

Transformation of UML Diagrams into Interval Temporal Logic and Petri nets for Real-Time Systems Design

Ryuji Gushiken†, Morikazu Nakamura‡, Shinji Kono† and Kenji Onaga‡

†Department of Information Engineering,
University of the Ryukyus

1 Senbaru, Nishihara, Okinawa, 903-0213 Japan
Tel: +81-98-895-8715, Fax: +81-98-895-8727

E-mail: ryuji@ads.ie.u-ryukyu.ac.jp,
{morikazu,kono}@ie.u-ryukyu.ac.jp

‡Okinawa Research Center,

Telecommunications Advancement Organization
1 Asahimachi, Naha, Okinawa, 900-0029 Japan

E-mail: onaga@ie.u-ryukyu.ac.jp

Abstract: We consider, in this paper, a UML-based design support system for real-time systems. However, the UML does not include any notion for verification of timing constraints. We presents transformation algorithms, as a function of the support system, of UML descriptions into Petri nets and Interval Temporal Logic models, which are very powerful for the verification. This paper shows also transformation example for simple elevator system.

1. Introduction

A real-time system is a system with timing constraints and strict requirements of reliability and high performance. Therefore, the verification and evaluation are very important at the designing stage of real-time systems.

The UML(Unified Modeling Language)[1] is the formal standard modeling language in the OMG (Object Management Group) and is becoming a standard description language for object oriented modeling. For real-time systems' designer, however, UML is insufficient in areas critical to the development of real-time systems[3]. Therefore, developing verification or performance analysis algorithm incorporating with the UML is strongly required in software engineering.

On the other hand, Petri nets and temporal logic can be regarded as description languages. Many algorithms well based on them to verify logical validity and performance have been developed since they are based on mathematical formulation.

In this paper, we propose a technique to transform a UML model into Petri nets and temporal logic (Interval Temporal Logic), then the transformed model is directly used for correctness verification and performance evaluation.

2. Real-Time Systems and UML

Real-time systems have strict timing constraints. In this paper, we consider a UML-based design support system for real-time systems. The UML combines graphical and textual notations to describe systems. The UML is primarily a set of notations and does not prescribe specific development process. The UML contains several different types of diagrams, which allow to express different aspects and properties of the system to be developed. The UML includes two forms of diagrams, sequence and

collaboration diagrams, to represent message trajectory among (active and passive) objects in the system. The state and activity diagrams represent the state and behavior of the system. These four diagrams can represent dynamic properties and can allow us to model real-time systems.

Let us consider an elevator system as a simple example of the real-time system. Figure. 1(a) shows a usecase digram of the system. The system is composed of four objects, "User", "Controller", "Box", and "Door". The usecase diagram represents relation among objects.

The sequence diagram shown in Figure. 1(b) explains the detailed *message* trajectory among objects: *User* object sends "request" to *Controller*, *Controller* object receives the "request", *Controller* sends *messege* to *Box* to move to the requested floor and also to *Door* to open or close, and so on. Figure. 1(c) depicts the statechart of the system. It represents the relation among the system states.

Verification and performance evaluation of a real-time system at the deigning stage is very important. The purpose of the research is to introduce these functions into a UML-based design support system for real-time systems.

It is known that a considerable number of studies have been made on verification and evaluation using Petri nets or temporal logic. Thus, in this paper we propose a transformation technique of UML models(sequence diagram, statechart diagram) into Petri nets and temporal logic. The detail algorithm of the transformation is taken up in the next section.

3. Transformation

We show detailed algorithms of transformation a UML model into Petri nets and temporal logic, respectively. Here we just consider simple UML sequence and statechart diagrams:

We assume that the sequence diagrams do not include any loops and the statechart diagrams are not composed of *composite states* and/or *concurrent substates*.

3.1 Dealing with Time

In the real-time systems' design, considering timing constraints is very important. Here we explain how to deal

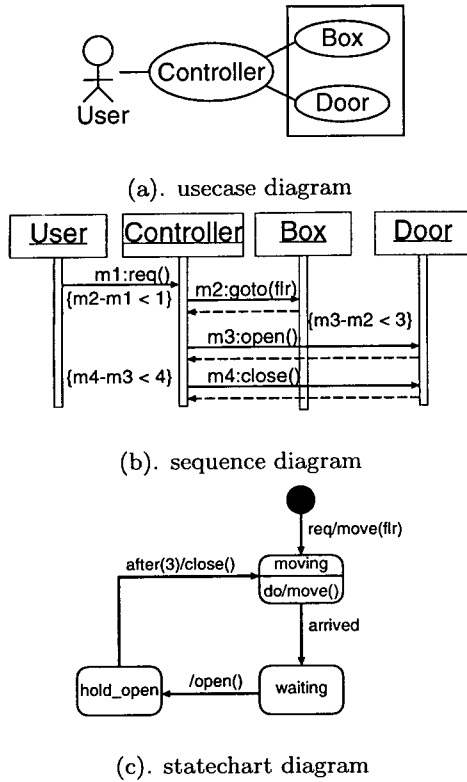


Figure 1. UML description of an elevator system

with Time in the UML design. Two kinds of timing information are needed to be modeled:

Latency: which is the amount of time it takes for two objects to communicate each other

Duration: which is the amount of time for an object to execute its processing since it has received *message*

To put it concretely, we consider the sequence diagram. As shown in Figure. 2, the life cycle of an object can be regarded as a sequence of activations and communications.

Latency is described in UML by attaching the value to the corresponding *message* and duration to the *activation*. For example, T_c shows a latency and T_a a duration in Figure. 2.

When it comes to statechart diagrams, we need to treat staying time at a *state*.

3.2 Transformation into Petri nets

In the real-time systems design, four types of communication primitive can be considered; synchronous sending, receiving, and asynchronous sending, receiving. These are denoted by different types of arrows in the UML as shown in Figure. 2.

Let us consider Petri nets representation of the sequence diagram. An *activation* is expressed by a timed transition such that the *activation* time (duration) corresponds to the firing time of the transition.

Synchronous sending is represented by the net in Table. 1 (c). The firing of t_1 means sending a *message* and

that of t_2 does receiving the acknowledge(or some return value). After firing of t_2 , the object can perform the following *activation*. On the contrary, the synchronous receiving can be represented by the net in Table. 1 (d). Timed transition t' corresponds to the processing in the procedure and $T(t')$ is the processing time.

Each pair of $message^-$ and $message^+$ is connected by the net in Table. 1 (b). Timed transition t_i represents the *message* transmission and $T(t_i)$ is its Latency.

Asynchronous sending requires just an immediate transition as shown in Table. 1 (e). For the asynchronous receiving, the *mail box* is introduced shown as Figure. 1 (f). During the activation the receiver object can fetch *messages* from the *mail box*. Both timed transitions t_1 and t_2 compose an *activation*, that is, t_1 is the beginning and t_2 the ending part. Therefore $T(t_1) + T(t_2) = T$.

Similar to the synchronous communication, each pair of $message^-$ and $message^+$ is connected by the net in Table. 1 (b). So now the transforming is summarized as follows.

«sequence diagram \Rightarrow PNs»

1. Transform each component in *lifeline* to subnet in accordance with the rules in Table. 1
2. By introducing a place, connect between subnets
3. Connect by the subnet in Table. 1 each pair of $message^-$ and $message^+$

In case of statechart diagrams, the transforming is easy, just replace each *messages* by a *transition* and each *state* by a timed-place and introduce arcs in Petri nets to connect transition-place pairs.

3.3 Interval Temporal Logic

ITL(Interval Temporal Logic) uses a sequencing model operator as its basis. In this logic, it is very easy to express control structures in conventional programming language[2].

We show informal visual representation of basic operators in ITL. An interval is a finite line which has the number of clock ticks. An operator *empty* is true on the length 0 interval.

The *nextoperator* $@P$ means P becomes true after one clock cycle. Thus, in ITL $@P$'s interval must be one clock cycle longer than P 's and $@P$ is false on the empty interval. P can be any temporal logic formula.

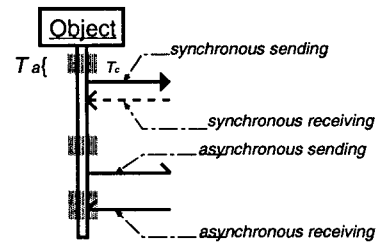
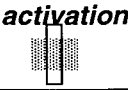
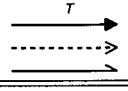
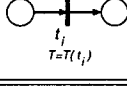
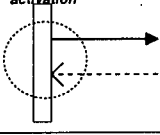
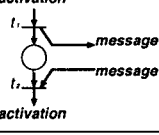
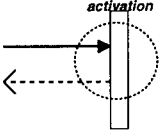

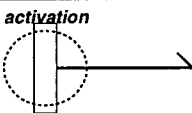
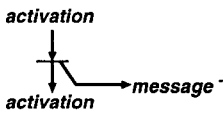
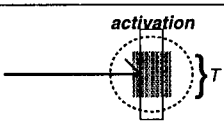
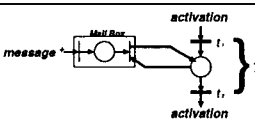


Figure 2. life-cycle of an object

Table 1. transformation rules

UML		Petri nets
activation		
(a)		$t \xrightarrow{\quad} T(t)$
message		
(b)		
synchronous sending		
(c)		
synchronous receiving		
(d)		
synchronous sending		
(e)		
synchronous receiving		
(f)		

We introduce the chop operator '&' which combines two intervals. $P&Q$ roughly means "do P then Q ".

We can verify timing constraints of real-time systems by these temporal operator apply to UML models, though we may leave the details to [2].

The following description illustrates that a technique to translation of sequence diagram and statechart diagram into ITL.

«sequence diagram \Rightarrow ITL»

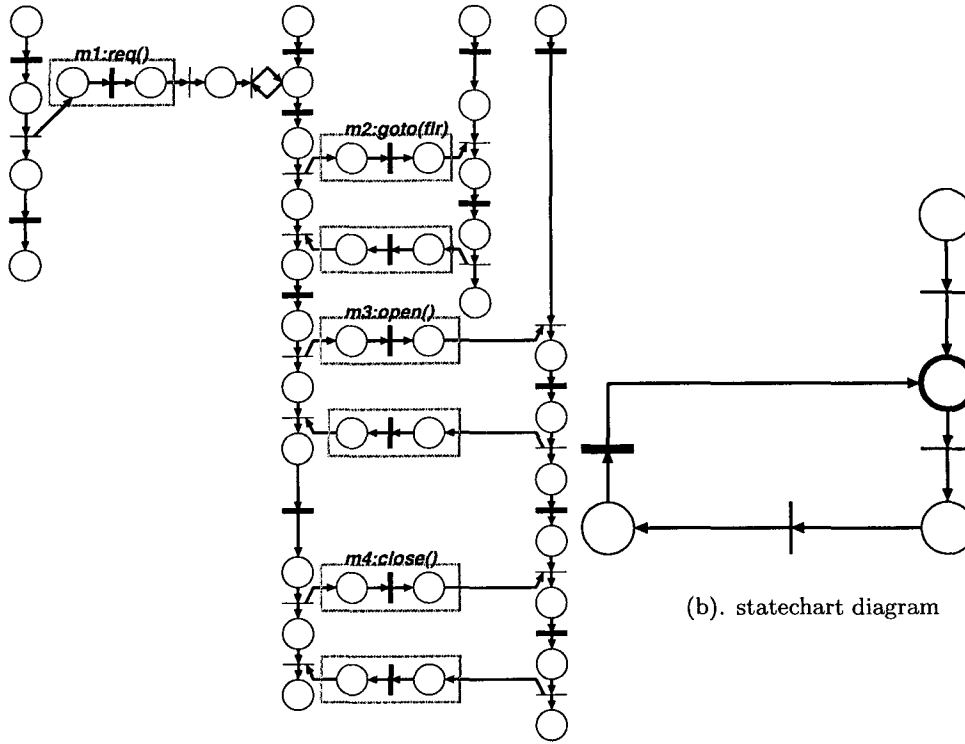
```
String seq2itl() {
  String expression;
  while(until message is empty) {
    /* to ignore return message */
    mi = current_message;
    mi+1 = next_message;

    ti = (mi+1) - (mi);

    expression.append( " ^ "+
      "(mi ^ @(less(ti) ^ mi+1)) " );
  }
  return expression;
}
```

«statechart diagram \Rightarrow ITL»

```
String state2itl() {
  String expression;
  while(to visit all states) {
    si = current_state;
    while( state is connected to si ) {
      si+1 = next_state;
      ei+1 = connected edge to si+1
      ti = si.process_time ;
      String current = "(" + si;
      String future = "(" + less(ti);
      if(ei+1 has event)
        current.append( ^ + event + "(" );
      if(ei+1 has guard_condition + "(" )
        current.append( ^ + guard_condition )
      if(ei+1 has action) {
        future.append( ^ + action&si+1 + "(" )
      } else future.append( ^ + si+1 + "(" );
      expression.append( ^ + ((current) ^
        @(future));
    }
  }
  return expression;
}
```



(a). sequence diagram

(b). statechart diagram

Figure 3. transformed real-time systems

4. Example

Let us consider the elevator system shown in Figure. 1. First, we transform the sequence diagram and statechart into Petri nets. Figure. 3(a) and Figure. 3(b) indicate the transformed results. There are many reports for performance evaluation based on Petri nets in the literature[4, 5].

Secondly, we express timing constraints of elevator system in ITL. The ITL representations of the elevator system(Figure. 1) transformed by our algorithm are presented in below.

«sequence diagram»

$$(m_1 \wedge (less(1) \wedge @m_2)) \wedge (m_2 \wedge (less(3) \wedge @m_3)) \wedge (m_3 \wedge (less(4) \wedge @m_4))$$

«statechart diagram»

$$(((moving \wedge arrived) \wedge @(less(1) \wedge waiting)) \wedge ((waiting) \wedge @(less(1) \wedge (open() \& hold_open)))) \wedge ((hold_open \wedge after(3)) \wedge @(less(1) \wedge (close() \& moving))))$$

[2] introduced several methods of verification, and implements automatic theorem provers for ITL.

In spite of importance of verification methods, it is not the main point in this paper. We will treat the details at other papers.

5. Conclusion and future works

This paper proposes a transformation technique of UML model(sequence diagram and statechart diagram) into Petri nets and ITL, respectively, for real-time systems design and shows a small example.

Our future works are to implement a support system based on UML and to evaluate its efficiency.

References

- [1] Object Management Group, "OMG Unified Modeling Language Specification version 1.3," 1999.
- [2] S. Kono, "Automatic Verification of Interval Temporal Logic," TM-92-007, 8th British Colloquium For Theoretical Computer Science, 1992.
- [3] M. J. McLaughlin and A. Moore, "Real-Time Extensions to UML," *Dr. Dobb's JOURNAL*, Dec., 1998.
- [4] G. Balbo and M. Silva ed., "Performance Models for Discrete Event Systems with Synchronous: Formalisms and Analysis Techniques," *MATCH Human Capital and Mobility CHR-X-CT94-0452*, vol. II, 1998.
- [5] C. Girault and R. Valle ed., "Systems Engineering: A Petri Net Based Approach to Modelling, Verification, and Implementation," *MATCH Human Capital and Mobility CHR-X-CT94-0452*, 1998.