Translating Real-Time UML Timing Constraints into Real-Time Logic Formulas

Gowri Aruchamy and Albert Mo Kim Cheng

Real-Time Systems Laboratory
Department of Computer Science
University of Houston
Houston, TX, 77204, USA
http://www.cs.uh.edu

Technical Report UH-CS-06-07

June 5, 2006

Keywords: UML Modeling, Real-Time Systems, Timing Constraints, Verification, RTL

Abstract

Real-time software development is a complex process and understanding the detailed system design is important for the correct operation of a successful system. Modeling can help to understand the complex system design and Unified Modeling Language (UML) is gaining popularity in real-time systems development. Even though UML is expressive and easy to use, the semantics of UML is not well defined; it lacks the support for formal analysis which is needed to prove the timing correctness and safety properties of real-time systems. This paper focuses on translating a UML specification into Real-Time Logic, the real-time specification language which can be then analyzed and verified.

*This work is supported in part by the UH Real-Time Systems Laboratory and the Institute for Space Systems Operations.

Translating Real-Time UML Timing Constraints into Real-Time Logic Formulas

Gowri Aruchamy and Albert M. K. Cheng

Abstract

Real-time software development is a complex process and understanding the detailed system design is important for the correct operation of a successful system. Modeling can help to understand the complex system design and Unified Modeling Language (UML) is gaining popularity in real-time systems development. Even though UML is expressive and easy to use, the semantics of UML is not well defined; it lacks the support for formal analysis which is needed to prove the correctness, safety properties of real-time systems. This paper focuses on translating UML specification into Real-Time Logic, the real-time specification language which can be then analyzed and verified.

Index Terms

UML Modeling, Real-Time Systems, Timing Constraints, Verification, RTL

I. Introduction

Real-time systems are used to control and monitor many applications including avionics, automotive, patient monitoring and these applications tend to become very large and complex in nature. These system developers are challenged to face complex system architectures, rapid changes in requirements, shorter time to market and cost constraints, and evolving technological platforms. Traditional code driven development approaches are not suitable to handle these challenges. In recent years, Object Oriented Development (OOD) approach that uses Model Driven Architecture (MDA) to model complex system design is increasingly used in real-time systems development. The object oriented modeling language UML is used to model the system design. The standard UML model provides ways to represent functional, structural requirements. To represent timing requirements, UML profile which includes the extensibility mechanisms such as stereotypes, tagged values and constraints can be used. But the information presented in the UML diagrams is often not well-defined and it lacks formal semantics. UML uses easy to use, semi-formal notation and it does not provide support for real-time modeling and timing issues.

On the other hand, formal methods are effective to ensure the system's validity in real-time safety critical systems but they are very expensive and time consuming and they are not suitable to face the rapidly changing market conditions. So UML model can be used to design the system, and timing related specifications from the model can be translated into formal notations which can be analyzed and verified for the correctness of the resulting system. This paper presents an approach in which real-time system timing constraints are specified using OMEGA-RT profile [1] concepts and translating these constraints into Real time Logic (RTL) formulas [7]. The resulting RTL formulas can be verified for safety, correctness properties with the constraint graph technique approach using the tool SDRTL [8]. Several approaches exist for UML model checking and verification based on formal methods including timed state chart, timed automata, timed CTL, etc. But no work has been done previously for RTL based verification of UML models. Since this is an initial approach for translation of UML specifications to RTL, this paper focuses only the timing constraints specified in class diagrams and sequence diagrams (with the necessary assumptions) for translation.

The paper is organized as follows. Section II gives an overview of real-time support in UML. Section III presents some of the existing approaches for UML model checking and verification based on formal methods. Section IV gives brief introduction to RTL based verification. Section V gives the programming approach for the translation of UML to RTL and section VI presents the conclusion.

II. REAL-TIME UML

When modeling software systems in general, we are not considering the time taken (assume zero time units) for messages/ tasks execution. But in real-time system, certain task must be executed at a precise, absolute or relative time and within predictable, constrained duration. Thus in real-time system modeling, the real-time properties that have to be expressed are [1]:

- Time dependent behavior using timers or system clock
- Time related assumptions on the external environment of the system like response times to requests, execution times of actions
- Time related requirements such as deadlines of actions, duration between two events

To support real-time modeling, UML 1.3 included graphical representation for timing mark to denote event occurrence time, time expression that evaluates to an absolute or relative value of time, and timing constraint which is a semantic statement about the relative or absolute value of time [3]. Also UML 2.0 includes time related types and time related concepts such as timer, clock. It used state diagrams and sequence diagrams to express timing statements and a new diagram called Timing diagram also introduced for reasoning about time and visualization or state changes over time. But it lacks the syntax for timing constraints expressions.

The Object Constraint Language (OCL) is an important part of UML [10] which was developed to express restrictions over UML models. This declarative expression language allows modeling constraints in the context of a given UML model. It is used for specifying invariants attached to classes, pre- and post conditions of operation, and conditions on state transitions. But, OCL doesn't provide the support to specify temporal constraints over the dynamic behavior of objects. Several [11, 12, 15] OCL extensions have been proposed to address this limitation.

Since the UML standard lacks a quantifiable notion of time and resources which was an impediment to its broader use in real-time systems domain, to address this problem its built-in extensibility mechanism, UML profiles for real-time systems are introduced. The following subsections list some of the existing profiles for real-time modeling.

A. UML Profile for Schedulability, Performance and Time

UML profile for real-time modeling also called UML profile for Schedulability, performance and Time (SPT profile) [4] was requested and later adopted by OMG to support real-time modeling. This increased interest to use Object Oriented technology and UML in real-time system development. It includes several sub profiles to describe time, resource, and performance concepts of real-time systems. It uses stereotypes and tagged values to annotate the UML model. The *RTtimeModeling* sub-profile defines a model for representing time related mechanisms. It includes data types such as Time and Duration and features to express both local and global time constraints [1]. However SPT profile definition is incomplete and abstract and this technique is not sufficient to be used in real world projects since the timing constraints expressed with these definitions lack a well defined relationship to functional model which is needed for exchange of specifications between different tools. So to be used effectively, a concrete framework for these definitions is needed.

B. OMEGA-RT profile

This profile aims to provide a concrete UML profile with formal semantics and is a refinement of SPT profile. It introduces events based time modeling where an event is used to represent an instant of state change and allows the expression of duration constraints between occurrences of events.

OCL-like expressions are used to represent constraints. In SPT, time constraints are expressed at instance level but Omega enables time constraints to be expressed at class level. It extends the events use from timed scenarios to timed state machines in the form of observer. This profile is sufficiently general and expressive, and the real-time concepts defined in the other real-time profiles can be expressed in this profile. It includes the following features [1]:

- Operational concepts as in UML 2.0, the operator *now* for using system time, *timers* for *timeout* event after specified time and *clocks* to measure the time passed.
- A *timed event* concept which represents patterns of state changes occurring during execution and it can store event parameters and system state information at event occurrence as local attributes.
- A formal notation based on constraints, *Duration* to represent duration between event occurrences which is able to express all durations and constraint patterns
- Explicit syntactic distinction between two types of constraints, assumptions and requirements
- *Observer* is used to express constraints involving more than two events.

The timing profile is structured into two layers as primitive and derived. Primitive timing extensions use the basic notations mentioned above with the semantic definitions given in OCL-like constructs. Derived concepts correspond to subset of SPT terminology and are defined through a transformation into primitive concepts. This profile is developed for OMEGA project and several specializations of this profile have been considered with tool support for validation. The IF language and tool-set defined in [5] translates the timed UML model into timed automata in which UML level concepts are mapped into more primitive concepts. IF language format is used for the mapping and existing model-checking tools can be used for validation.

C. TURTLE profile

TURTLE (Timed UML and RT-LOTOS Environment) is a real-time UML 1.5 compliant profile with formal semantics given in terms of RT-LOTOS [6]. It extends UML with structuring capabilities, advanced logical and temporal operators. The translation of TURTLE models into RT-LOTOS is not straightforward and the operators described in TURTLE have no direct counterpart in RT-LOTOS. This profile extends class diagrams and activity diagrams of UML 1.5. TURTLE class diagram consists of *Tclasses* and they communicate using gates, a *Tclass* attribute of type *Gate*. Activity diagrams are extended with synchronization and temporal operators. For real time modeling TURTLE offers four temporal operators. Those are: deterministic delay, nondeterministic delay, time-limited offer, and time capture operator. Time intervals are expressed by combining the deterministic and nondeterministic delays. The ability to express temporal latency in implementation independent manner makes TURTLE profile more powerful than UML profile which is limited in expressing fixed duration.

III. PREVIOUS APPROACHES FOR UML VALIDATION

The approach proposed in [13] translates UML state models into timed automata and model checking is done by using the tool Kronos. For this approach the TCTL formulas to be verified has to be written by the user. The tool vUML given in [14] translates UML models into Promela language and SPIN model checker is used for verification of behavioral properties. Since this approach uses its own definition of UML state machine, it lacks the ability to verify linear temporal logic formulas.

In [16], formal specification from the UML models is obtained by deriving an axiomatic description of the system from formal design specifications. Tool support is provided to simulate the system and PVS theorem prover is used to verify the system properties. However to use this approach users need to have the ability to use PVS which is non trivial. The approach in [17] uses the Omega project approach for validating the timed UML model by transforming it into IF automata. The resulting timed automata can be verified with existing model checking tools. A formal specification and validation method called FORTS [18] translates the UML models into timed automata and verifies timing assertions presented by UML sequence diagram using model checking and constraints solving technique.

IV. Real-Time logic based verification

Designing and analyzing software systems for real-time applications are difficult due to the complexity of these applications. Important issues to be considered during this process are safety and reliability. Formal analysis and verification methods are used to guarantee the system is safe and reliable. One of the formal analysis techniques uses the relation between system timing specification (SP), and the safety assertion (SA) that describes the required safety properties to show the system is safe [7]. If SA is a theorem derivable from SP then the system is safe. The event-action model is used to specify SP and SA which can capture the data dependency and temporal ordering of computational actions that must be taken in response to events in a real-time application. Since event-action model is not suitable to be automated, it is translated into Real Time Logic (RTL) formulas. Then the constraints graph technique using the SDRTL tool [8] can be applied to verify the satisfaction of the safety assertion.

The approach proposed in this paper, focuses on translation of UML timing constraints into RTL formulas. Then using the SDRTL tool [8], these RTL formulas are verified. RTL is first-order logic with features to capture timing requirements of real-time systems. It is better than temporal logic since temporal logic can only express the relative ordering of events but RTL can express the relative ordering of events with absolute timing characteristics. RTL has three types of constants which are events, actions and integer. Four types of events in RTL are [9]:

1. start events representing beginning of actions, preceded by ↑
2. stop events representing ending of action, preceded by ↓
3. transition events representing the change in certain attributes of the system state
4. external events preceded by Ω

Occurrence function @ assigns time values to event occurrences and @ (e, i) represents the ith occurrence of event e. Since the analysis problem for full set of RTL is not decidable, a subclass of RTL formulas called pathRTL is used for the analysis. The restricted RTL formula contains arithmetic inequalities of the form: occurrence function ± integer constant ≤ occurrence function. More details about RTL formulas can be found in [7].

V. Translation Approach

Considering all the above mentioned profiles, Omega profile seemed a good choice for the UML modeling of the system timing properties for the following reasons: Omega profile has concrete semantic based syntax for modeling timing properties and allows expressing durations between event occurrences using the duration constraint. The UML concepts Objects, classes, operations, class diagrams, and sequence diagrams are considered in this approach by assuming these are enough to model timing properties of a system. Sequence diagrams are annotated with timing constraints on object level and class diagrams are annotated with timing constraints on class/type level. The Omega profile concepts used for this approach are discussed in more detail below.

Omega profile uses the two data types, *time* and *duration* where *time* represents points in time and *duration* represents distances between time points. *TimedEvent* represents instant of state change, defined as a type level concept. In instance level, it is an event instance, defined as an attribute of a class. Each event is classified syntactically into event kinds and each event kind is associated to a syntactic entity and it defines the relevant parameters of an event. These event kinds allow capturing of all meaningful state changes in the underlying semantic model. List of event kinds identified are given in [2], and the events associated with operation call are:

- *invoke* - the emission of the call request by the caller,
- *receive* - the reception of the call by the callee,
- *accept* - the start of the actual processing of the call by the callee,
- *invokereturn* - the emission of the “return” response by the callee,
- *receivereturn* - the reception of the “return” by the caller, and
- *acceptreturn* - the consumption of the return.

Since we are interested in operations between classes and objects in the UML diagrams with timing constraints, the events *invoke* and *receive* are considered under the assumption that *invoke* event occurs when an operation call invoked and *receive* event occurs when the operation call received.

A. UML model in Omega-RT profile

To illustrate the translation of UML model timing constraints into RTL formulas consider the example given in [1]. In a real-time system, an *Engine* displays information (temperature, rotating speed, etc.) received from sensors on some *Display* device. UML class diagram for this is given in Figure 1. The requirements contain the timing constraints as follows:

(1) Between two consecutive calls made by an Engine to the operation update of its Display, less than 100ms pass if the Engine rotation speed (attribute rpm) exceeds 7000 at the moment the first call to update is made.

(2) Between the moment the engine temperature becomes critical (reception of signal criticalTemperature by the Engine from the TemperatureSensor) and the moment the engine reacts by decreasing its speed (invocation of the Engine operation accelerate with a negative parameter by the Engine to itself) less than 50ms pass. Moreover, the Display shall receive an update from the Engine less than 20ms after the call to accelerate.

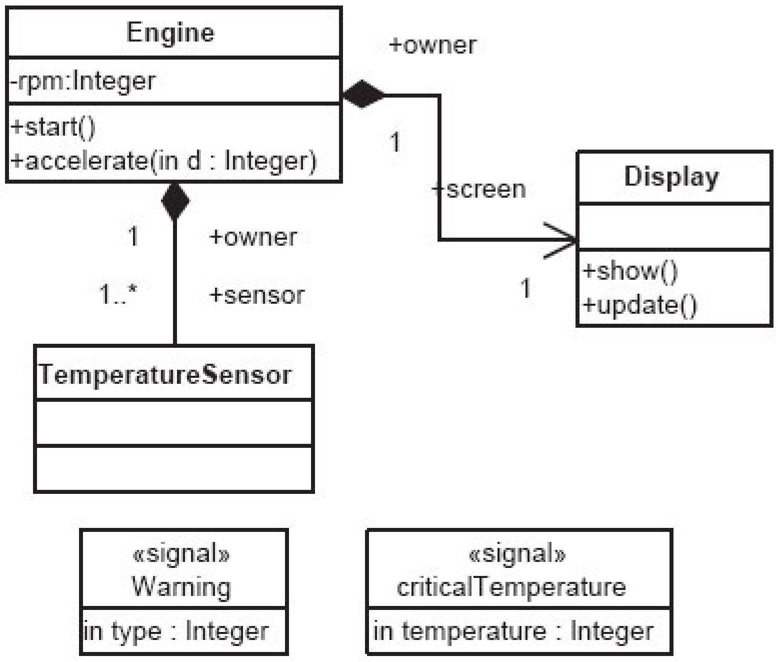
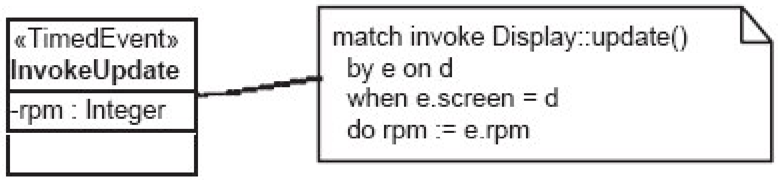


Figure 1. UML class diagram for the Engine Display example[1]

In property (1), the identified event is operation of Update from engine to display. The definition of timed event *InvokeUpdate* refers to this operation and is given in the following Figure 2.



```
match invoke Display::update()
  by e on d
  when e.screen = d
  do rpm := e.rpm
```

Figure 2. Definition of TimedEvent InvokeUpdate

In the above diagram the *match* clause expresses that an event of this type matches any occurrence of an invocation of Update of any object e of type Engine to an object d of type Display. The filter condition given with *when* restricts it to only the occurrences of call made by engine to display object defined by its screen attribute. The *do* statement saves the engine's rpm value at the point of time event occurred. The definitions of the events

[1] The figures used in this paper are from [1] Engine display example

extracted from the timing constraints from the property (2) are *RcvCritical, InvDecelerate* and *RcvUpdate* and they are given in Figure 3.

In Omega-RT profile time constraints in the requirements are defined as duration between event occurrences. The duration expressions for the above examples are given in the Figure 4. In property (1) because the constraint is local to the Engine class, the instances of events in its duration expression are defined as local attributes of Engine. With the OCL semantics, event instances are considered as objects that have a value in every semantic level state. Therefore in the syntax of events given in duration expressions, event E refers to an object that holds the most recent occurrence value of that event and E.pre represents an object holding at any time the previous value of E.

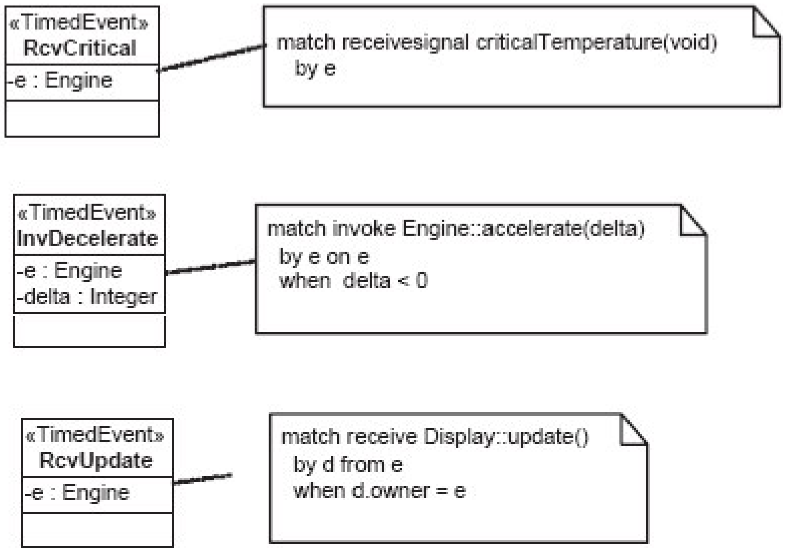


Figure 3. Event types for timing constraints in property (2)

The second property of the above example contains events of many objects and therefore it is expressed as global event with instances attached with the class <<*TimeAnnotations*>> and since more than one instance of Engine is involved in this, the match condition is given in [], to restrict the occurrences of events concerning the same engine *e*.

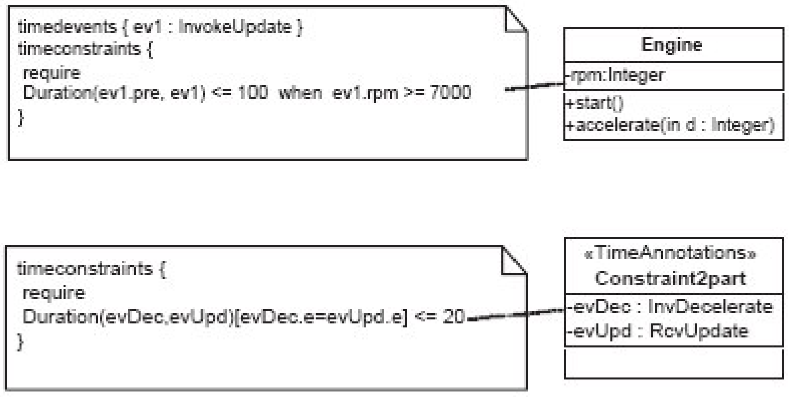

timedevents { ev1 : InvokeUpdate }
timeconstraints {
require
Duration(ev1.pre, ev1) <= 100 when ev1.rpm >= 7000
}
Engine
-rpm:Integer
+start()
+accelerate(in d : Integer)
timeconstraints {
require
Duration(evDec,evUpd)[evDec.e=evUpd.e] <= 20
}
«TimeAnnotations»
Constraint2part
-evDec : InvDecelerate
-evUpd : RcvUpdate

Figure 4. Duration expressions for the Engine Display example

B. Conversion Program

The timing constraints in the requirements are expressed in UML as duration expressions. A program is written to translate these constraints into RTL. The idea is to read the UML model time constraints, extract the duration expressions. Then duration expressions are parsed to identify the events and the time between those events in that constraint. Then the events are represented as occurrence functions of the form @ (event, occurrence) and converted as RTL formula of the form: occurrence function1 ± time ≤ occurrence function2.

The assumptions made for this program are: the timing constraints in UML model are in Omega-RT profile time constraints format. The Invoke event name starts with the prefix *Inv*, Receive event name starts with the prefix *Rcv* and External event name start with the prefix *Ext*. The UML timeconstraints annotations are extracted from the model and stored in the text file. This file is taken as command line argument when the program starts.

The program is written in java. The program algorithm works as follows. It reads the lines from the timing constraints file given in the command line argument and looks for the line that starts with the key word "Duration". If such a line found, then it extracts the events given in that duration expression and stores in an array named events. If the Duration expression line contains the keyword "when" then a conditional boolean expression that should hold for the attributes of the event occurrences is found. Then the time value that should be satisfied by the event occurrences is identified.

To be consistent with RTL notation, the event names are translated to RTL equivalent as given below.

- If the event name starts with *Inv* then it is identified as RTL start event and the prefix *Inv* is changed to "S_".
- If the event name starts with *Rcv* then it is identified as RTL stop event and the prefix *Rcv* is changed to "E_".

- If the event name starts with *Ext* then it is identified as RTL external event and the prefix *Ext* is changed to "Extern_".

The timing constraint related to the periodic event occurrences of the same event is identified if the first event in the duration expression contains the .pre suffix and is translated as follows. The two events are converted into occurrence functions such that the first event which denotes the previous occurrence value with the key word .pre in its name is written as @ (event_name, i) that represents the ith occurrence of the event. The second event changed into occurrence function as @ (event_name, i+1) that represents the i+1 occurrence of the event.

If the conditional expression found for this duration expression then it is assumed that a conditional event occurs at the time point when the condition's boolean expression evaluation becomes true. The conditional string is considered as an event and the conditional occurrence function is created as @ (conditional event name, i). In the engine display example given above, property (1) has a conditional expression *when ev1.rpm >= 7000*. In this case when the engine's rpm exceeds 7000 it is assumed an event named *RPM>=7000* occurs and the occurrence function corresponding to this event is written as @ (RPM>=7000, i).

Then the relationship between the conditional event occurrence and the timing constraint event occurrence is expressed in RTL. And also, the event occurrence functions with the time value are expressed in RTL. These RTL formulas are displayed as output with all quantified occurrence variable written at the beginning. The same way the timing constraints involving two different events are converted. This program is tested for the above engine display example and the resulting RTL formulas from this program output are:

For property (1)
```
Vi @(S_Upd, i) <= @(RPM>=7000, i) ^ @(S_Upd, i + 1) <= @(S_Upd, i) + 100
```

For property (2)
```
 Vi @(S_Dec, i) <= @(S_Upd, i) ^ @(E_Upd, i) <= @(S_Dec, i) + 20

Vi @(E_Cri, i) <= @(S_Dec, i) ^ @(S_Dec, i) <= @(E_Cri, i) + 50
```

The program is tested with test files containing timing constraints for the precedence relations, periodicity and priority assertions. The results show that this translation of UML to RTL is possible. Moreover, this program can be easily extended to handle the all possible timing constraints expressions modeled in UML which is not considered in this paper.

VI. Conclusions

Object-Oriented development techniques are widely used in the design and development of industrial real-time systems to manage their inherent complexity. These techniques use model driven approach in which UML is the standard modeling language. Even though, several approaches exist for validating UML model with timing properties, this approach of model checking UML model by translating into RTL formulas and using SDRTL tool is more beneficial for the following reasoning. The systematic debugging algorithm used in this tool is able to detect and fix the missing constraints. Also it can handle the incorrect constraints from the original specification. This approach uses incremental debugging, which avoids re-computing the satisfiability of the whole problem every time. Therefore model checking of UML model timing specifications using SDRTL will be a very useful contribution in Object-oriented development approach for real-time systems domain. In order to use the SDRTL tool for UML model checking, those modeling information has to be translated into RTL formulas. The conversion approach presented here shows that this translation can be effectively done.

This approach assumes the UML model timing constraints are present in a file. This assumption is made since the main focus is to translate the UML constraints given in duration expression into RTL. Therefore to avoid the complexity, UML model stored in XMI format is not considered. As future work, conversion program that reads the UML model in XMI format would be suitable for more sophisticated automated translation. Also a parser would be useful for handling the translation of all possible kinds of timing constraints expressions in UML models.

Acknowledgements

This work is supported in part by a 2006 grant from the Institute for Space Systems Operations.

References

[1] Susanne Graf, Ileana Ober, and Iulian Ober, “A real-time profile for UML”, http://www-omega.imag.fr/doc/d1000308_1/308-V1-main.pdf

[2] Susanne Graf, Ileana Ober and Iulian Ober, “Timed annotations with UML”, In Proceedings of SVERTS'2003

[3] Grady Booch, James Rumbaugh, and Ivar Jacobson, “The Unified Modeling Language User Guide”, Addison-Wesley Professional, 1st edition , September 30, 1998.

[4] OMG. Response to the OMG RFP for Schedulability, Performance and Time, v. 2.0. OMG document ad/2002-03-04, March 2002

[5] Iulian Ober, Susanne Graf, and Ileana Ober. “Validating timed UML models by simulation and verification”, http://www-omega.imag.fr/doc/d1000310_2/WP22-D223-310-V2-main.pdf

[6] Ludovic Apvrille, Jean-Pierre Courtiat, Christophe Lohr, and Pierre de Saqui-Sannes, “TURTLE: A Real-Time UML Profile Supported by a Formal Validation Toolkit” , IEEE Transactions on Software Engineering, July 2004 (vol. 30, No. 7), pp. 473-487.

[7] Farnam Jahanian, and Aloysiys Ka-lau Mok “Safety Analysis of Timing Properties in Real-Time Systems”, IEEE Trans. on Software Eng., 12(9), 1986, pp. 890-904,

[8] S. Andrei, W.-N. Chin, A. M. K. Cheng, and M. Lupu, ``Incremental Automatic Debugging of Real-Time Systems Based on Satisfiability Counting", IEEE-CS Real-Time and Embedded Technology and Applications Symposium, San Francisco, March 2005.

[9] Albert M.K. Cheng, “Real-Time Systems: Scheduling, Analysis, and Verification”, Wiley, August 2002.

[10] Object Management Group, “Unified Modeling Language 1.5 Specification” OMG Document formal/2003-03-01, March 2003. http://www.omg.org/technology/documents/formal/-
uml.htm.

[11] Stephan Flake, “Temporal OCL Extensions for Specification of Real-Time Constraints”, SVERTS Workshop at UML 2003, San Francisco, USA, October 20, 2003

[12] M. Cengarle and A. Knapp. “Towards OCL/RT”, In L.-H. Eriksson and P. Lindsay, editors, Formal Methods – Getting IT Right, International Symposium of Formal Methods Europe, Copenhagen, Denmark, volume 2391 of LNCS, pages 389–408. Springer, July 2002

[13] Vieri Del Bianco, Luigi Lavazza, and Marco Mauri, “Model Checking UML Specification of Real Time Software”, Eighth IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'02) p. 203

[14] Lilius, J., and Porres Paltor, I, “ vUML: a tool for verfying UMLmodels”, ASE'99, 1999, pp. 255-258.

[15] E. Roubtsova and W. Toetenel. “Specification of Real-Time Properties in UML”, In 22nd IEEE Real-Time Systems Symposium RTSS, Work-In-Progress Section, London, UK, December 2001

[16] Alagar, V.S., Muthiayen, D., Pompeo, F., “From behavioral specification to axiomatic description of real-time reactive systems”, Fifth IEEE Real-Time Technology and Applications Symposium, Work-In-Progress Session. Vancouver, BC, June 1999.

[17] Jean-Louis Houberdon, Pierre Combes, Jean-Philippe Babau, and Isabelle Aug_e-Blum, “Validating temporal properties of a deployed application with an MDD approach”, International Workshop MARTES, October 4, 2005 , Montego Bay, Jamaica

[18] Guoqiang Shu, Chao Li, Qing Wang, and Mingshu Li “Validating Objected-oriented Prototype of Real-time Systems with Timed Automata”, 13th IEEE International Workshop on Rapid System Prototyping (RSP'02), p. 99.

[19] Rice, L.E.P.; Cheng, A.M.K., “Timing analysis of the X-38 space station crew return vehicle avionics**”,** Proceedings of the Fifth IEEE Real-Time Technology and Applications Symposium, 1999, 2-4 June 1999 Page(s):255 – 264.

APPENDIX

Sample of the timing constraints for the X-38 Avionics system [19]:

```
; periodicity
timedevents { InvICP_I50FC_SENSOR : InvokeICP_I50FC_SENSOR }
timeconstraints {
require
Duration(InvICP_I50FC_SENSOR.pre,InvICP_I50FC_SENSOR) <= 20
}
; precedence relations
timeconstraints {
require
Duration(RcvICP_I50FC_SENSOR,InvFCP_I50FC)
}
```

```
; precedence between start and stop events
timeconstraints {
require
Duration(InvFCP_I50FC,RcvFCP_I50FC)
}
; priority assertions
; 50 HZ FC higher priority than 10 Hz FC
timeconstraints {
require
Duration(FCP_I50FC,InvFCP_I10FC)
}
```

RTL conversion:

Vi @(S_ICP_I50FC_SENSOR, i + 1) <= @(S_ICP_I50FC_SENSOR, i) + 20

Vi @(E_ICP_I50FC_SENSOR, i) <= @(S_FCP_I50FC, i)

Vi @(S_FCP_I50FC, i) <= @(E_FCP_I50FC, i)

Vi @(FCP_I50FC, i) <= @(S_FCP_I10FC, i)