ANDRIU. A Technique for Migrating Graphical User Interfaces to Android

Ricardo Pérez-Castillo, Ignacio García-Rodríguez de Guzmán, Rafael Gómez-Cornejo, Maria Fernandez-Ropero and Mario Piattini

Instituto de Tecnologías y Sistemas de Información (ITSI) at University of Castilla-La Mancha
Paseo de la Universidad 4 13071, Ciudad Real, Spain
{ricardo.pdelcastillo, ignacio.grodriguez, rafael.gomezcornejo, marias.fernandez, mario.piattini}@uclm.es

Abstract—Nowadays, pervasive environments force maintainers to provide agile solutions for migrating legacy information systems to mobile applications. While business knowledge can be easily reused in tier-based modularized systems, the migration of user interface tiers to a mobile application entails a bigger (but usually ignored) challenge. This paper presents AndrIU, a reverse engineering tool based on static analysis of source code for transforming user interface tiers from desktop applications to mobile ones. AndrIU has been specially developed for migrating traditional systems to Android applications. AndrIU is generic and extensible since it manages all the embedded knowledge in a common, standard repository according to the Knowledge Discovery Metamodel. This metamodel represents legacy knowledge in a platform-independent way. The main advantage is that AndrIU is designed to be extended for different migrations to others mobile platforms.

***Keywords*—Migration, User Interface, Android.**

I. Introduction

Pervasive environments are very common in our lives, which allow us to freely interact with a vast amount of services through a great variety of interactive devices (e.g., smartphones, laptops, pads, game consoles or digital television among other) [15]. For example, today's smartphones serve as email readers, calendars, car navigation systems or even entertainment systems, and they can provide almost constant connectivity between people via texting, voice calls, and video conferencing [3].

Pervasive environments force software producers to carry out agile developments to obtain just-in-time software applications for different devices [11]. Indeed, *App Store* and *Google Play* (two of the most important mobile applications markets) offer today more than 400,000 applications (with a growth of 20% per month) and they approximately consider 80,000 software vendors [16].

Recently, researchers and engineers have provided many methods and techniques to alleviate the challenge of agile software development for pervasive environments. Firstly, there exist multi-platform programming languages like Java. Secondly, various architectural design patterns have been proposed to facilitate the reuse of software such as three-tier architecture [4] or model-view-controller architecture [7]. The tier-based architectures encapsulate business rules in an isolated tier that minimize the coupling with other tiers in charge of user interface and data persistency.

Tier-based architectures facilitate the migration of software applications in pervasive environments since the core, business tier can be almost fully reused, while the remaining tiers has to be adapted. In fact, the sole migration of the part of an application that is interacting with the user is sometimes enough to migrate an application to different devices [2].

This paper addresses the migration of the graphical user interface (GUI) tiers of traditional applications to the user interfaces of mobile applications by integrating them with other source code tiers. This technique follows the model-driven development approach. The technique is particularly based on a static analyzer of Swing/AWT user interfaces which represent the information in a common, standard repository according to KDM (Knowledge Discovery Metamodel) [10]. KDM is the ISO/IEC 19506 standard for representing all the information retrieved by reverse engineering from every legacy software artifact (e.g., source code, databases, user interfaces, etc.). After that, the technique transforms the user interface model represented in the KDM repository to a user interface model for Android [8] applications based on a set of XML (eXtensible Markup Language) files.

The technique is supported by an Eclipse™-based tool, which was specially developed for Android applications due to their widespread use and open source nature, which facilitated the research. However, the technique is generic because it is based on the KDM standard, and therefore, additional transformations from KDM to others platforms such as iOS (based on Xcode) could easily be provided. The developed tool allows the applicability and adoption by the industry of the proposed technique for migrating user interfaces from Java-based applications to Android applications.

The remaining of the paper is organized as follows. Section II presents in detail the technique to migrate user interface tiers to Android applications. Section III introduces the supporting tool. Finally, Section 0 discusses conclusions and future work.

II. MIGRATION TECHNIQUE

AndrIU facilitates the migration of graphical user interfaces from desktop applications to mobile applications. On the one hand, the underlying process follows model-driven development principles, i.e., (i) it treats all the involved artifacts as models in accordance with particular metamodels, and (ii) it provides automatic transformations between such models at different abstraction levels. On the other hand, AndrIU is based on KDM [10] to represents all the extracted information in a platform-independent and standardized way.

AndrIU is specially developed to the migration of AWT/SWING user interfaces of desktop applications to Android user interfaces based on XML files (see Figure 1). However, AndrIU is generic due to the use of KDM, and may easily be extended for other platforms. AndrIU considers four different artifacts and a path of three progressive transformations between them.

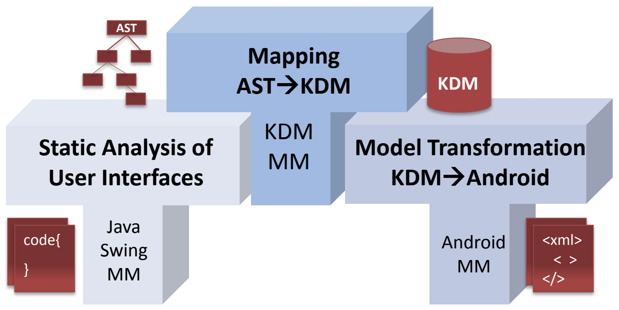


Figure 1. The user interface migration process

A. *Static Code Analysis of AWT/SWING interfaces*

In the proposed method, the first transformation is characterized by the use of static analysis as a reverse engineering technique to retrieve user interface information from the legacy source code. Static analysis consists of syntactically analyzing the source code of source files that belongs to the legacy system. Such static analysis detects code elements of the user interface tiers and ignores non-relevant pieces of source code belonging to different tiers.

This transformation is specifically tuned to analyze Java-based systems. Therefore, while the static analysis is digging up the information from a Java source file, a source code model is built on the fly according to the Java SWING metamodel. This metamodel contains some elements to represent containers such as *JFrame* or *JDialog*, and other elements to represent components within containers, e.g., *JTextField*, *JButton*, *JLabel*, etc. The parser searches for such elements in each source code file and builds the respective AST which represents a specific-platform model (PSM) (see Figure 1).

B. *Integration into the KDM repository*

After obtaining various ASTs from source code files, which contain relevant user interface elements, they are mapped to KDM elements to be integrated in the KDM repository. The mapping between Java SWING elements and KDM elements distinguishes three concerns: (i) the containers and control elements mapping; (ii) the actions mapping; and (iii) the navigability mapping.

Container and control elements mapping. The first mapping transforms container and control elements from SWING AST models to KDM UI models. Regarding containers, the mapped GUI elements are: windows (*JFrame*) and panels (*JPanel*) to *Screens*, and dialogs (*JDialog*) to *Report*. Concerning control elements, for example, buttons (*JButton*) and toggle buttons (*JRadioButton* and *JCheckBox*), which are transformed into *UIResource* elements; labels (*JLabel*) and text fields (*JTextField* and *JTextArea*) that are transformed into *UIField* elements in the KDM UI model. Since the KDM UI model has to be platform-independent, this mapping simplifies the semantics of user interface models. In fact, there are only four basic elements in the KDM UI model (*Screen*, *Report*, *UIResource* and *UIField*) to represents all the information. However, in order to avoid a semantic loss, such KDM elements incorporate an *Attribute* element that has a tagged value 'kind' which contains the classifier name of the SWING element, e.g., *JLabel*, *JButton*, etc.

Action Mapping. Since GUI elements are used to interact with other tiers of the system (e.g., the business domain tier), actions triggered under occurred events associated with GUI controls have to be collected in the KDM UI model. Such pairs of events-actions are handled in Java SWING through action listeners added in each GUI element. These elements are represented in the KDM UI model as *UIAction* elements, which have a feature *implementation* which contains a reference to a *CallableUnit* element within the KDM Code model. The implementation feature therefore represents a reference to the method that implements the triggered action. In this way, the UI and Code model are integrated within the KDM repository and feature location techniques could be used. Besides the action, *UIAction* elements contain a *kind* feature containing the type of the event (e.g., *actionPerformed*, *onMouseClick*, etc.).

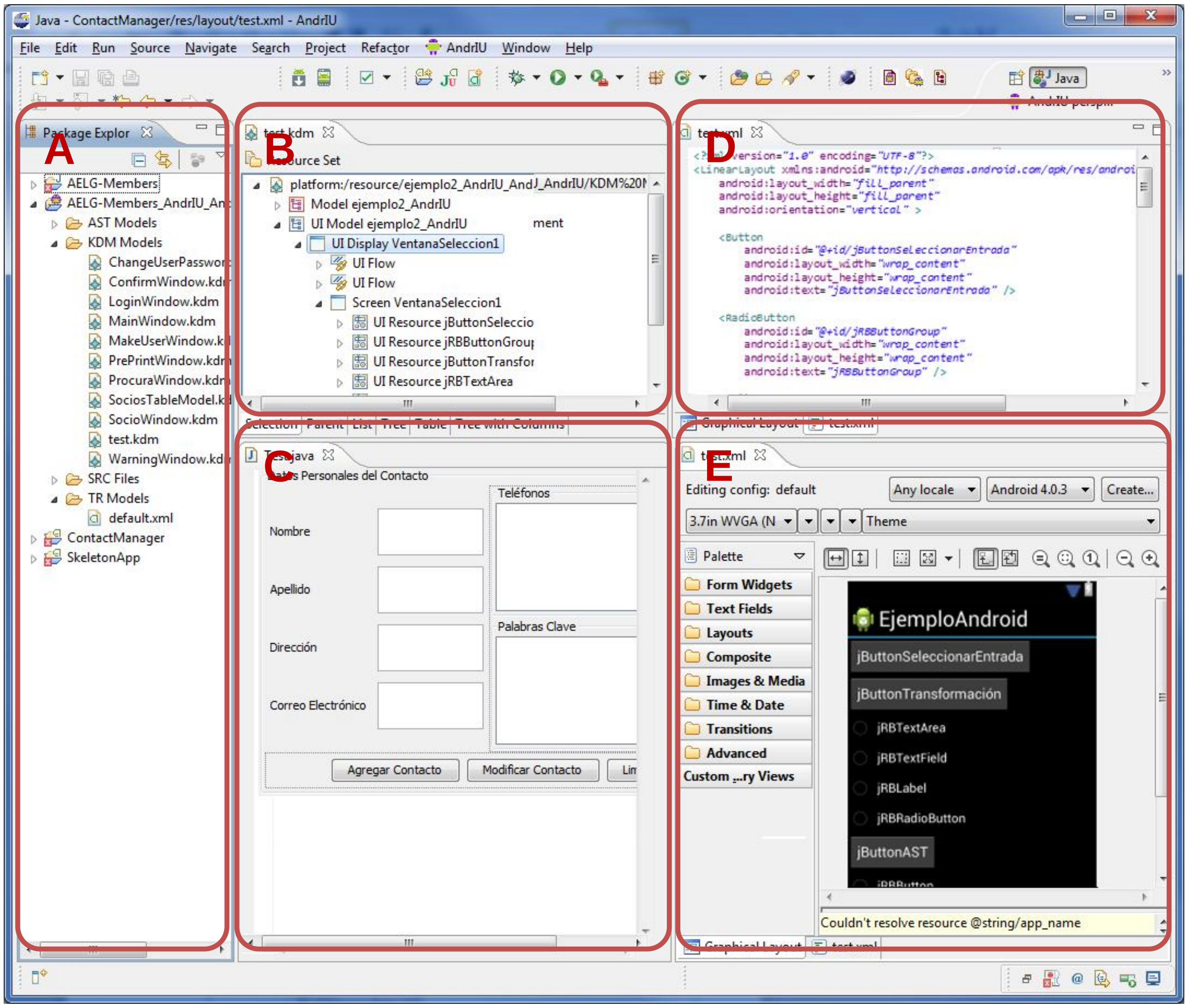
Navigability mapping. Finally, the mapping between the SWING AST model and the KDM UI model is completed with the navigability mapping. This mapping uses the *UIFlow* elements, as added to the *UIAction* elements to define a navigability relationship between two windows. *UIFlow* elements contain the features 'from' and 'to' that respectively represent the references to the source and target *UIDisplay* elements (*Screen* or *Report*).

In order to detect the navigability between two different windows the parser of the previous transformation searches for calls to the method *setVisible (true)* (open a certain window) and *setVisible (false)* (close a window). These calls represent that a window can be launched from another window.

C. *Generation of Android interfaces*

Once the information retrieved from the user interface tier is integrated into the KDM repository, it can be used for migrating the KDM UI model to Android-based user interfaces.

Android applications follow a Model/View/Control (MVC) architecture [7], which consists of four different components: activities, services, intents and resources. Graphical user interfaces of Android applications therefore consist of a set of Activities classes, which are built using *View* and *ViewGroup* objects. There are many types of views and view groups, each of which is a descendant of the *View* class. On the one hand, the *View* objects are the basic units of user interface expression on the Android platform and is the base for subclasses called 'widgets', which implements user interfaces controls (e.g., text fields and buttons).



```
<?xml version="1.0" encoding="UTF-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/androi
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <Button
        android:id="@+id/jButtonSeleccionarEntrada"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="jButtonSeleccionarEntrada" />

    <RadioButton
        android:id="@+id/jRBButtonGroup"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="jRBButtonGroup" />
```

Figure 2. AndrIU modules and functionalities.

On the other hand, the *ViewGroup* class serves as the base for subclasses called 'layouts', which offer different strategies for distributing widgets such as linear, tabular, relative, and so on. Android user interfaces are depicted through XML files together a java class (*R.java*) with the definition of all the UI resources. The XML files, depicting the Android user interfaces, are built by transforming the KDM UI model almost directly. For each *UIDisplay* element this transformation creates a *Layout* element in the Android user interface. After that, it transforms all the child elements of *Screen* or *Report* elements to different Android controls (e.g., *Button*, *EditText*, *TextView*) included within the previous *Layout* element.

III. ANDRIU TOOL

AndrIU [1] is a tool based on the Eclipse platform especially developed to automate the underlying migration process (see Figure 2). AndrIU allows maintainers to complete the entire technique, since it automates the three proposed model transformations. AndrIU additionally aids a manual post-intervention by maintainers and developers through various graphical editors so that the migration can be tuned or adapted for each case.

A. *Technologies Involved*

This tool has been developed for Java-based legacy systems and can be used to carry out case studies involving applications with AWT/SWING user interfaces. The tool is based on four key technologies. The first technology is *JavaCC*, which is a parser and scanner generator for Java [14]. It is used to develop the static analyzer. The second technology is EMF (Eclipse Modeling Framework), which is a modeling framework and code generation facility for building tools and other applications based on structured data models [6]. This framework makes it possible to build specific metamodels according to the ECORE meta-metamodel (i.e. ECORE is the metamodel proposed by the Eclipse platform to define metamodels). Then, from these metamodels, EMF provides tools to produce a set of Java classes for the model, along with a set of adapter classes that enable viewing and command-based editing of the model as well as a basic editor. Another

Eclipse framework, such as GMF (Graphical Modeling Framework), is also used together with EMF to generate graphical editors from the ECORE metamodels. Finally, the fourth technology is XMI (XML Metadata Interchange), which is a model-driven XML integration framework for defining, manipulating and interchanging XML data and objects [12]. Every model involved in the proposed technique becomes persistent with an XMI file.

B. *Tool Modules*

AndrIU is divided into five panels supporting different functionality. Firstly, the panel A (see Figure 2) provides a project explorer to navigate through all the different artifacts involved in the three transformations (e.g., source code of the input java application, KDM models, Android files, etc.).

To support the first transformation, a tool module to carry out static analysis was developed. In this case, the module was built specifically for parsing Java source code. This tool module was developed through JavaCC from the EBNF (Extended Backus–Naur Form) grammar of Java 1.5 [13]. This module takes a Java file as input and then generates an XMI file as the output that represents the Java code model, a PSM model in L1. The second module executes a set of QVT transformations to obtain a KDM model in L2 from the Java code model obtained previously. The transformation is executed using the open source *Medini QVT* [9], a model transformation engine for QVT Relations. Panel B (see Figure 2) provides a tree view editor, which was built through EMF, to visualize and manipulate KDM models. At the same time, Panel C shows the graphical representation of the existing GUI. The third module also executes a QVT transformation to support the third transformation based on pattern matching. Panel D (see Figure 2) visualizes in an XML editor the file that represents the target user interface according to the Android platform. Finally, Panel E allows maintainers to visualize in parallel the first sketch of the target Android-based GUI so that maintainers can refine it. Panels C and E can be used by maintainers for checking manually the results between the input and output graphical user interfaces.

IV. CONCLUSIONS

This paper presents AndrIU a static analysis-based tool for migrating the GUI layer of legacy, desktop application to mobile applications. AndrIU follows the model-driven development principles and uses the KDM standard to represent the intermediate information. The usage of KDM has two important advantages. Firstly, AndrIU considers a common KDM repository in which back-end parsers for different graphical user interfaces (e.g., SWING, GTK, etc.) can store the extracted information in the common KDM repository. In turn, many front-end analyzers could be plugged in the KDM repository to migrate interfaces to different platforms. Secondly, another advantage of the KDM repository is that it facilitates the integration of additional information related to other software artifacts such us source code, databases, event model. This information may be used to exploit synergies between different artifacts during user interfaces migrations. In this sense, feature location techniques [5] may be used by mapping, for example, user interface elements with database model in order to know which data is accessed from particular user interface controls.

The work-in-progress deals with the conduction of various experiments with several industrial, Swing applications in order to demonstrate the applicability of AndrIU. Additionally, the future work will address some open issues, e.g., the adaptation of user interfaces to different devices and screens; transform layout of desktop application to mobile devices; recognize GUI elements in legacy applications with *spaghetti* code.

ACKNOWLEDGMENT

This work was supported by the FPU Spanish Program and the R&D projects PEGASO/MAGO (TIN2009-13718-C02-01) and GEODAS-BC (TIN2012-37493-C03-01).

REFERENCES

[1] Alarcos Research Group. AndrIU v1.0. Eclipse Marketplace 2012 [cited 2012 28-06-2012]; Available from: http://marketplace.eclipse.org/content/andriu.

[2] Bandelloni, R., G. Mori, F. Paternò, C. Santoro, and A. Scorcia, Web User Interface Migration through Different Modalities with Dynamic Device Discovery, in 2nd International Workshop on Adaptation and Evolution in Web Systems Engineering (AEWSE'07). 2007: Como, Italy. p. 58-72.

[3] Ebling, M.R. and M. Baker, Pervasive Tabs, Pads, and Boards: Are We There Yet?, in IEEE Pervasive Computing Magazine. 2012. p. 42-51.

[4] Eckerson, W., Three Tier Client/Server Architecture: Achieving Scalability, Performance and Efficiency in Client Server Applications. Open Information Systems, 1995. 10(1): p. 3.

[5] Eisenbarth, T., R. Koschke, and D. Simon, Aiding Program Comprehension by Static and Dynamic Feature Analysis, in Proceedings of the IEEE International Conference on Software Maintenance (ICSM'01). 2001, IEEE Computer Society. p. 602.

[6] EMF, Eclipse Modeling Framework Project. http://www.eclipse.org/modeling/emf/. 2009, The Eclipse Foundation. IBM Corporation

[7] Gamma, E., R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Longman Publishing Co. ed. 1995, Inc. Boston, MA, USA: Addison Wesley.

[8] Google Inc. Android (http://www.android.com/). 2012 [cited 2012 05/04/2012].

[9] ikv++, Medini QVT. http://www.ikv.de/index.php?option=com_content&task=view&id=75&Itemid=77. 2008, ikv++ technologies ag.

[10] ISO/IEC, ISO/IEC 19506. Knowledge Discovery Meta-model (KDM), v1.1 (Architecture-Driven Modernization). http://www.iso.org/iso/iso_catalogue/catalogue_ics/catalogue_detail_ics.htm?ics1=35&ics2=080&ics3=&csnumber=32625. 2012, ISO/IEC. p. 302.

[11] Martin, R.C., Agile software development: principles, patterns, and practices. 2003: Prentice Hall PTR.

[12] OMG, XML Metadata Interchange. MOF 2.0/XMI Mapping, v2.1.1. http://www.omg.org/spec/XMI/2.1.1/PDF. 2007, OMG.

[13] Open Source Initiative, Java 1.5 grammar for JavaCC. https://javacc.dev.java.net/files/documents/17/3131/Java1.5.zip. 2009.

[14] Open Source Initiative, JavaCC 4.2. A parser/scanner generator for java. https://javacc.dev.java.net/. 2009.

[15] Schmidt, A., B. Pfleging, F. Alt, A.S. Shirazi, and G. Fitzpatrick, Interacting with 21st-Century Computers, in IEEE Pervasive Computing Magazine. 2012. p. 22-31.

[16] van Agten, T. Google Android Market Tops 400,000 Applications. 2012 January 3, 2012 04/04/2012].