# Requirements on Model Transformation for Model Refactoring in State Transition Model Description Language stmc

Takumi ICHIDA[1,a)] Nobuhiko OGURA[1,b)]

***Abstract:*** This article discusses the issues in the model transformation for a model refactoring supporting tool. As a basis of this discussion, the article shows examples of the model transformation to a state machine model described by state transition model description language stmc and provides a model-to-model transformation program for the stmc model. In the field of model refactoring, numerous approaches to describe the model and model transformations have been introduced. Although, the ambiguity of semantics in the modeling language makes it difficult to describe software models and model transformations and to preserve model behaviors before and after model transformations. In this article, to avoid such ambiguity, we focus on the state transition model description language stmc which can define the state machine model using clear and informal semantics.

***Keywords:*** Model Transformation, Model Refactoring, State Machine Diagram, stmc

## 1. Introduction

Software models help to represent the abstract structure of the system as the concrete diagram in the development process of embedded systems. Model refactoring techniques which is the application of refactoring techniques [1] to software models can improve readability and maintainability by changing the internal structure of models without changing their external behavior. Numerous works such as a model refactoring for UML class diagrams and a model transformation using graph transformation systems(GTS) have been developed to suggest approaches describing a model and a model transformation [2]. However, the ambiguity of semantics in the modeling language causes difficulties in describing the model and the model transformation and preserving the model's behaviors before and after the model transformation. The refactoring of Executable UML [3] is one of the cases which use the modeling language with clear semantics in the model transformation. Since ambiguity should be avoided, we focus on the state transition model description language stmc, which can express the elements of a state transition model with clear and informal semantics.

This article discusses issues in the stmc model transformation for a model refactoring supporting tool. We specify two issues that need to be overcome for users to use the model refactoring tool: how to depict the area to be transformed, and how to input prior information for the automated transformation process. In addition, we show issues of the behavior preservation and difficulties of the scope alteration derived from the language structure of stmc. To discuss issues, we develop concrete examples of stmc model modifications. We also develop a model-to-model transformation program for the stmc model to execute these example modifications. This discussion clarifies several requirements on a model transformation for creating a model refactoring supporting tool. Future works are needed to classify effective model modification examples and to identify the input forms for a model refactoring supporting tool. Furthermore, we plan to develop a model refactoring supporting tool for stmc models.

This article starts with explaining the language and stmc model modification examples. Then we describe the model transformation program to implement these examples. Based on these examples, we discuss issues on stmc model transformation. Lastly, we introduce related works and future works.

## 2. Model Modification Example

This section introduces the model modification examples for state machine models written in stmc. stmc is the programming language that can describe state machine models using a syntax resembling C programming language. This language can write the state machine model in the C program.

We develop following four modification examples which focus on the transition: (1) MoveCodeToTransitionAction (2) SeparateGuardString (3) SeparateSourceState (4) GroupSourceState.

### 2.1 MoveCodeToTransitionAction

If the action that should be executed at the transition appears in the execution program, the domains of the model and the model execution program will be confused. This transformation inserts a part of the execution program into the transition action. This transformation moves actions written in the execution program into actions in suitable transition.

[1] Tokyo City University, Setagaya, Tokyo, 158–8557, Japan
[a)] g2083102@tcu.ac.jp
[b)] ogura@tcu.ac.jp

### 2.2 SeparateGuardString

A composite condition is one of the factors that reduce the maintainability of a source code. This example decomposes a long transition condition and changes it into multiple transitions with short conditional expressions. It becomes easy to add or modify conditions in the future by subdividing the conditional expression into multiple statements. **Figure 1** shows the part of this modification example.

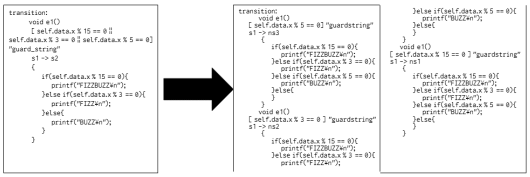

**Fig. 1** Example of SeparateGuardString

### 2.3 SeparateSourceState

When a transition has multiple source states, it is difficult to handle the current state in the transition action, which leads to more complicated transition actions. This transformation changes a transition from multiple states to a single state into multiple transitions with a one-to-one correspondence between the source state and the destination state. Although this transformation increases the overall number of transitions, it improves readability and maintainability of each transition.

### 2.4 GroupSourceState

stmc can define a state group that combines multiple states like one state. Another example of multiple source states is the use of this state group. In this case, the transformation grouping transition source states as a group facilitates the handle of the current state in the transition action.

## 3. stmc Model Transformation Program

We developed a program that receives the stmc model, edits it, and outputs it to the stmc model. **Figure 2** shows the process of the stmc model transformation program. This transformation program can modify the model by adding or deleting elements of the syntax tree and model data.

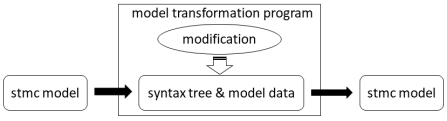


**Fig. 2** Process of the model transformation

## 4. Discussion

This section presents issues that can be discussed using transformation examples in Section 2 and the transformation program.

### 4.1 Issues in Using Tool

#### 4.1.1 How to Specify Area

It is necessary to make it clear how to specify the elements of the model which need to be transformed by refactoring. For instance, an intuitive method for users to indicate the source code in the execution program should be considered in *MoveCodeToTransitionAction*. Other than that, it is required to provide a method to specify a target transition that doesn't have an identifier in *SeparateSourceState* and *GroupSourceState*.

#### 4.1.2 How to Input Prior Information

Before implementing model refactoring, the user should know what kind of information is required as prior input forms. When new transitions are created in the process of *SeparateSourceState*, users should be able to input the value of "action_string" and "guard_string" arbitrarily. Moreover, although *SeparateGuardString* adds the state and *GroupSourceState* adds the state group, it is better for the users to decide these names than to name them automatically in the process.

### 4.2 Issues in Model Structure

#### 4.2.1 Behavior Preservation

Sometimes the state machine model generated after a model transformation has a different behavior than before the transformation. However, according to the definition of refactoring, the behavior must be the same before and after the transformation. The risk of changing the behavior is high in *MoveCodeToTransitionAction* because the order of elements in the source code such as the execution program and transition actions is changed. In addition, the process of separating conditional statements in *SeparateGuardString* may change the behavior, since an automatic transformation may result in a change of branching.

#### 4.2.2 Scope Alteration

stmc is an extension of the C language. Therefore, there are problems with the scope alteration that exists in an ordinal programming language. This problem occurs not only when moving source code without considering the scope structure between the transition action and the execution program in *MoveCodeToTransitionAction*, but also when moving a part of the source code across packages or classes.

## 5. Related Works

Previous studies have proposed model refactoring tools for UML models. Since we are targeting a language with explicit semantics, we can utilize refactoring in ordinal programming languages.

## 6. Conclusion

We present four issues in the stmc model transformation: how to specify the elements in the model, how to input prior information, behavior preservation, and scope alteration. As future works, we plan to classify effective model modification methods as model refactoring and identify the sufficient input forms for the implementation of a model refactoring. Furthermore, we shall develop a program that enables model refactoring for stmc models towards a model refactoring supporting tool.

### References

[1] Martin, F. and Kent, B.: *Refactoring: Improving the Design of Existing Code*: Addison-Wesley (2019).
[2] Misbhauddin, M. and Alshayeb, M.: UML model refactoring: a systematic literature review, *Empirical Software Engineering*, Vol.20, No.1, pp.206-251 (2015).
[3] Mellor, S.J. and Balcer, M.J.: *Executable UML: A Foundation for Model-Driven Architecture*, Addison-Wesley (2002).