IFVM Bridge: A Virtual Machine for IFML Models Execution in Combination with Domain Models

Sara Gotti and Samir Mbarki
MISC Laboratory, Faculty of Science, Ibn Tofail University, Kenitra, Morocco

Keywords: Interaction Flow Modeling Language IFML, Model Execution, Mda, Bytecode, Virtual Machine, Model Interpretation, Model Compilation, User Interfaces, Front End, Content Dependency, Input-Output Dependency.

Abstract: Many software systems are executable on various computing devices. No one denies that each of these devices has its own user interfaces. Nevertheless, this trend of computing everywhere is not accompanied by a solution that can be used to abstractly express the content, user interaction and control behavior of the software application front end without focusing on the implementation platform. Applying the concept of abstract models to user interfaces become a necessity. Accordingly, OMG adopted (in March 2013) the new Interaction Flow Modeling Language (IFML) for abstractly describing the system front end. It ensures executability in order to be mapped into executable applications for different kind of devices. In this paper, we propose a new model driven development approach to execute the logical description of UIs components and their interactions captured with IFML. We define IFVM, a virtual machine for executing IFML models with focus on the content-dependent navigation specification for passing parameters between the ViewElements, and the Data binding specification to specify the source of the published content.

1 INTRODUCTION

In the last few years, there has been an extensively increase in the complexity of computer systems in order to guarantee an efficient production respecting the time to market. Besides, technologies are widely evolving. In order to benefit from technological advances, it is necessary to adapt the applications to these technologies. But this operation is expensive for companies because it is often necessary to rewrite the code entirely. To solve this issue, several approaches have been proposed, in the context of model driven engineering (MDE), to software design and implementation via what we call models. Among these proposals we cite model driven architecture (MDA) (Blanc and Salvatori, 2005); it is the object management group's (OMG) vision of MDE. Therefore, modeling has been used more and more to try to control system's complexity to easily produce valid software.

Consequently, MDE has recently been presented in the building of software back-end through the UML language. However, several practical questions arise when dealing with the building the system's front-end. It should be noted that with the emergence of devices and technological platforms, front-end development become a complex task in which several requirement, perspectives, and disciplines intersect (Brambilla et al., 2014). So, to guarantee this trend of computing everywhere, it is recommended to create a platform independent representation for expressing user design and interaction to facilitate the generation and the execution on different computing platforms and technologies. To answer all these questions and needs, OMG group has adopted a solution in March 2013 as a standard which is the Interaction Flow Modeling Language (IFML) (Omg.org, 2018).

IFML is a modeling language used for expressing user interactions and front-end behavior of software systems regardless the implementation specific issue of their realization. IFML was designed considering many rules in which we site implementability which means that it supports code generation via model transformations and code generators. In other words, it could be mapped to executable applications for multiple devices.

In this present study, a new technique for executing IFML model is suggested. we propose a design of IFML virtual machine for directly executing IFML models without code generation in the hope of

Gotti, S. and Mbarki, S.
IFVM Bridge: A Virtual Machine for IFML Models Execution in Combination with Domain Models.
DOI: 10.5220/0007581604890494
In *Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2019)*, pages 489-494
ISBN: 978-989-758-358-2

elaborating interactive graphical user interfaces. the proposed IFVM (IFML Virtual Machine) considers not only the UI general structure, but also the different types of events that could be triggered as well as the caused navigations between the views especially the content-dependent ones.

The remainder of the paper is organized as follows. We start out, in section 2, with the related work on model execution approaches. In section 3, we explore conceptual modeling of Human Computer Interactions (HCIs) by presenting the user interface description language IFML and its executability. The section 4 is devoted to the proposed process of IFVM virtual machine for executing IFML models. Future works are presented in section 5. Section 6 concludes the paper.

2 RELATED WORK

In this work and in related references, it was observed that many researches have been proposed for directly executing models, without code generation, as those based on UML for the design of the back-end. Here (Gotti and Mbarki, 2016), the authors discuss many proposed works in this field through a comparative study.

The literature on front-end development shows a variety of model-based approaches for GUIs generation. As for (Frajták, Bures and Jelinek, 2015), authors discuss the automatic generation of automated application front-end from the IFML model. (Laaz et al., 2018) studied many other related solutions or tools for modeling IFML by providing a comparative analysis while considering various criteria.

Besides, other model driven solutions, based on IFML, were emerged for the development of code generators by affording extension for the IFML language, among these solutions we cite (Brambilla, Mauri and Umuhoza, 2014).

(Gotti and Mbarki, 2016) offer a new utilization of IFML language in the software modernization field. they propose an architecture-driven modernization (ADM) based approach to obtain knowledge of the structure and behavior of source code by generating three independent platform combined models: KDM to capture the knowledge discovery, IFML to extract the structure of the presentation and TaskModel for tasks aspect needed for the construction of the future user interface (UI).

However, to the authors' best knowledge, most of the previous studies do not take into account the IFML executability, while we refer to our previous work (Gotti and Mbarki, 2016), the focus is different. Although several solutions have suggested code generation of IFML, little or no one have proposed a direct execution of IFML without code generation except (Gotti and Mbarki, 2016).

Nevertheless, in this paper, we explore the possibility of executing IFML models by providing a new model driven design of a virtual machine called IFVM for directly executing the abstract models.

3 CONTEXT

A human can control and communicate with a machine by means of what we call Human-Computer Interaction (HCI). Engineers in this field are studying how humans interact with computers or with each other using computers, as well as how to design systems that are ergonomic, efficient, easy to use, or more generally adapted to their context of use.

Quite recently, considerable attention has been paid to the new trend of computing everywhere in the modern communication. It consists of the merging of various technologies to allow people to work everywhere in different devices. There is a need to adapt the UIs with their context of use, i.e., plasticity (Jean-Sébastien, Gaëlle and Jean-Marie, 2006). To solve this issue many solutions have been proposed. We focus on those model-based that permit plasticity by elaborating platform independent models describing the UIs and then it could be easy to generate the implementation via model transformations and code generators.

In fact, abstract modeling of UIs become a necessity to adapt platforms and technologies changes. So, the answer of how to permit UIs presentation at a high level of abstraction is by defining a language for abstractly describing the UIs regarding many criteria as described here (Selic, 2003). We call this language a User Interface Description Language (UIDL) (Orit et al., 2008).

OMG in its turn has adopted IFML in March 2013 as its vision of UIDL. IFML is designed to capture the user interaction, the content and the control behavior of the software systems front-end.

3.1 IFML Language

IFML allows the system modeler to design the structure of the general view part of the system in terms of ViewContainers with navigation relations, and what ViewContainers could contain in terms of ViewComponents. ViewComponents enable content display and data entry, they could be associated with a ContentBinding to express the source of the displayed content.

IFML ensure the events management that could be produced by user interactions, by actions in the application or by the system.

Events engender actions that could change the state of the UI and they can be associated with ParameterBinding that expresses the input-output dependency between the ViewElements.

Many aspects that could help understanding IFML language are presented in (Brambilla et al., 2014). we focus on the IFML language definition aspect through the IFML metamodel artifact, and the IFML executability aspect.

The IFML meta-model is the best definition artifact presented to describe the IFML language. It is divided into three packages: The Core package, the Extension package, and the DataTypes package, as shown in Figure 1.

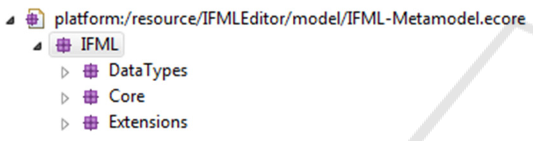
platform:/resource/IFMLEditor/model/IFML-Metamodel.ecore
IFML
DataTypes
Core
Extensions

Figure 1: IFML Packages.

The Core package contains the abstract and general concepts for building the infrastructure of the language such as InteractionFlowElements, InteractionFlows, and Parameters. These defined concepts are extended by concrete concepts in the extension package to treat more complex behaviors.

The IFML metamodel incorporates the basic data types defined in the UML metamodel into the third DataTypes package. It specializes some UML metaclasses as the origin for IFML meta-classes, and presumes that the IFML domain model is represented in UML.

the IFML model admits IFMLModel as a top level metaclass for including the InteractionFlowModel; that affords the general ViewElements, action and events of the system, and the Domain model; that specifies the main objects, their attributes and associations.

Model driven development processes start by elaborating models and then generate executable code (such as C, C++, Ada, Java, Forth, even VHDL) through a series of model transformations and code generators. To ensure this automatic generation, it is recommended to use executable models.

To execute models, i.e. generate executable code from it, they must have a complete definition in terms of executability respecting two kinds of elements as described in (Eric et al., 2011) which are: the static part; that could define the static view elements of a model, and the dynamic part; that describes behavior of the model.

IFML has been designed with executability aspect. This is obtained through model transformations and code generators already exist that ensures an easy mapping between conceptual IFML constructs and executable applications for various platforms and devices.

User interaction, within a view, produces events that could affect the state of the views and then execute actions that could signal another event and that are what the execution semantics of IFML.

The IFML execution semantics describes any IFML diagram as a machine that takes as input the interaction of the user and updates the state of the interface for the user to continue the interaction.

3.2 Content and Input-Output Dependency

Actually, the specification of the content and navigation aspects plays an important role in the description of the system front end. This is done through the using of ViewContainers, Events and NavigationFlows.

First, we consider the content dependency allowing the display of a data in the interface. In fact, the content to be published within ViewComponents could be provided from a different source. It is then important to identify the source of the content in each publishing ViewComponent. We use for this purpose the DataBinding concept associated with ViewComponent element.

The DataBinding concept expresses the source of the retrieved content from objects of the domain model. It ensures the automatic generation of data from classes and attributes of UML Class diagrams, Entity-Relationship models, ontologies, or other elements.

Figure 2 shows an example of a simple DataBinding. The "MovieList" ViewComponent retrieves its content from the "Movie" entity of the domain model of type UML Class model.

Besides, ViewComponents may allow user interaction that could produce a navigation, this navigation results in changing the content of ViewComponent to display other ones. We talk about the content dependent navigation specification expressed by means of Event and NavigationFlow view elements.

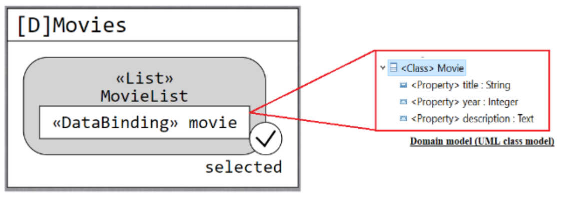
[D]Movies
«List»
MovieList
«DataBinding» movie
selected
<Class> Movie
<Property> title : String
<Property> year : Integer
<Property> description : Text
Domain model (UML class model)

Figure 2: DataBinding for Movie List content.

This specification permits the expression of the dependency in which one ViewComponent publishes content that relies on a precedent ViewComponent content after a user interaction is performed. This dependency is expressed by associating one or more ParameterBinding specifications to the NavigationFlow.

For example, as shown in Figure 3, a user accesses a list of movies published inside the List ViewComponent. He may select one object of the movie list and reach another ViewComponent that displays the detailed information of the selected movie.

4 APPROACH OVERVIEW

This present paper presents a modest contribution to overcome front end development problems. It proposes a new approach for model execution based on the definition of a virtual machine for the implementation of abstract UIs definition with IFML. The approach exploits the IFML language as the input for the virtual machine to cover the content and the user interactions of the software front end, as well as the binding to external sources for extracting information. We choose the acronym IFVM for designate the proposed IFML Virtual Machine.

Figure 4 illustrates the general process of the approach. It can be seen that we present a model driven process that allows the execution of IFML models. During this process, the focus of attention was on the two concepts provided by IFML which are the DataBinding and the content dependent navigation described in section before.

For building the virtual machine, several technics are suggested to ensure automatic execution of the IFML models. As can be seen from Figure 4, the process encompasses two major units: (i) the compilation unit and (ii) the interpretation unit. We suggest to merge the two concepts of implementation which are the compilation and the interpretation to benefit from their advantages.

In general, the hybrid approach of an interpretive compiler allows fast execution (Aggarwal, K. Singh and Jain, 2014), since it is based on virtual machine instructions of bytecode that has simple format to be quickly analyzed by an interpreter.

4.1 Compilation Unit

Here we start the process of execution, in a first stage, by elaborating the IFML model conforming to IFML metamodel. It captures the general structure and behavior of the different views of the system. As well, it will be accompanied with a UML model referencing the domain model that provides the content to be displayed in the interface.

Figure 3 gives an example of an IFML model expressing two views associated with a NavigationFlow for a movies manager system. As depicted in the figure, there are two windows, one for listing the set of movies, and a second one for publishing the details of the selected movie. The first window uses the List ViewComponent to display the movies list including some parameters:

- DataBinding referencing the Movie UML class (see Figure 5),

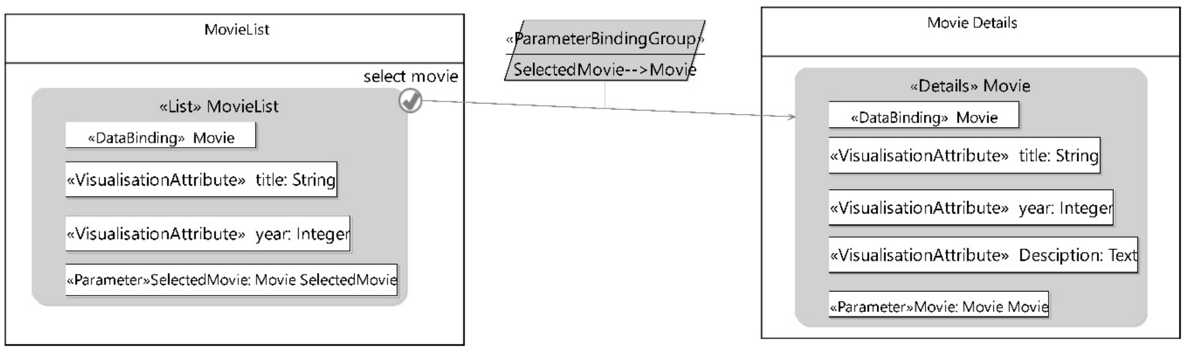
MovieList
select movie
«List» MovieList
«DataBinding» Movie
«VisualisationAttribute» title: String
«VisualisationAttribute» year: Integer
«Parameter»SelectedMovie: Movie SelectedMovie
«ParameterBindingGroup»
SelectedMovie-->Movie
Movie Details
«Details» Movie
«DataBinding» Movie
«VisualisationAttribute» title: String
«VisualisationAttribute» year: Integer
«VisualisationAttribute» Desciption: Text
«Parameter»Movie: Movie Movie

Figure 3: Example of List and Details ViewComponents of movies manager app.

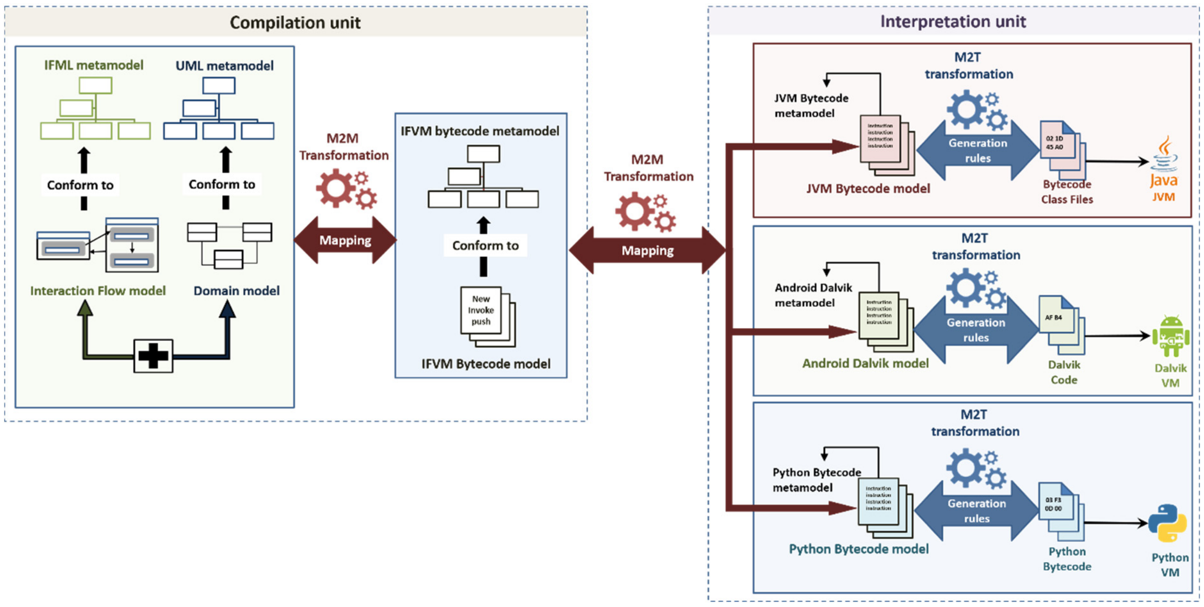

Figure 4: IFVM Process.

- VisualisationAttribute to locate the data to be shown in the interface,
- Parameter including the value of the selected movie,

The second window is devoted to the display of detailed information of the selected movie from the “MovieList” ViewComponent including some parameters as well.

The NavigationFlow is associated with a parameterBindingGroup that contains the declaration of an input–output dependency between the two windows. The value of the parameter “SelectedMovie” (output of the “MovieList” ViewComponent) is associated with the value of the parameter “Movie” (input of the “MovieDetails” ViewComponent).

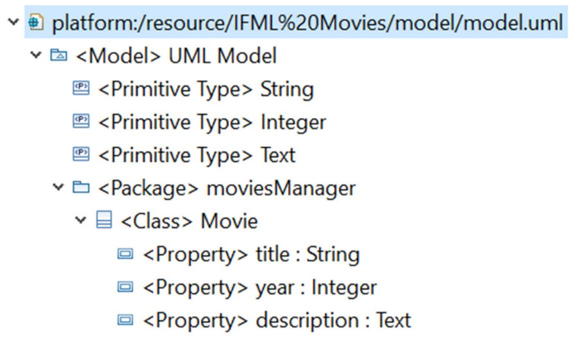
platform:/resource/IFML%20Movies/model/model.uml
<Model> UML Model
<Primitive Type> String
<Primitive Type> Integer
<Primitive Type> Text
<Package> moviesManager
<Class> Movie
<Property> title : String
<Property> year : Integer
<Property> description : Text

Figure 5: Example of UML domain model for movies concepts.

The purpose of this present work is to express the bytecode model elements in a bytecode format to be interpreted by a corresponding virtual machine in order to obtain the machine instructions understood by a computer's processor. Actually, there are a plenty of virtual machine architectures available on the market. However, in this paper we propose a new definition, through a metamodel, of IFVM bytecode form for the desired IFVM virtual machine. It has a similar syntax of Java bytecode with additional instructions for expressing events and navigations for example.

So, once the IFML and the UML models are built, we launch a M2M transformation that maps their elements to elements of IFVM bytecode model.

4.2 Interpretation Unit

IFVM has been chosen as an abstract intermediate form in order to permit its representation in the other forms of bytecode. So, after the execution of the compilation unit, that is to say getting the IFVM bytecode model, we begin the interpretation unit dedicated to firstly mapping via a M2M transformation the IFVM model to the other models of bytecode of existing virtual machines; the JVM bytecode model, the android Dalvik model and the Python bytecode model, in order to gain optimization and portability.

After getting the three bytecode models, we launch a model to text transformation, in a second time, for each generated bytecode model in order to generate the corresponding bytecode files.

Finally, the Bytecode files will be then treated by their corresponding virtual machines to be executed.

5 FUTURE WORK

On the basis of the IFVM process, we will build a framework based on the proposed process to enable the automatic front end execution of a system. It will provide both a textual and graphical model editor for designing the IFML and UML models. We will elaborate our own IFVM bytecode metamodel with all the required instructions according to bytecode instruction set of the existing virtual machines. Compilation will be done via a model to model transformation designed with QVTo language (Omg.org, 2016). In the interpretation unit we launch a second M2M transformation that will be designed with QVTo as well. For the model to text transformation, we will opt for a code generation using the open source Acceleo (Eclipse.org, 2005). Finally, we will extend the framework to support additional bytecode forms for other existing virtual machines.

6 CONCLUSION

In this position paper, we presented a model driven process for automatically executing the abstract representation of user interfaces with content provided by domain models. Our contribution reveals the possibility of building and maintenance of software front end easier and less expansive via transformations and model tools in comparison to the code driven contributions.

REFERENCES

Blanc, X. and Salvatori, O. (2005). MDA en action. *Paris: Eyrolles.*

Brambilla, M., Fraternali, P., Elliot, S., Herbert, K., Kumaraguruparan, P. and Rogers, M. (2014). *Interaction flow modeling language*.

Omg.org. (2018). The Interaction Flow Modeling Language. [online] Available at*: https://www.omg.org/ ifml/ [Accessed 22 Nov. 2018]*.

Gotti, S. and Mbarki, S. (2016). UML Executable: A Comparative Study of UML Compilers and Interpreters. In: *Information Technology for Organizations Development (IT4OD).*

Frajták, K., Bures, M. and Jelinek, I. (2015). Transformation of IFML Schemas to Automated Tests. In: *Conference on research in adaptive and convergent systems*.

Laaz, N., Wakil, K., Mbarki, S. and N.A. Jawawi, D. (2018). Comparative Analysis of Interaction Flow Modeling Language Tools*. Journal of Computer Science, 14(9)*.

Brambilla, M., Mauri, A. and Umuhoza, E. (2014). Extending the interaction flow modeling language (ifml) for model driven development of mobile applications front end. In*: International Conference on Mobile Web and Information Systems*.

Gotti, Z. and Mbarki, S. (2016). Java Swing Modernization Approach Complete Abstract Representation based on Static and Dynamic Analysis. In*: ICSOFT-EA. KENITRA*.

Gotti, S. and Mbarki, S. (2016). Toward IFVM Virtual Machine: A Model Driven IFML Interpretation. In: *ICSOFT-EA*.

Jean-Sébastien, S., Gaëlle, C. and Jean-Marie, F. (2006). Models at runtime for sustaining user in-terface plasticity. In: *Models@ run. time workshop (in conjunction with MoDELS/UML 2006 conference).*

Selic, B. (2003). The pragmatics of model-driven development. *IEEE Software, 20(5), pp.19-25*.

Orit, S., Robert JK, J., Mark, G. and K., L. (2008). User interface description languages for next generation user interfaces. In: *CHI'08 Extended Abstracts on Human Factors in Computing Systems*.

Eric, C., Cyril, B., Alexandre, F. and F., B. (2011). Contracts for model exe-cution verification. In: *European Conference on Modelling Foundations and Applications*.

Aggarwal, A., K. Singh, D. and Jain, S. (2014). A Hybrid Approach of Compiler and Interpreter. *International Journal of Scientific & Engineering Research, 5(6).*

Omg.org. (2016). About the MOF Query/View/ Transformation Specification Version 1.3. [online] Available at: *https://www.omg.org/spec/QVT/About-QVT/ [Accessed 23 Nov. 2018]*.

Eclipse.org. (2005). Acceleo | The Eclipse Foundation. [online] Available at*: https://www.eclipse.org/acceleo/ [Accessed 23 Nov. 2018]*.