Stepwise Approach to Design of Real-Time Systems based UML/OCL with Formal Verification

Takeshi NAGAOKA†, Eigo NAGAI†, Kozo OKANO†, and Shinji KUSUMOTO†

†Graduate School of Information Science and Technology, Osaka University
Yamadagaoka 1–5, Suita City, Osaka, 565–0871
{t-nagaok, e-nagai, okano, kusumoto}@ist.osaka-u.ac.jp

Abstract - This paper provides a stepwise method for the design of real-time systems with timeliness QoS guarantees. In the proposed method, the system components are designed using UML diagrams and are provided with the timeliness QoS annotated with OCL. The basis of this technique is to formally ensure that the required timeliness QoS is satisfied under the provided timeliness QoS, given the network property and the UML diagrams. In order to avoid the state-explosion problem during performance model checking, which can logically check the satisfiability, the problem is separated into two steps. The first step checks the satisfiability using an abstract model of each of the components derived automatically from the provided QoS. The second step independently performs model checking for each of the components using a more detailed version of the behavioral model of a given component. Such an approach reduces the number of total states to check. Furthermore, the approach can be extended into hierarchical design, which leads to good scalability. Experimental results are also included in this paper.

Keywords: timeliness QoS, UML/OCL, model checking, component based systems, hierarchical design

1 Introduction

This paper presents a new method to verify consistency of timeliness QoS of component-based designed real-time systems. We assume that timeliness QoS is not only given to a whole system (Required QoS) but also associated with each component of a given system (Provided QoS).

Timeliness QoS is a time aspect of QoS (Quality of Service) features[1]. In the paper, we treat jitter, latency and throughput as timeliness QoS.

Nowadays, most real-time systems are designed with help of UML diagrams[7]. Especially components and their relation through signal communication can be represented in a class diagram of UML. In UML based design, such timeliness QoS can be annotated in OCL[8]. The annotation is associated to each of components as a provided QoS and also to a network link as a network property. Recently, SysML (System Model Language)[13] also attracts interest. SysML extends from UML and presents mixed systems consisting of physical devices and software and network systems. Therefore, SysML supports diagrams for signal flows and physical flows. For behavioral diagrams, SysML supports four diagrams as same as UML. Among them, state diagram has powerful representation including parallel and hierarchical states.

The proposed method is revised version of paper [15]. The method in [15] uses Linear Programming (LP) for some of verification. The approach has a disadvantage that connection among components has to be acyclic, and it cannot be applied to hierarchical design. The method of this paper uses abstract QoS automata instead of using LP; thus it improves the former disadvantage.

The heart of the technique is formally to ensure that the required timeliness QoS is satisfied under the provided timeliness QoS, given network property and the class diagram.

In order to check the satisfiability, there are several approaches. Formal approaches are very useful. Model checking is one of such an approaches. Notion of Test Automata[3], [5] and its application is also useful. However, one of disadvantage of the method is the state-explosion problem.

In order to avoid state-explosion problem while performing model checking, we separate the problem into two steps. The first step checks the satisfiability using abstract model of each of components derived automatically from the provided QoS. The second step performs model checking each of components independently using more detailed version of behavioral model of a component. Such an approach efficiently reduces the number of total states to check. Moreover the approach can be extended into hierarchical design; therefore it has good scalability.

SaveCCM[14] is a technique for Component based Development (CBD). In a description of a component, it allows user to define ports where signals input or output and to represent behavior in a timed automaton[2]. An IDE over Eclipse is available. Therefore, our proposed method has an affinity for SaveCCM.

The paper organized as follow. Section 2 gives timeliness QoS. Section 3 shows how to design component based real-timed systems in UML/OCL. Section 4 demonstrates the proposed method which consists of two steps. Section 5 provides an experimental example. We conclude the paper in Section 6.

2 Timeliness QoS

Main building blocks of our model are components. Each component has one or more *interfaces* to the environment, where all interactions between components is conducted via the interfaces. Since, we are mostly dealing with real-time systems and timeliness QoS, we shall assume that the interaction of a component with its environment is carried out via

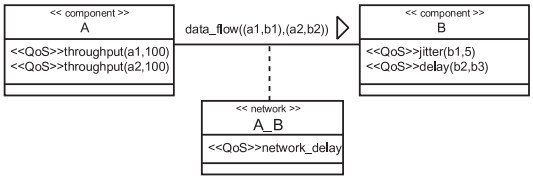

Figure 1: A Configuration of Components in UML Class Diagram

input and *output* signals. As a result, interfaces of a component specify signals that the component receives or emits.

Each component is associated with a number of input and output signals. In this paper, signals are denoted by x, y and z. Time of occurrence of a signal is denoted via a non-negative sequence of rational numbers. For example, the time of occurrence of a signal x is denoted with $x_1, x_2, \ldots$ representing time of first, second, ... occurrence of x.

Timeliness QoS expressions [9] such as jitter, throughput and latency can be expressed via first-order logic formulas on the set of time of occurrence of signals.

Throughput of at least (most) K within the time period T, for signal x can be written as the first order formula $\forall i \in \mathbb{N} : \quad x_{i+K-1} - x_i \leq T$ ($\forall i \in \mathbb{N} : \quad x_{i+K-1} - x_i \geq T$), respectively.

Notice, paper [5] refers to the above QoS constraint as Non-Anchored throughput.

Jitter , also called Non-Anchored jitter , of a signal x can be defined by the expression $\forall i \in \mathbb{N} : \quad T - m \leq x_{i+1} - x_i \leq T + M, \quad$ where T is the period of the jitter and m, M are constant rational numbers.

Latency of at most T unit of time between two signals x and y as $\forall i \in \mathbb{N} : \quad 0 < x_i - y_{Ki+K'} \leq T$. A special case of the above definition (for $K = 1$ and $K' = 0$) is the well-known definition of latency $\forall i \in \mathbb{N} : \quad 0 < x_i - y_i \leq T$ that applies to the time difference of the i-th occurrence of x and y.

3 UML/OCL based design of real-time systems

A real-time system can be designed as a set of components where signal communication links exist among pairs of components. We can describe such components in a UML class diagram in Fig. 1.

Type Component can be specified by Stereotyping "component," by which user can easily extend UML specification. A signal communication can be specified with Association.

Each of components has provided QoS, which can be represented via OCL annotation. Each of network links (which has association class with stereotype "network") also has network properties represented via OCL annotation. The network properties is the same as timeliness QoS. Attribute region of each class, includes special variables for QoS with "QoS" stereotype (in Fig. 1).

The following is the syntax of the variables.

Throughput Variable := "throughput(" signal "," period ")" ;
Jitter Variable := "jitter(" signal "," period ")" ;
Latency Variable := "delay(" output "," input ")" ;

A class with "component" stereotype has three categories of timeliness QoS (Jitter, throughput and latency), while a class with "network" has two categories of timeliness QoS (Jitter and latency).

The OCL description is given as follows[9].

QoS description := "context" className invariant* ;
invariant := "inv: self." constraint ;
constraint := variable op constant;
variable := Throughput Variable | Jitter Variable | Latency Variable
op := "$>$" | "$<$" | "$\geq$" | "$\leq$" ;

For example, the following are examples for Fig. 1, where – means a comment line.

context A
inv: self.throughput(a1,100) ≥ 20
– signal a1 is emitted at least 20 times in the period 100 units of time
inv: self.throughput(a2,100) ≤ 10
– signal a2 is emitted at most 10 times in the period 100 units of time

context B
inv: self.jitter(b1, 5) < 1
– signal b1 has jitter 1 with period 5 units of time
inv: self.delay(b2,b3) < 5
– latency between receiving signal b2 and sending signal b3 is less than 5 units of times

context A_B
inv: delay ≤ 100
– latency (network delay) between component A and component B is less than 100 units of time

4 The Verification Method

The verification consists of two steps; First Step and Second Step. If some of the components are not simple enough, then repeat the process again from First Step on each of the components. The following is the abstract level of steps of the proposed method.

input:

- system required timeliness QoS represented in OCL;
- component level provided timeliness QoS represented in OCL; and
- network configuration represented in UML/OCL class diagram.

output:

- component level behavioral specification represented in UML/OCL state-chart which satisfies required timeliness QoS under the configuration;

- or failure.

1. First step
   (a) We generate test automaton from the required timeliness QoS.
   (b) We generate abstract QoS automaton from each of the provided timeliness QoS.
   (c) We generate configuration automaton from the network configuration.
   (d) We check the consistency from parallel composition of the above automata.
   (e) If the result is deadlock then return failure. We have to reconfigure the requirement or provided conditions.
   (f) If the result is not deadlock then go to Second Step.
2. Second Step
   (a) If the component is not small enough to represent simple state-chart, then refine the component by;
      i. renaming provided QoS of the component to required QoS;
      ii. design sub components and provided QoS of each of them;
      iii. design network configuration; and
      iv. we repeat the First Step until the component is enough to small.
   (b) If the component is small enough to represent simple state-chart, then we describe state chart of the component.
   (c) We translate a test automaton from the provided timeliness QoS.
   (d) We design network of timed automata from the state-chart.
   (e) We check the consistency from parallel composition of the above automata.
   (f) If the result is deadlock then return failure. We have to reconfigure state-chart.
   (g) If the result is not deadlock then return success.

4.1 The First Step

Verification inputs are the following.

- Required QoS;
- a set of components with provided QoS; and
- a configuration automaton which represents network properties.

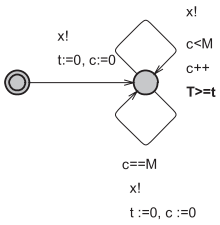

Figure 2: Abstract QoS automaton for Anchored Throughput

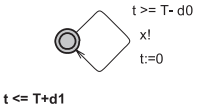

Figure 3: Abstract QoS automaton for Non-Anchored Jitter

The output is whether a given required QoS is satisfied under a given set of components with provided QoS and a given set of network links with network properties.

In usual methods, designer models behavior of each component in a network of timed automata and for the whole network of timed automata. Then the designer performs model checking, which often results in state explosion.

Here, we give a new method, in which timed automata (We call each of them an **abstract QoS automaton**) is derived automatically from the provided QoS. The important point is that derived automata are so small that state-explosion is avoided. Here, we give a translate rule for each provided QoS.

4.1.1 Throughput

A translated abstract QoS automaton for throughput is shown in Fig. 2. The automaton transmits signal x at least M times during the period T. The variable c and clock variable t are used for such the control. When "throughput must be at least k frames per P ms" is given as a provided QoS, the corresponding abstract QoS automaton is generated with substitution $M = k$ and $T = P$.

4.1.2 Jitter

A translated abstract QoS automaton for jitter is also shown in Fig. 3. The automaton transmits signal x with the priod T. The allowed jitter is $[T-d_0, T+d_1]$. Using the clock variable t, it transmits signal x at every $[T - d_0, T + d_1]$ period.

When "jitter must be $[-d'_0, d'_1]$ with a period T'" is given as a provided QoS, the corresponding abstract QoS automaton is generated from Fig. 3 with substitution $T = T'$, $d_0 = d'_0$ and $d_1 = d'_1$.

4.1.3 Latency

For signal x and y, let m and M be the minimum and maximum latency, respectively. A translated abstract QoS automaton for latency with above parameter is shown in Fig. 4. The

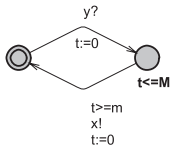

Figure 4: Abstract QoS automaton for Latency

automaton transmits signal x after receiving signal y with the latency $[m, M]$.

Unfortunately, the automaton does not accept input y until it emits output x. To avoid the problem, a set of the same automata is needed. The number of automaton decided from throughput property of the components.

When "latency for signal x and y must be $[m, M]$" is given as a provided QoS, and also parameter T representing period of signal y is given, the corresponding abstract QoS automata are generated from Fig. 4. The number of copies is T.

4.1.4 Configuration Automaton

A configuration automaton models interfaces among components. As each component has several inputs and outputs, such an I/O is represented as a channel in the configuration automaton. Each channel synchronizes with some I/O of some components with provided QoS (abstract QoS automaton). Abstract QoS automata and the configuration automaton communicate each other as described above.

4.1.5 Test Automaton

For a given required QoS, we can verify whether the required QoS is satisfied with the system by generating a corresponding test automaton from the required QoS. Fig. 5, 6, and 7 show templates of test automata for throughput, jitter, and latency, respectively. For such template, substituting each parameter with a concrete value specified by the required QoS, we can obtain a test automaton.

Throughput For a non-anchored throughput of which a signal e occurs at least k times in a period T and at most k times in a period $T0$, a network of test automata consisting of k processes of timed automata in Fig. 5 observes the throughput.

The test automaton observes if $T0 \leq T(e, i+k) - T(e, i) \leq T$ holds for some i, where $T(e, i)$ means the time of i-th occurrence of signal e. With k copies of such test automata, they can observe if $\forall i(T0 \leq T(e, i+k) - T(e, i) \leq T)$ holds. In other words, thay can observes at least k times signal e occures during $[T0, T]$. Parameters of the test automaton are k, T and $T0$.

In the network of test timed automata, the variables c is shared among automata globally. Each of timed automata is activated by turns along the value of variable K. When there exists common divisor k for $T, T0$ and n, we can reduce the number of copies of the test automaton to k/n with the parameters T/n, $T0/n$ and k/n. The discussion of such reduction technique is described in Sec. 5.4.

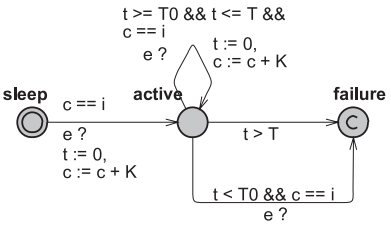

Figure 5: Test Automaton for throughput

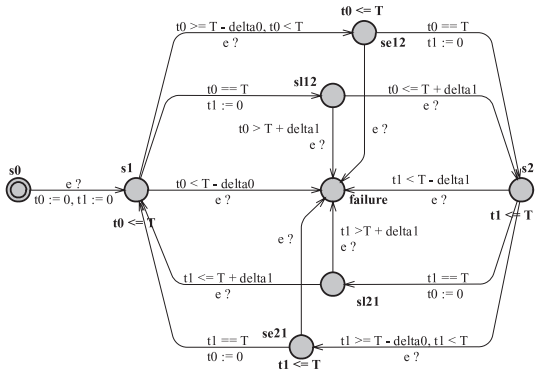

Figure 6: Test Automaton for jitter

Jitter Figure 6 shows a test automaton for anchored jitter. It observes whether a signal e occurs periodically in the period $[nT - delta_0, nT + delta_1]$, where $n = 1, 2, 3, \ldots$.

The automaton in Fig.6 has two clocks $t0$ and $t1$. A path form $s1$ to $s2$ via $se12$ or $sl12$ observes that the time of j-th occurrence of signal e is during the period $[jT - delta0, jT + delta1]$ using clock $t0$, while a path form $s2$ to $s1$ via $se21$ or $sl21$ observes that the time of $j+1$-th occurrence of signal e is during the period $[(j+1)T - delta0, (j+1)T + delta1]$ using clock $t1$.

Latency Figure 7 provides a component of test automata for latency between a signal x and y. The test automaton shown in Fig.7 observes if $\forall i(T(y, i) - T(x, i) \leq T)$ holds, where $T(e, i)$ means the time of i-th occurrence of signal e.

We have to use T/D copies of such test automata, where D is period of signal x. Variable cx and cy are shared variables with them, which serve to count the occurrence of signal x and y.

4.1.6 Verification

The behavior of such media with timeliness property also is modeled in a network of timed automata, we call such an automaton a configuration automaton. Parallel composition of

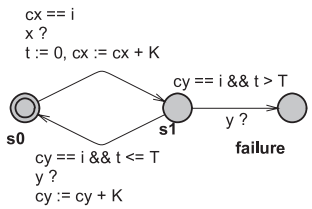

Figure 7: Test Automaton for latency

an abstract automaton for every component and the configuration automaton and test automaton for specified timeliness QoS decides whether the whole system satisfies the specified timeliness QoS. For the detail of process of the verification, refer [10].

4.1.7 Category Based Model Checking

Verification is performed for every timeliness QoS category (latency jitter and throughput). The idea and approach is very simple. When we want to check only latency as the required QoS, we build an abstract automaton for provided QoS of latency only. The divided and conquer approach, reduces the size of states.

4.2 The Second Step

For each of components, Second step has the following two cases depending on the component's abstraction

- We repeat First step to the given component recursively.
- We design detail behavior of the component and verify whether provided QoS is ensured by the design.

If the size of the given component is large and designer has to design the given component from more detail components, then repeats First step. Hereafter, we describe the later case.

At Second step (of the later case), verification is independently performed for each component. Before Second step, the designer has to give detailed behavior of each component. Such behavior is given in UML state-chart. In order to give time constraints on events, the state-chart has clocks.

Verification inputs are the following.

- component's behavior given in UML state-chart with clocks; and
- component's provided QoS.

The output is whether provided QoS is satisfied under a given UML state-chart with clocks.

The verification is performed based on test automaton. We have to translate a UML state-chart with clocks to a network of timed automata.

A state-chart can represent hierarchical architectures; while a network of timed automata is a simple flat structure model.

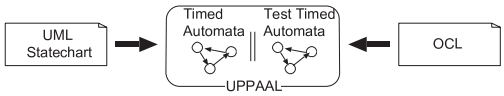

Figure 8: Verification on UPPAAL based on Test Automata

In general, hierarchical structure can be flatten, but such translation increases the number of states. There are several translations, and this paper adapts the one in [4]. The translation itself is an algorithm to translate Hierarchical Timed Automata (HTA) to a network of timed automata used by UPPAAL[11].

Thus, we have to translate state-charts to HTA. Fortunately, syntax and semantics of state-chart and HTA are both similar, the translation is simple.

We add the following constraints on the state-chart.

- the state-char diagram has clocks; and
- arcs in the stat-chart has clock constraints in a form of the one same as Timed automata in UPPAAL.

We also use test automata to check timeliness QoS. Test automata for jitter, latency and throughput are given in 4.1.5.

The verification can be performed with UPPAAL. Thanks to test automata, we just check deadlock property for each QoS. Logical expression for deadlock property in UPPAAL is "A[] not deadlock."

5 Experiment

The proposed method is applied to an example.

5.1 The example

Media Server is an application delivering video stream and audio stream to Digital Television and Audio System[6], [12]. Each of output devices required timeliness QoS (throughput). Figure 9 shows the class diagram of the application, which consists of twelve components.

In order to compare the proposed method to the old method, which uses LP solver to First Step, we merge the twelve components to three components (Server 3 components , Audio client 4 components, and Video clients 5 components).

5.2 First Step

The following is the provided QoS.

- Throughput of Component MS–Server is equal or greater than 100 frames/s.
- Processing latency of Component MS-Storage is equal or less than 5ms.
- Network latency between MS–Server and Digital–TV is equal or less than 100ms.
- Network latency between MS–Server and Audio–System is equal or less than 150ms.

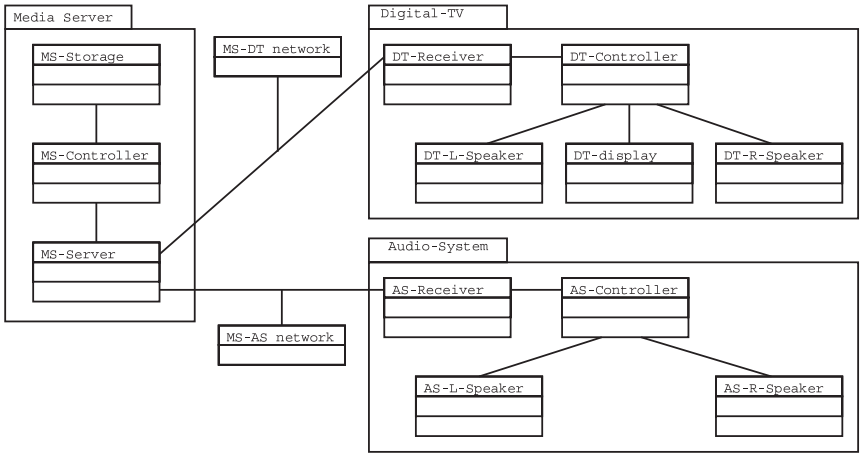

Figure 9: Class Diagram of Media Server

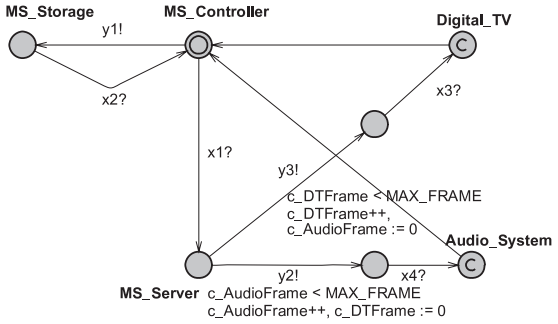

Figure 10: The configuration automaton

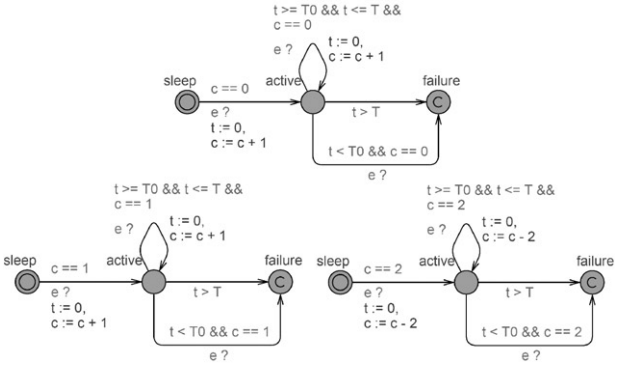

Figure 11: The network of test automata for the given required QoS

We give the following requirement for the Required QoS for the system.

- Throughput of Digital Display (DT Display) must be at least 30 frames/sec.

For these provided QoS, and a configuration automaton derived from the UML class diagram, and Required QoS for the whole system, we apply the verification along with First Step.

Figure 10 shows the Configuration automaton for the experiment. The Configuration automaton in Fig.10 represents connection among the components. It uses channels to communicate abstract QoS automata providing the provided QoS mentioned above. For example channel x is used for communication to an abstract QoS automaton with throughput 100 frames/sec at a transition between MS–Controller and MS–Server. In order to avoid unfairness that frame communication occurs only between MS–Server and Digital–TV (or only between MS–Server and Audio–System), we use a parameter MAX_FRAME, which is used in a condition that the maximum successive occurrence of signals between the same components. We use a condition $MAX\_FRAME = 1$ for the experiment.

Figure 11 shows the test automaton for throughput as Required QoS. The test automaton observes throughput of 3 frames per 100ms. As required QoS, the required value of throughput is 30 frame/sec, We have to need 30 processes of throughput test automata to observe the throughput exactly.

At the First Step, we have performed verification experiments for two configurations: (1) an abstract QoS which outputs ten frames per 100 msec, and (2) an abstract QoS which outputs 30 frames per 300 msec, respectively, are used for abstraction of provided QoS for MS.Server.

The provided QoS of MS.Server is at least 100 frames/sec. To prevent the jitter of frame signals, we adopt 100ms and 300ms as the period in the experiments. The values are set in the configuration (1) and (2).

Each of two experiments is performed with several numbers of test automata: 3, 6, 9, 12, 15 and 30. We have obtained CPU times and sizes of memory consumed. The experiments are performed in the following environment: CPU is Intel Core 2 Duo 2.33GHz, OS is Windows Vista Business and M.M. is 2GB. We used UPPAAL4.1.0 as a model

Table 1: The result (1) of first step

# of P	result	CPU time	Used memories
3	not valid	0.3 ms	23.9MB
6	not valid	1.4 ms	24.4MB
9	valid	28 ms	24.8MB
12	valid	50.6 ms	25.9MB
15	valid	83.6 ms	26.8MB
30	valid	480 ms	34.4MB

Table 2: The result (2) of first step

# of P	result	CPU time	Used memories
3	not valid	0.6 ms	23.9MB
6	not valid	0.7 ms	24.4MB
9	not valid	1.2 ms	24.7MB
12	not valid	1.6 ms	25.0MB
15	not valid	2.1 ms	25.1MB
30	valid	957 ms	40.4MB

checker. Table 1 and Table 2 show the results of (1) and (2), respectively. The column of "# of P" shows the number of processes (the number of test automata).

In the previous experiment, we have performed First Step with Linear Programming solver. In the experiment, we have performed it in 78 ms (although it has been performed in different environment).

5.3 The Second Step

After First Step, we design inner behavior of each component.

In the example, recursive application of First Step is not performed, because each component is small enough. Behavioral specification is described in UML state-chart. The design must meet the provided QoS. Figure 12 shows behavioral specification of Component MS-Storage.

In order to verify timeliness QoS for each component, Statechart must to be translated into a network of timed automata and also timeliness QoS is converted into test automata.

Figure 13 depicts translated result of *on* part in Fig.12.

The translating times are summarized as follows.

Translation time : 1153 ms
The number of states(before) : 89
The number of states(after) : 179

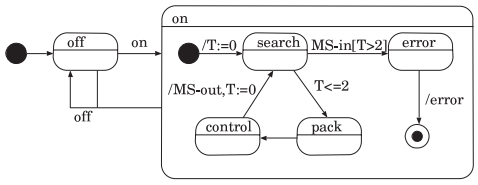

Figure 12: A UML Statechart Diagram of Component MS-Storage

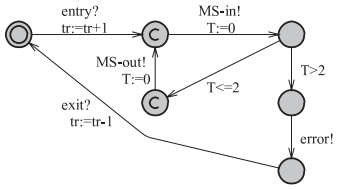

Figure 13: An UPPAAL Timed Automaton of Component MS-Storage

For every component and for every timeliness QoS, verification is performed. The total CPU time is about one seconds.

We found that for every component, the verification is performed within a few seconds with UPPAAL, without state explosion. Also we found that there is no deadlock.

5.4 Discussion

Table 1 shows that when we perform the experiment with nine test automata, it outputs the correct result. The result of Tab.2, however, shows that we cannot obtain the correct result until the number of test automata increases to 30. When we perform the experiment with 30 test automata, the CPU time increases exponentially. Therefore, we can conclude that there is a trade-off between degree of precision and CPU times. As shown in both tables, the CPU times of the experiments with 30 test automata are too large. Thus, as we consider the trade-off, in this experiment, the trade-off point is at which the number of process is 15.

Though we cannot exhibit that our proposed method is better than that of linear programming based method with respect to the CPU time, the performance of the proposed method is within useful reasonable time. The linear programming method has many constraints on configuration, while the new proposed method is flexible and is able to apply recursively along with component hierarchy, which are the advantage of the proposed method.

Our proposed method has more acceptable inputs than the former method. The difference between this and that of general class is very small. It is although not faster than the former method, it is more flexible than the former method.

6 Conclusions

This paper proposed a stepwise verification method for design of real-time system with UML/OCL focusing on timeliness QoS aspects. The method uses abstract QoS timed automata in order to reduce the possibility of state explosion.

The method can be applied to a design with complex connection of components.

Future works include simultaneous verification of several kinds of timeliness QoS, and utilization of feedback information such as verification counter-examples.

Acknowledgement

This work is being conducted as a part of Stage Project, the Development of Next Generation IT Infrastructure, supported

by Ministry of Education, Culture, Sports, Science and Technology.

REFERENCES

[1] R. Staehli, F. Eliassen, J. Aagedal and G. Blair "Quality of Service Semantics for Component-Based Systems," 2nd Int'l Workshop on Reflective and Adaptive Middleware Systems, pp.153-157, 2003

[2] R. Alur and D.L. Dill: "A Theory for Timed Automata," In Theoretical Computer Science 125 pp.183-235, 1994.

[3] L. Ageto, P. Bouyer, A. Burgueño and K. G. Larsen: "The Power of Reachability Testing for Timed Automata," LNCS, Vol.1530, pp.245-256, 1998.

[4] A. David and M. O. Möller: "From HUPPAAL to UPPAAL: Translation from Hierarchical Timed Automata to Flat Timed Automata," BRICS Technical Report Series, RS-01-11, 2001.

[5] H. Bowman, G. Faconti and M. Massink: "Specification and verification of media constraints using UPPAAL," In Proceedings of Design, Specification and Verification of Interactive Systems'98, pp.261-277 Springer, 1998.

[6] K. Havelund, A. Skou, K. G. Larsen and K. Lund: "Formal Modelling and Analysis of an Audio/Video Protocol: An Industrial Case Study Using UPPAAL," In Proceedings of the 18th IEEE Real-Time Systems Symposium, pp.2-13, 1997.

[7] Object Management Group: Unified Modeing Language Specification version 2.1, available at http://www.omg.org/.

[8] B. Bordbar, J. Derrick and A. G. Waters: "A UML approach to the design of open distributed systems," In Chris George and Huaikou Miao, editors, Formal Methods and Software Engineering, LNCS, Vol.2495, pp.561-572, 2002.

[9] UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms, Request for Proposal, available at `http://www.omg.org`

[10] B. Bordbar and K. Okano, "Verification of Timeliness QoS Properties in Multimedia Systems," 5th International Conference on Formal Engineering Methods (ICFEM '03), LNCS 2885, pp.523-540, 2003

[11] Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi: "UPPAAL . a Tool Suite for Automatic Verification of Real-Time Systems," LNCS, Vol.1066, pp.232-243, 1995.

[12] D. Akehurst, J. Derrick, and A. G. Waters: "Design and Verification of Distributed Multi-media Systems," LNCS, Vol.2884, pp.276-292, 2003.

[13] The official OMG SysML site, http://www.omgsysml.org/

[14] J. Carlson, J. Håkansson and P. Pettersson: "SaveCCM: An Analysable Component Model for Real-Time Systems," Proc. of FACS'05, Electronic Notes in Theoretical Computer Science, Vol.160, pp.127-140, 2005.

[15] E. Nagai, A. Makidera, K. Okano, K. Taniguchi: "A Method to Develop Distributed Real–Time Applications Based on UML/OCL (in Japanese)," In the IEICE Transactions, Vol.J89-D No.4, pp683-692, 2006.

(Received October 14, 2008)
(Revised July 15, 2009)

Takeshi Nagaoka received the M.I. degree in Computer Science from Osaka University in 2007. He currently belongs in a doctoral course. His research interests include abstraction techniques in model checking, especially timed automaton.

Eigo Nagai received the BE, and M.I. degree in Information and Computer Sciences from Osaka University, in 2005, and 2007, respectively. During his Master Course, his research interest was real-time application for model checking, especially timed automata. He is currently a researcher in NEC.

Kozo Okano received the BE, ME, and Ph.D degrees in Information and Computer Sciences from Osaka University, in 1990, 1992, and 1995, respectively. Since 2002 he has been an associate professor in the Graduate School of Information Science and Technology, Osaka University. In 2002, he was a visiting researcher of the Department of Computer Science, University of Kent at Canterbury. In 2003, he was a visiting lecturer at the School of Computer Science, University of Birmingham. His current research interests include formal methods for software and information system design. He is a member of IEEE, IEICE of Japan and IPS of Japan.

Shinji Kusumoto received the BE, ME, and DE degrees in information and computer sciences from Osaka University in 1988, 1990, and 1993, respectively. He is currently a professor in the Graduate School of Information Science and Technology at Osaka University. His research interests include software metrics and software quality assurance technique. He is a member of the IEEE, the IEEE Computer Society, IPSJ, IEICE, and JFPUG.