VII International Conference on Computational Methods for Coupled Problems in Science and Engineering
COUPLED PROBLEMS 2017
M. Papadrakakis, E. Oñate and B. Schrefler (Eds)

MIGRATING SOFTWARE TO MOBILE TECHNOLOGY: A MODEL DRIVEN ENGINEERING APPROACH

LILIANA FAVRE[*] AND FEDERICO BRICKER[†]

[*] Universidad Nacional del Centro de la Provincia de Buenos Aires
Comisión de Investigaciones Científicas de la Provincia de Buenos Aires (CICPBA)
Tandil, Argentina
e-mail: liliana.favre@gmail.com

[†] Dokko Group, Tandil, Argentina

Key words: Software Modernization, Reverse Engineering, Model Driven Engineering, Mobile Computing, C++ Language, Haxe Language

1 INTRODUCTION

Nowadays, organizations are facing the problematic of having to modernize or replace their legacy software. This software has involved the investment of money, time and other resources through the ages and there is a high risk in replacing it. The purpose of reengineering is to adapt software in a disciplined way in order to improve its quality in aspects such as operability, functionality or performance. The focus of reengineering is on improving an existing system with a higher return on investment than would be achieved by developing a new system.

In the context of reengineering, the term legacy was associated with software that survived several generations of developers, administrators and users. The entry into the market of new technologies or paradigms is increasingly occurring and, motivates the growing demand for the adaptation of systems developed more recently. Mobile Computing is crucial to harvesting the potential of these new paradigms. Smartphones are the most used computing platform worldwide. They come with a variety of sensors (GPS, accelerometer, digital compass, microphone and camera) enabling a wide range of applications in Pervasive Computing, Cloud Computing and Internet of Things (IoT).

Pervasive Computing, also called Ubiquitous Computing is the idea that almost any device can be embedded with chips to connect the device to a network of other devices. The goal of pervasive computing, which combines current network technologies with wireless computing, voice recognition and Internet capability, is to create an environment where the connectivity of devices is unobtrusive and always available. Cloud Computing is an Internet-based computing for enabling ubiquitous, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly supplied with minimal management effort. Cloud computing has long been recognized as a paradigm for Big Data storage and analytics providing computing and data resources in a dynamic and pay-per use model. Finally, there is no single universal definition for IoT, we could define it as the interconnection via the Internet of computing devices embedded in everyday objects, enabling them to send and receive data. The IoT is becoming so pervasive and several studies predict that will be more than 30 billion IP-connected devices and sensors in the world by 2020.

Pervasive Computing, Cloud Computing and IoT face similar problems related to similar use cases, including smart cities, environmental monitoring, agriculture, home automation, and health. Smartphones have been one of the greatest facilitators of the solution for them.

Frequently, the development of software component and applications aligned to these new paradigms requires adapting existing desktop software to mobile platforms. For instance, there is a need to migrate C/C++ desktop applications developed in different domains of engineering to mobile platforms in order to adapt them to new technologies. On the one hand, most challenges in this kind of software migration are related with the proliferation of mobile platforms that makes mobile development difficult and expensive and, on the other hand with the need to define systematic, reusable processes with a high degree of automation that reduce risks, time and costs.

With respect to the first challenge, the ideal situation is to define multiplatform development. New languages are emerging to integrate the native behaviors of the different platforms targeted in development projects. In this direction, the Haxe language is an open-source high-level cross-platform programming language and compiler that can produce applications and source code for many different platforms from a single code base [7].

With respect to the systematic modernization process, novel technical frameworks for information integration, tool interoperability and reuse have emerged. Specifically, Model Driven Engineering (MDE) has emerged as a new software engineering discipline which emphasizes the use of models and model transformations to raise the abstraction level and the degree of automation in software development. Productivity and some aspects of software quality such as maintainability or interoperability are goals of MDE [4]. A branch of MDE linked to reengineering is Model-Driven Software Modernization (MDSM) [5]

This paper describes an MDE-based modernization framework defined in the context of a software modernization project aimed at migrating non-mobile software to various mobile platforms. An instantiation of the framework for the migration of C/C++ code to different mobile platforms through Haxe is presented. The proposal is validated in Eclipse Modeling Framework (EMF) considering that some of its tools and environments are aligned with MDE [23]. Our approach is supported by metamodels to describe existing systems, discoverers to automatically create models of these systems and, tools to understand and transform complex models.

The structure of the paper is as follows. Section 2 describes background, emphasizing on MDE and multiplatform development. Section 3 presents some relevant work related to our approach. Section 4 describes a MDE framework for software modernization. Section 5 includes a realization of the framework for the migration of C/C++ code to mobile App deployed on different mobile platforms. Finally, Section 6 presents a discussion of our approach and future work.

2 BACKGROUND

This section includes background on Model Driven Engineering (Section 2.1) and multiplatform development in the Haxe language (Section (2.2).

2.1 Model Driven Engineering

Different acronyms are associated with model-driven developments: MBE (Model Based Engineering), MDE (Model Driven Engineering), MDD (Model Driven Development) and MDSM (Model Driven Software Modernization).

MBE is the branch of software engineering in which software models play an important role being the basis of development. However, there is no direct link between models and generated software that is defined through precise transformations.

MDE can be viewed as a subset of MBE. It is the branch of software engineering in which processes are driven by models, i.e. models are the primary artifacts of different software processes. MDE has emerged as a new software engineering discipline that emphasizes the use of models and model transformations to raise the abstraction level and the degree of automation in software development. Productivity and some aspects of the software quality such as maintainability or interoperability are goals of MDE.

Model Driven principles can be summarizes as follows: all artifacts involved in a MDE process can be viewed as models that conform to a particular metamodel, the process itself can be viewed as a sequence of model transformations and all extracted information is represented in an standard way through metamodels. Then, model, metamodel and transformations are crucial in MDE.

Model-Driven Development (MDD) refers to forward engineering processes that use models as primary development artifacts. In an MDD development, everything is a model that conforms to a metamodel and the development itself is seen as a sequence of model-to-model transformations ranging from abstract to concrete levels. A specific realization of MDD is the Model-Driven Architecture (MDA) proposed by the Object Management Group (OMG) [10].

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. Three concepts are crucial in MDA: model, metamodel and model transformations. Models play a major role in MDA, which distinguishes at least Platform Independent Model (PIM) and Platform Specific Model (PSM) [16]. An MDA process focuses on the automatic transformation of different models that conform to MOF metamodels. The essence of MDA is Meta Object Facility (MOF), an OMG standard for defining metamodels that provides the ability to design and integrate semantically different languages such as general-purpose languages, domain specific languages and modeling languages in a unified way. The modeling concepts of MOF are classes, which model MOF meta-objects; associations, which model binary relations between meta-objects; Data Types, which model other data; and Packages, which modularize the models [17]. Consistency rules are attached to metamodel components by using OCL [18].

Model transformation is another fundamental concept in MDA. A transformation is the process of converting a source model that conforms to a source metamodel, in a target model that conforms to a target metamodel, both metamodels (source and target) conforming to MOF.

The standard proposed by OMG to specify model-to-model transforms is the QVT (Query, View, Transformation) [21]. ATL (Atlas Transformation Language) is the most mature transformation language aligned with MDE that provides ways to produce a set of target models from a set of source models [3].

A particular form of reengineering for the technological and functional evolution of legacy systems begins to be identified in the early 21st century under the designation of Model Driven Software Modernization (MDSM). It is based on model-driven processes of reverse engineering, restructuring and forward engineering [5]. In MDSM, models representing legacy software are discovery semi-automatically through a reverse engineering process and then transformed into models that meet the modernization requirements from which it is possible to forward engineering a new modernized software. The OMG Architecture-Driven Modernization Task Force (ADMTF) is developing a set of specifications and promoting industry consensus on modernization [2].

2.2 Multiplatform Development

The high cost and technical complexity of targeting development to a wide spectrum of mobile platforms has given rise to the cross-platform development. It allows using the same code to deploy an application on multiple platforms such as iOS, Androis or WindowsPhone. In this direction, the Haxe language emerges as an open-source high-level multiplatform programming language and compiler that can produce applications and source code for many different platforms from a single code-base [7].

Reference [6] summarizes the Haxe principles as follows: “support mainstream platforms”, “write once, reuse everywhere”, “always native, no wrapper”, “generated but readable” and “trust the developer”. The Haxe programming language is a high level programming language that mixes features of object oriented languages and functional ones. It is similar (but not pure) to object-oriented languages. The compiler supports novel features such as type inference, enforcing strict *type safety* at compile time. Currently, Haxe supports nine target languages which allow for different use-cases: JavaScript, Neko, PHP, Python, C++, ActionScript3, Flash, Java and C#. Haxe includes a set of common functions that are supported across all platforms, such as numeric data types, text, arrays, binary and some common file formats.

The idea behind Haxe is to allow developers choose the best platform for a specific development. To achieve this, it provides a standardized language, a standard library that works the same on all platforms and platform specific libraries that allow us accessing the full API for a given platform from Haxe.

3 RELATED WORK

Various authors describe challenges of mobile software development, for example, in [8] 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 uncertainty.

A DSL (Domain Specific Language), named MobDSL, to generate applications for multiple mobile platforms is described in [15]. Authors perform the domain analysis on two cases in the

Android and iPhone platforms. This analysis allows inferring the basic requirements of the language defined by MobDSL.

The proliferation of mobile devices generated the need to adapt desktop applications to mobile platforms. Reference [9] describes a migration process from Java to mobile platforms through the multiplatform language Haxe.

ANDRIU, a reverse engineering tool based on static analysis of source code for transforming user interface tiers from desktop application to Android, is described in [20]. ANDRIU has been developed for migrating traditional systems to Android applications although it was designed to be extended for different migrations to others mobile platforms.

Reference [13] describes six major trends affecting future smartphone design and use: personal computers, internet of things, multimedia delivery, low power operation, wearable computing and context awareness.

Reference [22] reports a practical experience in two transfer of technology projects. Authors analyze the factors that can be taken into account in transfer of technology projects applying metrics to give hints about the potential productivity gains that MDE could bring.

Reference [11] presents a solution for facilitating the migration of applications to the cloud, inferring the most suitable deployment model for the application and automatically deploying it in the Cloud. Reference [14] describes the challenges in mobile computing offloading to cloud through experimentation.

4 A MODERNIZATION FRAMEWORK

We propose a framework for the modernization of desktop software to new technologies. According to the three crucial concepts of MDE, the framework provides sets of models, metamodels and transformations. Figure 1 depicts the main components of the framework.

Three different types of models are distinguished: Platform Independent models (PIM), Platform Specific Model (PSM) and Implementation Specific Model (ISM). A PIM is a model with a high level of abstraction that is independent of an implementation technology. A PSM is a tailored model to specify a system in terms of specific platform such J2EE,.NET, web or mobile. PIM and PSM are expressed in UML and OCL [18]. The subset of UML diagrams that are useful for PSM includes class diagram, object diagram, state diagram, interaction diagram and package diagram. On the other hand, a PIM can be expressed by means of use case diagrams, activity diagrams, interactions diagrams to model system processes and, state diagrams to model lifecycle of the system entities. An ISM is a specification of the implementation (source code) in terms of models.

The framework includes PSMs and ISMs related to the source and target platform. The target PSM and target ISM are related to a cross-platform language that allows writing mobile applications that target all major mobile platforms.

Metamodeling is a powerful technique to specify families of models. A metamodel is a model that defines the language for expressing a model, i.e. “a model of models”. A metamodel is an explicit model of the constructs and rules needed to build specific models. It is a description of all the concepts that can be used in a model. MOF metamodels use an object modeling framework that is essentially a subset of UML 2.5 core. The modeling concepts are

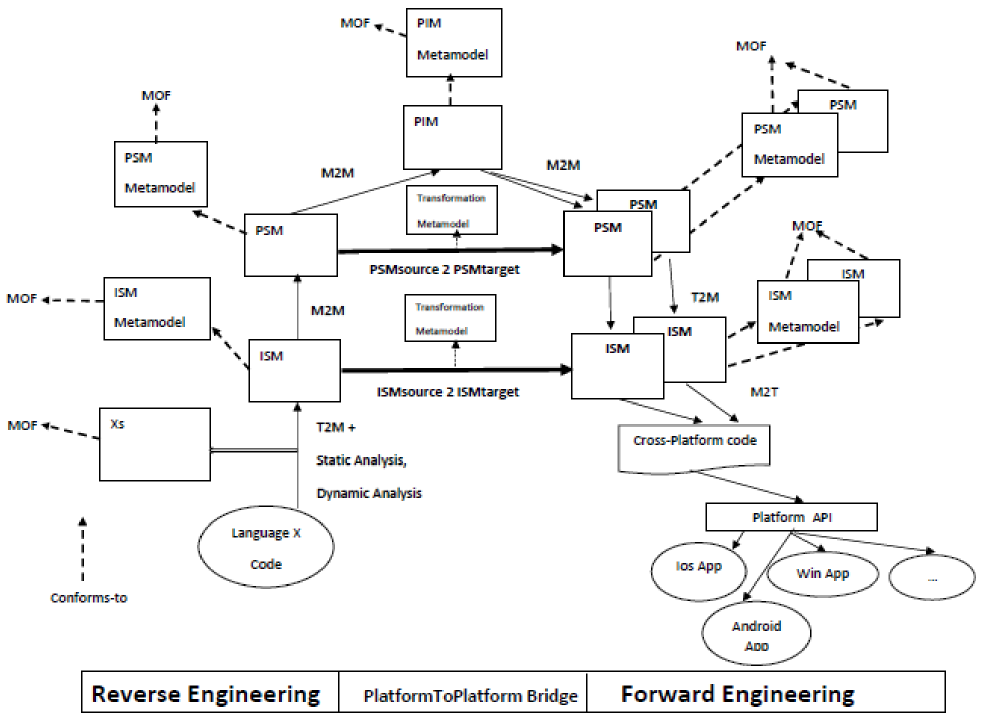

Figure 1. A modernization framework

metaobjects, data types which model other data, and packages which modularize the models. At this level MOF metamodels describe families of ISM, PSM and PIM. Every ISM, PSM and PIM conforms to a MOF metamodel.

The framework includes different kinds of transformations: T2M (Text-to-Model), M2M (Model-to-Model) and M2T (Model-to-Text).

T2M transformations allow representing the source code of the program in terms of a model compatible with MOF. They require to have a metamodel that describes the grammar of the source language. First, a representation of the original code in terms of an Abstract Syntax Tree (AST) is built. The next step in the reverse engineering process involves applying traditional techniques for static and dynamic analysis. The basic representation of the static analysis is an oriented graph that represents all data flow. Static analysis can be complemented with dynamic analysis that analyses traces of execution for different test cases.

Model-to-model (M2M) transformations provide a mechanism for automatically creating target models based on information contained in existing source models.

The framework distinguishes vertical and horizontal model-to-model transformations. Vertical transformations occur when a source model is transformed into a target model at a different abstraction level. They are useful in reverse engineering processes (ISM-to-PSM, PSM-to-PIM transformations) or forward engineering (PIM-to-PSM, PSM-to-ISM). Horizontal transformations involves transforming a source model into a target model that is at the same abstraction level. They are bridges between different platforms at the same abstraction level (ISM or PSM), for instance *ISMsource2ISMtarget* and *PSMsource2PSMtarget*.

M2T transformations focuses on the generation of textual artifacts from models. In our context, M2T transformations are the processes to extract code from models following the MDE principles.

The framework shows different scenarios of modernization to adapt software to diverse mobile platforms. In the most general form, reverse engineering processes extract PIM models from the code, which are transformed into code through MDD processes for forward engineering. Reverse engineering processes can also recover PSMs that can be restructured at the same level of abstraction through a migration between different platforms. Different instantiations of this framework were analyzed. Next we will describe the instantiation of the framework for the migration of C / C ++ code to the multiplatform Haxe language at ISM level.

5 A FRAMEWORK REALIZATION: FROM C/C++ TO MOBILE PLATFORMS

This section is about an instantiation of the framework. First, we partially show the metamodels that had to be defined in order to realize the objectives of our project: the C/C++ metamodel (Section 5.1) and the Haxe metamodel (Section 5.2). Finally, Section 5.3 describes an instantiation of the framework for migrating (at ISM level) C/C++ code to Haxe that, at the same time, allows generating code on different mobile platforms.

The proposal was validated in the open source application platform Eclipse considering that some of its frameworks and tools are aligned with MDE standards. For example, EMF (Eclipse Modeling Framework) has evolved starting from the experience of the Eclipse community to implement a variety of tools and to date is highly related to MDE. Ecore is the core metamodel at the heart of EMF and can be considered the official implementation of MOF. The subproject M2M supports model transformations that take one or more models as input to produce one or more models as output [12]. ATL is a model transformation language (Atlas Transformation Language) and a toolkit that provides ways to produce a set of target models from a set of source models develop on top of the Eclipse platform [3].

5.1 The C++ Metamodel

The C++ metamodel conforms to Ecore and is partially shown in Figure 2. The root metaclass is *Program* that represents a C++ program, which owns source files, instances of *TranslationUnit*. A translation unit contains declarations such as block declaration, function definitions, template declarations, among others. A *SimpleDeclaration*, instance of *Block-Declaration*, has a *DeclSpecifierSeq* that is a sequence of *DeclSpecifiers* which refers to a declaration specifiers and a type specifier. In addition, a simple declaration has an *InitDeclaratorList* containing a variable declaration list that is a list of specifiers and the name of a variable and its corresponding initialization. A *FunctionDefinition* has a *Declarator* containing the function identifier and the parameter list. *Function* and *CtorOrDestFunction*, instances of *FunctionDefinition*, have a body that contains compound statements such as declarations, iterations, and selections. In addition, a *Function* has a *DeclSpecifierSeq* that is a sequence of *DeclSpecifiers* such as function specifiers and a type specifier. *TypeSpecifier* subclasses are *SimpleTypeSpecifier*, *ClassSpecifier* and *EnumSpecifier* among others. A *ClassSpecifier* has a *ClassHead* containing the class key (class or struct) and a *MemberSpecification* that contains *MemberDeclarations* such as variables, function declarations, function definitions, constructors, destructor and template members. The full C/C++ metamodel can be found at [10]

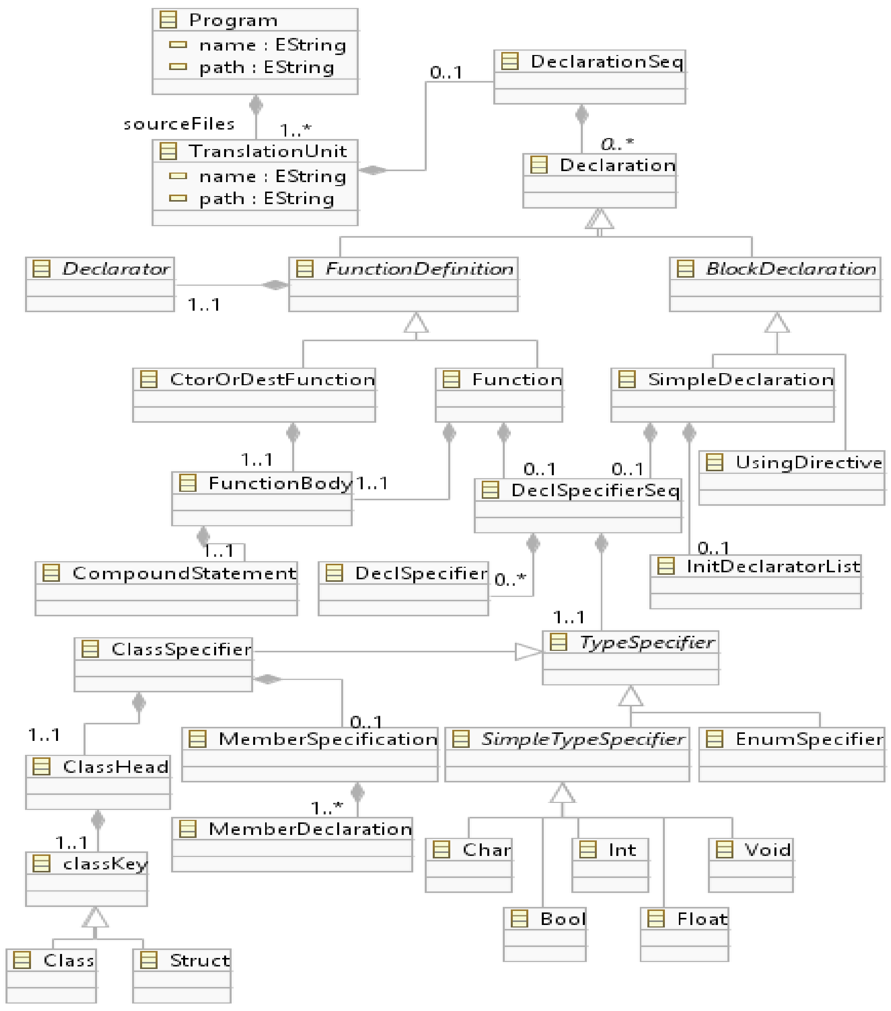

Figure 2. C++ Metamodel

5.2 The Haxe Metamodel

The Haxe metamodel conforms to Ecore metamodel. It is partially shown in Figure 3. The main metaclasses of the HAXE metamodel are those that allow specifying an application using Haxe as language.

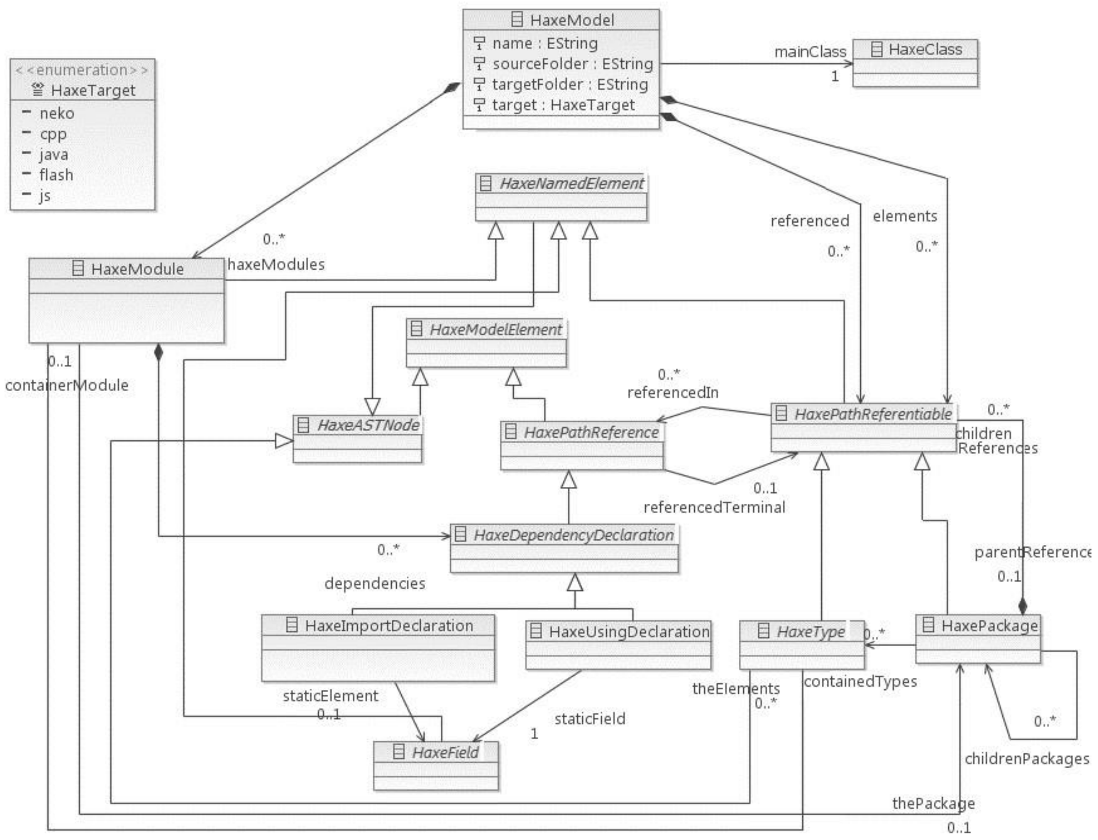

Figure 3. Haxe metamodel

One of the main metaclasses of the metamodel is *HAXEModel*, that serves as element container used to describe an application and store additional information on it, for example, some options of compilation and different metaclasses for modeling such as modules, classes and packages. *HAXEModel* owns *HAXEModule* and *HAXEPathReferentiable*.

Starting from the relations *HaxeModules*, *referenced* and *elements*, the class *HAXEModel* allows storing different information. Relation *HaxeModules* allows accessing the different HAXE modules used in the project. Through relation *elements*, it is possible to access the different elements of the package tree. Relation *referenced* provides access to elements, which are referenced in the project but are not defined completely. In the case of relations and referenced elements, the type used is *HAXEPathReferentiable*, which is the parent type of metaclasses such as *HAXEType* and *HAXEPackage*. The HaxeXE language includes different kind of types such as class (the types class and interface), function, abstract type, enumeration, and anonymous structures.

5.3 The Migration Process

Figure 4 depicts a realization of the framework. The initial transformation T2M obtains a code model that conforms to the C/C++ metamodel. This transformation was based on the generation of a parsing tree with the ANTLR tool through the C ++ grammar. Also a discoverer of a C ++ model that conforms to the C ++ metamodel was built.

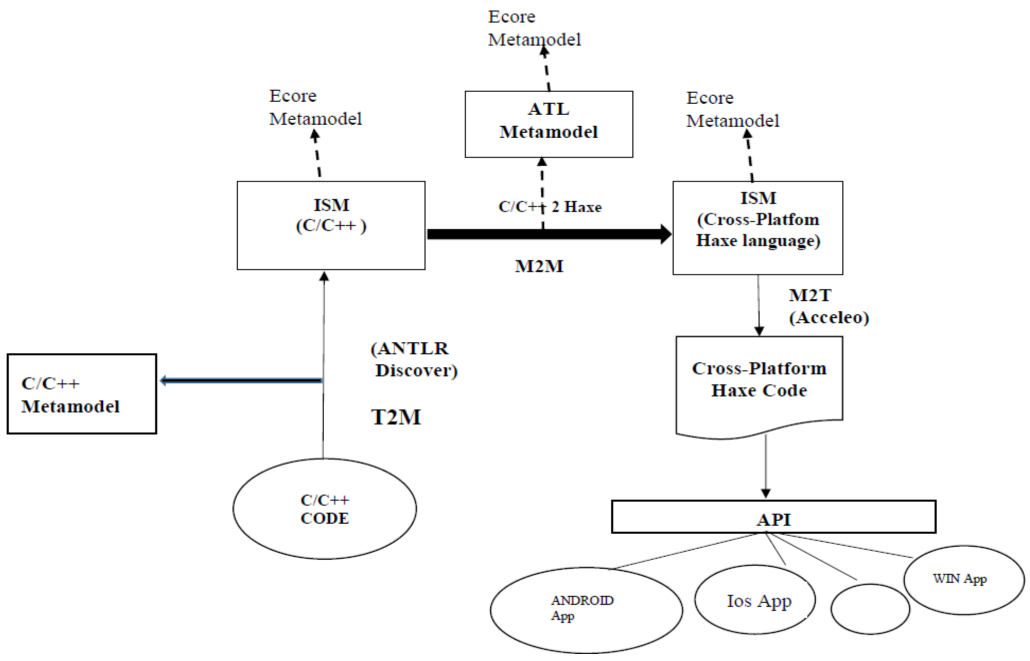

Figure 4. From C/C++ to mobile platforms

The Discover is a Java program whose input is the syntax tree and its output an ISM, the C++ model of the code.

M2M transformations were defined in ATL, the most mature transformation language in the context of MDE. ATL is a model transformation language (Atlas Transformation Language) and toolkit developed on top of the Eclipse platform. The ATL Integrated Development Environment (IDE) provides a number of standard development tools (syntax highlighting, debugger, etc.) that aims to facilitate the development of ATL transformations. ATL is a hybrid language that provides a mix of declarative and imperative constructs.

A model-to-model transformation from C++ to Haxe, called *C/C++ 2 Haxe*, was defined in ATL. It takes as input the model obtained in the reverse engineering phase and release an Haxe model. The transformation specifies families of transformations that produce Haxe models (target) from C++ models (source). Both source and target models must conform to the C++ metamodel and Haxe metamodel respectively.

The *C/C++ 2 Haxe* transformation conforms to the ATL metamodel, that, in the same way conforms to Ecore.

All models obtained in this chain of transformations are saved in the interchange format XMI, an OMG standard that combines XML, MOF and UML. It allows integrating tools, repositories, and applications in distributed heterogeneous environments [24].

Finally, from a model Haxe, it is possible to generate a source code in Haxe by using an M2T transformation defined in Acceleo, that is a code generation system based on MDE. It contains all the tools expected of a quality code generation IDE: simple syntax, efficient and advanced code generation. Its approach, based on prototypes and models, facilitates the creation of text generators based on the source code of existing prototypes [1].

Haxe allows writing mobile applications that target all major mobile platforms in a straightforward way. The generated code is syntactically correct, although, it does not compile on other platforms without doing changes due to the code refers to proprietary technologies of C++. To run on mobile environments, these technologies can be replaced with OpenFL and HAXEUI, that is an open source, multi-platform application-centric user interface framework designed for Haxe and OpenFL [19].

6 CONCLUSIONS

This paper describes migration of desktop applications to new technologies in which smartphones are its great facilitators. A modernization framework based on MDE principles and multiplatform development is presented. A framework realization that allows adapting C/C++ software to different mobile platforms through Haxe is described. Our approach covers a migration method and the presentation of models, metamodels and model transformations. These artifacts could be reused, modified for evolution purposes or, extended for other purposes.

Our approach is supported by metamodels to describe existing systems, discoverers to automatically create models of these systems and, tools to understand and transform complex models. The migration process can be divided in smaller steps focusing in specific activities, and be automated thanks to the chaining of model transformations. Model transformations allow developers to concentrate on the conceptual aspects of the relations between models and delegate the implementation of the transformation. The metamodel approach enables covering different levels of abstraction and satisfying several degrees of detail depending on the needs of the migration. Metamodeling is the key for interoperability of languages and tools.

We believe that our approach provides benefits with respect to processes based only on traditional migration techniques. One of the benefits of applying MDE techniques is to increase productivity in software development due to the automation that is introduced in the generation of artifacts. Besides, the use of MDE in migration projects is more cost-effective when the same process must be repeated frequently as the case of migrations to different platforms.

The advances achieved in MDE infrastructure and training of human resources, could allow us to apply the results in real projects in the scientific and industrial field.

REFERENCES

[1] ACCELEO. Obeo. ACCELEO Generator. http://www.eclipse.org/ACCELEO/ (2017)
[2] ADM. Architecture-driven modernization task force. http://www.adm.org (2017)
[3]ATL. Atlas Transformation Language Documentation. http://www.eclipse.org/atl/documentation/ (2017)
[4] Brambilla, M., Cabot, J. and Wimmer, M. *Model-Driven Software Enginneering in Practice*, Synthesis Lectures on Software Engineering. Morgan & Claypool Publishers (2012)
[5] Bruneliere, N., Cabot, J., Dupé, G. and Madiot, F. MoDisco: A Model Driven Reverse Engineering Framework. *Information and Software Technology*, (2014) 56(8), 1012–1032
[6] Cannasse, N. *HAXE. Too Good to be True?* GameDuell Tech Talk. http://www.techtalk-berlin.de/news/read/nicolas-cannasse-introducing-HAXE/ (2014)

[7] Dasnois, B.. *HAXE 2 Beginner's Guide*. Packt Publishing (2011)
[8] Dehlinger, J. and Dixon, J. Mobile application software engineering: Challenges and research directions. *Proceedings of the Workshop on Mobile Software Engineering*. Berlin, Springer-Verlag (2011) 29-32
[9] Diaz Bilotto, P. and Favre, L. Migrating Java to Mobile Platforms through HAXE: An MDD Approach. Chapter 13. *Modern Software Engineering Methodologies for Mobile and Cloud Environments.* Antonio Miguel Rosado da Cruz, Sara Paiva, eds., IGI GLOBAL (2016) 240-268
[10] Duthey, M. and Spina, C. Migration of C/C++ software to mobile platforms through MDD (Undergraduate Thesis, Thesis Advisor, Favre, L); Computer Science Department, Universidad Nacional del Centro de la Provincia de Buenos Aires, Tandil, Argentina (2016)
[11]Ejarque, J., Micsik, A. and Badia, M. Towards Automatic Application Migration to Clouds, *IEEE 8th Int. Conf. on Cloud Computing* (CLOUD) (2015) pp. 25-32
[12] EMF. Eclipse Modeling Framework (EMF). http://www.eclipse.org/modeling/emf/ (2016)
[13] Islam, N. and Want, R. Smarthphones: Past, present and future. *Pervasive Computing*, (2014) 13(4), 82-92
[14] Joshi, P. Nivangune, A., Kumar, R., Kumar, S., Ramesh, R., Pani, S. and Chesum, A. Understanding the Challenges in Mobile Computation Offloading to Cloud through Experimentation. *2nd ACM Int. Conf. on Mobile Software Engineering and Systems* (P. Joshi MOBILESoft) (2015) 158-159
[15] Kramer, D., Clark, T. and Oussena, S. MobDSL: A domain specific language for multiple mobile platform deployment. *Networked Embedded Systems for Enterprise Applications (NESEA), 2010 IEEE International Conference*, Los Alamitos: IEEE Press. doi:10.1109/NESEA.2010.5678062 (2010) 1-7
[16] MDA. The Model-Driven Architecture. http://www.omg.org/mda/ (2017)
[17] MOF. Meta Object Facility (MOF) Core Specification Version 2.5, OMG Document Number: formal/2015-06-05 http: //www.omg.org/spec/MOF/2.5 (2015)
[18] OCL. OMG Object constraint language (OCL), version 2.4. http://www.omg.org/spec/OCL/2.4 (2014)
[19] OPEN FL, http://www.openfl.org/ (2016)
[20] Pérez Castillo, R., García Rodriguez, I., Gómez Cornejo, R., Fernández Ropero, M. and Piattini, M. ANDRIU. A Technique for Migrating Graphical User Interfaces to Android. *Proceedings of The 25th International Conference on Software Engineering and Knowledge Engineering (SEKE 2013)* Boston: Knowledge Systems Institute (2013) 516-519
[21] 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/SMM (2012)
[22] Sánchez Cuadrado, J., Cánovas, J. and García Molina, J. Applying model driven engineering in small software enterprises. *Science of Computer Programming* (2014) (89)176-198
[23] Steinberg, D., Budinsky, F., Paternostro, M. and Merks, E. *EMF: Eclipse Modeling Framework* (2nd ed.). Addison-Wesley (2009)
[24] XMI. XML Metadata Interchange (XMI) Specification. OMG Document Number: formal/2014-04-04. http://www.omg.org/spec/XMI/2.4.2/PDF/ (2014)