In this specific case, it determines that the components found in *component*, are firstly evaluated to see if there are rules that define which elements should be transformed and then are assigned to *mChildren*. Because of this, we define rules to transform each of the possible elements found in *Component* to their equivalent in the Android model.

E. Target Application: Android Platform

From the design recovered by reverse engineering techniques and the transformation process created using metamodeling concepts, an equivalent design on the Android platform is created. By using this design we can complete the migration of the application under study. 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 in order to compare it with Fig. 6 which shows the resulting screen on the mobile device.

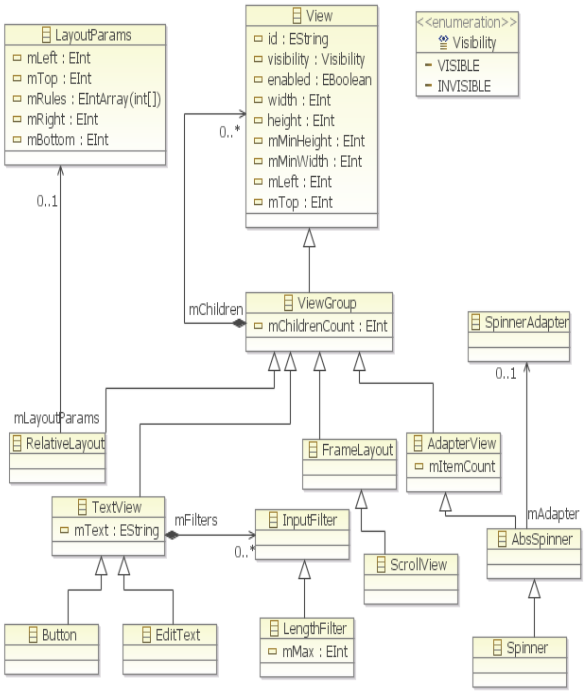

Fig. 4. Java/Android Metamodel

V. Conclusions

This research integrates traditional reverse engineering techniques and MDA to simplify the modernization of legacy desktop applications to mobile platforms. We exemplify our approach with a case study on modernization of legacy CRM to mobile platforms. The idea is to create applications for mobile platforms by reverse engineering a high level and platform independent model of a desktop application, and automatically transforming this high level model to platform specific code.

MDD allows developers to pay attention to the required functionality. It moved the focus from code to design, reducing the development effort to produce nearly native applications across multiple platforms.

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. Similarly, poor documentation tools and development in metamodeling and metamodel transformation also cause inconvenience.

Beyond these difficulties we believe that the case study illustrates the acceptable feasibility of the proposed reengineering process.

Future activities in reverse engineering should push towards a tight integration of dynamic analysis and human feedback into automatic reverse engineering techniques. The idea is to learn from expert feedback to automatically produce results.

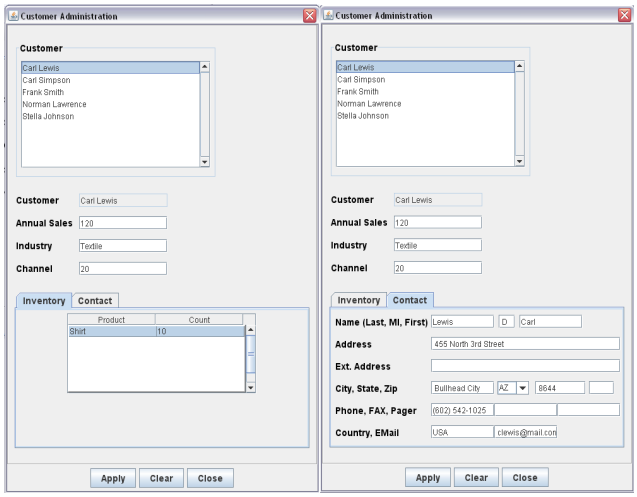

Fig. 5. Original screen of client management

Fig. 6. Resulting screen of client management

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] G. Canfora and M. Di Penta. "New Frontiers of Reverse Engineering. In Future of Software Engineering", FOSE'07, IEEE Press, 2007, pp. 326-341.

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

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

[7] S. Demeyer, S. Ducasse, and O. Nierstrasz. "Object-Oriented Reengineering Patterns." Amsterdam:Morgan Kaufmann, 2002.

[8] J. Dunkel and R. Bruns. "Model-Driven Architecture for Mobile Applications". LNCS 4439, Springer-Verlag, 2007, pp. 464-477.

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

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

[11] S. Maoz, and D. Harel “On tracing reactive systems”. Software & System Modeling. Springer-Verlag. 2010.

[12] MDA. The Model-Driven Architecture. http://www.omg.org/mda/, 2013.

[13] Medini . Medini QVT. http://projects.ikv.de/qvt, 2013.

[14] MoDisco . Model Discovery. http://www.eclipse.org/MoDisco, 2013.

[15] 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.

[16] ObjectAid. http://www.objectaid.com/home, 2013

[17] 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.

[18] SellWin, http://sellwincrm.sourceforge.net/, 2013.

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

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

[21] 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.

[22] UML. Unified Modeling Language: Superstructure. Version 2.4.1, OMG Specification: formal/2011-08-06. http://www.omg.org/spec/UML/2.4.1/, 2011.

[23] Unhelkar, B. Murugesan, S. “The Enterprise Mobile Applications Development Framework”. ITProfessional, (vol 12, no.3), IEEE, 2010, pp. 33-39.