ECCOMAS Congress 2016
VII European Congress on Computational Methods in Applied Sciences and Engineering
M. Papadrakakis, V. Papadopoulos, G. Stefanou, V. Plevris (eds.)
Crete Island, Greece, 5–10 June 2016

MODERNIZING SOFTWARE IN SCIENCE AND ENGINEERING: FROM C/C++ APPLICATIONS TO MOBILE PLATFORMS

Liliana Favre[1,2]**, Claudia Pereria**[1]**, and Liliana Martinez**[1]

[1] Universidad Nacional del Centro de la Provincia de Buenos Aires
Tandil, Argentina
[2] Comisión de Investigaciones Científicas de la Provincia de Buenos Aires,
Argentina

{lfavre, cpereira, lmartine}@exa.unicen.edu.ar

Keywords: Software Modernization, Mobile Development, Model Driven Development, Model Driven Architecture, Migration, HAXE, C++.

Abstract. *The evolution of software technology leads to continuous migration of software components and applications. Particularly, most software applications in applied science and engineering are for desktop computers and there is a need to migrate them to mobile technologies. This kind of migration faces many challenges due to the proliferation of different mobile platforms. New programming languages are thus emerging to integrate the native behaviors of the different platforms targeted in development projects. In this direction, the HAXE language allows writing mobile applications that target all major mobile platforms. Novel technical frameworks for information integration and tool interoperability such as the Model Driven Development (MDD) can help to manage a huge diversity of mobile technologies. A specific realization of MDD is the Model Driven Architecture (MDA) proposed by the Object Management Group (OMG). In this work, we propose a migration process from C/C++ software to different mobile platforms that integrates MDA standards with HAXE. C/C++ is one of the most commonly used programming language in science and engineering domains and numerous legacy software components written in C++ require to be modernized. On the one hand, the proposed process follows model-driven principles: all artifacts involved in the process can be viewed as models that conform a particular metamodel, the process itself can be viewed as a sequence of model-to-model transformations and all the extracted information is represented in a standard way through metamodels. On the other hand, HAXE easily adapts the native behaviors of the different platforms targeted in development projects enabling extremely efficient cross-platform development, ultimately saving time and resources. The proposal was validated in Eclipse Modeling Framework considering that some of its tools and run-time environments are aligned with MDA standards. The paper includes a simple case study, the migration of a C++ application, "the Set of Mandelbrot", that allow us to exemplify the different steps of the process.*

1 INTRODUCTION

Mobile devices such as smartphones or tablets come with engineers and researchers 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. These features offer the possibility to adapt functionality to user needs and preferences and can be used to build context-aware applications [29].

The evolution of software technology leads to continuous migration of software components and applications. Particularly, most software applications in applied science and engineering are for desktop computers and there is a need to migrate them to mobile technologies. This kind of migration faces many challenges due to the proliferation of different mobile platforms. The high cost and technical complexity of targeting development to a wide spectrum of platforms, has forced developers to make applications tailored for each type of device. New programming languages are thus emerging to integrate the native behaviors of the different platforms targeted in development projects. In this direction, the HAXE [10] language allows writing mobile applications that target all major mobile platforms, such as Android, iOS and BlackBerry, in a straightforward way.

Novel technical frameworks for information integration and tool interoperability such as the Model Driven Development (MDD) can help to manage a huge diversity of mobile technologies [6]. MDD provides principles and techniques to represent software through models at different abstraction levels. A specific realization of MDD is the Model Driven Architecture (MDA) proposed by the Object Management Group (OMG) [23]. 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. Models play a major role in MDA which distinguishes Computation Independent Model (CIM), Platform Independent Model (PIM) and Platform Specific Model (PSM). Some authors also distinguish Implementation Specific Model (ISM) as a description (specification) of the system in source code.

The essence of MDA is MOF (Meta Object Facility), 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. Significant advantages can be made of this unification to construct powerful mobile design environments. 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 [21]. Consistency rules are attached to metamodel components by using OCL [22]. 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. The MOF 2.0 Query, View, Transformation (QVT) metamodel is the standard for expressing transformations [25].

The Architecture Driven Modernization (ADM) approach has established a set of solutions for information system modernization [2]. ADM is defined as "the process of understand and evolve existing software assets for the purpose of software improvement, modifications, interoperability, refactoring, restructuring, reuse, porting, migration, translation, integration, service-oriented architecture deployment". The OMG ADM Task Force (ADMTF) is developing a set of standards (metamodels) to facilitate interoperability between modernization tools.

The success of approaches such as ADM and MDA depend on the existence of CASE tools that make a significant impact on software processes such as forward engineering and reverse engineering processes [8]. The Eclipse Modeling Framework (EMF) was created for facilitating system modeling and the automatic generation of Java code [26]. 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 MDD [15]. The subproject M2M supports model-to-model transformations that take one or more models as input to produce one or more models as output. For instance, ATL (Atlas Transformation Language) is a model transformation language that is developed on top of the Eclipse platform [4]. Another subproject is ACCELEO, which is an implementation of the M2T (Model-to-Text) transformation standard of the OMG for EMF-based models [1]. It is used in forward engineering processes.

In the Eclipse environment, MDA is integrated with Java language but it is weakly supported for other programming languages such as C++ [26]. In particular, C++ is one of the most commonly used programming language in science and engineering domains. Numerous legacy software components written in C++ require to be modernized. EMF4CPP is the first step at providing a set of tools for MDD in C++ as an alternative to the Eclipse tools for Java [16]. It is a C++ implementation and type mapping for the Eclipse Modeling Framework core, the ECORE metamodel. The main facilities provided by EMF4CPP are to generate C++ code from ECORE metamodels and to parse and serialize models and metamodels from and into XMI documents [30]. However, an implementation of a MOF-compliant C++ metamodel would be necessary for other MDD processes (for example, reverse engineering or software modernization).

In this work, we propose a migration process from C/C++ software to different mobile platforms that integrates MDA standards with HAXE. An ECORE metamodel for the C++ language is provided. On the one hand, the process follows model-driven principles: all artifacts involved in the process can be viewed as models that conform a particular metamodel, the process itself can be viewed as a sequence of model-to-model transformations and all the extracted information is represented in a standard way through metamodels. On the other hand, HAXE easily adapts the native behaviors of the different platforms targeted in development projects enabling extremely efficient cross-platform development, ultimately saving time and resources.

The paper includes a simple case study, the migration of a C++ application “the Set of Mandelbrot” that allow us to exemplify the different steps of the process. We believe that our approach provides benefits with respect to processes based only on traditional ad-hoc migration techniques increasing productivity due to the automation introduced in the generation of the new software.

The paper is organized as follows. Section 2 presents related work. In Section 3, we summarizes our contribution: a migration process from C++ to mobile platforms. Section 4 describes the HAXE language and its metamodel. Section 5 describes the C++ metamodel. Section 6 details the different stages of the migration process through of a simple case study. Finally, in Section 7 and 8 we present discussion and conclusions respectively.

2 RELATED WORK

In this section, we describe existing approaches for the development of mobile applications related to MDD.

Reference [7] proposes a new software architecture with the objective of providing the same service as mobile Web service as well as mobile application. The authors report on the feasibility study that they conducted in order to evaluate whether to use model driven software development for developing mobile applications. They argue that the architecture is flexible enough to support mobile Web services and mobile applications at the same time. They have develop a metamodel to describe mobile application and have shown how to generate mobile application from that model.

The project BAMOS and an architecture designed and implemented for the generic and flexible development of mobile applications is described in [14]. A declarative description of the available services supports the architecture. The authors describe a model driven approach for generating almost the complete source code of mobile services.

Reference [19] goes through mobile development process and architectural structures and their analysis with empirical mobile application development. The architecture and architecture role on the development has been studied in mobile application and multiplatform service development.

Various authors describe challenges of mobile software development, for example, in [11] 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. Issues related to ensuring that the application provides sufficient performance while maximizing battery life are remarked in [28].

A proposal for supporting mobile application development by using models as inputs to an emulator is outlined at [5]. The authors describe an MDD-based emulator for using in the design of graphical interfaces and interactions. They propose transform functional behavior and requirement models with design restrictions into emulated applications.

Reference [20] describes a DSL (Domain Specific Language), named MobDSL, to generate applications for multiple mobile platforms. They 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. A reengineering process that integrates traditional reverse engineering techniques such as static and dynamic analysis with MDA is presented at [3]. The article describes a case study that shows how to move CRM (Customer Relationship Management) applications from desktop to mobile platforms. The proposal was validated in the open source application platform Eclipse, EMF, EMP, ATL and Android platform. Reference [13] 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 [24]. 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 [18] 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.

In this paper, we describe an original model-driven migration process that includes automatic analysis of existing code, reverse engineering of abstract high-level models, model

transformation to target models in the HAXE platform and code generation. Our approach offers an opportunity for increasing the automation in software migration. In the following section, we detail the main steps of the proposed migration process.

3 OUR CONTRIBUTION : THE MIGRATION PROCESS

The migration process follows model driven development principles: all artifacts involved in the process can be viewed as models that conforms to ECORE meta-metamodels, the process itself can be viewed as a sequence of model-to-model transformations and the extracted information is represented in a standard way through ECORE metamodels. For each transformation, source and target metamodels are specified. A source metamodel defines the family of source models to which transformations can be applied. A target metamodel characterizes the generated models.

Figure 1 summarizes the proposed process. The first step is the reverse engineering of the C/C++ code to build a high-level model of it by using a model injector. Then, the obtained model is refactored to improve the design and eliminate dependencies of C/C++ language. The following step is related to model-to-model transformations that allow integrating C/C++ models with HAXE models. Specifically, this transformation is an ATL transformation that generates a model of the HAXE platform from a C/C++ model. Next, it is possible to generate source code in HAXE from the HAXE model by using M2T transformations expressed in ACCELEO. Considering that HAXE has one cross-platform standard library and various platform specific APIs used to natively access platform features, it is possible to write a mobile application once and have this application instantly available to different mobile devices.

The proposal was validated in the open source application platform Eclipse considering that some of its tools and run-time environments are aligned with MDA. C++ and HAXE metamodels were specified as ECORE instances and model-to-model transformations were implemented in ATL.

The HAXE Language and, HAXE and C/C++ metamodels are described in the following sections. The HAXE metamodel was previously described in a related work [12]. It is worth mentioning that our research also contributes the definition of an ECORE metamodel for the C/C++ language.

4 THE HAXE LANGUAGE

HAXE is 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 [10].

Reference [9] summarizes the HAXE principles as follows: “support mainstream platforms”, “write once, reuse everywhere”, “always native, no wrapper”, “generated but readable” and “trust the developer”. Some languages allow cross-platform development, but neither their features nor their standard libraries are designed to run on multiple platforms. HAXE was designed from scratch to run and compile for many different platforms.

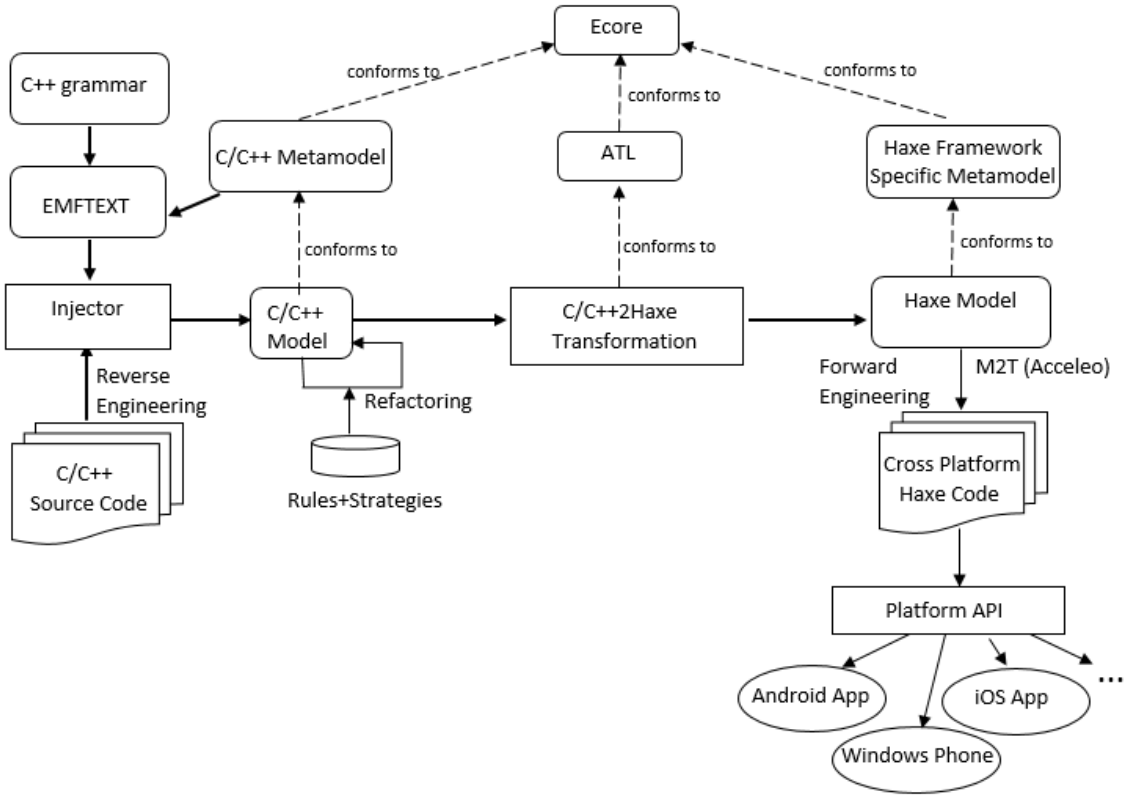

Figure 1. Our contribution: A Migration Process

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.

Since language can be compiled for different platforms, it is useful for a wide variety of applications such as games, web and mobile. In previous work, we show an integration the HAXE with MDD defining an ECORE metamodel of the Cross-Platform Framework HAXE. This metamodel corresponds to the HAXE version 3.1.3. In the context of MDA, the instances of this metamodel are platform specific models (PSM). This metamodel allowed us to integrate HAXE with MDA migration process from Java or C/C++ to mobile platform.

HAXE easily adapts the native behaviors of the different platforms targeted in development projects enabling extremely efficient cross-platform development, ultimately saving time and resources. Currently there are nine supported target languages: Javascript, Neko, PHP, Python, C++, Actionscript3, Flash, Java and, C#. In the context of Mobile App Development, HAXE allows writing mobile apps that target all major mobile platforms and run at native speed. The C++ target allows us to target Android or iOS and OpenFL (www.openfl.org) provides support for creating interfaces using a Flash-like API. OpenFL is a free and open source software framework and platform for the creation of multiplatform applications and video games. OpenFL programs are written in HAXE and may be published to Flash movies, or standalone applications for Microsoft Windows, Mac OS X, Linux, iOS, Android, BlackBerry OS, Firefox OS, HTML5 and Tizen.

Next, we summarizes the main characteristics of the HAXE metamodel.

4.1 The HAXE Metamodel

The HAXE metamodel conforms to ECORE and is partially shown in Figure 2. The main metaclasses of the HAXE metamodel are those that allow specifying an application using HAXE as language. 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 HAXE language includes different kind of types such as class (the types class and interface), function, abstract type, enumeration, and anonymous structures. The full HAXE metamodel can be found in [12].

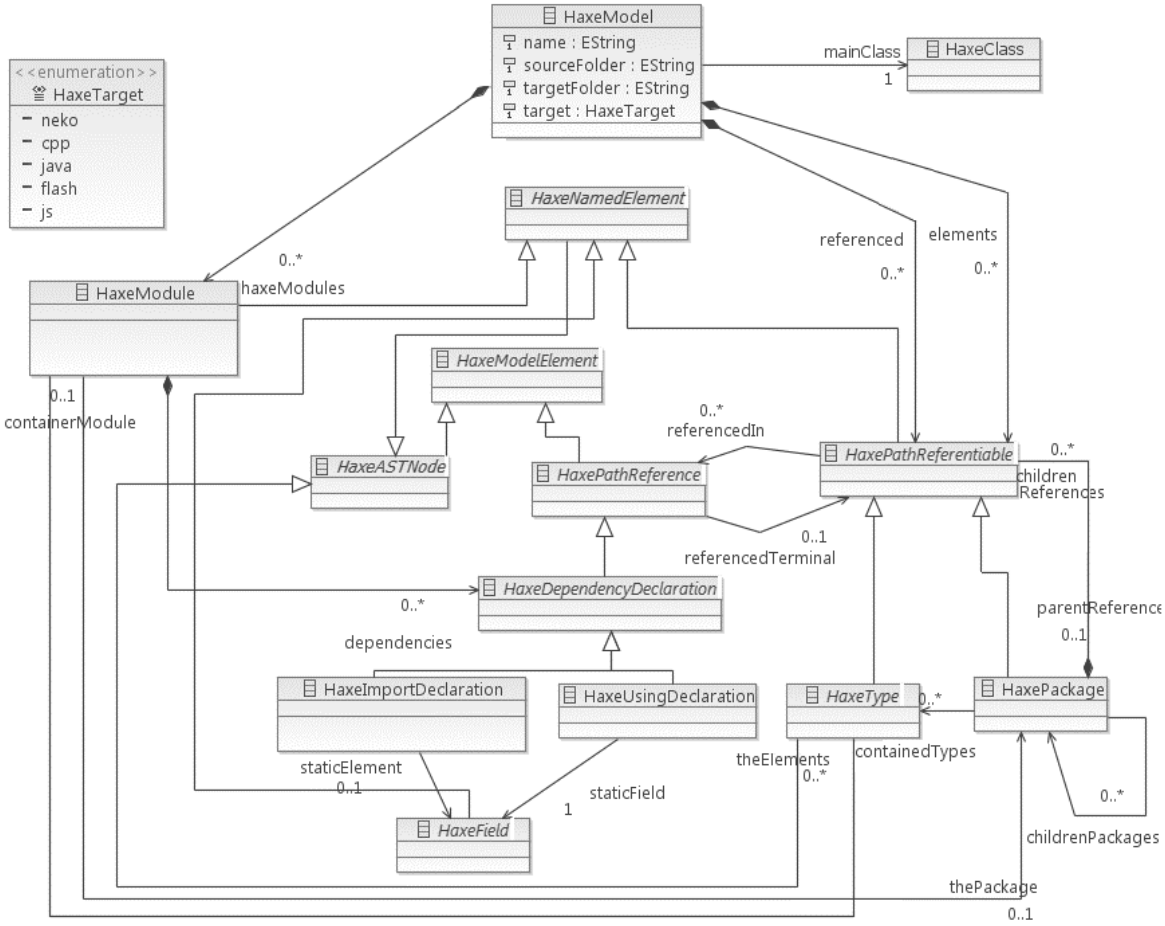

Figure 2. HAXE metamodel

5 THE C++ METAMODEL

The C++ metamodel conforms to ECORE and is partially shown in Figure 3. 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, template members, etc.

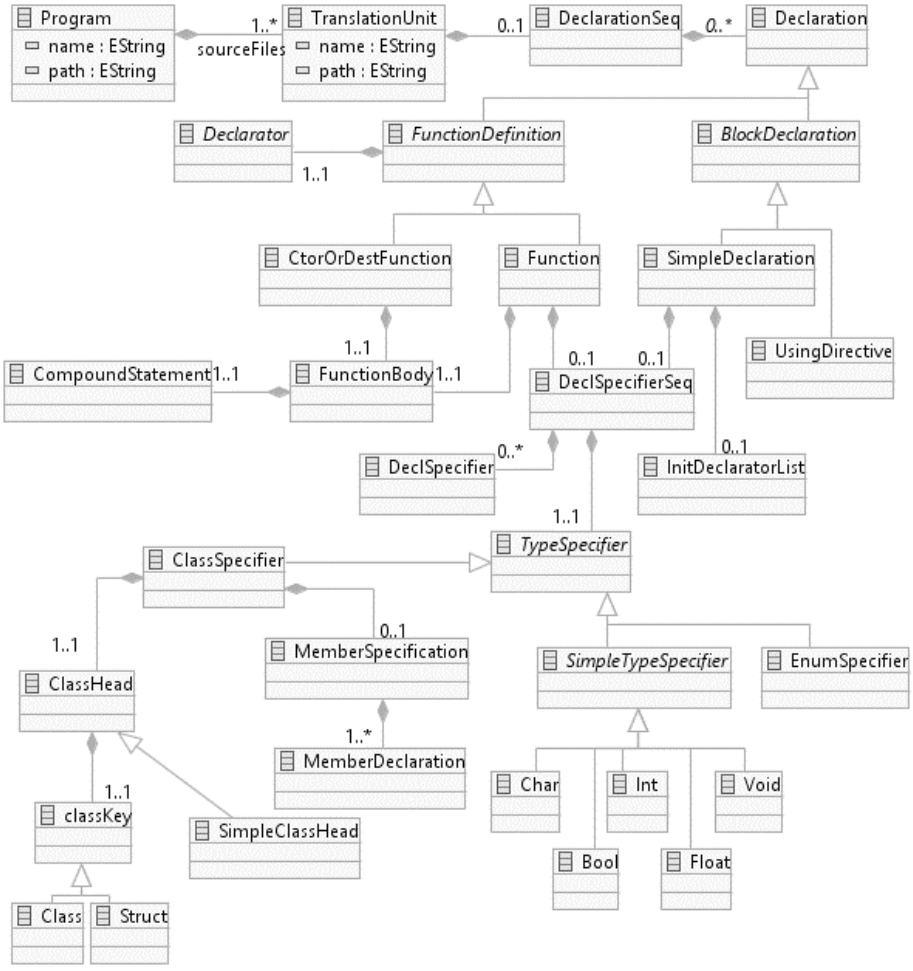

Figure 3. C++ metamodel

6 MIGRATING C/C++ TO MOBILE PLATFORMS

In this section, we describe a migration process from C++ code to different mobile platforms through HAXE. This process starts extracting models from the C++ legacy system. Next, these models are transformed into HAXE models that allow generating HAXE code which can be compiled to multiple target languages in a straightforward way.

To illustrate the migration process we describe a simple case study, how to migrate the C++ code of "the Set of Mandelbrot" to HAXE code. The original application consists of a main class, called Mandelbrot, that is responsible for the calculation of the set of Mandelbrot and serves as entry point for the application. It also includes several classes that collaborate in the realization of the task. The class Mandelbrot is responsible for generating the set and displaying it as image. To perform these tasks, the class depends on Picture and Complex classes, the first is used as a data type that supports the manipulation of digital images. The second class is a data type used to model complex number with their respective operations.

The following subsections describes the steps of the migration process.

6.1 Injection

This first step extracts a complete model of code from C++ code. To carry out this task, we constructed a model injector by using EMFText [17]. To generate this injector, EMFText requires the language metamodel and the concrete syntax specification. In our approach, to generate the injector we first specified the C++ metamodel based on the C++ grammar [27]. Then, we specified the concrete syntax that defines the textual representation of all metamodel concepts. Taking these specifications, the EMFText generator derives an advanced textual editor that uses a parser and printer to parse language expressions to EMF models or to print EMF models to languages expressions respectively.

Figure 4 exemplifies the first step of the process. It partially shows C++ code of Mandelbrot Set that is the input of the model injector and the C++ model of the application in XMI format [30].

6.2 C++ Model Refactoring

This refactoring reorganizes and modifies the syntactic elements to improve the design and to adapt own behavior of the C++ language in order to be directly translate to HAXE, such as remove multiple inheritance, remove multiple class constructors.

The refactoring is implemented as a model-to-model transformation whose source and target models are instances of C++ metamodel. In our case study, refactoring is applied to remove the multiple class constructors of Mandelbrot class.

6.3 Transformation of C++ models into HAXE models

Model-to-model transformations were implemented in ATL that is a model transformation language in the field of MDE developed on top of the Eclipse platform. ATL is a hybrid language that provides a mix of declarative and imperative constructs. 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.

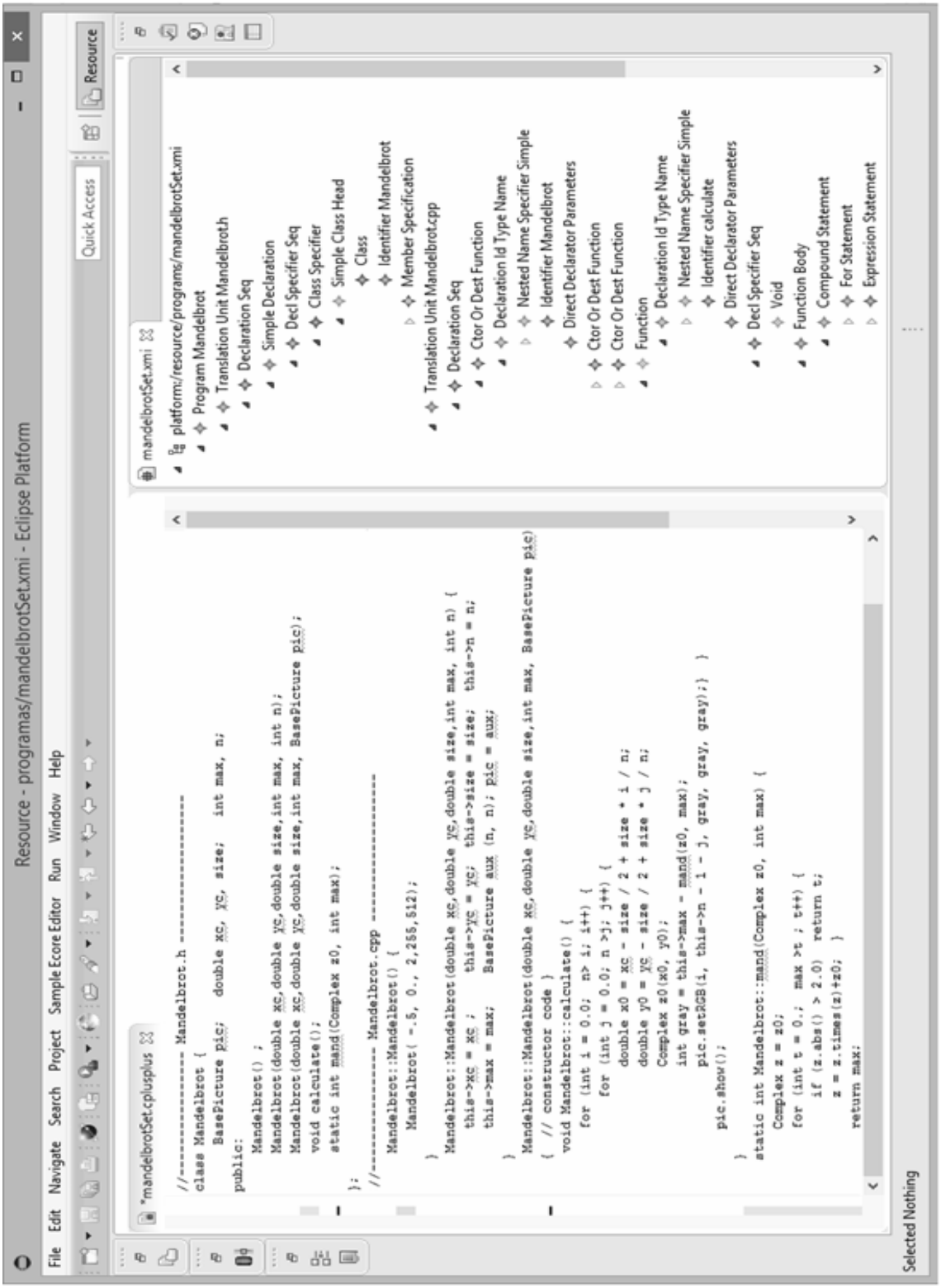

Figure 4.Mandelbrot Class: Code and Model

- A set of rules that defines how source model elements are matched and navigated to create and initialize the elements of the target models.

In our case study, the model-to-model transformation from C++ to HAXE takes as input the model obtained in the previous step and release an HAXE model conforming to the HAXE metamodel. This ATL transformation is partially depicted in Figure 5. The module *CPP2HAXE* that corresponds to the transformation specifies the way to produce HAXE models (target) from C++ models (source). Both source and target models must conform to their respective metamodels. The main rules that carry out the transformation are the followings:

- The rule Program2Model transforms each C++ program into a HAXE model that contains, among others, elements such as *HAXETypes* and *HAXEPackages*.
- The rule TranslationUnit2Module transforms each C++ translation unit into a HAXE module that contains a set of *HAXEDependencyDeclarations*, references a set of elements instances of *HAXETypes*, etc.
- The rule *Classifier2HAXEClass* transforms each C++ classifier, such as class and structure, into a HAXE class whose fields are derived from the classifier members.

All models obtained in this chain of transformations are saved in the interchange format XMI, an OMG standard that combines XML, MOF and UML for integrating tools, repositories, and applications in distributed heterogeneous environments [30].

6.4 HAXE Code Generation

From a model HAXE, it is possible to generate a source code in HAXE by using ACCELEO. 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). The code obtained is partially shown in Figure 6.

7 DISCUSSION

Our work shows the viability of semi-automatic migration processes based on MDD (MDA in particular). Due the fact that the objective of the migration is not only “compile” an application in a mobile platform but also to create a modified version of the application using quality criteria, the process can not be fully automated. Next, we informally compare the model-driven migration process with brute-force re-development migration.

A crucial limitation of our approach is to require preliminary activities that requires time and cost, for instance we need to define metamodels if they do not exist. It is assumed that using a brute-force redevelopment, developers do not need training to write model transformations, however the programming interfaces of these languages generally restrict the kind of transformation that can be performed. In addition, general-purpose languages do not provide a sufficient level of abstraction to specify them. Changes will be difficult to write and understand and, therefore their maintenance is hard. On the contrary, model driven transformations are expressed in specialized languages for that purpose. Model transformations allow developers to concentrate on conceptual aspects of the relations between models and then to delegate the production of the transformation rules. We can consider that the generation of models by model transformations in ATL, aims to generate

models "Correct-by-Construction" with respect to metamodel specifications. Even with these issues, there is still activities done by hand and the migrated application has to be tested.

A general limitation on both processes is the cost of testing due to the fact these activities in general are handled manually. In the context of model driven approaches there is a need to reduce the cost of testing by defining semiautomatic process based on metamodels. Beyond the previous issues, we consider that mobile developers need frequently adapt software components and applications developed in Java or C/C++. Then, model driven migration processes could be reused and the cost of preliminary activities is recovered.

```
CPP=/CPP2HAXE/CPP.ecore
-- @path HAXE=/CPP2HAXE/HAXE.ecore
module CPP2HAXE;
create OUT: HAXE from IN : CPP;
--  HELPERS
helper context CPP!Program def: get_Types(): Set(HAXE!HAXEType)= ... ;
helper context CPP!Program def: get_Packages(): Set(HAXE!HAXEPackage)= ... ;
...
-- RULES
rule Program2Model {
      from   s : CPP!Program
      to     t : HAXE!HAXEModel (
             name <- s.name,
             elements <- s.get_Types(),
             elements <- s.get_Packages(),
             HAXEModules <- s.get_sourceFiles(),
             mainClass <- s.get_mainClass()
             ...
             )
}
rule TranslationUnit2Module {
      from   s : CPP!TranslationUnit
      to     t : HAXE!HAXEModule (
                    name <- s.name,
                    dependencies <- s.get_Dependencies(),
                    theElements <- s.get_Declarations()
                    ...
             )
}
rule Classifier2HAXEClass {
      from   s : CPP!CPPClassifier
      to     t : HAXE!HAXEClass (
                    name <- s.get_Name()
                    ,HAXEFields <- s.getMemberVariables()
                    ,HAXEFields <- s.getMemberMethods()
                    ,generalization <- s.get_superClass()
                    ,implementation <- s.get_superAbstractSuperClasses()
                    -- ...
             )
}
rule EnumSpecifier2HAXEenum {
      from   s : CPP!CPPClassifier
      to     t : HAXE!HAXEEnum ( ...    )
}
...}
```

Figure 5. *CPP2HAXE* Transformation

```
class Mandelbrot
{
   public static function new_Mandelbrot_9 (xc : Float, yc : Float, size : Float,
                                max : Int, pic : BasePicture)  : Mandelbrot {
      var tmp : Mandelbrot = new Mandelbrot();
      tmp.ctor_Mandelbrot_9(xc, yc, size, max, pic);
      return tmp;
   }
   public static function mand (z0 : Complex, max : Int)  : Int {
      ...
   }
   public function calculate ()  : Void
   {
      {var i : Int = 0;
       while (this.n > i){  {  {
         var j : Int = 0;
         while (this.n > j) {  {
            var x0 : Float = this.xc - this.size / 2 + this.size * i / this.n;
            var y0 : Float = this.yc - this.size / 2 + this.size * j / this.n;
            var z0 : Complex = new Complex(x0, y0);
            var gray : Int = this.max - mand(z0, this.max);
            pic.setRGB(i, this.n - 1 - j, gray, gray, gray);
            }  j++;  }
         }; }  i++;  }  };
      pic.show();
   }
   function new () {
   }
   function ctor_Mandelbrot_9 (xc : Float, yc : Float, size : Float, max :
                                Int, pic : BasePicture)
   {
      if (pic.getHeight() != pic.getWidth())
         throw new IllegalImageSize() ;
      this.xc = xc;
      this.yc = yc;
      this.size = size;
      this.n = pic.getWidth();
      this.max = max;
      this.pic = pic;
   }
   public function getPic()  : BasePicture
   {
      return pic;
   }
   public var pic: BasePicture;
   public var xc: Float;
   public var yc: Float;
   public var size: Float;
   public var max: Int;
   public var n: Int;
}
```

Figure 6. HAXE Code

8 CONCLUSIONS

This paper describes an approach for adapting object-oriented software in C/C++ to mobile platforms. A migration process, based on the integration of MDA and the HAXE platform, has been proposed. The main contributions of our approach are the definition of a metamodel for C/C++, the specification of a metamodel transformation between a source C++ metamodel and a target HAXE metamodel, and a generic and extensible migration process for the implementation of cross-platform, multi-device mobile applications from C/C++ code.

The proposal was validated in the open source application platform Eclipse considering that some of its tools and run-time environments are aligned with MDA standards. We believe that our approach provides benefits with respect to processes based only on traditional migration techniques.The migration process can be divided in smaller steps focusing in specific activities, and be automated thanks to the chaining of model transformations. All the involved artifacts can be reused, modified for evolution purposes or extended for other purposes. The metamodel approach enables covering different levels of abstraction and

satisfying several degrees of detail depending on the needs of the migration and is the key for interoperability. All artifacts can be actually represented as models so that there is no information loss during the migration process. Model transformations allow developers to concentrate on the conceptual aspects of the relations between models and delegate the implementation of the transformation.

Our approach has already shown to work on real applications of medium size. We foresee to apply our approach in real industrial projects.

REFERENCES

[1] ACCELEO.*Obeo.* ACCELEO Generator. http://www.eclipse.org/ACCELEO/, 2016.

[2] ADM. Architecture-driven modernization task force.

http://www.adm.org, 2016.

[3] F. Améndola, L. Favre. Adapting CRM systems for mobile platforms: An MDA perspective. *14th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD´13)* (pp. 323-328), Los Alamitos:IEEE Computer Society, 2013.

[4] ATL. Atlas Transformation Language Documentation. http://www.eclipse.org/atl/documentation/, 2015.

[5] J. Bowen, A. Hinze. Supporting mobile application development with model-driven emulation. *Journal of the ECEASST*, *45*, 1–5.

[6] M. Brambilla, J. Cabot, M. Wimmer. *Model-Driven Software Enginneering in Practice*, Synthesis Lectures on Software Engineering. Morgan & Claypool Publishers, 2012.

[7] P. Braun, R. Eckhaus. Experiences on model-driven software development for mobile applications. *Proceedings of Engineering of Computer-Based Systems, IEEE International Conference andWorkshop on the Engineering of computer Base Systems* (pp. 490-493), Los Alamitos: IEEE Computer Society, 2008.

[8] H. Bruneliere, J. Cabot, G. Dupé, F. Madiot. MoDisco: A Model Driven Reverse Engineering Framework. *Information and Software Technology*, *56*(8), 1012–1032, 2014.

[9] N. Cannasse. *HAXE. Too Good to be True?* GameDuell Tech Talk. http://www.techtalk-berlin.de/news/read/nicolas-cannasse-introducing-HAXE/ , 2014.

[10] B. Dasnois. *HAXE 2 Beginner's Guide*. Packt Publishing, 2011.

[11] J. Dehlinger, J. Dixon. Mobile application software engineering: Challenges and research directions. *Proceedings of the Workshop on Mobile Software Engineering* (pp. 29-32). Berlin, Springer-Verlag, 2011.

[12] P. Diaz Bilotto. *Software development for mobile applications through an integration of MDA an HAXE*. (Undergraduate Thesis). Computer Science Department, Universidad Nacional del Centro de la Provincia de Buenos Aires, Argentina, 2015.

[13] P. Diaz Bilotto, L. Favre. 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. (pp.240-268), IGI GLOBAL, 2016.

[14] J. Dunkel, R. Bruns. Model-driven architecture for mobile applications. *Business Information Systems* (LNCS), (vol. 4439, pp. 464-477). Berlin: Springer-Verlag, 2007.

[15] EMF.Eclipse Modeling Framework (EMF). http://www.eclipse.org/modeling/emf/ 2016.

[16] EMF4CPP :What is EMF4CPP? https://code.google.com/archive/p/emf4cpp/ , 2016.

[17] EMFText, www.emftext.org , 2016.

[18] N. Islam, R. Want. Smarthphones: Past, present and future. *Pervasive Computing*, 13(4), 82-92, 2014.

[19] H. K. Kim. Frameworks of process improvement for mobile applications. *Engineering Letters, 16*(4), 550-555, 2008.

[20] D. Kramer, T. Clark, S. Oussena,. MobDSL: A domain specific language for multiple mobile platform deployment. In *Networked Embedded Systems for Enterprise Applications (NESEA), 2010 IEEE International Conference* (pp. 1-7), Los Alamitos: IEEE Press. doi:10.1109/NESEA.2010.5678062, 2010.

[21] MOF. Meta Object Facility (MOF) Core Specification Version 2.4.2, OMG Document Number: formal/2014.

[22] OCL. *OMG Object constraint language (OCL), version 2.4*. http://www.omg.org/spec/OCL/2.4, 2014.

[23] MDA. The Model-Driven Architecture. http://www.omg.org/mda/, 2016.

[24] R. Pérez Castillo, I. García Rodriguez, R. Gómez Cornejo, M. Fernández Ropero, M. Piattini. ANDRIU. A Technique for Migrating Graphical User Interfaces to Android. In *Proceedings of The 25th International Conference on Software Engineering and Knowledge Engineering (SEKE 2013)*(pp. 516-519) Boston: Knowledge Systems Institute, 2013.

[25] 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.

[26] D. Steinberg, F. Budinsky, M. Paternostro, E.Merks. EMF: Eclipse Modeling Framework (2nd ed.). Addison-Wesley, 2009.

[27] B. Stroustrup, *The C++ Programming Language.* Addison-Wesley, Third Edition.1997

[28] C. Thompson, D. Schmidt, H. Turner,J. White. Analyzing Mobile Application Software Power Consumption via Model-Driven Engineering. *Proceedings of PECCS, 2011*, 101–113, 2011.

[29] A. I. Wasserman . Software engineering issues for mobile application development. In *Proceedings of the FSE/SDP workshop on Future of software engineering research* (FoSER '10), (pp. 397-400).New York, NY: ACM, 2010.

[30] XMI. XML Metadata Interchange (XMI) Specification. OMG Document Number: formal/2014-04-04. http://www.omg.org/spec/XMI/2.4.2/PDF/, 2014.