A Study of the Model Method Driven Architecture (MMDA) and Its Modeling Environment

Tadashi KURATA *, Hiroshi DEGUCHI *, and Manabu ICHIKAWA **

Abstract: Model Driven Architecture (MDA) is a program development method designed for experts in modeling, but not in programming, and it has not been widely accepted yet. In this paper, we propose a new program development method that enables modeling experts to develop a program based on the dynamic model description in accordance with the modeling method in their domain. Furthermore, we discuss the structure of a programming language specialized for this modeling method and the modeling environment to develop this new program development method.

Key Words: MDA, MMDA, modeling method specialized programming language, modeling GUI.

1. Introduction

Model Driven Architecture (MDA) [1] is one of the program development approaches, which enables experts to make a program (Java, C++, C#, Python, and so on) automatically from a specification written in abstract description language such as UML [2]. It does not depend on specific platforms and programming languages, thus enables non-technical users to reflect their understanding of the needs and changes to the software safely and effectively [3]. When this approach was first introduced, it has attracted attention since it enables non-technical users to create programs only from their conceptional specifications. However, unfortunately this trend did not last for too long. Intuitively if non-technical users can create an application by only describing specifications, there is no doubt that the productivity of software development could be dramatically improved. Therefore, the question is why it does not work well any more. In this paper, we discuss the reasons and propose a new program development approach that could replace MDA. In addition, we describe an application that we have designed and implemented as the modeling environment to achieve that non-technical users can create programs only from their conceptional specifications.

2. The Problem with MDA and a New Program Development Approach

The reasons of why MDA is not widely accepted are as follows. The model description by MDA targets at modeling methods of a wide and general domain, rather than a specific one, and it adopts the abstract specification description language such as UML, which does not actually work. In MDA, users must describe a specific target domain model by using the general model description method such as UML.

By using UML, it is impossible to describe the model by the specific model description method based on the modeling method of each specific domain such as dynamic multi agent simulation, system dynamics, ladder logic and robot simulation and so on. Here, the modeling method comes from a logical thinking method of a specific domain. For domain expert users, who do not have enough programming skill, it is very hard to create a model by using a general model description method such as UML. This relationship is shown in Fig. 1.

Fig. 1 The relationship among modeling method, abstract specification description language and MDA.

JCMSI 0005/17/1005–0450 © 2016 SICE
generate a program from an abstract specification description language such as UML. The program has to be written by general programming language (Java, C++, C#, Python, etc.) in order to work well. In fact, the program automatic generator of MDA can only create class groups with member variables and empty methods with specific platform, programming language and target domain. To make the program work, programmers must write code in an empty methods by using a text editor. Although it is possible to reduce the amount of code if users write more detailed description of the abstract language such as UML, the description of the abstract language becomes complicated, closer to the code and no longer abstract, and the original significance of MDA is lost. For example, we try to describe an infection epidemic simulation model by UML, which is expressed by the processing flow in the flowchart as shown in Fig. 2. Each agent possesses the information of infection state (uninfected, infected, recovery, and death) and position (workspace and home). While the position information of each agent varies at random, an infected agent is generated by the contact with other infected agents based on the SIR model [4]. In UML, the use case diagram is described as in Fig. 3, the state chart diagram is described as in Fig. 4, and the class diagram is described as in Fig. 5. We must create the agent class, the variable class and the controller class, and describe the member variables and methods based on the class diagram by using a text editor. The definition and description should follow the general programming language specifications and platforms which we use. As we described above, even the automatic program generator creates only the classes which have the member variables and empty methods as follows.

```java
public class Variable {
    public String name;
    public String value;
}

public class Agent {
    public String name;
    public Variable infection;
    public Variable location;
    public void updateLocation() {
        // Do something
    }
    public void updateInfectionStatus() {
        // Do something
    }
}

public class Controller {
    private List<Array> agents;
    public void updateLocation() {
        for (Agent agent:agents)
            agent.updateLocation();
    }
    public void updateInfectionStatus() {
        for (Agent agent:agents)
            agent.updateInfectionStatus();
    }
}
```

Therefore, it is difficult for non-technical users to write a program since programming skill is required. Against this background, we propose an alternative program development approach. It enables the specific domain experts to create and run the program. It can be realized when the following three requirements are satisfied.

1) The model is described by a user interface built based on each specific modeling method such as system dynamics.

2) The model is described by a modeling method specialized programming language, which is specific for the domain experts.

3) The modeling method specialized programming language satisfies nine requirements described later.

Experts in the target domain are competent to create the model description based on its model construct concept naturally, if this model description is limited to a specific method.
For example, experts of system dynamic modeling can create the model description based on stock flow, those of role-oriented agent based simulation [5] based on definition of agents and their behaviors, and those of Real World OS [6] based on agent interactions embedded on the cyber physical space.

Since a general programming language (Java, C++, C#, Python, etc.) has a number of variable types, it is required to use the program integrated development environment (IDE) such as Eclipse [7]. Moreover, it requires to describe the program by a text editor. Therefore, it hinders those specific domain experts with no programming skills from describing the model. Repast [8] using Java and MASON [9] using C++ or Java are the applications that adopt this type of method. On the other hand, the modeling method specialized programming language has only limited variable types that represent the objects of the target domain such as Agent class, Variable class and Controller class as shown in Fig. 5. Therefore, by taking advantage of features such as the variable type selector, a graphic user interface (GUI), which is specific to the modeling method specialized programming language, can be presented without a text editor. Further, if the program written by the modeling method specialized programming language is executable, the model description can be converted to a workable program. Moreover, users can extend the GUI or visualize the program execution results if such functions are enabled. Therefore, if the specific domain experts use this new program development approach, they can execute the program by only describing the model, extend the GUI and visualize the program execution results. We especially refer the modeling environment to describe a model as Modeling GUI, which is a comprehensive program integrated development environment.

Figure 6 shows the relationship among a modeling method, a modeling method specialized programming language and Modeling GUI for modeling method specialized programming language for MMDA. Role-oriented agent-based simulation simulates the movement of agents with different roles between spots. System dynamics simulates the state changes of a system in natural science, economy and society, etc, to observe and analyze the system state changes.
transition. Ladder logic simulates a logic circuit. Robot simulation simulates the behavior of a robot on software.

Namely, in the new program development approach, as the modeling method is established by a specific modeling method and a modeling method specialized programming language, the model could be described through building Modeling GUI. Modeling GUI enables the definition of processing flows, objects and variables in order to generate statements, execute the program, extend the GUI and visualize the program execution results. Here, the processing flow is similar to the flow chart in a model, the connection of each flow in system dynamics, and the loop processing procedure in role-oriented agent-based simulation and the Real World OS. In Modeling GUI, it is possible to generate an object by only mouse drag-and-drop, select an object and a variable via a combo box, and specify a variable name via a text field. Therefore, in the presence of Modeling GUI, non-technical users could create and modify the software safely and effectively.

As described above, unlike the MDA, this new program development approach allows to develop a program only by Modeling GUI. Therefore, we refer this program development approach by Modeling GUI as MMDA. In this paper, we discuss the construction of Modeling GUI that enables to develop a dynamic simulation program by MMDA. Here, a dynamic simulation refers to the ever changing state of a system that we could observe and analyze.

3. The Modeling Method Specialized Programming Language

In MMDA, a program is created by Modeling GUI. Therefore, the realization of Modeling GUI depends on the specifications of the modeling method specialized programming language.

3.1 Modeling Method Specialized Programming Language and Modeling GUI

In order to realize the Modeling GUI, the language should satisfy the following requirements.

1) Causal definition of the processing flow: When we define the processing flow, it is required to describe the causal model along the time axis.

2) Number of object types: An object is an element of the model. For instance, stock and flow are object types in system dynamics, and agent’s variables are object types in agent-based simulation. For easy selection of an object from a combo box, the number of available object types should be limited.

3) Selection of instruction types: When we select an instruction from a combo box, it is required to select the instruction and the object as its argument, for creating a statement without a text editor.

4) Clear dependency relationships between objects: The type and number of objects that an object could hold should be clear. It is required to select the object from a combo box hierarchically when create a statement.

5) Definition of a set of objects: It is required to define a set of objects as a single object. When we create a statement, the instruction is required to be able to process this object.

6) Limited behavior of objects: It is required a limited type of objects and instructions; otherwise Modeling GUI is hard to build.

7) Extensibility in the language specification: It is required to extend the language specification such as additional features of new instructions and to reflect it to Modeling GUI.

8) Variable to represent the elapsed time: When we do dynamic simulation, it is required to perform visualization of program execution by chart and animation with a time variable.

9) Work as coded: Programs written in this programming language should work as coded.

As mentioned above, since Modeling GUI is a comprehensive program integrated development environment rather than just a GUI, the specification of the modeling method specialized programming language has to satisfy the requirements 1)–9). Stella [10] for system dynamics and structured language (ST language) for programmable logic controller (PLC) [11] satisfy some of these requirements as follows.

Stella for system dynamics enables the definition of stock and flow objects, thus experts can write a simulation program based on their mathematical modeling concept. Stella has its embedded program development environment to create a Stella program. Figure 7 is the edit screen of the influenza epidemic simulation [12] by Stella. Users with mathematical knowledge of system dynamics could create and run the program with only model description by using this GUI.

In Stella, a new stock object is created by clicking the editing window while selecting the stock object icon on the icon menu as shown in Fig. 7. The stock objects’ mathematical expressions could be defined via the expression editor, as shown in Fig. 8. In Stella, a new processing flow is defined by connecting stock objects on the editing window while selecting the flow object icon on the icon menu as shown in Fig. 9; the flow objects’ mathematical expressions can be defined by the expression editor, as shown in Fig. 10. The mathematical expressions defined in the stock objects and flow objects are the statements as shown in Fig. 8 and Fig. 10. As already mentioned, since these mathematical expressions are defined by using expression editor and it is standard to use the format of the mathematical expression, the users are not required to remember new ones.

PLC is a control device of a relay circuit. It is commonly referred to as a sequencer. PLC requires to describe the control program by the modeling method specialized programming language called ST language. The relay control program could be created through the definition of process flow based on the modeling building concept of ladder logic (logic circuit description method). GX Developer (Mitsubishi Electric Corporation) is a program development environment to create a program for ST language. Figure 11 is the edit screen of a sequencer program by GX Developer. Users with the knowledge of reply circuit and ladder logic could create and run the reply control program only based on the model description by this GUI. In GX Developer, a circuit can be created by connecting any circuit part selected with the mouse operation in the circuit input dialog box as shown in Fig. 11.

Stella and ST language do not satisfy all of the requirements for Modeling GUI. They do not satisfy the requirements 7).
They do not have GUI extensibility in the language specification. In addition, Stella and GX Developer (the program development environments of a Stella program and a ST language program) are not designed and implemented as Modeling GUI for MMDA. Since both languages have no extensibility, it is impossible to add new instructions into their GUI and utilize any other programs. However, these GUIs include many parts common to the Modeling GUI. For example, the processing flow definition by mouse operation is based on the specification of the programming language that enables to define the processing flow. The number of object types is limited. Then it is easy to generate the object my mouse operation.

NetLogo [13] and artisoc [14] are called coordinate system-oriented agent-based simulation [15]. They have own program development environments with GUI. By using them, the state transition of the space can be observed and analyzed as agents’ variables change while moving and interacting. Also they do not satisfy all of the requirements for Modeling GUI any more than Stella and ST language.

Figure 12 shows the relationship among a modeling method, a modeling method specialized programming language and GUI for a program development environment. However, each GUI does not satisfy all of the nine requirements to realize MMDA.
Based on the above discussion, we select the modeling method specialized programming language which satisfies the nine requirements for Modeling GUI and build Modeling GUI which enables to create a program written by the modeling method specialized programming language in MMDA. Its Modeling GUI has to satisfy all of the above requirements to realize Modeling GUI, not only the features of the Stella and GX Developer’s GUI.

### 3.2 SOARS - Role-Oriented Agent-Based Simulation Programming Language

Spot oriented agent role simulation (SOARS) [16] developed by Deguchi laboratory of Tokyo Institute of Technology is the programming language for role-oriented agent-based simulation and has been continuing to evolve.

SOARS allows to simulate the movement of agents with different roles between spots. The resulted nonlinear state transition of spots as well as agents can be observed and analyzed.

SOARS is constituted by agents/spots as the objects and roles defining their behaviors. Every agent can move between spots. Agents/spots have variables such as strings, numbers, arrays, hash table, and so on. Role has instructions (conditions and actions) that agents/spots can execute. The simulation progresses as agents/spots execute instructions of the role. The mechanism of role-oriented agent-based simulation by SOARS is shown in Fig. 13.

SOARS has a concept called stage for the processing flow definition. According to the stage concept, the processing flow of simulation is represented as a sequence of one or more stages. In each stage, each agent and spot executes all instructions in one stage. The instructions in a stage are parallelly executable. It is required to execute the instructions which cannot be executed in parallel in a different stage. Therefore, by designing the processing flow based on the stage concept, all instructions in the simulation run are in the correct order.

SOARS satisfies nine requirements to realize Modeling GUI as follows.

1) Processing flow could be defined by the stage concept.
2) Types of objects are limited to agent, spot, role and variables (string, number, array, hash table, etc.).
3) Types of instructions are limited to value assignment and comparison.
4) Dependency relationships between agent, spot, role, and variables are clear: agents are on spots, and roles and variables are defined for agents and spots.

Therefore, SOARS can be used to create a program via Modeling GUI in MMDA. If the user has the modeling method of role-oriented agent-based simulation, he/she can create and run the program to express the complex interactions among agents by using the Modeling GUI without any programming skill. Therefore, we have designed and implemented SOARS VisualShell as the Modeling GUI for SOARS.

Figure 14 shows two types of relationship. One is the relationship among a modeling method, a modeling method specialized programming language and MMDA based GUI such as SOARS VisualShell.
4. SOARS VisualShell

By using SOARS VisualShell, a program can be created only by the model description. Only mouse operation and keyboard input are required to define the variable name of objects. In the following, we will use the infection epidemic model again to demonstrate the modeling environment.

4.1 Generation of Object

In SOARS VisualShell, a new object (agent, spot, agent role, and spot role) is created by dragging and dropping the icon from the icon menu to the editing window as shown in Fig. 15. The variables of the object can be defined in Fig. 16. Like this, it is possible to generate a new object by mouse operations only.

4.2 Definition of the Processing Flow

In SOARS VisualShell, the processing flow is defined by adding a new stage and specifying the stage execution order in the GUI as shown in Fig. 17. It is required to use the keyboard input to define only the stage names and the mouse operation to specify the stage execution order.

4.3 Generation of the Statement

In SOARS VisualShell, statements executed at each stage are defined by the GUI as shown in Fig. 18. Each statement is generated by selecting an instruction and arguments (the variables of the agent or spot) with mouse operation. It is possible...
to define statements by mouse operation without any keyboard input.

4.4 Extensibility of GUI

The specification of statements is expected to change frequently. In SOARS VisualShell, if there is a need to add a new instruction and change the argument of the instruction, it is a heavy burden to modify SOARS VisualShell. Also since it takes time, the user cannot immediately use the new features. We have created a new scripting language to define the user interface for role editing. SOARS VisualShell can generate a GUI screen by interpreting the script written by the language. Since SOARS can run any Java program by its extensibility, this fea-
tecture of Modeling GUI can be realized. By implementing this feature, the user can define all the statements in SOARS VisualShell.

4.5 Visualization of Program Execution

In SOARS VisualShell, the changing values of agents/spots can be recorded and displayed, as shown in Fig. 19. Besides, the simulation results can be visualized as shown in Fig. 20.

5. Discussion

In MDA, it is impossible to have a model creating method because it is required to describe a wide and general domain model by using the abstract specification description language. On the other hand, MMDA enables a model creation method by a specialized programming language. This model creation method makes it possible to build Modeling GUI and allows non-programmers to create and run the program by only describing the model by Modeling GUI.

Also, in Modeling GUI, since it is not required to remember the instructions and statement format for the modeling method specialized programming language, even non-experts of the target domain can easily describe the model. It is even possible to learn the specific modeling method through Modeling GUI.

To create an environment which enables experts to develop a role-oriented agent-based simulation program by MMDA, we selected SOARS as the modeling method specialized programming language for the role-oriented agent-based simulation. SOARS VisualShell requires only mouse operations and keyboard input without programming by a text editor and remembering the instructions and statement format. In addition, syntax errors will not occur because the string from keyboard input is checked automatically. Therefore, the user only needs to analyze the semantic bugs of the program.

SOARS Project [17] holds SOARS Workshop every year. By attending the intensive tutorials organized by experienced SOARS programmers, anyone can construct an agent-based simulation model by SOARS through the SOARS VisualShell within a short period.

We have started to develop onsite workflow language for internet of everything (OWLIE) that is a new processing flow description by using message queue telemetry transport (MQTT) protocol [18]. The environment contains both the real world and the virtual world constructed by role containers. OWLIE is a modeling method specialized programming language for describing the Real World OS.

Also, we have already started to develop Modeling GUI for OWLIE. In the internet of everything (IoE) era, for the cooperation of OWLIE and SOARS, it is required to evolve SOARS into a language to collect the sensor data and control the actuators in the real world as well to simulate autonomous agents in the virtual space.

In order to make the definition of MMDA clear, it is required to describe the model not only intuitively but also abstractly. Thereby, we will devise a new method for the abstract description of the model.

6. Conclusion

In this paper, we have focused on the model description in the domain of dynamic processes. In the dynamic process, the initial states are defined in entities based on their initial conditions, and the states of the entities and the environment surrounding them change from moment to moment due to the change of the boundary conditions. The system dynamics, the sequencer and role-oriented agent-based simulation all follow the same pattern. However, GUI design based on MMDA is not limited to the modeling method specialized language in dynamic process domain. If the modeling method is actually established by a specific modeling method and the modeling method specialized programming language in another domain, it is easy to develop GUI based on MMDA for the modeling method specialized programming language of the domain.

References


---

Tadashi Kurata  
He, born in 1960, received B.E. degree from Waseda University in 1984. He is a doctoral candidate in the Department of Computational Intelligence and Systems Science, Tokyo Institute of Technology, Japan. He has been developing software in PieCake Incorporated (http://www.piecake.com/).

Hiroshi Deguchi (Member)  
He, born 1955 in Tokyo, is Professor in the department of Computational Intelligence and Systems Science, Tokyo Institute of Technology, Japan. He holds a Dr. of Science from Tokyo Institute of Technology and a Dr. of Economics from Kyoto University. He serves in the board of directors of several associations, e.g.: Japan Association for Social and Economic Systems Studies, Pacific-Asian Association for Agent-based Approach in Social Sciences, Japan Association for Philosophy of Science, and he is President of Japan Association of Simulation and Gaming (JASAG). He works as editor in chief of “Agent-Based Social Systems” and (http://www.springer.com/series/7188), “Translational Systems Sciences” (http://www.springer.com/series/11213#) and is author of the book “Economics as an Agent-Based Complex System” (Springer, 2004).

Manabu Ichiyama (Member)  
He received B.E., M.S., and Ph.D. degrees from Tokyo Institute of Technology in 2004, 2005, and 2009, respectively. From 2009 to 2015, he was an assistant professor at Tokyo Institute of Technology. He is currently a senior researcher at National Institute of Public Health in Japan. His research interests include social simulation, simulation and gaming, crisis management and agent-based approach.