Electronic Notes in Theoretical Computer Science 39 No. 3 (2000)
URL: `http://www.elsevier.nl/locate/entcs/volume39.html` 13 pages

Specification of Real-Time Systems in UML

E.E.Roubtsova [a,1], J.van Katwijk [b,2], W.J.Toetenel [b,3], C.Pronk [b,4], R.C.M.de Rooij [b,5]

[a] *Faculty of Mathematics and Computing Science TU Eindhoven Den Dolech 2, P.O.Box 513, 5600 MB Eindhoven, The Netherlands*

[b] *Faculty of Information Technology and Systems Delft University of Technology Zuidplantsoen 4, NL-2628 BZ Delft, The Netherlands*

Abstract

We introduce time semantics into UML class and statechart diagrams. This extends the expressiveness of UML for specification of real-time systems and allows to specify verification properties of real-time systems by means of Timed Computation Tree Logic. We furthermore propose a way to collect stereotypes for specification of real-time systems. The approach is illustrated by a case study.

1 Introduction

Most industrial systems today are real-time systems: communication servers, process control systems, traffic control systems, etc. In this class of systems, timing constraints are often as important as functional and ordering ones. Managing temporal constraints as an afterthought is not an optimal solution: applying formal methods already during the design stage can help to meet the constraints.

Designers of real-time systems seldom use an object-oriented approach because they find difficulties in constructing models of these systems [12] and in specifying their properties. The main reason of these difficulties is that there is no standard way handling of time in common object-oriented approaches.

The Unified Modeling language (UML) has become the standard for object oriented design [13], however there is no explicit representation of time in

[1] Email: `E.Roubtsova@tue.nl`
[2] Email: `J.vanKatwijk@its.tudelft.nl`
[3] Email: `W.J.Toetenel@twi.tudelft.nl`
[4] Email: `C.Pronk@twi.tudelft.nl`
[5] Email: `ruud@ruud.org`

many of diagrams of UML and this limits its possibilities for real-time systems specification.

In this paper we present extensions of UML that enable specification of real-time systems and their properties and that make verification of UML projects of real-time systems possible.

We propose a UML data type *DenseTime*, the type to be used for clock variables. The choice of dense time is justified by applications we selected. We assume that each class of a UML specification of a real-time system can define a set of clocks to represent temporal aspects of behaviour.

We limit our paper the cases when both objects and associations between objects are statically defined. The system specification consists of the class diagram, object diagram and the statechart diagram of the real-time system. This tuple of diagrams can be transformed into a specification based on timed automata which can be translated into computation tree.

To specify properties of real-time systems we define specification-classes. Each property is represented as a specification-class that has a predefined constraint with parameters. The constraint is presented in an extended variant of Timed Computation Tree Logic (TCTL) that has been put into basis of the verification tool Prototype Model Checker (PMC) [4].

Specification classes extend the UML class diagram. The formal constraint specification of the system is derived from the extended class diagram. As a result of our extensions, we are able to give both a specification of a system and a specification of its properties to be verified.

The structure of this paper is as follows. Section 2 clarifies the problems of specification of real-time systems in UML and reviews related work. Section 3 introduces our UML extensions for system specification and for properties specification. In section 4 we present some conclusions.

2 Problems of Specification in UML

2.1 An illustration of specification problems

The purpose of this section is to clarify the problems with specifying real-time systems and their properties in UML. Consider as an example a system for concentration control [11], intended to react to the external event e=*'the value of concentration c is more than a given value M'*, where the separation time T of two consecutive events e is given as a property of the system.

Based on the solution presented in [11], we construct the system using two subsystems: the **control subsystem** CS and the **registration subsystem** RS. There is a **buffer** B in the system for an exchange of information between the subsystems. When event e happens, the subsystems begin to work simultaneously. The control subsystem CS can work in one of two alternative modes. If the concentration c is in the range $(M \leq c < Max)$, CS works in **normal mode.** In this mode the subsystem calculates the position x of a control lid, records the result to the buffer and gives a command to move the lid. If ($c \geq Max$), then CS operates in **emergency mode.** In this mode the subsystem opens several lids maximally, notifies personnel and records the current position x of the control lid into the buffer. The registration subsystem RS can be switched off and on by personnel. If it is switched on then in reaction to the occurrence of event e, RS registers the current concentration and the current position of the control lid from the buffer.

Some properties of the system of concentration control are:

Property 1: There are no two consecutive reactions to events e, such that subsystem CS works in the emergency mode both times.

Property 2: The registration of the lid position x from the buffer by the registration subsystem should be completed U time units earlier than the recording a new lid position to the buffer by the control subsystem begins.

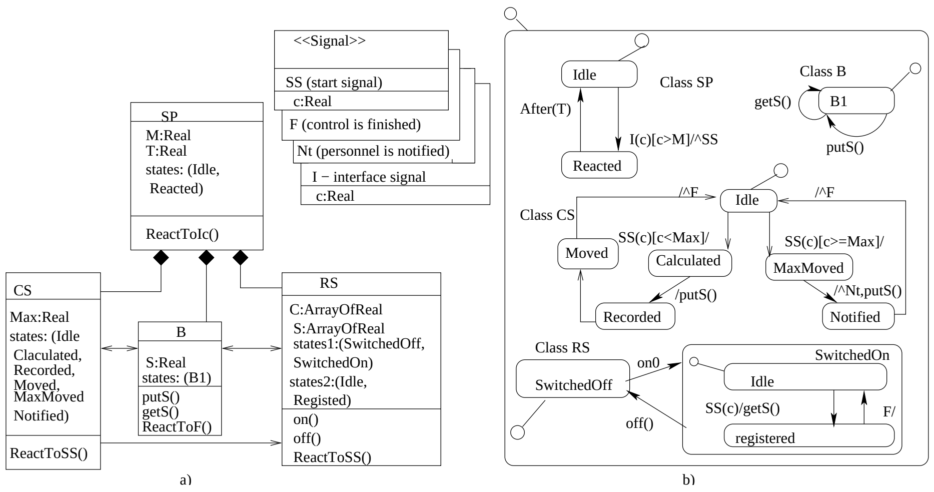

Fig. 1. The class- and the statechart diagrams of the concentration control system.

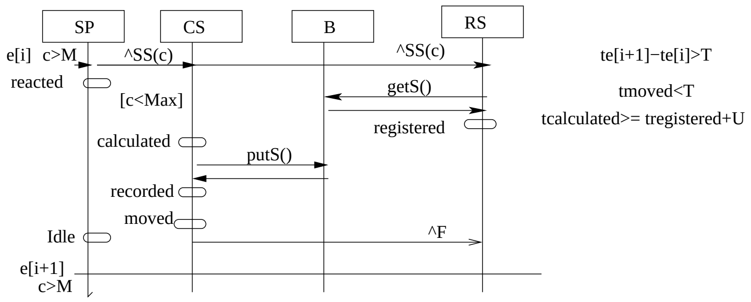

Fig. 2. The sequence diagram of the concentration control system.

Property 31: A reaction to occurrence of event e (of the control system in normal mode) should be completed within the separation time of the event.

Property 32: A reaction to occurrence of event e (of the control system in emergency mode) should be completed within the separation time of the event.

In fig.1 (a) the UML class diagram of the system of concentration control is shown.

- There exists an external interface signal $I(c)$ with concentration c as a parameter.
- Class SP models a sporadic reaction to event e. The class has an operation $ReactToIc()$ and two attributes: T : *Real* - is a separation time;
  M : *Real* - is a value of the concentration on which the system has to react. When interface signal $I(c)$ takes place and guard condition
  $[c > M] = true$ class SP makes a transition from state $Idle$ to state *Reacted*. Transition *Reacted* $\rightarrow Idle$ is possible after the separation time.
- Occurrence of signal $SS(c)$ starts both the control subsystem CS and the registration subsystem RS. Concentration c is the parameter of the signal.
- Class CS corresponds to the control subsystem. Its attribute Max : *Real* - is a value of the concentration. The value of this attribute defines the mode of the control subsystem reaction.
- F is the signal, that is sent by the control system after it has finished its work.
- Nt is a signal of notification of personnel.
- Class RS is a model of the registration subsystem. The class has operations $on()$,$off()$, ReactToSS() and two attributes:
  C : *ArrayOfReal* - is an array of values of the concentration;
  S : *ArrayOfReal* - is an array of values of the lid positions.
- Class B models the buffer. The class contains the attribute S : *Real* to save a value of the lid position and two operations: $putS()$ - to write the attribute S, $getS()$ - to read the attribute S.

The statechart diagram models the behaviour of the system (fig.1(b)).

The sequence diagram (fig.2) is an attempt to specify system properties. The specification of properties in the sequence diagram shows problems. First, we are unable to specify that event e happens sporadically. However, this problem can be solved by syntax extensions. Second, we can not represent all sequences of reactions to event e by sequence diagrams and we can not define their properties. So, it is not possible to present e.g. *Property 1* using UML.

In general, UML does not allow temporal-logic properties such as reachability and properties of sequences of reactions to different instances of an event to be specified.

2.2 Related work

The issue of real-time system specification in UML has been addressed before. A set of suitable stereotypes for hard real-time systems was investigated in the *OOHARTS* (Object Oriented Hard Real Time System) approach [3]. A special 'real-time clock' stereotype is proposed by B. Selic in [12]. However, an attempt to use the classes of this stereotype in a design process encumbers class diagrams by clock-classes and associations with these classes, encumbers statechart diagrams by statecharts of clocks and, as the result, complicates model of a real-time system.

There are extensions of UML that solve the problem of specification for a subclass of properties of real-time systems. Extensions of sequence diagrams were proposed by B.P.Douglass [2], by J. Seeman and J. Wolff v. Gudenberg [1]. B.P. Douglass has proposed dashed syntax boxes to specify periodicity and state marks which bridge the gap between sequence and state diagrams, such that time intervals between states can be defined. J.Seeman and J.Wolff v. Gudenberg have introduced loops and a graphical notation very similar to that used in Message Sequence Charts [7]. To present time constraints a language based on Real Time Logic is used. The authors assume a global discrete clock. An event has a unique name and time mark, different occurrences of an event are distinguished by index values. A textual language UMLscript-RT is proposed to define time constraints as boolean expressions and to describe UML sequence diagrams. Constraints are composed of comparisons of event occurrences.

The possibility of using the Specification and Description Language (SDL) [6] with Message sequence charts (MSC) and UML diagrams for properties specification was discussed in [10].

Nevertheless, none of these extensions allows the specification of reachability of a state or properties of sequences of reactions to different occurrences of an event.

3 An Approach to Specification

3.1 An outline of the approach

We use the *metamodel of UML* [13] and the *Profile Extension Mechanism* [14] for our extensions. A **profile** contains definitions of stereotypes and a set of UML diagrams. Each profile selects elements of the UML metamodel that are useful for the design in a specific application domain. A profile can also include rules for validation and transformation of the diagrams.

We present a profile, which supports our **approach to specification of real-time systems and their properties in UML**. This profile contains definitions of

- a type $DenseTime$,
- UML notions for the specification of a real-time system: real-time class, class diagram, object diagram, statechart diagram,
- UML notions for the specification of properties of a real-time system: specification-class, extended class-diagram, extended statechart diagram, rules of deriving the properties specification from the extended class-diagram.

The outline of our approach in specifying a real-time system is:

(i) A class-diagram for a real-time system S is constructed from both real-time classes and traditional classes.

(ii) An object diagram statically defines the set of objects and associations between objects.

(iii) A statechart diagram for the system S is constructed from the statechart diagrams of the classes.

(iv) An extended class-diagram of the system S is built using specification-classes with predefined constraints. The specification of properties of system S is derived automatically from the extended class diagram.

(v) An extended statechart-diagram is constructed using the statechart diagram of the system and the statechart diagrams of specification classes and the object diagram.

  The specification of system S is derived automatically from extended class- and statechart diagrams. This specification is semantically related to a timed state-transition graph.

This approach prepares data for verification of the UML-project with respect to a timed state-transition graph of a real-time system.

3.2 Specification of real-time systems

An appropriate **representation of time** is a key element in any formalization of real-time systems. Since occurrences of events in this kind of systems can be arbitrarily close to each other [2], we prefer some form of dense time. The type *DenseTime* is represented by nonnegative real values $R^{\geq 0}$. Variables of the type are special, in the sense that they represent clocks. Such a clock can be reset to a new value, a new value of the clock speed can be given, after which clocks value continuously increases with the given speed. The current value of a clock can be observed. For the sake of simplicity we assume here that all clocks run at the same speed.

Let us remind, that a class in correspondence with the UML metamodel [13] is a tuple $Cl = (At, Op, C)$, where At is a finite set of **attributes**, Op is a finite set of **operations**, C is a **constraint**. Assume for now that C is $True$.

A real-time class is a class which contains attributes of traditional types as well as attributes of the type DenseTime.

To present the computational aspect of behaviour of a class, **the UML statechart diagram** is used [13]. The operational semantics of UML statechart diagrams is given in [9]. A **configuration** of a statechart is defined as a static tree of **states** (composite states, history states, substates, synch states etc.). A **transition** is presented by the following notation: $t : s \xrightarrow{e[g]/\wedge a} s'$, where s is a source configuration, s' is a target configuration. A trigger function assigns a triggering event e to a transition, a guard function maps a predicate $[g]$ which is defined on the set of attributes of the corresponding class, and an effect function assigns an action a to a transition [9]. **A transition is enabled if its guard condition $[g] = True$ and its event e takes place.** If several transitions are enabled, one of them begins immediately, however the choice is nondeterministic.

A statechart diagram is a tuple $Sch = (S, E, G, A, TR)$, where S is a set of configurations. E is a set of events. G is a set of guard conditions. A is a set of actions. $TR = \{(s_i, s_j, e, g, a) | s_i, s_j \in S; e \in E, g \in G, a \in A\}$ is a set of transitions.

The introduction of time, as given before, defines the following semantics of the statechart:

- The set of events may include events over clocks, for example, $t = T$ (clock t has a value T), $t_1 \geq t_2$ (the value of the clock t_1 is more or equal than the value of the clock t_2).
- A guard predicate $[g]$ can be defined on the set of clocks.
- Among actions there are actions of resetting of clocks.
- Time will pass in a configuration as long as no transitions are enabled.
- Time will pass in a transition as long as its action runs to completion.

The specification of a real-time system is now a tuple

$$SP = (Class\, Diagram,\ Object\, Diagram,\ Statechart\, Diagram).$$

In this tuple *Class Diagram =(Classes, Associations)* is a finite set of **classes** and **associations** between classes. An association of two classes A, B in the class diagram is a tuple

$r = (AType, A, B, N_A, N_B),\ where\, AType \in ATypes.$

The set of possible association types of two classes on the class diagram is represented by edges between classes with specific ends [13]:

$ATypes = \{\overset{\diamond}{aggregation},\ \overset{\blacklozenge}{composition},\ \overset{\triangleright}{generalization},\ \overset{\rightarrow;-}{association}\}.$

N_A, N_B are numbers of objects of the class A and the class B in the relation correspondingly. The set of associations can be empty. There is a set of signals $Signals \subset Cl$ among classes. They are marked as instances of the $\langle\langle Signal \rangle\rangle$ stereotype. The class diagram does not allow to define the set of instances of classes. To make the specification unambiguous we use UML object diagrams.

An object diagram is a pair $OD = (Objects, Associations_o)$, where $Objects$ is a set of objects of classes from the class diagram, $Associations_o$ is a set of relations of these objects. $Associations_o \subseteq Associations$, only one-to-one relations of objects are allowed.

Each object diagram fixes the sets of objects, attributes and operations.

Statechart Diagram is a parallel composition of the statecharts of the classes from the *Class Diagram* with one initial composite state. The statechart diagram fixes the set of configurations, events, guard conditions and actions.

Transition semantics defines a **timed computation tree** for the specification SP of the real-time system. A node of this tree is a vector $n = (s, v, q)$, where $s \in S$ is a configuration of the *Statechart Diagram*, v is a tuple of values of all attributes of objects of classes from the *Class Diagram* , q is a set of active instances of events. If event $e \in E$ has been activated m times, then there are m instances of the event in the set. An arrow (n_i, n_{i+1}) of this tree corresponds to the transition of the system from state n_i to state n_{i+1}. A path of this tree is a sequence $n_1, ..., n_m$ such that there is an arrow between n_i and n_{i+1}, $i = 1, .., m-1$.

The properties of the timed computation tree can be specified using Timed Computation Tree Logic (TCTL) [8].

3.3 Specification of properties

To specify properties we use a variant [4] of TCTL which was put into basis of Prototype Model Checker developed in our group. This variant contains reset quantifiers over variables, and location predicates. By introducing variables that are not a part of the system specification and by using reset quantifiers, additional properties can be expressed.

A specification of properties is based on a specification of a class, a subsystem or a system. In our variant of TCTL the specification has the following syntax:

$$\psi ::= p|X@x|\neg\psi|\psi \wedge \psi|\psi \vee \psi|\psi AU \psi|\psi EU \psi|u.\psi,$$

where p is a predicate about attributes of a class or about an event. The event predicate e means, that event e is the first in the queue of events. $X@x$ is a predicate about state x of the statechart of class X, $u.\psi$ is a construction that resets new values to variables for specification of properties.

The formula $\psi_1 AU \psi_2$ is satisfied in a node of the computation tree if for all computation paths starting from it, there is a node along it which satisfies ψ_2, and until that node ψ_1 is satisfied. The formula $\psi_1 EU \psi_2$ is satisfied if there is at least one such computation path. There are other convenient predicates used in TCTL. Predicate $AF\psi$ means that on all paths, there is a node, satisfying ψ. Predicate $AG\psi$ means that all nodes on all paths satisfies ψ. The semantics of TCTL was defined in [4] and is not given here. The semantics definition is based on one which can be found in literature [5].

In our experience in specifying of real-time systems we found that many TCTL specifications with the same interpretation are used over and over again, for example, to define the deadline of an operation or to specify an order in which states of the system are reached. For specification of properties some additional classes, such as counters, are often used.

We consider such finds to be stereotypes for specification of real-time systems. Having such an experience of specification, we abstract from concrete classes and we define a specification class.

Let SP be a real-time system specification. A **specification-class** for this real-time system is a tuple $SC = (At_{sc}, Op_{sc}, C_{sc}(SP))$, such that At_{sc} is a finite set of attributes of the specification class. These attributes extend the set of system variables and clocks for property specification. Op_{sc} is set of operations of the specification class. Operations can change attributes values of the specification-class. $C_{sc}(SP)$ is a constraint of the specification-class. A constraint is expressed by a TCTL formula on the set of attributes of specification-class SC, attributes and clocks of classes from the system specification, states, guard conditions and events of statecharts from the system specification.

A **specification-stereotype** is a construct representing a group of specification classes with the same set of attributes and operations and structural-equivalent TCTL-formulas of constraints with the same interpretation.

Examples of specification-stereotypes

1. Deadline-stereotype. Let the specification SP_X of a class X be given. $ClassDiagram_X$ defines this class. The $Statechart_X$ fixes the set of states of the class X, the guard conditions and the actions.

Practice often demands the definition of a deadline T of one state of class X being reached from another state of the class. Therefore a *Deadline-stereotype* should be specified. The constraint of the deadline-stereotype is the following

$$C_D : AG((X@x) \wedge [g] \wedge e \Rightarrow (r := 0).(AF((X@y) \wedge (r \leq T)))),$$

where *r: DenseTime* is an attribute . We define the following set of parameters of the *Deadline-stereotype* :

- T - the value of the deadline,
- $X@x$ - the statechart configuration in which we begin the time calculation.
- g - the guard condition of a transition from the $X@x$ configuration.
- e - the predicate about event e of a transition from the $X@x$ configuration.
- $X@y$ - the statechart configuration in which we finish the time calculation.

The constraint means that for every path, for every state, if we are in configuration $X@x$ and guard $[g]$ of a transition from state $X@x$ is *true*, event e is activated and we reset clock r to zero, then in every path, which begins from this state, state $X@y$ is reachable before the deadline $(r \leq T)$.

2. Counter-stereotype. When we specify sporadic reactions on an event, we use a *Counter* stereotype to count different occurrences of an event e and distinguish these occurrences by unique value of the counter. Let a specification of a class SP_X be given. The specification defines an event e. The stereotype *Counter* has an attribute $(i : Integer)$. The constraint of the stereotype is $TRUE$. When event e happens, the value of the attribute is incremented.

3. Earlier-stereotype. Let the specification of two classes X, Y be given. When the classes are in the state $X@b \wedge Y@b$, they react to the same occurrences of event e. Let the reactions of classes be aperiodic sequences of states.

A stereotype *Earlier* models the property that the state $X@x$ of the class X has been left U time units earlier than the state $Y@y$ of the class Y is reached.

The the *Earlier*-stereotype contains a clock z and the set of parameters:

- real parameter U that define the time interval,
- event e,
- configurations $X@b$, $Y@b$ in which we begin react to event occurrences,
- configuration $X@x$ which has to be earlier,
- configuration $Y@y$ that has to be later.

The constraint of the specification-stereotype

$$\begin{aligned} C_{Earlier}: & AG(X@b \wedge Y@b \wedge e \Rightarrow \neg(Y@y) AU(X@x \wedge \neg Y@y))\wedge \\ & AG(X@b \wedge Y@b \wedge e \Rightarrow AF(Y@y))\wedge \\ & AG(X@b \wedge Y@b \wedge e \Rightarrow AG(X@x \Rightarrow (z := 0).AG(z < U \Rightarrow (\neg(Y@y))) \end{aligned}$$

means that in reaction to event e :

• for all paths state $X@x$ will be reached such that state $Y@y$ can be reached neither before $X@x$ no simultaneously with $X@x$,

• for all paths state $Y@y$ sometime will be reached,

• for all paths, always, if we reset clock in the state $X@x$, then the state $Y@y$ can not be reached before $z \geq U$.

3.4 Extended UML specification

To specify properties we extend the UML state diagram and the UML statechart diagram of the real-time system we specified. Both the specification of the system and the specification of its properties are now represented by a pair of an extended class-diagram and an extended statechart diagram (fig.3). A specification-class has **one-to-one specification associations** with a set of traditional classes, depicted by $- \cdot -$. Specification classes may introduce some additional states in the specification of the system behaviour in form of statecharts of specification classes. However, the behaviour of traditional classes which compose the system will not be changed.

In fig.3 we present the extended class- and statechart diagrams of our case study. There are four specification classes in this diagram, each corresponding to properties which were defined in section 2. *Property 3.1* and *Property 3.2* are modeled by instances of the *Deadline-stereotype. Property 2* is represented by an instance of the *Earlier-stereotype* (fig.3).

Property 1 is modeled by the specification-class *Property 1.* The class has an attribute $k : Integer$ which contains an amount of consecutive reactions of control system CS to event e in emergency mode. The statechart of specification class *Property 1* is shown in fig. 3. When the control subsystem reacts in the emergency mode ($SS(c)[c \geq Max]$), specification-class *Property 1* makes

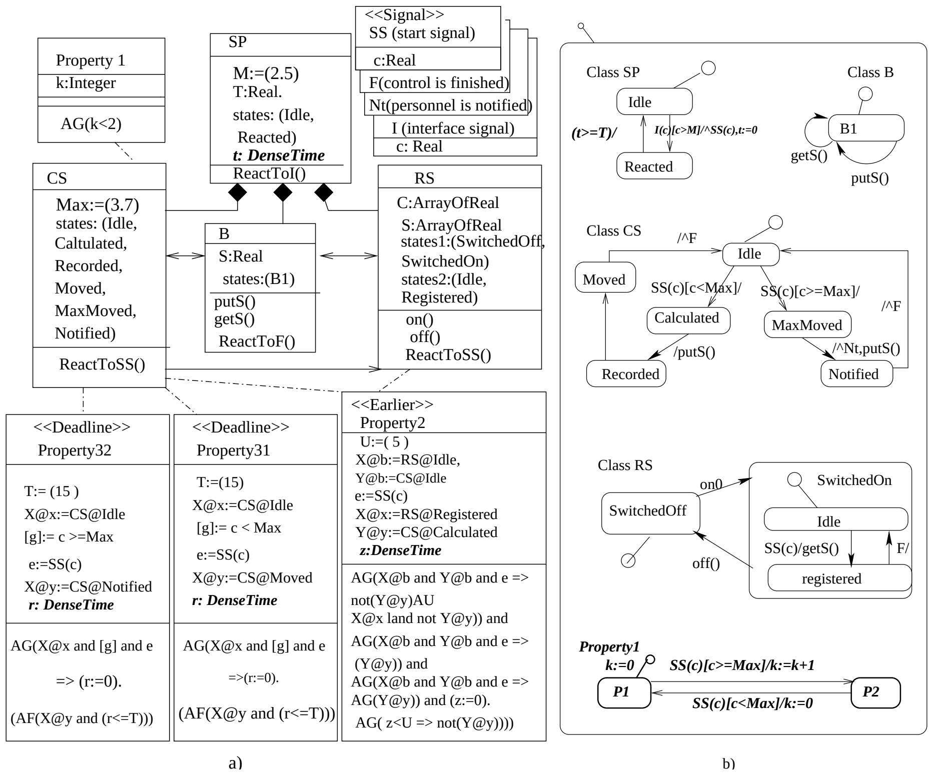

Fig. 3. The extended class- and the statechart diagrams of the system for the concentration control

a transition from state P_1 to state P_2 and the attribute k is incremented. When the control subsystem reacts in the normal mode $(SS(c)[c < Max])$, specification-class *Property 1* makes a transition from state P_2 to state P_1 and the attribute k is reset to zero. The constraint of the specification-class

$$C(SP)_{Property1} : AG(k < 2)$$

means that for every path and for every state of the computation tree of the system of concentration control $(k < 2)$, i.e. *there are no two consecutive reactions on event e such that subsystem CS works in emergency mode both times.*

A constraint of a specification class declares a property of the computation tree of the system. The property of the system is the conjunction of constraints of all specification classes from the extended class diagram of the system.

The pair of extended statechart of the real-time system and extended class diagram is a basis for verification, because this pair of diagrams defines a state-transition graph model of real-time system behaviour and TCTL formulas of properties of the system to be verified.

4 Conclusion

This paper investigated possibilities for the specification of real-time systems in UML. The main problem addressed is the formal specification of properties. We have shown that UML diagrams do not allow temporal-logic properties such as reachability and properties of sequences of reactions to different instances of an event to be specified.

In this paper we have proposed an approach to solve this problem. The benefits of our approach are:

We have introduced dense time into UML class and statechart diagrams. This solution gives rise to a new specification instrument for real-time systems.

We have defined new sorts of classes: real-time classes and specification classes. This opens a universal way to specify all kinds of properties in real-time systems.

The selection of specification-classes from traditional classes allows to collect stereotypes of specification on the base of parameterized TCTL formulas.

The repeated using of this approach in design can help to collect an appropriate set of specification stereotypes of real-time systems. Using formal methods in everyday practice of design is not possible without stereotypes of specification.

We understand that our paper does not answer to many important questions about associations between traditional classes and specification classes, about multiplicity. Currently we are focusing on translation the UML specification into input language of our Prototype Model Checher [4]. This translation will allow to use the tool support for our approach with case studies and it will help to answer to most of opened questions.

References

[1] Seeman J. and J.W.v.Gudenberg *Extension of UML Sequence Diagrams for Real-Time Systems*, The Unified Modeling Language. UML'98: Beyond the Notation, LNCS **1618** (1998), 240–252.

[2] Douglass B. P. *"Real-Time UML. Developing Efficient Objects for Embedded Systems"*, Addison-Wesley (1998).

[3] Kabous L. and W.Nebel, *Modeling Hard Real Time Systems using UML: The OOHARTS Approach.*, UML'99, LNCS **1723** (1999),339–355.

[4] Lutje Spelberg R.F. , W.J.Toetenel and M.Ammerlaan, *Partition Refinement In Real-Time Model Checking*,Formal Techniques in Real-Time and Fault-Tolerant Systems. LNCS **1486** (1998), 143–157.

[5] Alur R., C.Courcoubetis, N.Halbwachs, T.A. Henzinger, P.-H.Ho, X.Nicollin, A.Olivero,J. Sifakis and S.Yovine, *The algorithmic analysis of hybrid systems*, Theoretical Computer Science **138** (1995), 3–34.

[6] Ellsberger J., D.Hogrefe and A.Sarma, *"SDL - Formal Object-oriented Language for Communicating Systems"*, Prentice Hall Europe (1997).

[7] Reniers M.A. *Message Sequence Chart: Syntax and Semantics*, Eindhoven University of Technology (1999).

[8] Alur R., C.Courcoubetis and D.L.Dill , *Model-Checking in Dense Real-Time,* Information and Computation **104(1)** (1993), 2–34.

[9] Lilius J. and I.P.Paltor, *Formalizing UML State Machines for Model Checking*, UML'99. Beyond the Standard, LNCS **1723** (1999), 430–445.

[10] Selic B., Ph.Dhaussy, A.Ek, O.Haugen, Ph.Leblanc and B.Moller-Pedersen, *SDL as UML: Why and What.* UML'99. Beyond the Standard, LNCS **1723** (1999), 446–455.

[11] Novoselov O.N. and A.F. Frolov, *"Theory and calculation of control-measure systems"*, Moscow, Mashinostroenie (1991).

[12] Selic B. *Turning Clockwise: Using UML in the Real-Time Domain*, Communications of the ACM **42, October, 10** (1999), 339–355.

[13] OMG*"Unified Modeling Language Specification v.1.3"*, ad/99-06-10 (1999), URL:`http://www.rational.com/uml/resources/documentation/index.jsp`.

[14] OMG *"Analysis and Design Platform Task Force. Version 1.0 "*, ad/99-04-07 (1999),URL:
`www.iti.upv.es/iti/i+d/mirrors/ftp.omg.org/pub/docs/ad/99-04-07.txt`.