Development of a Tool to Keep Consistency between a Model and a Source Code in Software Development Using MDA.

Yuuki Kikkawa*, Tetsuro Katayama*, Yoshihiro Kita†,
Hisaaki Yamaba*, Kentaro Aburada‡ and Naonobu Okazaki*
**University of Miyazaki, Japan,†Tokyo University of Technology, Japan*
‡Oita National College of Technology, Japan
kikkawa@earth.cs.miyazaki-u.ac.jp, kat@cs.miyazaki-u.ac.jp, kitayshr@stf.teu.ac.jp,
yamaba@cs.miyazaki-u.ac.jp, aburada@oita-ct.ac.jp, oka@cs.miyazaki-u.ac.jp

Abstract

This study improves efficiency of software development using MDA. This paper develops the tool that keeps consistency between a model and a source code in software development using MDA. The tool has two functions: (i) Generating a source code and (ii) Keeping consistency between a model and a source code. A simple ATM example is used in order to confirm effectiveness of the tool. The tool can reduce time and effort to keep consistency between models and a source code.

Keywords: MDA (Model Driven Architecture), Extended Activity Diagram, Activity diagram, Detail specification.

1. Introduction

MDA (Model Driven Architecture) is a concept of software development.[1] MDA defines five models: business model, requirement model, platform independent model (PIM), platform specific model (PSM), and physics model. Each model has different abstraction level. A developer defines models and generates a less abstract model by software development in MDA. Here, MDA tools are used to generate a less abstract model. A developer uses UML (Unified Modeling Language)[2] for modeling PIM and PSM.

Before generation of a less abstract model, a developer must define generation rules of high abstract model. A method to support the definition of a generation rule is researched.[3]

One of the MDA's problems is how to keep consistency between the original model and an edited model which is generated from the original. A modification tool of PIM to keep consistency with PSM is researched.[4]

Also, there is no consistency if a developer edits the original model. A developer can keep consistency if a MDA tool generates models from the edited models again. Here, some MDA tools can generate a complete model from models including detail specification. A framework that generates the executable source code from a class diagram and a state machines diagram is researched.[5] However, MDA tools cannot generate complete models from abstract models because these models do not have detail specification of a system. The developer must modify generated models to fit the modified original models or generate a new model from the modified models with a MDA tool and then add the detail specification to the new model by hand again.

This study improves efficiency of software development using MDA. We proposed a modification method of a source code to correspond with a modified model in MDA.[6] However, we did not implement the method. In this paper, we develop the tool that keeps

consistency between a model and a source code in software development using MDA. This paper introduces the tool and describes how we implement the tool.

2. The Developed Tool

This chapter describes functions and an overview of the tool. Here, the tool treats with activity diagram of UML and programming language C++.

2.1. *Functions*

The tool has two functions: (i) generating a source code, (ii) keeping consistency between a model and a source code. (ii) consists of 3 steps. Fig. 1. shows each function.

An input of (i) is an activity diagram, and an output of (i) is the source code based on the activity diagram. Inputs of (ii) are an unmodified activity diagram, the modified activity diagram, and the source code including the detail specification. An output of (ii) is a source code which is consistent with the modified activity diagram and includes the detail specification.

In executing (ii), the tool generates an EAD (Extended Activity Diagram) as intermediate data. An EAD is a format of the data added a part of the source code which has the detail specification to the activity diagram.

2.2. *Overview*

Fig. 2 shows an overview of the tool. The tool consists of 4 parts shown below.

(i) Menu bar.
(ii) Mode select panel.
(iii) Multipurpose panel.
(iv) Paint panel.

A developer can execute a function of the tool by (i). By using (ii), a developer can select an operation he wants to do on (iv). (iii) assists the selected operation on (iv). (iv) shows activity diagram.

3. Implementation

This chapter describes how to implement each function. Here, the tool sets node ID and edge ID in the node and the edge that the developer describes. Node ID and edge ID are unique number in a diagram.

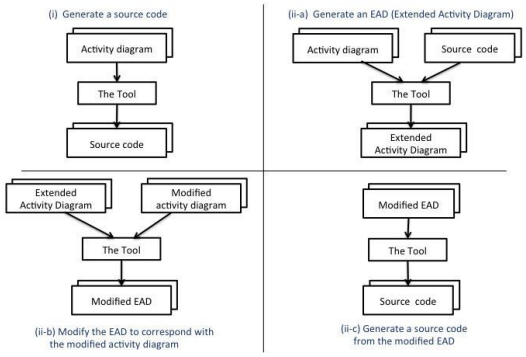

Fig. 1. Functions of the tool.

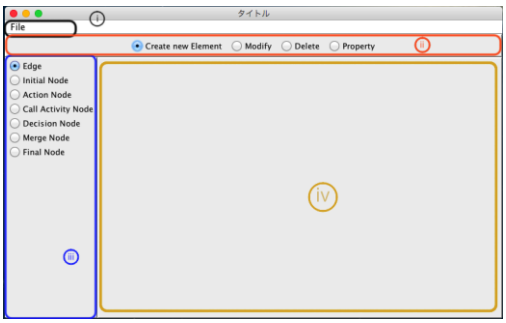

Fig. 2. Overview of the tool.

3.1. *Generating a source code*

The tool generates a source code from the activity diagram. The steps to generate a source code are shown as below.

(i) Acquire the function name
The tool generates a skeleton of source code. The function name is the activity name that is described in the activity diagram. Here, the type and the parameter of the function are void.

(ii) Select the initial node
The tool selects the initial node.

(iii) Implement the function
The tool executes the process as below depending on the type of the selected node.

- Call activity node
  Write the name of the call activity node to the source code.
- Decision node
  Write an if-statement to the source code. Conditions of if-statement are guard conditions of this node.

- Activity final node
  Finish the generation of the source code.
- Other than the above
  Do nothing.

(iv) Select another node
The tool reselects the node connected by the outgoing edge of the selected node and go to (iii).

3.2. *Generating an EAD*

The tool keeps consistency between a model and a source code. At the first step of the function, the tool generates an EAD. The steps to generate an EAD are shown as below.

(i) Select the first line of the source code added the detail specification.
- We call the selected source code "LOS (line of selected)"

(ii) Generate and extract a line of source code from the activity diagram.
- Generation rule of a source code is same as the function (i) of the tool.

(iii) If (ii) and the LOS are not same character string, execute the process shown as below.
  (a) Write the LOS to the activity diagram.
  (b) Encircle lines written in (a) as a node.
  (c) Select the original node of the generated source code in (ii).
  (d) Connect the incoming edge for the selected node to the node generated in (b).
  (e) Make an edge connected with the selected node and the node generated in (b).
  (f) Change the LOS to the next line of the current LOS.
  (g) Go to (iii).

(iv) Change the LOS to the next line of the current LOS.

(v) Go to (ii).

Later, we call an encircle node a platform specific node. We define a node other than a platform specific node as a platform independent node. We define an edge connected with a platform specific node as a platform specific edge. We define an edge other than a platform specific edge as a platform independent edge.

3.3. *Modifying the EAD*

At the second step of the function which keeps consistency between a model and a source code, the tool modifies the EAD to correspond with the modified activity diagram. The steps to modify the EAD are shown as below.

(i) Add an edge adjacent to a platform specific node.
  (a) Select a platform independent edge from an EAD.
  (b) If selected edge connects a platform specific node to a platform independent node and the platform independent node does not exist in the modified activity diagram, execute the process shown as below.
    1. Select a node in the modified activity diagram which has the same node ID as the selected node.
    2. Select a next node of the current selected node in the modified activity diagram.
    3. Select a node in the EAD which has the same node ID as the selected node in the modified activity diagram.
    4. Make an edge connected with the first selected node and the second selected node in the EAD.
  (c) If the selected edge connects a platform independent node to a platform specific node and the platform independent node does not exist in the modified activity diagram, execute the process shown as below.
    1. Select a node in the modified activity diagram which has the same node ID as the selected node.
    2. Select a previous node of the current selected node in the modified activity diagram.
    3. Select a node in the EAD which has the same node ID as the selected node in the modified activity diagram.
    4. Make an edge connected with the first selected node and the second selected node in the EAD.
  (d) If there is an unselected node, select an unselected node and go to (b).

(ii) Delete edges and nodes.
  (a) Select a node or edge from the EAD.

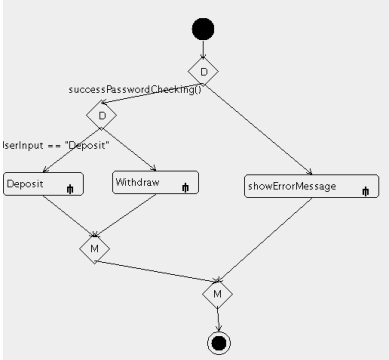

Fig. 3. The activity diagram of the ATM system.

(b) If the modified activity diagram does not consist of a node or edge which has the same ID as a selected node or an edge, delete the selected node or edge.
(c) If there is an unselected node or edge in the EAD, select an unselected node or edge and go to (b).

(iii) Add edges or nodes.
(a) Select a node or edge from the modified activity diagram.
(b) If the EAD does not consist of a node or edge which has the same ID as a selected node or edge, add the selected node or edge to EAD.
(c) If there is an unselected node or edge in the modified activity diagram, select an unselected node or edge and go to (b).

3.4. *Generating the source code from the EAD*

The tool generates a source code from the EAD. The steps to generate a source code are same as section 3.1. If the tool selects a platform specific node, the tool writes its name to the source code.

4. Application Example

We use a simple ATM as an example to confirm effectiveness of the tool. This ATM system executes password checking process. If a user successes password checking, this system executes a withdrawal process or a depositing process by a user's input. Fig. 3 shows the activity diagram of the ATM system.

```
void Transaction(){
if(successPasswordChecking()){
if(UserInput == "Deposit"){
Deposit();
}else{
Withdraw();
}
}else{
showErrorMessage();
}
}
```

Fig. 4. The generated source code.

The tool generates a source code from the activity diagram. Fig. 4 shows a generated source code.

The developer adds the detail specification to the generated source code in order to execute it. Fig. 5. shows the source code added the detail specification.

Suppose a case that the requirement of specification is changed, after adding the detail specification. The developer needs to modify the activity diagram by deleting and adding some elements of the activity diagram. Fig. 6 shows a modified activity diagram. The developer deletes the password checking process to sequence of deposit.

The tool generates a new source code from the source code including the detail specification, the modified activity diagram, and original activity diagram. Fig. 7 shows a generated source code. The source code includes the detail specification.

5. Discussion

MDA Tools such as EA[7] (Enterprise Architecture) can generate a skeleton of a source code from a class diagram. In addition, EA can generate a source code from an activity diagram or a state machine diagram. However, the source code generated by EA does not have the detail specification. It takes time and effort that the developer adds the detail specification to the source code generated from the modified activity diagram.

```
void Transaction(){
    string UserInput;
    cin >> UserInput;
if(successPasswordChecking()){
if(UserInput == "Deposit"){
Deposit();
}else{
Withdraw();
}
}else{
showErrorMessage();
}
}
```

Fig. 5. The source code added the detail specification.

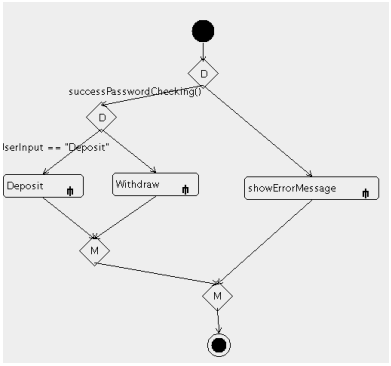

Fig. 6. The modified activity diagram.

The developed tool can generate a source code including the detail specification. The tool can reduce time and effort to add the detail specification to the source code generated from the modified activity diagram. Moreover, it can reduce time and effort to keep consistency between models and a source code after the requirement specification is modified. Therefore, the tool is useful for efficiency of software development using MDA.

6. Conclusion

This paper develops the tool that keeps consistency between a model and a source code in software development using MDA. The tool can generate the source code that corresponds with the modified activity diagram and has information about the detail specification.

We have confirmed that the tool can generate a source code including the detail specification from the original activity diagram, the modified activity diagram, and the original source code. Therefore, the tool is useful for efficiency of software development.

```
void Transaction(){
    string UserInput;
    cin >> UserInput;
if(UserInput == "Deposit"){
Deposit();
}else{
if(successPasswordChecking()){
Withdraw();
}else{
showErrorMessage();
}
}
}
```

Fig. 7. The source code corresponded with the modified activity diagram.

Future issues are as follows.

- Improvement of the tool to treat with other statements except if-statement.
- Improvement of the tool to treat with programming language Java.

References

1. MDA (Model Driven Architecture), http://www.omg.org/mda (accessed November 30, 2016).
2. UML (Unified Modeling Language), http://www.omg.org/spec/UML/2.4.1 (accessed November 30, 2016).
3. D. Lopes, S. Hammoudi, J. Bézivin, F. Jouault: Mapping Specification in MDA: From Theory to Practice, *Interoperability of Enterprise Software and Applications*, (Springer-Verlag London Ltd 2006), pp.253-264,
4. Lionel C. Briand, Yvan Labiche, Tao Yue: Automated traceability analysis for UML model refinements, *Information and Software Technology*, Vol. 51 (2009), Issue2, pp. 512-527.
5. A. Derezinska, Code Generation and Execution Framework for UML 2.0 Classes and State Machines, *IMCSIT*, (2008), pp. 517-524.
6. Yuuki Kikkawa, Tetsuro Katayama, Yoshihiro Kita, Hisaaki Yamaba, Kentaro Aburada and Naonobu Okazaki, Proposal of a Modification Method of a Source Code to Correspond with a Modified Model in MDA, *International Conference on Artificial Life and Robotics*, (2015), pp. 384-387.
7. Enterprise Architect, http://www.sparxsystems.jp (accessed November 30, 2016).