2012 International Conference on Information and Network Technology (ICINT 2012)
IPCSIT vol. 37 (2012) © (2012) IACSIT Press, Singapore

MDA Driven xUML Plug-in for JAVA

A.M.Magar [1], S.S.Kulkarni [1], Pooja Arora [2], Sandeep Gupta [2 +], Savita Khetmalis [2] and Vikas Shetty[2]

[1] Department of Information Technology, SAE, Pune University
[2] UG (8th SEM) B.E [IT], Department of Information Technology, SAE, Pune University

Abstract. This paper is an attempt to focus on challenges for creating system executables from directly analysis models using MDA. MDA is a standard approach to using the UML as a programming language. Modelers represent a particular application by creating a Platform Independent Model (PIM). The PIM is a UML model that is independent of any particular technology. Tools can then turn a PIM into a Platform Specific Model (PSM). The PSM is a model of a system targeted to a specific execution environment. Some of the key benefits of MDA are Portability of models, Cross-platform Interoperability and Platform Independence.

Keywords: EMF, GEF, MDA, MOF, OMG, PIM, PSM, XUML

1. Introduction

Mellor and Balcer noted that "the history of software development is a history of raising the level of abstraction". Computers were initially programmed in machine code, which consisted of a series of zeroes and ones to be interpreted by the processor. Later, the assembly language substituted some of these codes by easier-to remember mnemonics.

The act of programming continued machine centered, but it was made easier, some of these mnemonics could be reused as new processors were introduced, Productivity was improved. Later, high-level programming languages such as FORTRAN and BASIC were born.

These new languages were closer to human language than to machine language. They also increased the possibilities of re-use, as the same code could be translated to different processors without being altered.

Executable UML, as the name implies, is an attempt at making UML directly executable or translatable to 3rd or 4th generation programming languages. UML in its current state is not directly executable. It lacks action semantics to describe the steps executed by the system in response to events.

Executable UML tries mitigating the risk described above, by making the UML artifacts an integral part of the production of systems, not just non-imperative design.

Executable UML, coupled with Model-Driven Architecture, may be the foundation for a new software development methodology in which domain (business) experts would be involved in the crafting of high-level models and technologists would be concerned with building model compilers that would translate the models to 3rd or 4th generation code. These model compilers would be created by highly skilled computer specialists using the best practices and advanced knowledge of the computer science and software technology.

+ Corresponding author. Tel.: +91-020-263-614-96.
E-mail address: sandeep3010@gmail.com.

Table. 1: Concepts in executable UML

CONCEPT	CALLED	MODELLED AS	EXPRESSED AS
The world is full of things	Data	Classes Attributes Associations Constraints	UML Class Diagrams
Things have life cycle	Control	States Events Transitions Procedures	UML State Chart Diagrams
Things do things at each stage	Algorithm	Actions	Action language

Executable UML thus also aspires to become the next level of abstraction, more accessible to domain experts, who could more easily design and specify the characteristics of the system taking into account expert domain knowledge. According to the words of Mellor and Balcer Executable UML is at the next higher layer of abstraction, abstracting away both specific programming languages and decisions about the organization of the software so that a specification built in Executable UML can be deployed in various software environments without changes.

There are already tools in the market that provide for round-trip engineering. Round-trip engineering consists of generating code from models and updating the model as the code is changed by other means. While this approach is known to improve productivity, it does not raise the level abstraction. Models become just graphical representations of the code, which determines the level of abstraction. This approach is obviously tied to the platform for which the code is generated, providing for little actual software platform independence.

2. Existing Methodologies

UML uses elaborative approach where Executable UML advocates a different approach to modeling, which is called the “translative” approach. According to the executable UML methodology the activities of analysis and design should be completely separated. Modeling is done mostly in analysis. The models that are the output of the analysis phase is not to be refined in design but rather translated to executable code using the model-translators, which are the output of the design phase. In other words, the design activities aim at developing a model-compiler to the target platform using the most adequate design patterns.

The analysis activities aim at creating executable models of the subject matter in study. The same executable model can be translated using different model-compilers, or a single model-compiler can be used to translate many executable models.

The translative approach is far from widespread in today’s software development, but we believe that its main ideas are sufficiently powerful to have the potential to change how software is produced and delivered and perhaps could lead to a new software development paradigm based on the automated translative approach.

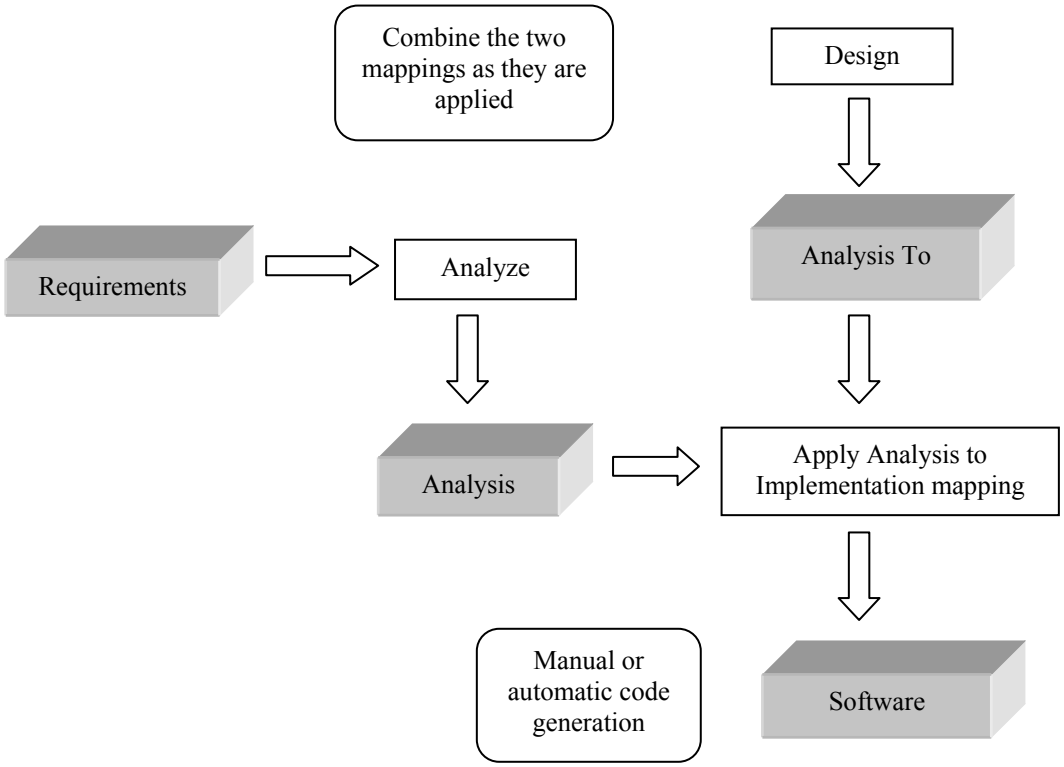

Fig. 1: The translative approach

2.1. Action semantics

The Unified Modelling Language (UML) lacks precise semantics for several modelling constructs, such as transition guards or method bodies. This prevents executability; making early testing and validation are out of reach of UML tools. Furthermore, the semantic gap from high-level UML concepts to low-level programming constructs found in traditional object-oriented language prevents the development of efficient code generators. Another problem is UML does not supply timing and synchronization rules.

The first problem of writing method bodies can be solved by using Action Languages. There are several languages exists for this purpose such as

- Action Language (OAL),
- Shlaer-Mellor Action Language (SMALL),
- Action Specification Language (ASL),
- That Action Language (TALL),
- Starr's Concise Relational Action Language (SCRALL),
- Platform-independent Action Language (PAL)
- Path MATE Action Language (PAL).

In addition to these languages OCL can be extended for this purpose. The adoption of the Precise Action Semantics for the UML by the OMG supports the viability of Executable UML. The standard was the work of a large industry consortium comprised of Rational Software, Kabira Technologies, Project Technologies, Kennedy-Carter, and others, in response to a Request for Proposals (RFP) from the OMG. Prior to that adoption the only way to attain 100% code generation from a UML-modelled solution was to use a third generation language or a proprietary language to specify the algorithmic behaviour of the solution.

In the executable UML OCL4X can be used as a ASL to express the operation body in class diagram, action effect in activity diagram or behaviour of state in state machine diagram.

The breakthrough notions of using an action language to specify behaviour in a UML model are that:

• The action language allows modellers to define behavioural specification at a higher level of abstraction.

• The language is independent of any specific underlying technology in the execution environment.

Second problem of synchronization can be solved by introducing set of rules consistently across all the state charts. The synchronization rules may be simple, but they must be simultaneously offering the ability to capture complex application requirements and be implementable on wide variety of platform technologies.

2.2. Model compilers/ interpreters

An executable UML model completely specifies the semantics of a single subject matter, and in that sense, it is indeed a "program" for that subject matter. An executable UML model compiler turns an executable UML model into an implementation using a set of decisions about the target hardware and software environment.

As discussed rules for synchronization model interpreter or model virtual machine (MVM) must be constructed so that the rules observed by the modeler must be correctly implemented.

The model interpreter will allow us to put all of our objects into initial state, trigger an initial event and then let the models execute actions and move state by state. If there are pending events, the MVM selects and presents one to the waiting object. If not, the MVM will present the next incoming event.

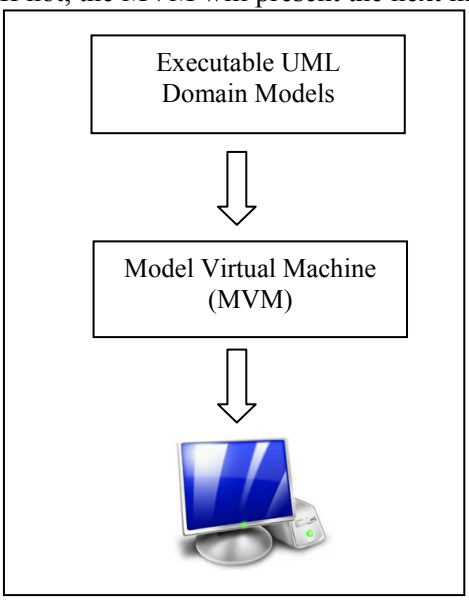

Fig. 2: Model Virtual Machine

The MVM sees two kinds of events mainly as self-directed and non-self-directed. When object sends a signal to itself, usually as part of an if-then action or just to trigger an advance at the end of a procedure, it results in the occurrence of a self-directed event.

MVM is the support environment of xUML, which allows building, debugging and processing the xUML models.

It contains four parts:

- XUML
- MVM Modeller, which is based on a third-party UML modelling tool and provides environment to build executable models;
- MVM Engine loads XMI formatted xUML models exported by MVM Modeller, performs model validation, and processes the models in compilation mode and generates test case from the models;
- MVM Debugger, which provides adapters to a third-party UML tool so that the developer can debug models visually.

MVM Modeller is independent of MVM Engine. XMI is used to exchange the model information between the two parts.

The MVM Engine supports both interpretation and compilation of the models. It accepts class diagrams, state machine diagrams and activity diagrams as inputs. In the interpretation mode, the model is executed without any intermediate step from the view of the user.

At first, the class diagrams are transformed to target code. When a new object is created in interpreting the state machine diagram, another state machine interpreter will be created to interpret the state machine diagram of the new created object. When state machine interpreter comes across the activity of an activity connection point or an operation calls, an activity diagram interpreter will be created to interpret the corresponding activity diagram.

In the compilation mode, the models are completely transformed to target codes, which can be deployed and executed without the models. MVM Debugger allows user to add/remove breakpoints on the model element and debug in the behaviour diagrams step by step. It sends debug commands to MVM Engine and receives debug events from MVM Engine while debugging.

3. Conclusion

Model Driven Architecture (MDA) is gaining more focus in many organizations. MDA stresses the benefits of modeling at various levels of abstraction and the integration and flow of information among models. With MDA, first the object model is built, which differentiates it from the traditional approach of server side development. Models are built and after modeling completion, development of software and systems is enabled.

MDA can be achieved by using UML, DSL, or other modeling solutions. DSL is one of the most efficient and proven ways of creating an MDA-based solution. There are many frameworks available for DSL like xText, EMF, GMF, Groovy, etc.

Eclipse Graphical Modeling Framework (GMF) is a Domain Specific language framework which is used to develop graphical editors based on Eclipse Modeling Framework (EMF) and Graphical Editing Framework (GEF).

It helps in defining the domain model, their properties, and relationships among them. A set of configuration files like Meta model, graphical model, tooling model, and the mapping model are defined. These configuration files provide graphical representation of domain model, its constraint etc.

xUML models help in creating PIM to PSM mapping. This project helps creating such models for java project by generating the code and executing it.

With the help of Reverse Engineering, the model can be obtained from the code available.

4. Acknowledgements

We would like to thank the entire Department of Information Technology at Sinhgad Academy of Engineering for their sincere guidance and continuous encouragement to gain superior degree of knowledge in the vast domain of computer science.

5. References

[1] A.M. Magar.; M.J. Chouhan; “Executable UML (xUML) and MDA”, International Conference GIT-2010 “Green-IT & Open Source” Conference Proceedings. No: 978-93-80043-89-0/13.

[2] Feiler P.H.; de Niz D; Raistrick C; Lewis B.A.; “From PIMs to PSMs”, Engineering Complex Computer Systems, 2007, 12th IEEE International Conference.

[3] Ke Jiang; Lei Zhang; Miyake S; “An Executable UML with OCL based Action Semantics Language”, Software Engineering Conference, 2007.

[4] Uri Shani; AviadSela; “Integrating Domain-Specific Programming into Software Design”, 2010 IEEE International conference on Software Science, Technology & Engineering. No: 978-0-7695-4061-0/10

[5] Enrico Biermann ; KarstenEhrig ; Claudia Ermel ; Jonas Hurrelmann; “Generation of Simulation Views for Domain-Specific Modeling Languages based on the Eclipse Modeling Framework” , 2009 IEEE/ACM, International conference on Automates Software Engineering. No: 978-0-7695-4061-0/10.