Migrating C/C++ Software to Mobile Platforms in the ADM Context

Liliana Martinez[1], Claudia Pereira[1], Liliana Favre[1,2]

[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

***Abstract* — Software technology is constantly evolving and therefore the development of applications requires adapting software components and applications in order to be aligned to new paradigms such as Pervasive Computing, Cloud Computing and Internet of Things. In particular, many desktop software components need to be migrated to mobile technologies. This migration faces many challenges due to the proliferation of different mobile platforms. Developers usually make applications tailored for each type of device expending time and effort. As a result, new programming languages are 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 Architecture-Driven Modernization (ADM) proposed by the Object Management Group (OMG) can help to manage a huge diversity of mobile technologies. The Architecture-Driven Modernization Task Force (ADMTF) was formed to create specifications and promote industry consensus on the modernization of existing applications. In this work, we propose a migration process from C/C++ software to different mobile platforms that integrates ADM standards with Haxe. We exemplify the different steps of the process with a simple case study, the migration of "the Set of Mandelbrot" C++ application. The proposal was validated in Eclipse Modeling Framework considering that some of its tools and run-time environments are aligned with ADM standards.**

***Keywords* — Architecture-Driven Modernization, Haxe, Migration, Metamodeling, Mobile Platform, Model-Driven Development**

I. Introduction

Today, mobile phones are the most used computing platform worldwide. The wide spread use of mobile computing, that emerged from the integration of cellular technology with the Web, has contributed to opening up opportunities for new paradigms such as 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. Smartphones come with a variety of sensors (GPS, accelerometer, digital compass, microphone, and camera), enabling a wide range of mobile applications in Pervasive Computing.

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. This generates enormous amount of data, which have to be stored, processed and accessed. 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. Mobile Cloud Computing is the combination of Cloud Computing, Mobile Computing and Wireless Network to provide computational resources to mobile users, network operators, as well as cloud computing providers.

There is no single universal definition for Internet of Things. The Oxford Dictionaries defines IoT as "the interconnection via the Internet of computing devices embedded in everyday objects, enabling them to send and receive data". Gartner defines the Internet of Things (IoT) as "the network of physical objects that contain embedded technology to communicate, sense or interact with their internal states or the external environment" [1]. This can generate volumes of real-time data that can be used by enterprises for a variety of business applications. 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.

Connectivity is central in Internet of Things. IoT extends Internet connectivity beyond traditional mobile devices to a diverse range of devices and everyday things that utilize embedded technology to communicate and interact with the external environment, all via Internet. Every object is integrated to interact with each other, allowing for communications between objects, as well as between human and objects, and the control of intelligent systems.

Pervasive computing, Cloud Computing and IoT face similar problems and challenges and smartphones have been one of the greatest facilitators of them. They are pursuing similar use cases, including smart cities, environmental monitoring, agriculture and home automation, and health and monitoring. These technologies will evolve and merge into only one following the vision of Mark Weiser: "The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it" [2]. IoT hardly could exist without the existence of Cloud Computing, as IoT need a network, storage, very cheap analytical possibilities to collect this information and analyze it in a meaningful way. IoT is also based on the same concept of the Pervasive Computing: having sensors and processors in everyday objects to determine their operation.

IoT is possible because thanks to mobile computing, the electronic miniaturization advances allow cutting-edge computing and communication technology to be added into very small objects. Besides mobile computing promoted the globalizations of 3G and 4G networks and today it is promoting 5G. Mobile Computing also facilitated the development of distributed processing to create a network of billions of devices. In summary, Mobile Computing is crucial to harnessing the potential of Pervasive Computing, Cloud Computing and IoT and, without the existence of smartphones these paradigms would not exist.

The development of applications aligned to these new paradigms requires adapting desktop software components to mobile platforms

DOI: 10.9781/ijimai.2017.436

facing 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 [3] 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 [4]. 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) [5]. 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. 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. 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 [6]. Consistency rules are attached to metamodel components by using OCL [7]. 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 Architecture-Driven Modernization (ADM) approach has established a set of solutions for information system modernization. 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" [8]. The OMG ADM Task Force (ADMTF) is developing a set of standards (metamodels) to facilitate interoperability between modernization tools. To date, ADMTF has published the standards such as KDM (Knowledge Discovery Metamodel) and ASTM (Abstract Syntax Tree Metamodel) [9][10].

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. The Eclipse Modeling Framework (EMF) was created for facilitating system modeling and the automatic generation of Java code [11]. 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 [12]. In this context, the subproject Model to Model Transformation (MMT), hosts model-to-model transformation languages. Transformations are executed by transformation engines that are plugged into the Eclipse Modeling infrastructure. For instance, Atlas Transformation Language (ATL) is a model transformation language and toolkit that provides ways to produce a set of target models from a set of source models [13]. Another subproject is Acceleo, which is an implementation of the Model-to-Text (M2T) transformation standard of the OMG for EMF-based models [14]. Acceleo is used in forward engineering processes.

Today, the most complete technology that support ADM is MoDisco, which provides a generic and extensible framework to facilitate the development of tools to extract models from legacy systems and use them on use cases of modernization. As an Eclipse component, MoDisco can integrate with plugins or technologies available in the Eclipse environment [15].

In the Eclipse environment, ADM is integrated with Java language but it is weakly supported for other programming languages such as C++ [11]. In particular, 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. 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 EMF 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 [17]. However, an implementation of a MOF-compliant C++ metamodel would be necessary for other MDD processes (e.g., reverse engineering or software modernization).

In this work, we propose a migration process from C/C++ software to different mobile platforms that integrates ADM standards with Haxe. On the one hand, the process follows model-driven principles: all artifacts involved in the process are viewed as models and the process is viewed as a sequence of model-to-model transformations. 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. It is worth mention that an Ecore metamodel and a model injector for the C++ language are provided.

The article 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 advantages over processes based only on traditional ad-hoc migration techniques increasing productivity due to the automation introduced in the generation of the new software.

The article is organized as follows. Section II presents related work highlighting our contribution. Section III, Background, briefly describes OMG standards for modernization and the Haxe and C++ metamodels. In Section IV, we present the migration process from C++ to mobile platforms. Section V details the different stages of the migration process through of a simple case study. Finally, in Sections VI and VII we present a critical analysis of our approach and conclusions respectively.

II. Related Work

In this section, existing approaches for the development of mobile applications that are related in some way with our approach are described.

Reference [18] 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 developed 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 are described in [19]. The architecture is based on the declarative description of the available services. The authors describe a model-

driven approach for generating almost the complete source code of mobile services.

Reference [20] 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 [21] 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 [22].

A proposal for supporting mobile application development by using models as inputs to an emulator is outlined at [23]. 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 [24] 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.

A reengineering process that integrates traditional reverse engineering techniques such as static and dynamic analysis with MDA is presented at [25]. 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 [26] 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 [27]. 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 [28] describes six major trends affecting future smartphone design and use: personal computers, IoT, multimedia delivery, low power operation, wearable computing and context awareness.

Reference [29] describes a comprehensive tool suite called WebRatio Mobile Platform for model-driven development of mobile applications. It is based on an extended version of OMG standard language called IFML (Interaction Flow Modeling Language) empowered with primitives tailored to mobile systems that enable specification of mobile specific behaviors.

Reference [30] brings out the findings of the experiments carried out to understand the impact of application characteristics, cloud and architecture and the android emulator used, on application performance when the application is augmented to cloud.

Reference [31] 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 available Cloud providers.

A. Our Contribution: The Migration Process

In this article, we describe an original model-driven migration process based on ADM standards (Figure 1). The process includes:

1. Recovering the generic abstract syntax tree (AST) model, instance of GASTM, from code: this step is different for each programing language.
2. Transforming the AST model to a target model in the Haxe platform through an intermediate transformation to obtain the KDM model. The advantage of this intermediate step is that, starting from the KDM model it is possible to obtain high-level models such as UML class diagrams, activity diagrams and use cases diagrams. These models could be refactored and be the starting point for generating code. This step is common for each source language.
3. Generating Haxe from the Haxe model. Then, Haxe allows compilation of programs to multiple target languages such as Javascript, Neko, C++ and Java and to all major mobile platforms.

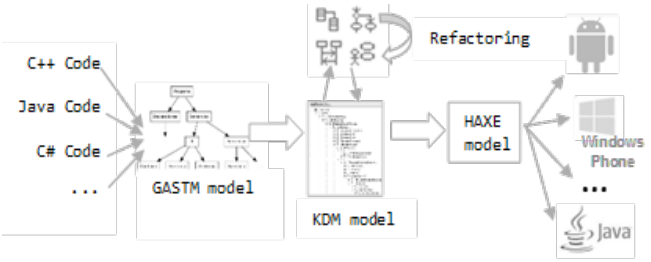

Fig. 1. Our contribution: The Migration Process

III. Background

In this section, we describe OMG standards for modernization. Next, we briefly describe the Haxe language and the Haxe metamodel. Finally, we describe the C++ metamodel that we defined via the Ecore metamodel.

A. Standards for Modernization

The purpose of standardization is to achieve well-defined interfaces and formats for interchange of information about software models to facilitate interoperability between the software modernization tools and services of the adherents of the standards. This will enable a new generation of solutions to benefit the whole industry and encourage collaboration among complementary vendors.

ADMTF is developing a set of standards of which we are interested in KDM and ASTM, both metamodels are defined via MOF. KDM is a metamodel for knowledge discovery in software that allows representing information related to existing software assets, their associations, and operational environments regardless of the implementation programming language and runtime platform. KDM is the foundation for software modernization representing entire enterprise software systems, not just code. ASTM is a specification for modeling elements to express abstract syntax trees. KDM and ASTM are two complementary modeling specifications. KDM establishes a specification that allows representing semantic information about a software system, whereas ASTM establishes a specification for representing the source code syntax by means of AST. ASTM acts as the lowest level foundation for modeling software within the OMG ecosystem of standards, whereas KDM serves as a gateway to the higher-level OMG models.

B. 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 [3].

Reference [32] 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.

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. 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 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 [33]. 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.

In previous work, we show an integration the Haxe with MDD defining an Ecore metamodel of the Cross-Platform Framework Haxe [26]. This metamodel allowed us to integrate Haxe with MDA migration process from Java or C/C++ to mobile platform.

C. 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 [34].

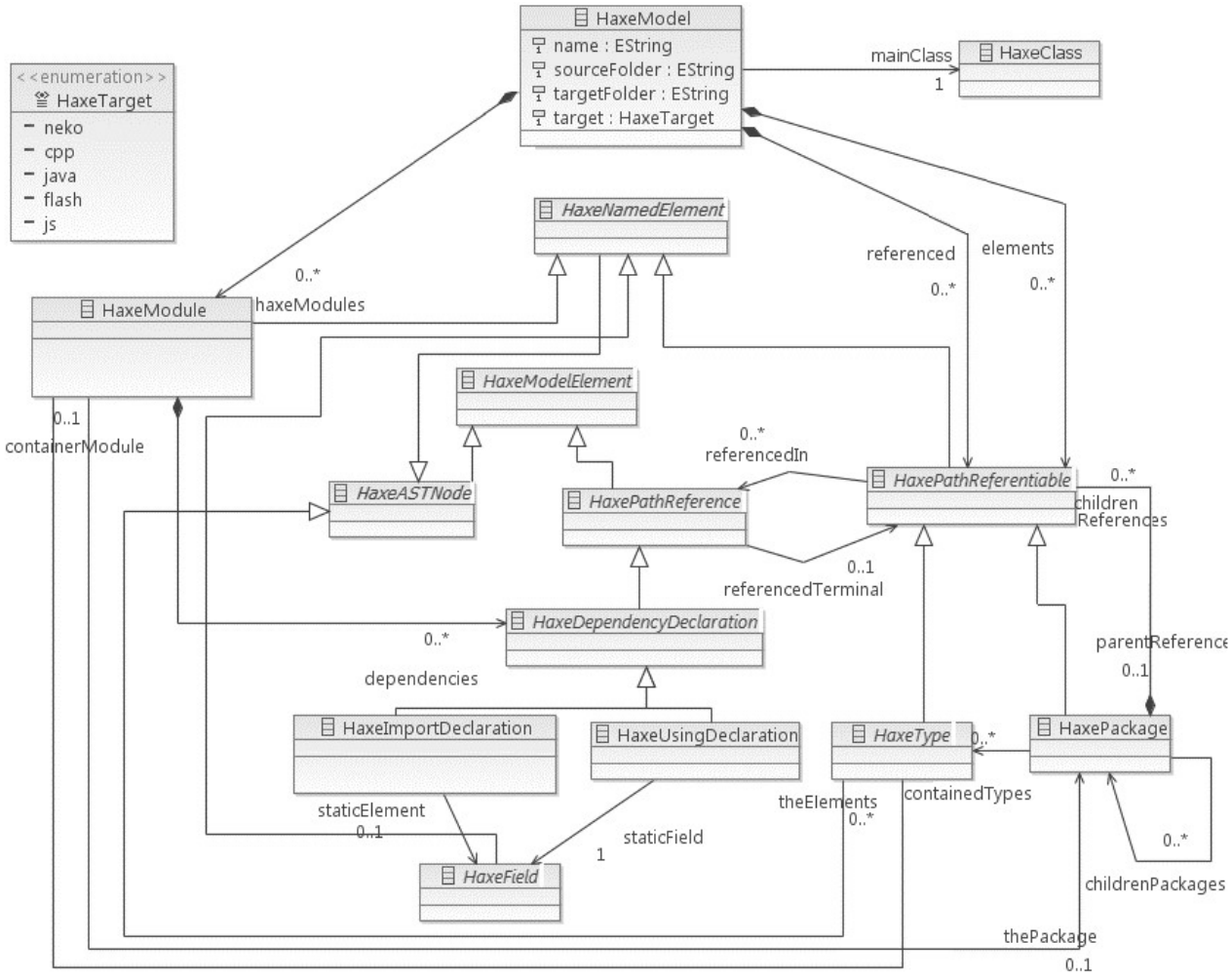

Fig. 2. The Haxe Metamodel

D. 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 and template declarations. A *SimpleDeclaration*, instance of *BlockDeclaration*, 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 a compound statement that owns 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.

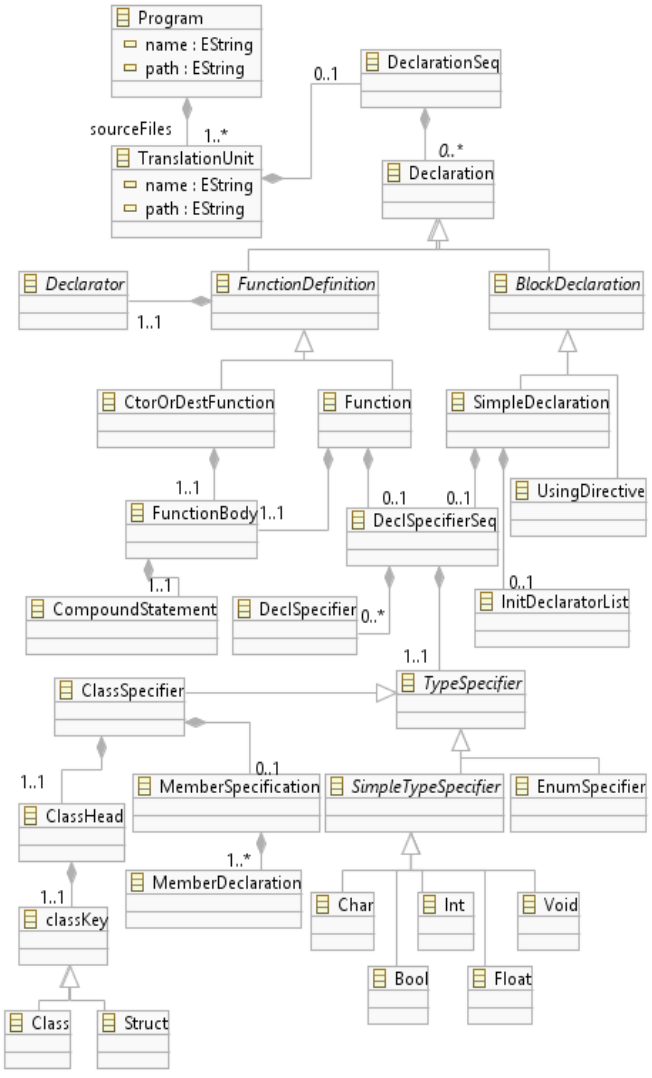

Fig 3. The C++ Metamodel.

IV. Migrating Legacy Code In The ADM Context

In this article, a process to migrate legacy code to Haxe in the ADM context is proposed. The migration process follows model-driven development principles: all artifacts involved in the process can be viewed as models that conforms to Ecore meta-metamodel, 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 4 summarizes the proposed process.

The first step is the reverse engineering of source code to obtain the abstract syntax tree of the code and consists of two stages:

1. Generating the first model of the code by using a model injector. This model conforms to source code metamodel, such as C++ and Java. The obtained model could be refactored to reorganize and modify the syntactic elements to improve the design. The refactoring is implemented as a model-to-model transformation whose source and target models are instances of source code metamodel.
2. Generating the abstract syntax tree model, instance of the GASTM metamodel, from the model obtained in the previous stage by an ATL model-to-model transformation.

In this first step of the process, an injector and a transformation to obtain the GASTM model must be implemented for each language, whereas the sequence of transformations involved in the followings steps of the migration process are independent of the language of the legacy code.

The second step generates the KDM model. This process is carried out by means of an ATL model-to-model transformation that takes as input a model conforming to the GASTM metamodel and produces a model conforming to the KDM metamodel.

The next step is related to an ATL model-to-model transformation that generates a model of the Haxe platform from a KDM model. Then, it is possible to generate Haxe code from the Haxe model by using model-to-text 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 ADM. Eclipse provides implementations of several metamodels such as Java, GASTM and KDM conforming to Ecore metamodel. We contribute with the implementation of C++ and Haxe metamodels, instances of Ecore metamodel. 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.

V. Case Study: Migrating C++ Code 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++ code. 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.

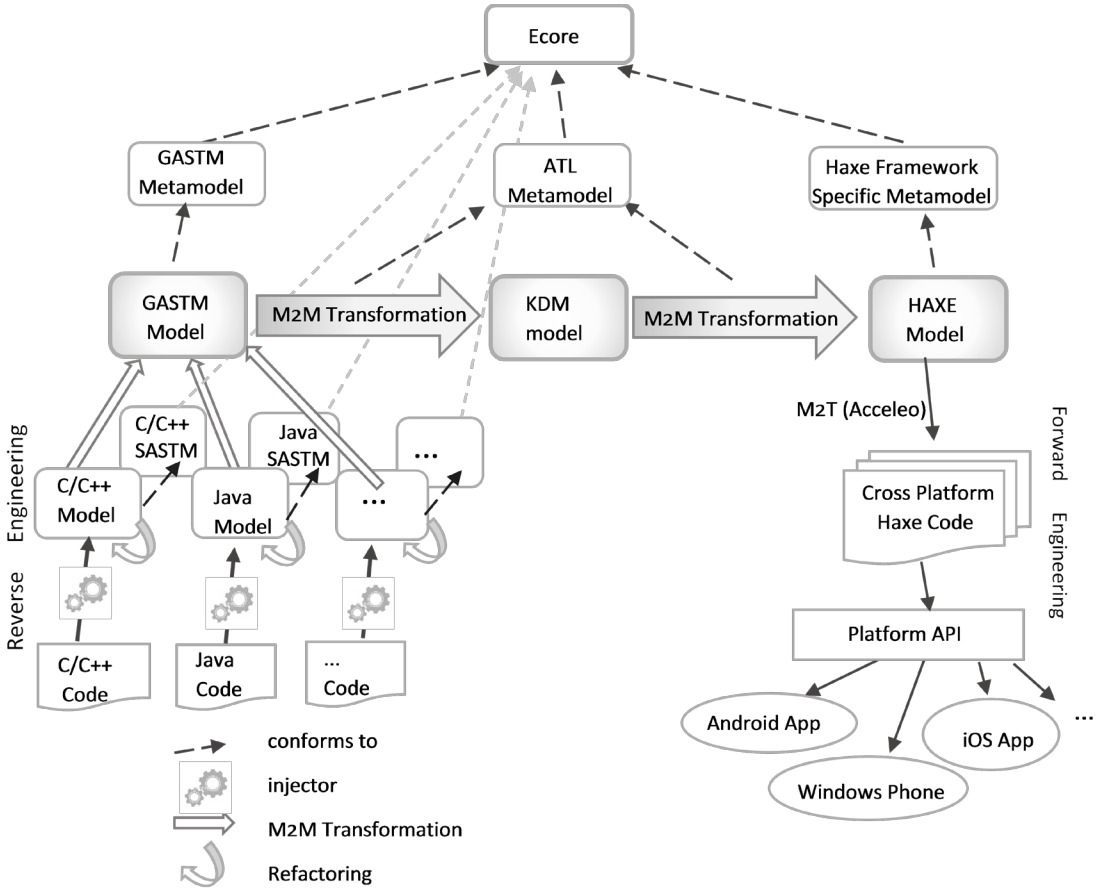

Fig. 4. The Migration Process.

The original application consists of a main class, called Mandelbrot, that is responsible for the calculation of the set of Mandelbrot and displaying it as image. To perform these tasks, the Mandelbrot 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.

A. Obtaining GASTM models from C++ code

Below, we describe the model-to-model transformations to generate generic abstract syntax tree (AST) models from C++ models.

This first transformation extracts an AST model specific to C++ from code. To carry out this task, we constructed a model injector by using EMFText [35]. 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 [36]. 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 5 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.

The second transformation takes as input the model obtained in the previous step and release a generic AST model conforming to the GASTM metamodel. This transformation specifies the way to produce GASTM projects (target) from C++ programs (source). Figure 6 partially shown the obtained GASTM model from the transformation. A project owns files that are obtained from the source files of the program. Each file, instance of *CompilationUnit*, owns fragments such as aggregate type definition, function definition and variable definition obtained from the translation of classes, function definitions, variables, etc.

B. Obtaining Haxe models

The previous transformations are dependent of the legacy code language, that is, for each language, the model injector and the transformation to obtain the generic AST model must be implemented.

In contrast to the previous stage, the sequence of transformations from GASTM models to Haxe models are independent of the language, that is, these transformations are common for all language of the legacy code:

- Transforming a GASTM model to a KDM model.
- Transforming a KDM model to a Haxe model: This transformation takes into account the characteristics of the Haxe language, for example, Haxe does not support neither multiple inheritance nor multiple class constructors.

These transformations are implemented in ATL and specified by means of ATL modules composed of the following elements:

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

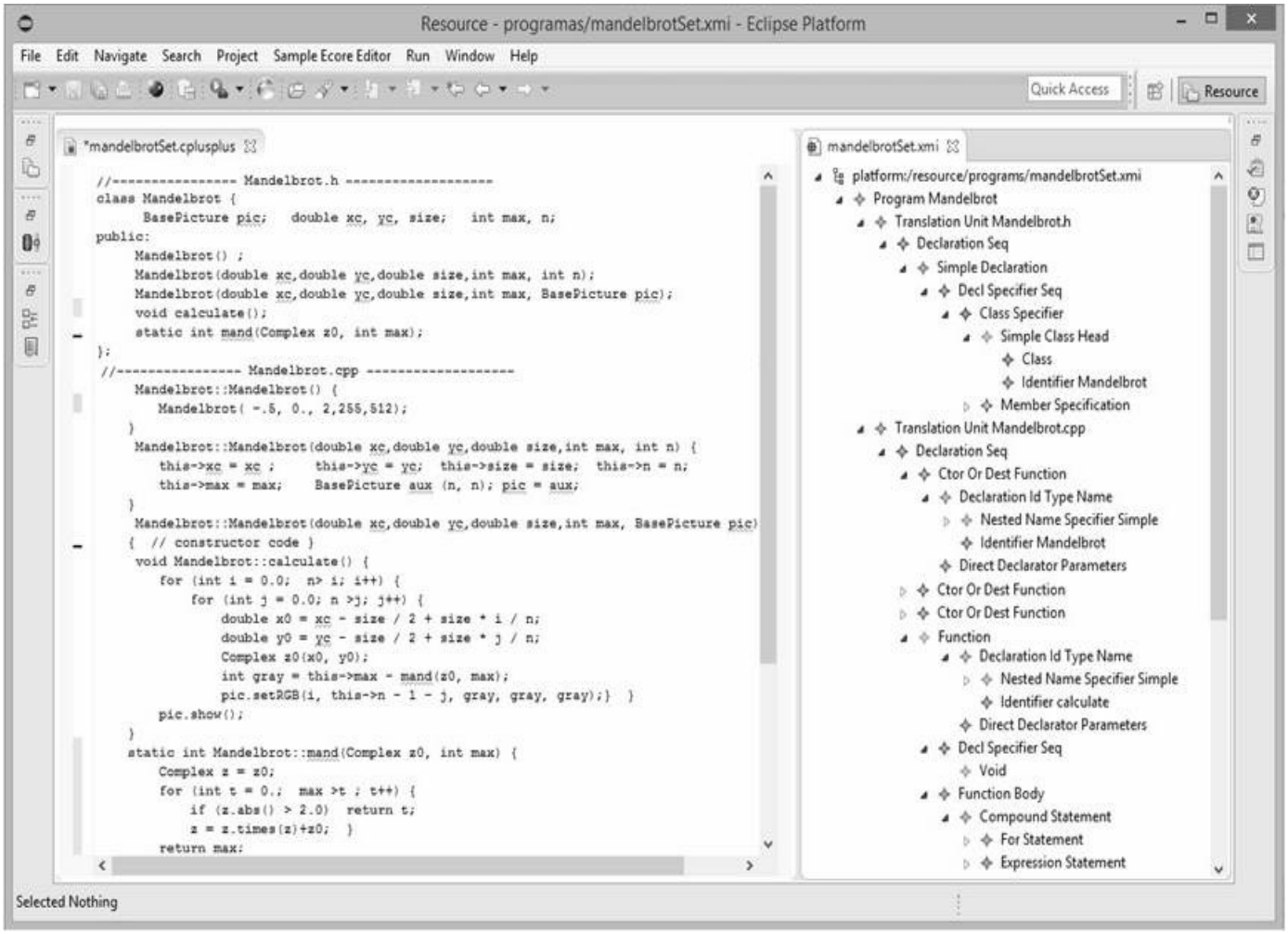

Fig. 5. The Mandelbrot Class: Code and Model.

To exemplify the ATL transformations, the GASTM-to-KDM transformation is partially shown in Figure 7. The module *ASTM2KDM* specifies the way to produce KDM models (target) from GASTM models (source). Some rules that carry out the transformation are the followings:

- The rule *Project2Segment* transforms each ASTM project into a KDM segment that owns models such as CodeModel containing code elements (data types, methods, variables, etc.) and InventoryModel that contains physical artifacts of the existing software system (source file, binary file, etc).
- The rule *AggregateTypeDefinition2ClassUnit* transforms each ClassType into a ClassUnit. Code elements are obtained from the variables and function of the original class; source is obtained from the source code location.
- The *VariableDefinition2StorableUnit* transforms each variable definition into a storable unit.
- The *FunctionDeclaration2MethodUnit* transforms each function declaration into a method Unit.

The resulting model of this transformation is partially shown in Figure 8.

C. Generating Haxe Code

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 9.

VI. Critical Analysis of Our Approach

This section analyzes critically our approach. First, we discuss advantages and limitations with regard to the application of an ADM approach. Next, we analyze the proposed migration process.

With regard to ADM, one of the well-known benefits is the increment of development productivity due to automation introduced in the generation of artifacts of the final system.

When a migration process is defined, it is important to consider that it is independent of the source and target technologies. In ADM, the intermediate models act as decoupling elements between source and target technologies. The independence is achieved with injectors and, M2M and M2T transformations. Besides, in a transformation sequence, models are an extension point to incorporate new stages.

ADM is based on MOF-like metamodeling that is a powerful approach for interoperability. For instance, a reverse engineering process recovers knowledge that must be represented using any formalism. For example, the XML technology seeks to solve the problem of expressing structured data in an abstract and reusable

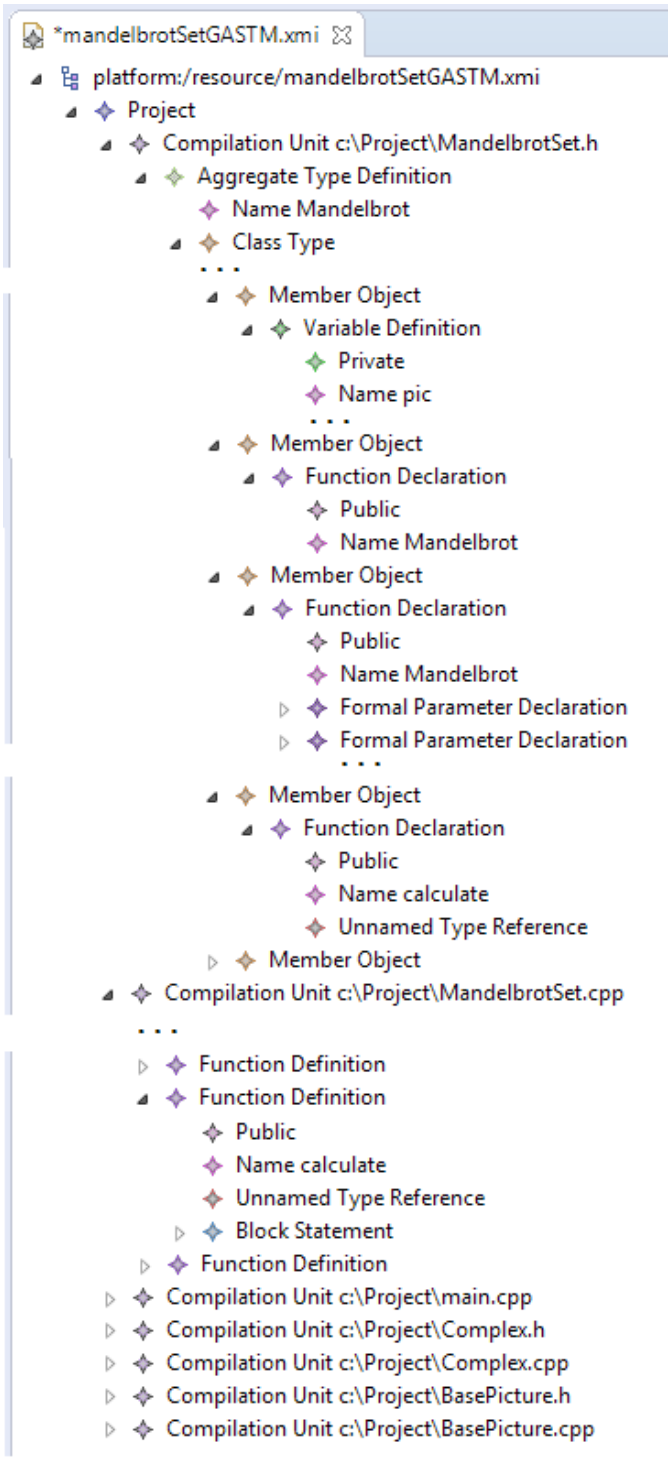

Fig. 6. The Mandelbrot Set: GASTM Model.

way. Metamodeling languages as MOF or its implementation called Ecore will outperform XML for its expressive power and the existence of powerful model transformation languages to implement transformations that are required at the different stages of the migration process. MOF-like metamodeling also includes the possibility to attach OCL restrictions to complete the model specification. In addition, MOF-like metamodels allow a clear separation of abstract and concrete syntax and thus, associate different notations for a model.

On the other hand, we can mention the following limitations of ADM. There are no available open and free injectors for different languages and it is often necessary to implement them. As regard

```
-- @path CPP=/CPP2ASTM/Metamodelos/gastm.ecore
-- @nsURI KDM=http://www.eclipse.org/MoDisco/kdm/action
module ASTM2KDM;
create OUT : KDM from IN : ASTM;
--------------- RULES  --------------------
rule Project2Segment{
 from    src : ASTM!Project
 to   kdmSegment : KDM!Segment (
     model <- kdmModel
    ,model <- sourcesModel ,
    ,model <- externalModel ...
  )
  ,kdmModel : KDM!CodeModel(
   codeElement<- src.files->collect(f|f.fragments)...)
  ,sourcesModel : KDM!InventoryModel (
   name <- 'source references'
   ,inventoryElement <- src.files)
  ,externalModel : KDM!CodeModel (...)
}
rule AggregateTypeDefinition2ClassUnit {
 from
  src: ASTM!AggregateTypeDefinition
   (src.aggregateType.oclIsTypeOf(ASTM!ClassType)
 to
  tgt: KDM!ClassUnit(
   name<-src.typeName.nameString
   ,codeElement<- src.get_VariableDefinition()
   ,codeElement<- src.get_FunctionDefinition()
   ,codeElement<- src.get_FunctionDeclaration()
                   ->collect (d|d.defRef)
   , ...
   ,source <- file
   )
  file: KDM!SourceRef (...)
}
rule VariableDefinition2StorableUnit {
 from
  src: ASTM!VariableDefinition
 to
  tgt: KDM!StorableUnit (
   name <-src.identifierName.nameString
   ,type <- src.getType()
   ,kind <-  src.accessKind
       ... )
}
rule FunctionDefinition2MethodUnit {
 from
  src: ASTM!FunctionDefinition
 to
  tgt: KDM!MethodUnit(
   codeElement <- signature
   ,name <-src.identifierName.nameString
   ,codeElement <-  body
  )
  , signature: KDM!Signature (
    name<- src.identifierName.nameString
   ,parameterUnit<- src.formalParameters
   ,parameterUnit<- if (src.returnType.oclIsUndefined())
                then OclUndefined else returnType endif
  )
  ,returnType: KDM!ParameterUnit (
   type <- src.returnType
   ,kind <- 'return'
  )
  ,body: KDM!BlockUnit ( ... )
}
rule FunctionDeclaration2MethodUnit {
 from
  src: ASTM!FunctionDeclaration
 to
  tgt: KDM!MethodUnit(
   name <-src.identifierName.nameString
   ,codeElement <- signature
  )
 , signature: KDM!Signature (    ... )
}
```

Fig 7. The ASTM2KDM Transformation.

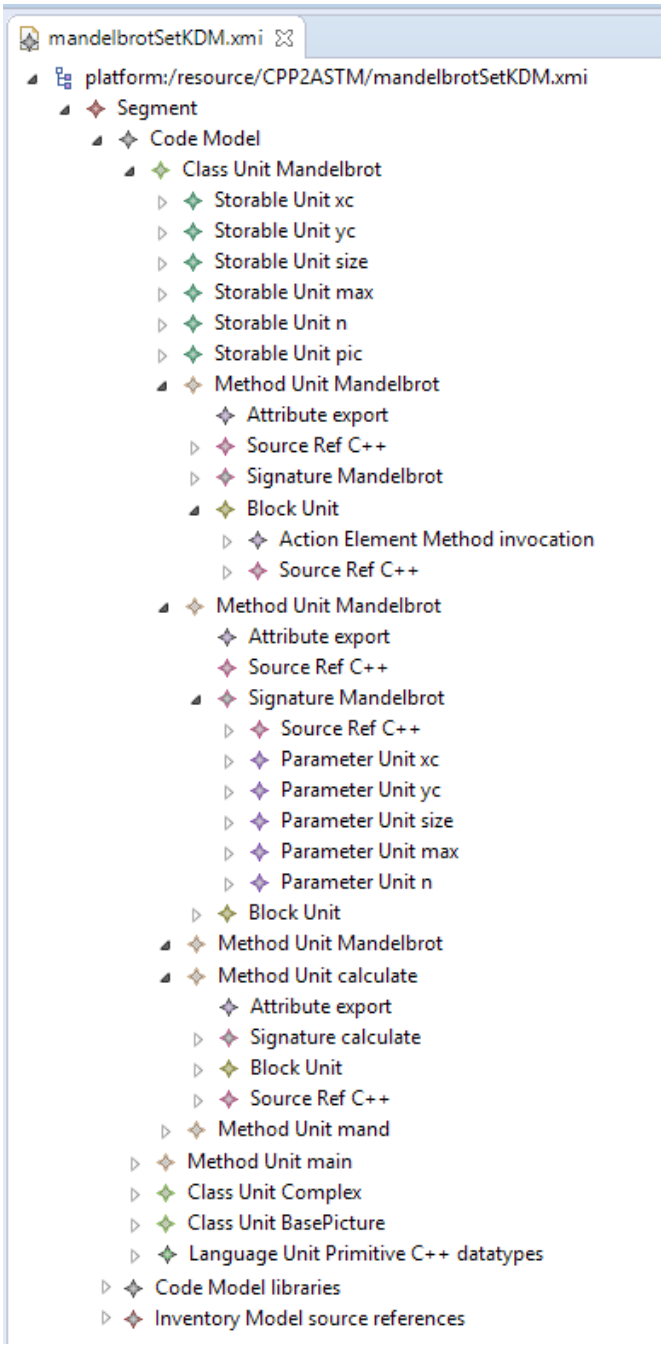

Fig 8. The Mandelbrot Set: The KDM Model.

```
class Mandelbrot
{
 public static function new_Mandelbrot_9 (xc: Float,
      yc: Float, siz : 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 (n > i){      {       {
    var j : Int = 0;
    while (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.get_height() != pic.get_width())
   throw new IllegalImageSize() ;
  this.xc = xc;
  this.yc = yc;
  this.size = size;
  this.n = pic.get_width();
  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;
}
```

Fig 9. The Mandelbrot Set: Haxe Code.

KDM metamodel, it is designed to support interoperability between modernization tools. For some aspects such as user interfaces, KDM provides a reduced level of detail and does not allow to express common concepts to many technologies. This requires to extend the metamodel (which leaves to conform to a standard) or define stereotypes. With regard to model transformation languages, they suffer limitations due to it does not allow defining complex data structures and algorithms.-

Regarding the proposed migration process, we show the viability of semi-automatic migration processes based on ADM. 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 redevelopment migration.

Our approach involves preliminary activities that require time and cost, for instance we need to define metamodels and transformations if they do not exist. It is assumed that using a brute-force redevelopment, developers do not need training to write metamodels and model transformations. However, model transformations allow developers to concentrate on conceptual aspects of the relations between models and then to delegate most of the migration process to the transformation rules, whereas in the brute-force redevelopment, developers need to migrate by hand the legacy systems, making over and over again the same task. It is worth mentioning that the generation of models by ATL transformations, aims to generate models "Correct-by-Construction" with regard to metamodel specifications.

A general limitation on migration processes is the cost of testing due to these activities in general are handled manually. In the context of model-driven approaches, the cost of testing could be reduced by defining semiautomatic process based on metamodels.

Beyond the previous issues, we consider that mobile application developers need frequently adapt software components and applications developed in languages such as Java or C/C++. Then, model-driven migration processes could be reused and the cost of preliminary activities is recovered.

VII. Conclusion

This paper describes an approach for adapting object-oriented software in C/C++ to mobile platforms. A migration process, based on the integration of ADM and the HAXE platform, has been proposed. The main contributions of our approach is a sequence of transformations implemented to migrate C++ code to mobile platform based on ADM standards allowing reusing both the transformations and the generated models. Besides, we provide a definition of the C++ metamodel via the Ecore metamodel and the implementation of an injector to obtain the first model from C++ code.

We believe that our approach provides benefits with regard 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.

The proposal was validated in the open source application platform Eclipse considering that some of its tools and run-time environments are aligned with ADM standards. 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] Gartner, http://www.gartner.com/it-glossary/internet-of-things/, 2016.

[2] M. Weiser. The Computer for the 21st Century, Scientific American, Vol. 265 No.9, pp. 66-75, 1991.

[3] B. Dasnois. *Haxe 2 Beginner's Guide*. Packt Publishing, 2011.

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

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

[6] *Meta Object Facility (MOF) Core Specification*, Version 2.5, OMG Document Number: formal/2015-06-05. Available: http://www.omg.org/spec/MOF/2.5/

[7] *OMG Object constraint language* (OCL), version 2.4. OMG Document Number: formal/2014-02-03. Available: http://www.omg.org/spec/OCL/2.4

[8] ADM. Architecture-driven modernization task force. http://www.adm.org, 2016.

[9] *Knowledge Discovery Meta-Model* (KDM), OMG Document Number: formal/2011-08-04. Available: http://www.omg.org/spec/KDM/1.3, 2011.

[10] *Abstract Syntax Tree Metamodel*, version 1.0, OMG Document Number: formal/2011-01-05. Available: http://www.omg.org/spec/ASTM, 2011.

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

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

[13] ATL. Atlas Transformation Language Documentation. http://www.eclipse.org/atl/documentation/, 2016.

[14] Acceleo. Obeo. Acceleo Generator. http://www.eclipse.org/Acceleo/, 2016.

[15] MoDisco. https://eclipse.org/MoDisco/, 2016.

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

[17] *XML Metadata Interchange (XMI) Specification*, OMG Document Number: formal/2015-06-07. Available: http://www.omg.org/spec/XMI/2.5.1

[18] P. Braun, R. Eckhaus, "Experiences on model-driven software development for mobile applications" in *Proc. of Engineering of Computer-Based Systems, IEEE International Conference and Workshop on the Engineering of computer Base Systems ,* Washington, DC, USA, IEEE Computer Society, 2008, pp. 490-493.

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

[20] H. K. Kim, Frameworks of process improvement for mobile applications. *Engineering Letters*, 16(4), 2008, 550-555. Available: http://www.engineeringletters.com/issues_v16/issue_4/EL_16_4_13.pdf

[21] J. Dehlinger, J. Dixon. "Mobile application software engineering: Challenges and research directions" in *Proc. of the Workshop on Mobile Software Engineering,* Berlin, Springer-Verlag, 2011, pp. 29-32.

[22] C. Thompson, D. Schmidt, H. Turner,J. White, "Analyzing Mobile Application Software Power Consumption via Model-Driven Engineering", *Advances and Applications in Model-Driven Engineering,* Chapter 16, IGI GLOBAL, 2014, pp.342-366.

[23] J. Bowen, A. Hinze, "Supporting mobile application development with model-driven emulation", *Journal of the ECEASST*, vol. 45, 2011, pp. 1–5.

[24] D. Kramer, T. Clark, S. Oussena, "MobDSL: A domain specific language for multiple mobile platform deployment", in *IEEE Int. Conf. on Networked Embedded Systems for Enterprise Applications* (NESEA), 2010, pp. 1-7.

[25] F. Améndola, L. Favre, "Adapting CRM systems for mobile platforms: An MDA perspective", *14th ACIS Int. Conf. on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing* (SNPD´13), 2013, pp. 323-328.

[26] 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.

[27] R. Pérez Castillo, I. García Rodriguez, R. Gómez Cornejo, M. 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 *Proc. of The 25th International Conference on Software Engineering and Knowledge Engineering* (SEKE 2013), Boston: Knowledge Systems Institute, pp. 516-519.

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

[29] R. Acerbis, A. Bongio, M. Brambilla, S. Butti, Model-Driven Development Based on OMG's IFML with WebRatio Web and Mobile Platform. *Engineering the Web in the Big Data Era*. Lecture Notes in Computer Science, vol. 9114, 2015, pp. 605-608.

[30] P. Joshi, A. Nivangune, R. Kumar, S. Kumar, R. Ramesh, S. Pani, A. Chesum, Understanding the Challenges in Mobile Computation Offloading to Cloud through Experimentation. In *2nd ACM Int. Conf. on Mobile Software Engineering and Systems* (MOBILESoft), 2015, pp.158-159.

[31] J. Ejarque, A. Micsik, R. M. Badia, "Towards Automatic Application Migration to Clouds", in *IEEE 8th Int. Conf. on Cloud Computing* (CLOUD), 2015, pp. 25-32.

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

[33] OPEN FL, http://www.openfl.org/, 2016.

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

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

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

Liliana Martinez has a Master degree in Software Engineering. She is an assistant professor in Computer Science area at the Universidad Nacional del Centro de la Provincia de Buenos Aires (UNCPBA), Tandil, Argentina. She is a member of the Software Technology Group, which develops its activities at the INTIA Research Institute at the UNCPBA. Her research interests are focused on system modernization, reverse engineering in the ADM context in particular. She has published book chapters, journal articles and conference papers. She has been member of the program committee of international conferences related to software engineering.

Claudia Pereira is an assistant professor in Computer Science area at the Facultad de Ciencias Exactas, Universidad Nacional del Centro de la Provincia de Buenos Aires (UNCPBA), Tandil, Argentina. She is a member of the Software Technology Group, which develops its activities at the INTIA Research Institute at the UNCPBA. She has a Master degree in Software Engineering from Universidad Nacional de La Plata, Argentina. Her research interests are focused on system modernization. She has published book chapters, journal articles and conference papers. She has been member of the program committee of international conferences related to software engineering.

Liliana Favre is Liliana Favre is a full professor of Computer Science at Universidad Nacional del Centro de la Provincia de Buenos Aires in Argentina. She is also a researcher of CIC (Comisión de Investigaciones Científicas de la Provincia de Buenos Aires). Her current research interests are focused on model driven development, model driven architecture and formal approaches, mainly on the integration of algebraic techniques with MDA-based processes. She has been involved in several national research projects about formal methods and software engineering methodologies. Currently she is research leader of the Software Technology Group at Universidad Nacional del Centro de la Provincia de Buenos Aires. She has published several book chapters, journal articles and conference papers. She has acted as editor of the book UML and the Unified Process. She is the author of the book Model Driven Architecture for Reverse Engineering Technologies: Strategic Directions and System Evolution.