2008 International Symposium on Nonlinear Theory and its Applications
NOLTA'08, Budapest, Hungary, September 7-10, 2008

Constructing State Machine Models by Using Petri Nets for the Extended ROOM Method

Toshiyuki Miyamoto†, Hiroyuki Kurahata†, Taku Fujii‡, and Sadatoshi Kumagai†

†Graduate School of Engineering, Osaka University, Osaka, Japan
‡Software Engineering Center, Osaka Gas Info. Syst. Res. Inst., Osaka, Japan
Email: miyamoto@eei.eng.osaka-u.ac.jp

Abstract—Service Oriented Architecture is an architecture style to build up a large-scale networked system composed of a set of components or functions, each of which is called a service. The Extended ROOM method has been proposed as a method to generate behavioral model of a system based on SOA and to validate the system on the behavioral model from the early state of system development. In this paper, we proposed a method to generate state machine models of the target system by using Petri nets.

1. Introduction

Service Oriented Architecture (SOA)[1] is an architecture style to build up a large-scale networked system composed of a set of components or functions, each of which is called a service. Recently, SOA attracts attention because of the expectation to reduce time and cost of information systems development. Since application area of SOA includes mission critical systems such as the business process management system, correctness assurance of the developed system is important.

The Real-time Object-Oriented Modeling, ROOM[2] for short, is well-known for both an object-oriented language and a methodology based on the language to the specification, design, and construction of software for distributed real-time systems. In ROOM, high-level architectures are expressed by using a simple graphical notation, and also by using UML *class*, *collaboration*, *state machine* and *sequence diagrams*[3]. The functionality of distributed entity is realized by the state machine diagram, and their interaction can be shown as the sequence diagram. In UML2.0, a collaboration diagram is a kind of *composite state diagram*[4], therefore we use the composite state diagram instead of the collaboration diagram.

In the early stage of system development it is not easy to define state machine models, SMs for short, of a distributed entity due to the following two essential difficulties for developers. At first, there is huge semantic gap between requirements and state machine models. Secondly, not all developers are familiar with the state machine diagram. Jon et al.[5] proposed a generation method of SMs from sequence diagrams and Object Constraint Language (OCL). Leue et al.[6] proposed a generation method of SMs and ROOM models from Message Sequence Charts. Liang et al.[7] survey studies of these generation method of SMs from scenarios, and they classify such methods based on class of scenarios, class of generated SMs, ability of detecting concurrency, and degree of automate.

Even if the model of the target system is designed, we will face the problem of validation of the design model. Since, interactions between services are highly complicated, it is difficult for developers to validate manually without overlooking design defects. Therefore, manual validations in the design stage often leave errors till the implementation stage. Errors detected in the implementation stage increase development costs and prolong development term.

We have proposed the Extended ROOM method[8] to overcome the above problems, and by using the Extended ROOM method developers can define design models in the following steps:

1. Define connection between services by the composite structure diagram.
2. Define interactions between services by *communication diagrams*, CDs for short.
3. Generate SMs from CDs
4. Define guards and actions in generated SMs.
5. Simulate these SMs by the simulator against test cases, and refine these models.

We think that writing CDs and composite structure diagrams may be much easier than writing SMs, and generating SMs from CDs supports developers to design systems based on SOA.

In this paper, we discuss the step 3 in the above procedure. Our method makes developers possible to generate SM models from CDs semi-automatically. However, due to the limitation of page numbers, formal description of our method is omitted in this paper. Outline of this paper is as follows: In Sect. 2, we show the construction method of SMs by using Petri nets. In Sect. 3, we remark conclusion of this paper.

2. Constructing State Machine Model

2.1. Outline

At the beginning, we assume that composite structure diagrams and CDs for the target system have already written.

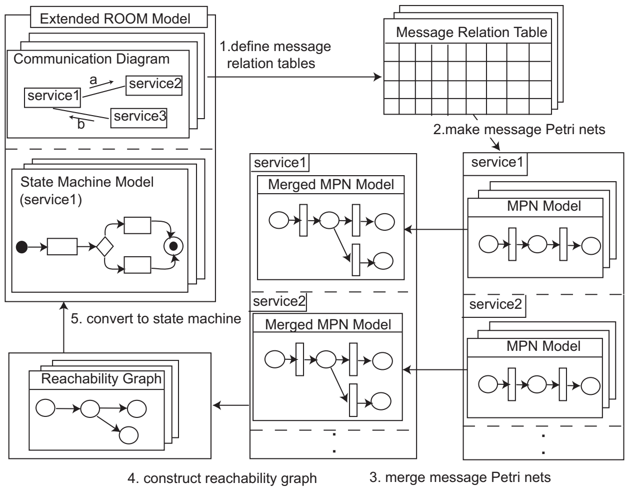

Figure 1: Outline to construct state machine models.

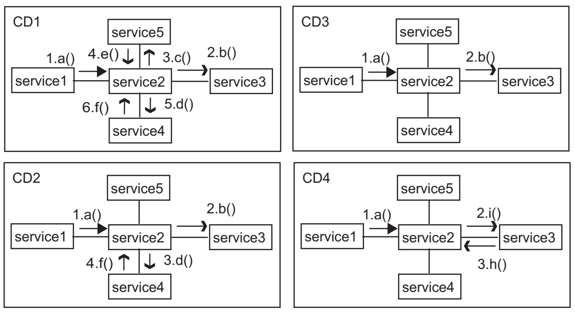

Figure 2: CDs for the example system.

This section describes how SMs are constructed from CDs by our proposed method.

Figure1 shows the outline to construct SMs from given CDs. Our construction method consists of five steps: (1) defining message relation tables step, (2) making message Petri nets step, (3) merging message Petri nets step, (4) constructing reachability graphs step, and (5) converting into SMs step.

2.2. Example System

We use the same example through this paper. The example system is composed of five services, and their interaction cases are described by four CDs shown in Fig. 2.

A CD shows the interaction among the services. The rectangular node represents a service, and lines between nodes show communication paths. Messages between services are shown by labeled arrows near connector lines. There exist two message types: synchronous and asynchronous messages. The synchronous, resp. the asynchronous, message is indicated by an arrow with a filled solid arrowhead, resp. a stick arrowhead. The return message for synchronous messages is not written in CDs normally. The number around the message is called the *sequence number*.

	a	b	c	d	e	f	re_a
a		→	→	→	→	→	→
b			→	→	→	→	→
c					→		→
d						→	→
e							→
f							→
re_a							

Figure 3: The MRT for CD1 in Fig. 2.

2.3. Message Relation Tables

Order relations between messages in CDs have to be defined precisely to generate SMs. Using the sequence number is one of methods to define the order relations. We, however, think that the sequence number does not have enough potential to describe order relations precisely. One of the major reasons is that it is difficult to express concurrent relations among messages, especially for asynchronous messages. Another reason is that the return message for a synchronous message does not appear in CDs.

To resolve this problem, we introduce a table to express ordering relations among messages, called the message relation table(MRT). Fig.3 shows an example of the MRT. This MRT express ordering relations among messages of CD1 in Fig.2. The MRT shows that the message `a` is a predecessor of the message `b`, for example, and `re_a` is the return message of the synchronous message `a`.

2.4. Message Petri Nets

Once CDs and their MRTs are defined, we are ready to generate SM models. For the effective construction, we use a kind of Petri nets[9] as an internal modeling language. The Petri nets are called the message Petri net, MPNs for short, and it is a labeled Petri net such that each transition has a label to a message.

A MPN is constructed from CDs and MRTs, and then the MRT is simplified by using several rules that merges nodes.

The initial MPN can be directly made from CDs and MRTs. Fig. 4 shows MPNs made from CDs in Fig. 2, where MRT of CD1 is shown in Fig. 3, and for other CDs we assume that the sequence numbers defines total order among messages.

Since the initial MPN contains redundant transitions and places, we reduce the MPN by using merging rules. Fig. 5 shows intuitive description of the merging rules, and Fig. 6 shows the merged MPN by applying the merging rules to the MPN in Fig. 4.

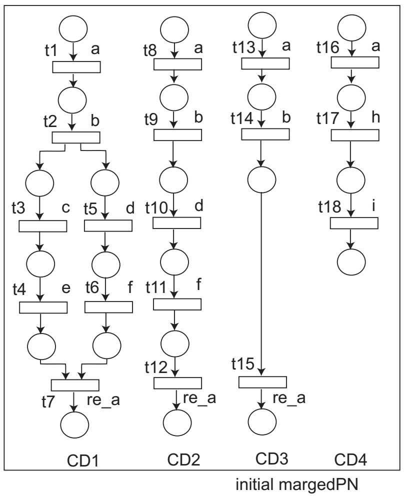
Figure 4: MPNs for service2 constructed from CDs in Fig. 2.

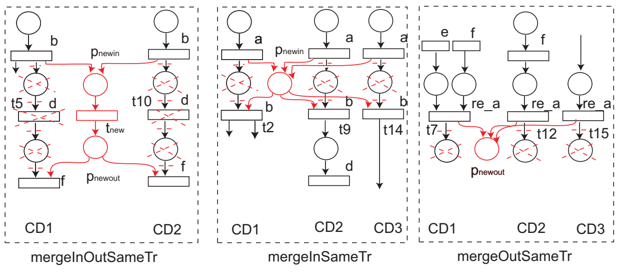
Figure 5: Rules for merging MPNs.

2.5. Constructing Reachability Graphs

At this step, we construct the reachability graph of the MPN so as to generate SM model of the system. Fig. 7 shows the reachability graph constructed from the MPN shown in Fig.6. Each label in a node shows marked places, namely a marking, and each label outside of a node is the node name. Label of an edge shows a message. In the reachability graph, there may still exist redundant structures. For example see node `n1` in Fig. 7, there exist two outgoing edges with the same label of message `a`. Before converting a reachability graph into a state machine, we simplify the reachability graph by using a rule.

The simplified reachability graph from Fig.7 is shown in Fig.8. In the figure a label in a node indicates the node name of the original reachability graph.

2.6. Converting to State Machine Models

Finally, we can get a SM model for each service by using a conversion rule from the reachability graph to the SM. Fig. 9 shows the resulting SM model.

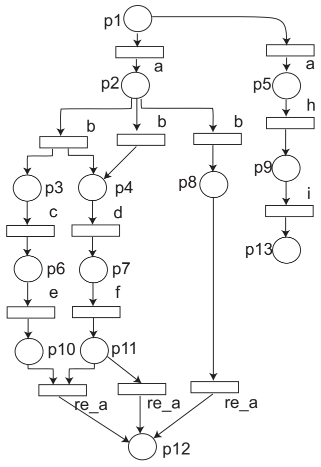
Figure 6: Merged MPN for the MPN in Fig. 4

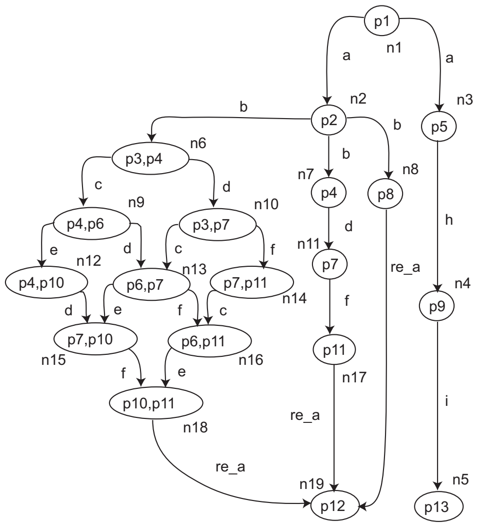
Figure 7: Reachability graph of the MPN in Fig. 6.

2.7. Discussions

Let us discuss the derived SM model. From the CDs in Fig. 2, the following 9 message sequences can be derived.

1. `a,b,c,d,e,f,re_a`
2. `a,b,c,e,d,f,re_a`
3. `a,b,c,d,f,e,re_a`
4. `a,b,d,f,c,e,re_a`
5. `a,b,d,c,e,f,re_a`
6. `a,b,d,c,f,e,re_a`
7. `a,b,d,f,re_a`
8. `a,b,re_a`
9. `a,h,i`

The SM shown in Fig.9 can execute all of these message sequences. However, another sequence can be executed in this SM, for example `a,b,c,d,re_a`. The cause

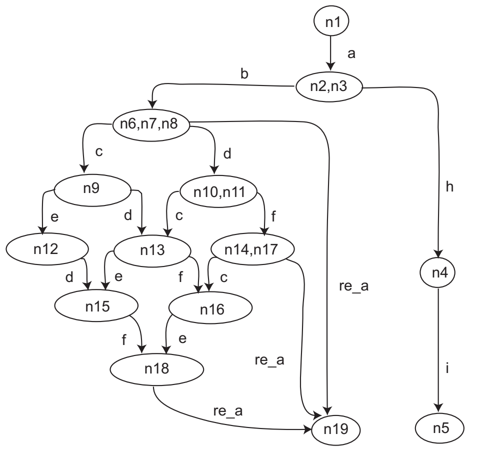

Figure 8: Simplified reachability graph for Fig. 7.

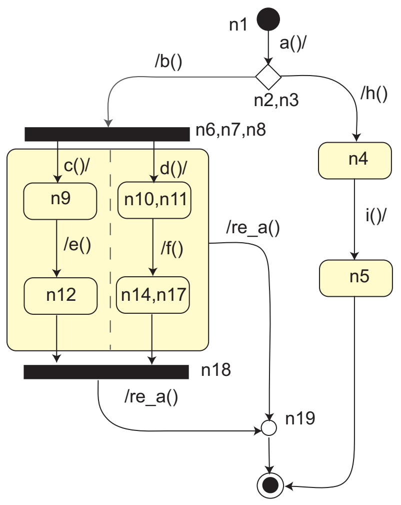

Figure 9: Generated SM model from the reachability graph in Fig. 8.

of these extra sequences is the transition from a composite state having two concurrent regions. When the SM is generated, several transitions from child states in the composite state to states in outside of the composite state are merged into these transitions. These extra sequences may be undesirable behavior from the point of service semantics. We think whether those extra message sequences are valid or not can be checked by developers.

After generation of SMs for services, developers need to define following additional elements in SMs to simulate behaviors of the system.

Guard Condition Transitions from a decision and transitions from a composite state need guard to determine the transition to be executed. The SM is able to become deterministic with guard expression.

Actions To simulate system behavior in detail, developer need to define actions to define detailed semantics of services.

3. Conclusions

In this paper, we have improved extended ROOM method in following two points. At first, we proposed the MRT to resolve ambiguity of message orders between asynchronous and synchronous messages in CDs. Secondly, we proposed an improved generation method of SM models from CDs and MRTs with by using the message Petri nets. By these two proposals, we succeed in generating SM models which are able to simulate service behaviors.

Our future work includes generating test cases and testing scripts for the SM models in order to effective validation of systems based on SOA.

References

[1] E. Thomas, Service-Oriented Architecture, PRENTICE HALL, 2004.

[2] B. Selic, G. Gullekson, and P.T. Ward, REAL-TIME OBJECT-ORIENTED MODELING, John Wiley & Sons, Inc., 1994.

[3] B. Selic and J. Rumbaugh, “Using UML for Modeling Complex Real-Time Systems,” 1998. http://www.ibm.com/developerworks/rational/library/139.html.

[4] J. Rumbaugh, I. Jacobson, and G. Booch, The Unified Modeling Language Reference Manual, Second Ed., Addison-Wesley, 2004.

[5] W. Jon and S. Johann, “Generating Statechart Designs From Scenarios,” 22nd International Conference on Software Engineering, pp.314–323, 2000.

[6] S. Leue, L. Mehrmann, and M. Rezai, “Synthesizing ROOM Models from Message Seauence Chart Specifications,” Technical report, Dept. of Elictrical and Computer Engineering, University of Waterloo, 1998.

[7] H. Liang, J. Dingel, and Z. Diskin, “A Comparative Survey of Scenario-based to State-based Model Synthesis Approaches,” 5th International Workshop on Scenarios and State Machines: Models, Algorithms and Tools (SCESM’06), pp.5–11, 2006.

[8] H. Kurahata, T. Fujii, T. Miyamoto, and S. Kumagai, “A UML Simulator for SOA Based on Agent Net Model,” IPSJ SIG Technical Reports, Vol.2007, NO.97, 2007. (in japanese).

[9] T. Murata, “Petri Nets: Properties, Analysis and Applications,” Proc. of The IEEE, Vol. 77, No. 4, 1989.