Synthesis of Timing Scenarios for Embedded Systems using Modular Petri nets

Woo Jin Lee*, Young Joon Park*, and Ho Kyoung Lee**
EECS School*, Network Infra Laboratory of Korea Telecom**
Kyungpook National University
1370 Sangyug-dong, Book-gu, Daegu,
KOREA
http://selab.knu.ac.kr

Abstract:- In developing time-critical systems such as real-time systems and embedded systems, it is important to check timing conflicts between timing requirements as earlier as possible. For checking timing conflicts, at least, a formal notation should be introduced for a concrete and unambiguous requirements specification. However, in an earlier development phase it is not easy to describe timing requirements by using formal methods. In this paper, we propose a systematic procedure for transforming and synthesizing timing scenarios of embedded systems into a Petri net-based model. Although our approach is based on the Petri net formalism, users only focus on describing timing requirements in the scenario concepts, since the detailed transformation and integration procedures based on Petri nets are hidden to users.

Key-Words: - Embedded system, real-time system. Requirement analysis, scenario composition, Petri nets

1 Introduction

When developing real-time or embedded systems such as process control systems, patient monitoring systems, flight control systems and weapon systems, whose timely response is critical, it is very important to analyze the timing behaviors in as earlier a development phase as possible. Otherwise, rectifying requirement errors may need much more efforts and costs. In order to allow checking of requirement conflicts, requirements need to be unambiguously described using formal methods.

There have been many researches on representing and analyzing timing behaviors using formal methods in the literature. However, since formal approaches mainly focus on verification of the specification, the requirements elicitation process was not addressed and the specification process was not user-friendly. Therefore, non-experts find it difficult to formally describe requirements, since modeling works require one's knowledge and experiences of using formal methods.

The use case approach is arguably one of the best known and most widely employed in the industry. The use case approach has a few practical advantages in describing external behaviors; it is scalable, traceable, and relatively insensitive to requirement changes [1]. Although a few many researches such as message sequence chart and labeled transition system [1,2], Statecharts [3], Finite State Automata [4], and Timed automata [5] have been reported for formalizing use cases, few works on timing scenarios can be found. Sóme [5] suggested a systematic approach to modeling and integrating timing scenarios with timed automata. However, their specifications may not properly preserve scalability and traceability of use cases.

In this paper, we extend Time ER nets formalism [6] to Modular TER nets (MTER), for easily describing timing scenarios and for systematically analyzing timing conflicts in requirements while preserving the main advantages of use cases. A MTER model is composed of a set of Timing Constraint Nets (T-CN), each of which describes individual timing scenario. Dependencies among timing scenarios are described by sharing labels of places and transitions of T-CNs. This modular structure makes MTER models scalable and traceable by reflecting properties of timing scenarios. For easily describing timing scenarios and formalizing them by MTER, we propose a systematic procedure to filling event-condition tables and to transforming event-condition tables to T-CN models.

The rest of the paper is organized as follows. In Section 2, we discuss on the characteristics of timing scenarios in the real-time systems. In Section 3, we extend Time ER nets to Modular TER nets, which is suitable for independently describing timing constraints of timing scenarios. Section 4 describes the systematic procedure for formalizing and integrating timing scenarios into a MTER model. Section 5 provides an analysis procedure for checking timing conflicts in MTER models. Finally, in Section 6, we conclude the paper.

2 Characteristics of Timing Scenarios

The scenario concept is suitable for describing the external behaviors of real-time systems such as reactive systems. Generally, a scenario which is composed of a sequence of operations describes a partial behavior of the interactions between a system and its environment. For specifying the timing behaviors of real-time systems, the scenario should have mechanisms for handling the timing constraints. For example, Fig. 1 presents a user- described requirement in natural language for the gate movement of the Generalized Railroad Crossing (GRC) system [7].

> The gate moves 'up' and 'down' states repeatedly. When a train approaches to the railroad crossing, the gate must start to move down within 1 second. The gate must be 'down' state between 6 and 7 seconds after starting to move down. When the train leaves the railroad crossing, the gate starts to move to 'up' state. After starting to move up, the gate must reach 'up' state within 7 seconds but at least 6 seconds are needed to move the gate up.

Fig.1. A gate requirement of the GRC system

From the scenario perspective, the requirement description shown in Fig. 1 is a sequence of events in a restricted situation such as resources and previous events, and so on with satisfying the timing constraints, for example, that "The gate must be 'down' state between 6 and 7 seconds after starting to move down". We can represent the requirements in Fig. 1 in the form of scenarios shown in Table 1.

Table 1. A scenario of gate movement

Event	Pre-condition	Timing constraints
move_down	<train, Approach>	@*move_down* ≤ @<train, Approach>+1
down		@*move_down*+6 ≤ @*down* < @*move_down* + 7
move_up	<train, Leave>	
up		@*up* ≤ @*move_up* + 6, @*up* ≤ @<train, Leave>+7

Conditions for a system or environments can be represented by a pair of a state variable and its value. Denoting the state of a train as a state variable 'train', we can rewrite the pre-condition of the 'move down' event as '<train, Approach>'. Conditions or events may include time-stamps as in @down and @<train, Approach> in timing constraints of Table 1, where '@' denotes the time of event occurrence. The occurrence time of an event is specified with respect to the time of either a previously occurred event or a previously satisfied condition, which we call 'event referencing' and 'condition referencing', respectively.

For example, in Scenario 1 of Fig. 2, both of the time constraints TC1 and TC2 reference the same events (Multiple referencing). In Scenario 2, the timing constraint TC3 uses event referencing, but TC4 uses condition referencing. One event can by constrained by several events or conditions as shown in Scenario 2 (Multiple constraining).

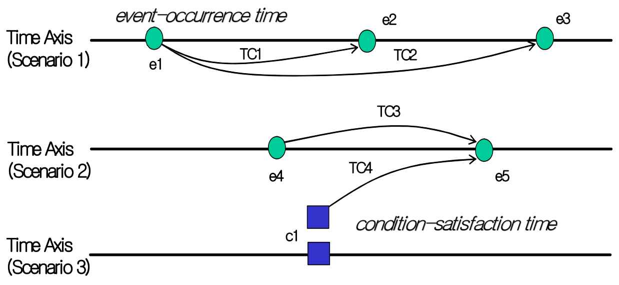


Fig.2. Usage patterns of timing constraints

Timing constraints are closely related to each other since event referencing or condition referencing can occur in several timing scenarios. Since the time behaviors of real-time systems are also closely related with functional behaviors, timing constraints cannot be handled independently. In order to analyze the time behaviors of real-time systems, a formal model which can be obtained by synthesizing timing scenarios is required. The formal method should also take into account the characteristics of timing constraints as well as the characteristics of scenarios such as partiality, traceability, and scalability.

3 Modeling the Scenarios by MTER

In requirements analysis phase, timing constraints are generally given in the form of absolute time manner, which means that every scenario use only one time axis to describe the timely behaviors of a system. On the other hand, when modeling the timely behaviors of scenarios in a local viewpoint, we can use a relative time axis, in which a scenario has an independent time axis against other scenarios. As a result, there exist several relative time axes in the scenario model. Therefore we need the modeling tool that can support the both timing viewpoints to analyze the timely behavior of the whole system.

The UML [11] state diagram is the best known notation to describe the behaviors of each system component, but there exist some difficulties in representing the relationships among the scenario models. And the time concepts, which it can support, are only the relative time with respect to some state in a state diagram. So, it is difficult to model the time constraints related with events in several components. And state diagrams have a weakness in handling concurrency.

To overcome these problems, we chose Petri Nets as a base formalism since some Petri nets such as Time ER nets [6] and TCPN [8] can handle both timing viewpoint and have several analysis techniques for timely behaviors. And Petri nets have many advantages of representing the concurrency among models naturally and representing multiple referencing of conditions and events. However, some Petri nets such as Time Petri nets [9], and timed Petri nets [10] do not support absolute timing viewpoint since they associate time with only transitions.
Among them we select Time ER nets since they are general enough to support the two broad families of time extended Petri nets as well as they have formal semantics which the existing time Petri nets usually lack. Time ER nets are descriptive enough to express the various timing constraints that appeared in scenarios and behavior models. To improve understandability and modeling simplicity we introduce some graphical notation, where a time pair is associated with input arcs of a transition. In this case, the arc can control the time slot through which the token in input place could pass.

In Time ER nets, each token (called environments) contains a variable, called *chronos*, whose value is of numerical type, representing the timestamp of the token and action associated with transition controls the timely behavior of token. To independently represent the behaviors of scenarios in a system, the methodology should support the modular concept. Since existing time extended Petri nets do not support the modular concept, it is impossible to represent the behaviors of a target system in the modular manner. Therefore, the Time ER net is extended with the modular concept as follows.

Definition 1 For a character set Σ, a Timing Constraint net(T-CN) is a 6-tuple TC net = (P, T, E, A, L, M_0), where

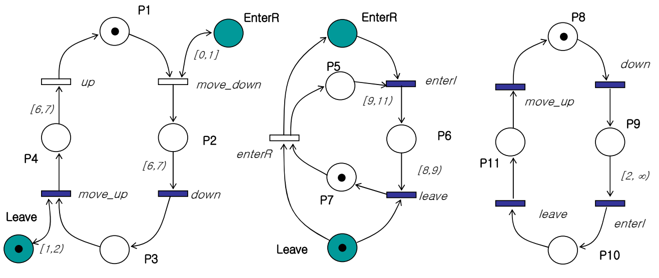
- P, T, E, A and M_0 are the same as those of a Time ER net, each of which stand for the sets of places, transitions, environments, actions and an initial marking respectively,
- L : T → Σ+ is a label function that associates a distinct label taken from strings (Σ+) with each transition of T.

Definition 2 Modular Time ER nets (MTER nets) are defined as MTER nets = { T-CN_i | i = 1…n} satisfying the following conditions:

- Ti should be disjoint for T-CN_i,
- M_0 for shared places should be the same.

In MTER nets, shaded transitions mean shared transitions. Fig. 3 shows an example of MTER nets, which describes three timing scenarios of the railroad crossing example. Fig. 3(a) represents the gate movement shown in Fig. 1 and an additional timing constraint for 'move up'. Fig. 3(b) illustrates the timing scenario for the train movement. And Fig. 3 (c) represents the safety constraint for the gate movement; the gate should be in the down state while a train is passing. In Fig. 3, shared transitions such as *down* and *move_ up* represent the events common to T-CN_1 and T-CN_3 and shared places such as **EnterR** and **Leave** represent the conditions common to T-CN_1 and T-CN2. Shared transitions and places are shaded to differentiate them from local places and transitions.

A local transition enabled in T-CN_i is also enabled in MTER since it does not appear in other timing constraint nets. On the other hand, a shared transition can be enabled in MTER only when it is simultaneously enabled in all the T-CN_i in which it appears. A globally enabled transition can fire in a common firable interval of all appearances in T-CN_i.



(a) T-CN_1 for gate movement (b) T-CN_2 for train movement (c) T-CN_3 for safety

Fig.3. An example of MTER nets

The markings of MTER are composed of a set of pairs *(Place, TimeStamp)*. In Fig. 3, the initial marking of MTER is $\mathbf{M_0}$ = { (P_1, 0), (P_7, 0), (Leave, 0), (P_8, 0) }, assuming all the initial tokens are generated at time 0. In M_0, if a local transition *enterR* is fired at time 0, the marking of MTER is changed into $\mathbf{M_1}$ = { (P_1, 0), (P_5, 0), (EnterR, 0), (P_8, 0) }. Then M_2, M_3, and M_4 are obtained by firing *move_down*, *down*, and *enterI* at time 1, 7, and 10, respectively (see follows). Fig. 4 shows how to get the Firable Interval(FI) of *enterI*.

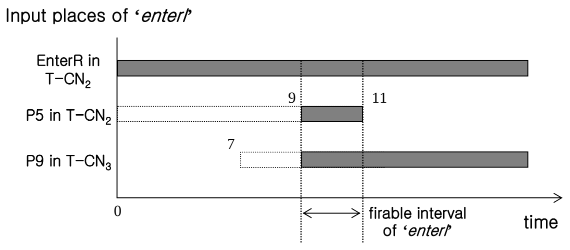


Fig.4. A firable interval of *enterI* transition.

$\mathbf{M_0}$ =[*enterR*,0]=> $\mathbf{M_1}$, *FI*(*enterR*) = [0,∞)
$\mathbf{M_1}$ =[*move_down*,1]=> $\mathbf{M_2}$ ={ (P_2, 1), (P_5, 0), (EnterR, 0), (P_8, 0) }, *FI*(*move_down*) = [0, 1]
$\mathbf{M_2}$ =[*down*, 7]=> $\mathbf{M_3}$ = { (P_3, 7), (P_5, 0), (EnterR, 0), (P_9, 7) }, *FI*(*down*) = [7,8)
$\mathbf{M_3}$ =[*enterI*, 10]=> $\mathbf{M_4}$ = { (P_3, 7), (P_6, 10), (P_{10}, 10) }, *FI*(*enterI*) = [9,11)

4 Synthesis of Timing Scenarios

A system is usually composed of a large number of timing scenarios. These timing scenarios are often described and managed by using UML [11] use case diagrams. In this section, we present a systematic procedure for converting such timing scenarios into a MTER model, which consists of three major steps. Firstly, through interactions between users and domain experts, each timing scenario is clarified by filling out an Event-Condition (EC) table. Secondly, the event-condition tables are systematically transformed into a T-CNs without users' interventions. In the final step, T-CNs are combined into MTER models.

The GRC system has five functionalities: timing scenarios for gate movement, train movement, safety constraint, gate utility1 and gate utility2. The safety constraint describes the behavior of train passing when the gate is down. Gate utility1 describes a timing constraint that the gate should move down as late as possible for increasing gate availability. Gate utility2 describes a timing constraint that the gate should move up as soon as possible for increasing gate availability.

4.1 Filling out event-condition tables

Action names initially given in the scenario description may need to be clarified by associating them with events. Generally, it is inappropriate to describe timing constraints by referencing only the occurrence times of actions, because they may have performance duration. In addition, users may use different terms for the same action and the same name to refer to distinct actions. Therefore, it is necessary to clarify actions by transforming action names to event names.

Table 2. The EC table of gate movement

Events	Pre	Post	Timing Constraints
move_down	<train, EnterR>		*@move_down* ≤ @<train,EnterR> +1
down			*@move_down*+6 ≤ *@down* < *@move_down*+7
move_up	<train, Leave>		@<train, Leave>+1 ≤ *@move_up* < @<train, Leave>+2
up			*@move_up*+6 ≤ *@up* < *@move_up*+7

Table 3. A table for state variables

State Variable	Descriptions	Domain Values
train	The status of a train	Approach, EnterR, EnterI, Leave
...	...	...

Table 4. The EC table of train movement

Events	Pre	Post	Timing Constraints
enterR		<train, EnterR> **out**<train, Leave>	
enterI		**out**<train, EnterR>	*@enterR*+9 ≤ *@enterI* < *@enterR*+11
leave		<train, Leave>	*@enterI*+8 ≤ *@leave* < *@enterI*+9

Table 5. The EC table of safety constraint

Events	Pre	Post	Timing Constraints
down			
enterI			*@down+2 ≤ @enterI*
leave			
move_up			

Table 2 shows an event-condition table designed to help modelers clearly represent timing scenarios. The table is composed of event names, pre-conditions, post-conditions, and timing constraints. Before filling out the table, relevant state variables and their domain values should be identified since each pre- and post-conditions of the table is expressed by a pair of a state variable and its value. In the gate movement example (see Table 2), we identified a state variable 'train' whose values are 'Approach', 'EnterR', 'EnterI', and 'Leave' (see Table 3).

As a result of event occurrence, some conditions may be satisfied (called generated conditions) and/or dissatisfied (called dissatisfied conditions). When writing down the post-conditions column, we differentiate the dis-satisfied conditions from the generated conditions by inserting prefix ‘**out**’ (see Table 4). A timing constraint is described in terms of inequalities including event occurrence time (denoted as @event) and condition satisfaction time (denoted as @condition). Tables 4 and 5 show the completed event-condition tables for train movement and safety constraint, respectively.

4.2 Convert event-condition tables to T-CNs

Event-condition tables can be systematically converted into timing constraint nets. Since each timing scenario is considered to be a concurrent unit of system’s functionalities, the corresponding event condition table is translated into a single T-CN model, in which a sequence of events is transformed into a loop of transitions with intermediate places (called control places) and other components such as pre-/post-conditions and timing constraints are transformed into places (called condition places or time places, respectively) and connecting arcs.
In the case of a pre-condition, the condition place is connected to the referencing transition by bi-directional arcs, which indicates that the event references only the condition without changing its status (see the arc between **EnterR** and *move_down* in Fig. 3(a)). In the case of a generated post-condition, it has an incoming arc from the transition (see the arc from *enterR* to **EnterR** in Fig. 3(b)). And in the case of a dissatisfied post-condition, it has an outgoing arc to the transition (see the arc from **EnterR** to *enterI* in Fig. 3(b)).

In the case of event time referencing in a timing constraint such as $@enterR+9 \le @enterI < @enterR+11$ as shown in Table 4, the referenced event time (*@enterR*) is transformed into a time place, with an incoming arc from *enterR* transition and an outgoing arc into *enterI* transition. A pair of min and max values such as [9,11) is associated with the outgoing arc. If the referenced event is an immediately preceding event, the time place can be replaced by the control place (see $\mathbf{P_5}$, *enterR*, and *enterI* in Fig. 3(b)). In the case of condition time referencing in a timing constraint such as *@move_down ≤ @<train, EnterR> +1* as shown in Table 2, a bi-directional arc between the *move_down* transition and the condition place (**EnterR**) is added. And a pair of min and max values such as [0,1] is placed on the arc (see the arc between **EnterR** and *move_down* in Fig. 3(a)).

4.3 Combining T-CNs into MTER

The procedure for integrating timing scenarios begins with locating shared transitions and shared places. The events appearing in multiple timing scenarios are declared to be shared transitions. Pre and post-conditions as well as the time places appearing in several T-CNs are marked as shared places. The relationships between scenarios such as UML ‘include’ and ‘extend’ are described by shared places and transitions. The ‘include’ relationship can be described by using shared **S** and **E** condition places. The ‘extend’ relationship, which means a branch structure, can be described by a shared transition.

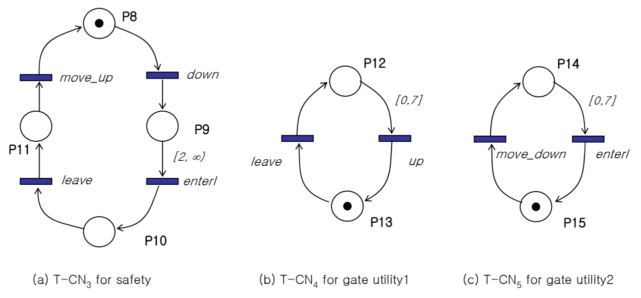


Fig.5. A MTER model of the GRC example

Shared conditions such as <train, EnterR> and <train, Leave> are represented by shared places **EnterR** and **Leave**, respectively as shown in Fig. 3. Fig. 5 shows the MTER model of the railroad crossing example which is composed of three remaining timing constraint nets such as safety, gate utility 1, and gate utility 2.

5 Analysis of Conflicting TCs

A MTER model is composed of a set of timing constraint nets, T-CNs. In order to detect timing anomalies among timing constraints, each T-CN$_i$ is locally analyzed and reduced in order to minimize the size of models. Then T-CN models are incrementally and iteratively composed, verified, and reduced for checking non-firable transitions. Fig. 6 summarizes a procedure for checking timing anomaly in MTER models.

In MTER net models, composition is performed by unifying shared transitions while preserving incoming/outgoing arc information of each places and timing constraints, as shown in Fig. 7(a). When there are several pairs of corresponding shared transitions, they are unified independently as shown in Fig. 7(b). A unified transition is transformed to a local transition. In parallel composition of MTER nets, the number of places is preserved.

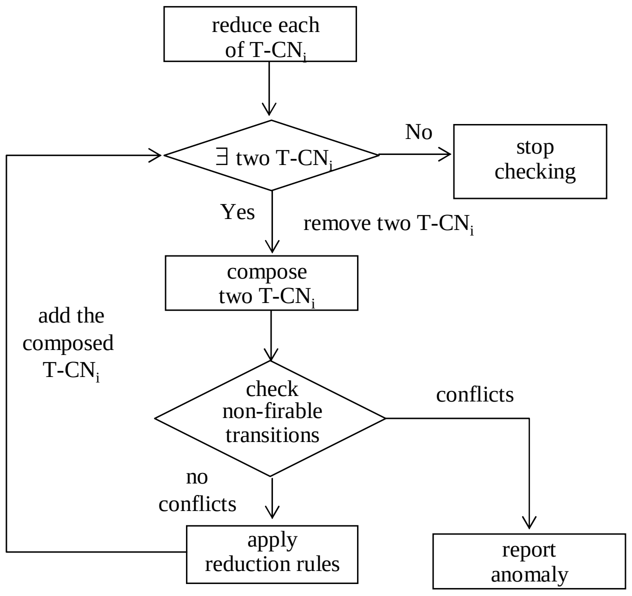


Fig.6. A procedure for checking timing anomaly

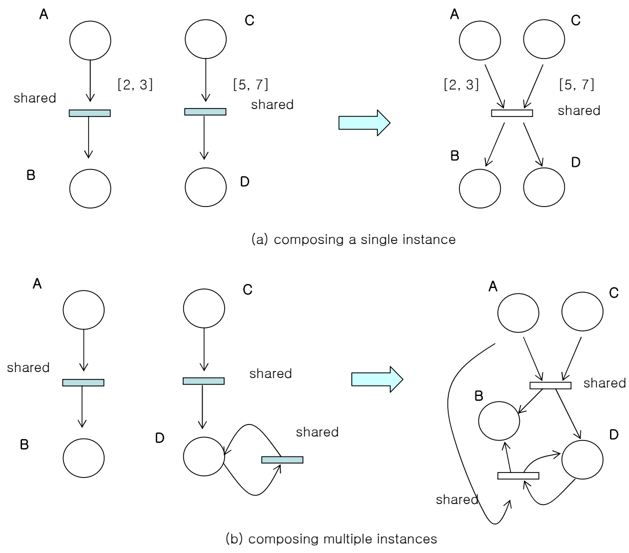

A
C
shared
[2, 3]
[5, 7]
shared
B
D
(a) composing a single instance
shared
(b) composing multiple instances

Fig.7. Composing rules of shared transitions

6 Conclusion

In earlier development phases of real-time systems, for easily obtaining a formal model and checking timing inconsistency among timing requirements, we proposed a systematic procedure of transforming and integrating timing scenarios into a Petri net-based model and provided a static timing analysis technique. By using our approach, users can easily obtain an initial requirement model of a system from timing scenarios and check the timing conflicts between timing constraints before proceeding to design and implementation phases.

As future works, software tools to support timing scenarios-based modeling and analysis on MTER models are needed because the productivity gains one can expect when applying our approach manually are limited.

References:

[1] C. Damas, B. Lambeau, P. Dupont, and A. Lamsweerde, Generating Annotated Behavior Models from End-User Scenarios, *IEEE Trans. on Soft. Eng.*, Vol. 31, No. 12, Dec. 2005.

[2] S. Uchitel, J. Kramer, and J. Magee, Synthesis of Behavioral Models from Scenarios, *IEEE Trans. on Soft. Eng.*, Vol.29, No. 2, Feb. 2003.

[3] M. Glinz, An integrated formal model of scenarios based on Statecharts, *European Software Engineering Conf. '95*, Sept. 1995, pp. 254-271

[4] F. Lustman, A formal approach to scenario integration, *Annals Software Engineering*, Vol. 3, 1997, pp. 255-271

[5] S. Sóme, R. Dssouli, and J. Vaucher, Toward an automation of requirement engineering using scenario, *Journal of Computing and Information*, Vol. 2, No. 1, 1996, pp. 1110-1132

[6] C. Ghezzi, D. Mandriodi, S. Morsca, M. Pezze, A unified high-Level Petri net formalism for time-critical systems, *IEEE Trans. on Soft. Eng.*, Vol. 17, No. 2, 1991, pp. 160-172

[7] T. Bolognesi, Constraint-Oriented Specification Style for Time-Dependent Behaviors, *Formal methods in real-time computing*, John Wiley, 1995

[8] J. J. P. Tsai, S. J. Yang, and Y.H. Chang, Timing Constraint Petri Nets and Their Application to Schedulability Analysis of Real-Time System Specifications, *IEEE Trans. on Soft. Eng.*, Vol. 21, No. 1, Jan. 1995

[9] P.M. Merlin, D.J. Farber, Recoverability of communication protocols implications of a theoretical study, *IEEE Trans. on Commun.*, 1976, pp. 1036-1043

[10] C. Ramchandani, Analysis of asynchronous concurrent systems by Petri nets, *MIT TR-120(Project MAC)*, MIT, 1974

[11] OMG, *Unified modeling language (UML) 2.0 Specification*, URL:http://www.uml.org, 2004