Toward IFVM Virtual Machine: A Model Driven IFML Interpretation

Sara Gotti and Samir Mbarki
MISC Laboratory, Faculty of Sciences, Ibn Tofail University, BP 133, Kenitra, Morocco

Keywords: Interaction Flow Modelling Language IFML, Model Execution, Unified Modeling Language (UML), IFML Execution, Model Driven Architecture MDA, Bytecode, Virtual Machine, Model Interpretation, Model Compilation, Platform Independent Model PIM, User Interfaces, Front End.

Abstract: UML is the first international modeling language standardized since 1997. It aims at providing a standard way to visualize the design of a system, but it can't model the complex design of user interfaces and interactions. However, according to MDA approach, it is necessary to apply the concept of abstract models to user interfaces too. IFML is the OMG adopted (in March 2013) standard Interaction Flow Modeling Language designed for abstractly expressing the content, user interaction and control behaviour of the software applications front-end. IFML is a platform independent language, it has been designed with an executable semantic and it can be mapped easily into executable applications for various platforms and devices. In this article we present an approach to execute the IFML. We introduce a IFVM virtual machine which translate the IFML models into bytecode that will be interpreted by the java virtual machine.

1 INTRODUCTION

The software development has been affected by the apparition of the MDA (OMG, 2015) approach. The trend of the 21st century (BRAMBILLA et al., 2014) which has allowed developers to build their system starting with abstract models, as well, to align with the imposed changes and respond to industry requirements related to cost and time to market. It is not expensive to update a system from its high level representation which is the abstract models than to update the source code after the targeted system was built.

UML (OMG, 2005) is the most successful software modeling language, standardized by Object Management Group (OMG) since 1997. It permits abstract concepts presentation that allows analysis and program description from textual syntax to graphical diagrams.

A new model driven approach was developed which is the execution of UML models without code generation. It is based on the compilation or interpretation of UML models.

There are solutions that execute models as those based on unified modeling languageUML. They define a subset of behavioral models whose semantics are executable. The OMG group proposed a fundamental standard fUML (OMG, 2011), which is a subset of UML that contains the most relevant part of class diagrams for modeling the data structure and activity diagrams to specify system behavior; it contains all UML elements that are helpful for the execution of the models.

UML (OMG, 2005) is the best language used to easily and successfully model many things but it may not be suited for all domains, for example when it comes to design user interfaces and user interactions, there has been no standard way to model user interfaces.

Therefore, a solution was adopted by the OMG group in March 2013 which is the interaction flow modeling language IFML (OMG, 2015), a concept of modeling language that allows the system modeler to express the content, user interaction and control behavior of application front-end.

IFML (OMG, 2015), permits a high level abstract representation of the different front end aspects such as content, interface organization, interaction and navigation options, and connection with the business logic and the presentation style without considering the implementation-specific issues.

IFML is a platform independent language that has been elaborated with an executable mind. IFML executability expresses the execution semantics and

Gotti, S. and Mbarki, S.
Toward IFVM Virtual Machine: A Model Driven IFML Interpretation.
DOI: 10.5220/0005986102200225
In *Proceedings of the 11th International Joint Conference on Software Technologies (ICSOFT 2016) - Volume 1: ICSOFT-EA*, pages 220-225
ISBN: 978-989-758-194-6

maps the PIM model of the executable IFML with behavior in the selected specific user interface platform. The executability is found through model transformations and code generators that make models easily mapped into executable applications in various platforms and devices.

As depicted in fig. 1, two main approaches were defined for executing models: Code generation and Model implementation that has two different forms of execution which are model interpretation and model compilation.

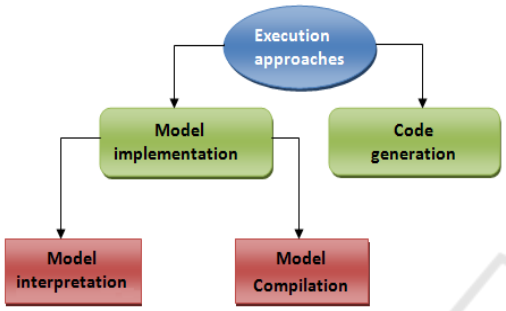

Figure 1: Types of model implementation.

In this paper, we opted for model interpretation approach that aims to execute IFML models to produce the bytecode equivalent.

The general plan of this article revolves around the second part, which briefly presents the interaction flow modeling language IFML, while the third part is devoted to IFML executability, and description of the kind of computation that an IFML model specifies. The fourth part describes the bytecode form that has been chosen as the target of IFML execution. The IFML virtual machine process will be shown in section five. The sixth section will be devoted to the related works. It will be followed by a conclusion in the seventh section indicating the objectives status of the performed execution.

2 INTERACTION FLOW MODELING LANGUAGE IFML

The user interface design became a more complex task where many aspects intersect: graphic design and aesthetics, enterprise visual identity, interaction design, usability, multi-screen support, offline-online usage, integration with backend business logic and data, and coherence with the enterprise organization models (BRAMBILLA et al., 2014).

Figure 2 shows the UI design problems.

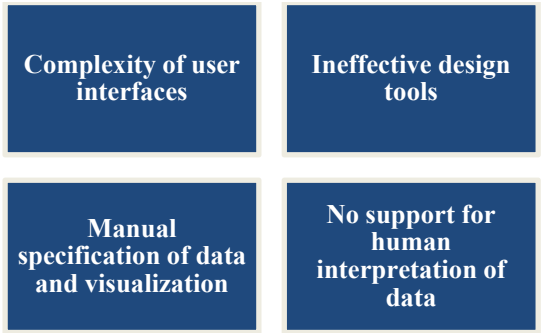

Figure 2: UI design problems.

Furthermore, the execution of software systems in multiple computing platforms is essentially needed for today's technologies, and there is still no support for multiple interfaces on multiple computing platforms.

In order to prevent these problems to affect the software system development, the interaction flow modeling language was developed in 2012 and 2013 under the leadership of WebRatio. It was inspired by the vast experience of 10 years of WebML (WebRatio, 2008) that was dedicated to the production of data-intensive Web applications. IFML was adopted by the Object Management Group (OMG) in March 2013.

Figure 3 describes the UI design solution, which is the IFML.

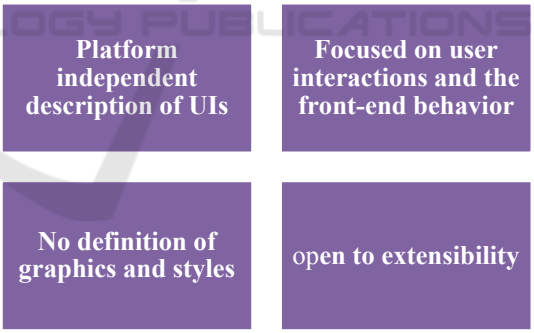

Figure 3: UI design solution: IFML.

IFML supports the platform-independent description of applications front end deployed on systems such as desktop computers, laptops, PDAs, mobile phones, and tablets regardless of the technological details of their implementation, thing that enables the communication of interface and interaction design to nontechnical stakeholders, enabling the early validation of requirements.

Each interface from UIs will be designed in IFML as a ViewContainer; a ViewContainer contains ViewComponents that enable content

display and data entry. A view component can have input and output parameters.

The user interactivity is expressed by Events in association with ViewContainers and View-Components. The event triggers actions that can affect the interface; the reaction is denoted by InteractionFlows that connect the event to a View-Container, aViewComponent, or an Action. An input-output dependency between elements can be specified through parameter bindings associated with navigation flows or through data flows that only describe data transfer.

3 IFML EXECUTABILITY

The Model Driven Approach has appeared due to the constraints related to the productivity and time to market. It prompted the developers to focus on modeling in the hopes of improving productivity by increasing the levels of abstraction, automation, and analysis.

According to MDA (OMG, 2015), developers start with modeling their systems, after that they generate an executable code (C, C++, Ada, Java, Forth, even VHDL) automatically from their models or directly execute them in order to generate the equivalent binary.

For executing models, we need to define a complete model that outlines the structure and the behavior required enough to be executed.

IFML executability defines the execution semantics of IFML language. It is an informal description of the kind of computation that an IFML model specifies. It defines the computation of values shown in the views.

With the execution semantics we can reach any executable behaviors in a specific user interface platform from the platform independent IFML.

A user interacts with an interactive application within a view and produces events that affect the software system which reacts by changing the status of views and executing actions that signal another event and that are what the execution semantics of IFML. The action functioning is not described by the semantics of IFML.

Triggering events is one of execution semantics aspects. With IFML execution semantics the Event computation is treated after an event came out, it can update the state of the ViewContainers and the ViewComponents.

In fact, there are two forms of triggering events produced by a user: event in a ViewContainer that affect another ViewContainer by a NavigationFlow and an event that affect an element inside the same ViewContainer.

Events have effects on the state of user interfaces. A state of interface gather visible Viewcontainers, active ViewComponents, and events.

AViewContainer is visible when it respects its visibility turn according to a composition model that contains the entire ViewContainers of the system.

A ViewComponent is active if its ViewContainer is visible and its input parameters values are available.

4 BYTECODE INTERMEDIATE REPRESENTATION

Bytecode is a small and easy-to-understand set of instructions designed for efficient execution by a software interpreter.

It is a binary code that is usually processed by a program, and then converted by a virtual machine into a specific machine instructions understood by a computer's processor. It includes instructions that are executable by a virtual machine interpreter.

Bytecode is compiling source code result of a language that comes with a virtual machine for each platform thing that make the source language statements compiled only once and then run on any platform.

The most famous language today that uses the bytecode and virtual machine approach is Java.

The Java compiler translates the java source code into bytecode (class files), which can be executed after by the Java Virtual Machine interpreter.

A bytecode, result from a source code compilation, can be run in java virtual machine even if the source code is not written in java.

One of the advantages of Java bytecode is that it can be recompiled at each particular system platform by a just-in-time compiler. Usually, this will enable the Java program to run faster.

It is an output programming language implementation used as an intermediate representation which is hardware and operating system independent, thing that makes interpretation easy and fast.

After a Java file is compiled, all references to variables and methods are stored in the constant pool table as a symbolic reference in the java class file.

The constant pool table contains many structures of various string constants, class and interface names, field names, and other constants.

Figure 4: example of java class file (bytecode) from java source code.

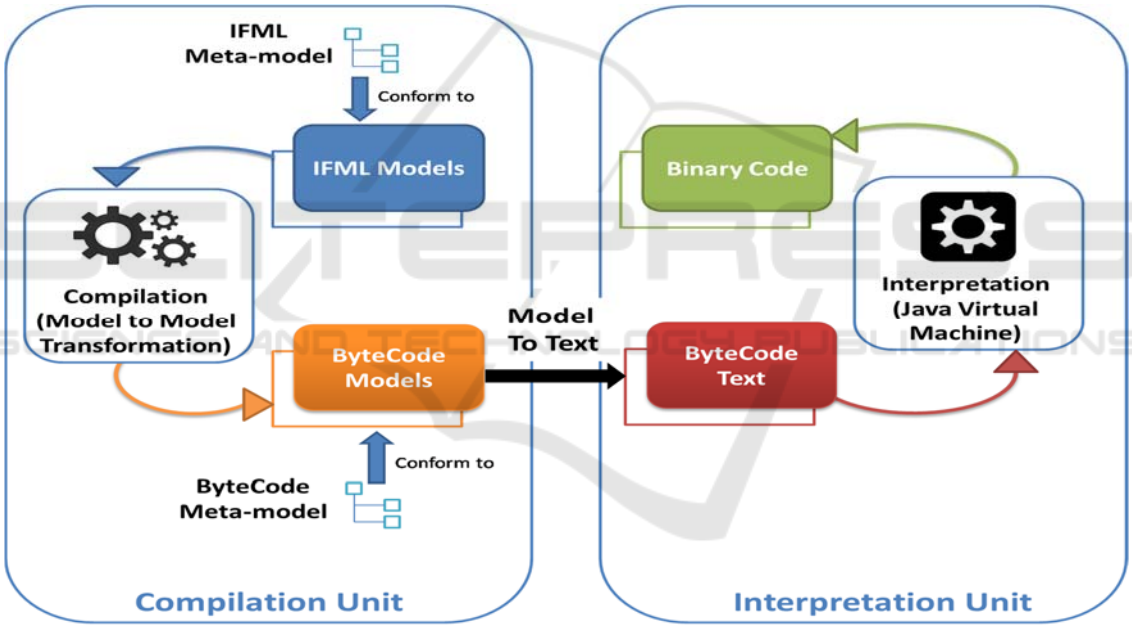

Figure 5: IFVM virtual machine architecture.

All computation in the JVM centers on the stack. Everything must be pushed onto the stack.

As depicted in figure 4, after compiling our java example code, we obtain a java class file which contains the constant pool table with bytecode instruction equivalent.

The constant pool has 15 entries in total. Entry #1 is Method public static void main, while #2 is for integer value 0xFEEDED (decimal 16707053). Also we have two entries #3 and #4 which corresponds to this class and super class. Rest is the symbol table storing string literals.

In the example's bytecode sequences (see figure 4), the integer 16707053 is pushed onto the stack with ldc instruction, then it is popped off the top of the stack and stored back to the local variable by the istore_1 instruction. After that, local variable is multiplied by two by first pushing the local variable onto the stack with the iload_1 instruction, and then pushing two onto the stack with iconst_2.

After both integers have been pushed onto the stack, the imul instruction effectively pops the two integers off the stack, multiplies them, and pushes the result back onto the stack. The result is popped

off the top of the stack and stored back to the local variable by the istore_1 instruction.

5 IFML VIRTUAL MACHINE

Our contribution is to produce an IFML virtual machine called IFVM for executing user interfaces designed with IFML models.

In order to make possible an execution of such models we opted for an approach that uses a model driven transformation to equivalent executable binary models.

A IFML model is translated to executable IFVM virtual machine bytecode.

The IFVM virtual machine consists of two major units: the compilation unit and the interpretation unit (see Figure 5). The IFVM benefits from the advantages of the famous java virtual machine.

The compilation unit compiles the IFML models in order to generate the bytecode. While the second unit which is the interpretation unit, it is dedicated to the java virtual machine that will interpret the bytecode generated in the previous unit in order to produce the binary code in a specific platform.

The Java Virtual Machine represents an interface between the bytecode and computer. Its implementations for different platforms are called Java runtime systems. The Java Virtual Machine allows running the same bytecode on any platform.

So, "write once, run anywhere" java makes it become a reality.

As introduced before, the bytecode produced from the first unit (compilation) is equivalent to IFML elements. It is the java bytecode which is based on the instruction set of the java virtual machine.

However, since IFML can be mapped to executable programs and structures, such as a relational database and a set of JSP templates and components of the MVC architecture according to (BRAMBILLA et al., 2014), it is easy to map IFML models to the instruction set of IFVM.

Bytecode instructions are passed to and from interactions through a stack without using registers, like it is given in other compiled high-level languages.

The IFVM bytecode has instructions similar to those of java bytecode. IFVM instruction set includes all java virtual machine instruction set.

The compilation unit represents a model to model transformation aiming to produce IFVM bytecode from the IFML model according to IFML and bytecode metamodels.

The second unit starts by a step which is a model to text transformation in order to produce the bytecode text that will be used as input to the java virtual machine.

6 RELATED WORKS

As depicted in figure 1, there are two basic types of model implementations: the compilation and the interpretation. While compilers admit compilation approach and virtual machines admit interpretation approach.

After the apparition of the MDA approach, many researches developed approaches for executing UML models applied on such a subset of UML which semantics are executable. Among these approaches, we cite (J. Chanda et al., 2010), (Knapp et al, 2002), (Fredriksen et al., 2005).

(Knapp et al., 2002) Implements the code generation approach, it develops a solution called HUGO based on a set of tools for model checking and generating Java code from UML model.

(J. Chanda et al., 2010) Implements the model compilation approach; it takes context free grammars for the two UML diagrams: Class diagram (depicting the static design) and sequence diagram (depicting behavioural design) and verifies the syntactic correctness of the individual diagrams and semantic correctness in terms of consistency verification with other diagrams.

(Fredriksen et al., 2005) implements the model interpretation approach. It develops a UML virtual machine that executes UML models. It proposes a subset of UML models and operational semantics for directly executing those models.

It has been shown, in this section, that there are many solutions proposed for allowing UML execution. But no defined solution has been presented for executing the user interface front end that can be designed with IFML models. This work is the first approach proposed for executing IFML models.

7 CONCLUSIONS

IFVM is our virtual machine proposed for executing IFML models after a big study of executability of IFML platforms which depends on the events and the status of IFML elements.

Our approach is based on an executable IFML that represents the front end of a system. We have

defined an IFVM Virtual Machine for the efficient execution of IFML models, which can be implemented in many platforms due to java virtual machine advantages.

Our approach eliminates erroneous intermediate step which is the code generation before the execution and increases the portability that came from the advantages of virtual machine concept.

The domain modeling (back end) is one of the most consolidated disciplines of information technology. Many languages have been proposed such as UML.

With IFML and the UML executable standards fUML (OMG, 2011) (executable subset of UML) and the Alf (OMG, 2013) the action language that follows the fUML subset, we can propose as a future work a framework that produces a complete model driven executable system with both the front end which can be presented by IFML and the back end that can be designed by UML executable subset.

REFERENCES

OMG, "OMG Unified Modeling Language (OMG UML)", *Superstructure, Version 2.0*, http://www.omg.org/spec/UML/2.0, 2005.

OMG, "Semantics of a Foundational Subset for Executable UML Models (fUML)", *v1.0, 2011*.

OMG, "Action Language for Foundational UML (Alf)". http://www.omg.org/spec/ALF/, 2013.

Knapp, A., and Merz, S., "Model checking and code generation for UML state machines and collaborations", *Proc. 5th Wsh. Tools for System Design and Verification, 2002*, p. 59-64.

Fredriksen, K.,"UMLexe–UML virtual machine: a framework for model execution", *2005.*

BRAMBILLA, Marco et FRATERNALI, Piero. Interaction Flow Modeling Language: Model-Driven UI Engineering of Web and Mobile Apps with IFML. *Morgan Kaufmann*, 2014.

OMG, MDA. "MDA Guide Version 2.0." ,2015.

OMG, Interaction Flow Modeling Language. *Version 1.0. IFML (2015)*, available at http://www.omg.org/spec/IFML/1.0/

WebRatio: http://www.webml.org (2008).

J. Chanda, A. Kanjilal, and S. Sengupta, "UML-compiler: a framework for syntactic and semantic verification ofUML diagrams". Distributed Computing and Internet Technology. *Springer Berlin Heidelberg, 2010*, p. 194-205.