

# Transformation of UML Specification to XTG

Ella E. Roubtsova<sup>1</sup>, Jan van Katwijk<sup>2</sup>, Ruud C.M. de Rooij<sup>2</sup>, and  
Hans Toeteneel<sup>2</sup>

<sup>1</sup> Department of Mathematics and Computer Science  
Eindhoven University of Technology  
The Netherlands

`E.Roubtsova@tue.nl`

<sup>2</sup> Faculty of Information Technology and Systems, Delft University of Technology  
The Netherlands

**Abstract.** We use tuples of extended class, object and statechart UML-diagrams as UML specifications of real-time systems. The semantics of the UML specification is defined by transformation to the eXtended Timed Graphs (XTG). The correctness of our transformation is demonstrated by showing that the XTG computation tree can be projected into the computation tree of the corresponding UML specification. The transformation opens the possibility to specify temporal-logic properties at the UML level and to verify them at the XTG level using the PMC model checker.

## 1 Introduction

The Unified Modeling Language (UML) is the object modeling standard [1] that provides a set of diagrams for system specification from static and dynamic perspectives [6]. Nowadays, it is becoming more and more custom for designers to use formal methods during the design. The formal methods allow to verify a system specification with respect to its property specification. The system specification in UML can be formalized [4], however, the property specification is limited by the logic of the Object Constraint Language (OCL) [6] which does not allow properties of computational paths, reachability, etc. to be specified [2].

This paper shows our approach to specification of systems and properties in UML. In section 2 we consider a tuple of UML class, object and statechart diagrams as a new input language of the Prototype Model Checker (PMC) being under development in Delft University of technology [7]. PMC is intended to verify system specification with respect to properties represented in a variant of Time Computation Tree Logic (TCTL). However, the level of the original system specification language of PMC, namely Extended Timed Graphs (XTG), is too low for the specification purpose. The specification language of the acceptable level is UML. In section 3 we represent the transformation of the UML specification into XTG. In section 4 we conclude about the results of the transformation that allows to specify system properties by TCTL at the UML level and to verify them by means of the PMC.

## 2 System Specification. Property Specification

1. *We have chosen three kinds of UML diagrams to specify a system.*

– Class and object diagrams define a static view of the system. A class diagram specifies sets of classes  $Cl_S$  with their attributes  $At_S$  and operations  $Op_S$ . An object diagram defines a current set of class instances.

*To enable the measurement of time* we introduce clock attributes of special *DenseTime* type. A clock attribute can be reset to a nonnegative real value. After resetting the value of the clock attribute continuously increases.

– A UML statechart diagram addresses a dynamic view of the system. All labels of the statechart use names that are defined by the UML class and object diagrams [3].

*We define a UML statechart as a tuple  $SchD = (S, S_0, Rl)$ , where*

–  $S$  is a tree of states. The states are depicted in the statechart diagrams by boxes with round corners. There are states of three types in this tree: *AND*, *XOR* and *Simple*. *AND* and *XOR* nodes are hierarchical states. One of them is a root of the tree of states. Nodes  $A_1, \dots, A_n$  of a hierarchical state  $C$  are drawn inside of  $C$ . An *AND*-state is divided by dotted lines to put other states between such lines. A simple state does not contain other states. In general, a state  $s$  is marked by  $(Name_s, History_s, In_s, Out_s)$  labels. The labels  $History_s$ ,  $In_s$ ,  $Out_s$  can be empty.

–  $S_0 \subseteq S$  is a set of initial states.

–  $Rl$  is a set of relations among states  $Rl = \{TR, Conn, Synch\}$ .

–  $TR$  is a set of transitions that are represented by labeled arrows.

$$TR = \{(s_i, s_j, e, g, a) \mid s_i, s_j \in S; e \in Op_S,$$

$$g \in Boolean\ Expression(At_S, Op_S), a \in Op_S\}.$$

–  $Conn = \{(I, O) \mid I \subset S, O \subset S, \}$ . The relation is represented by a black rectangle (fork-join connector) and by arrows that are directed from each state  $I_i \in I$  to the connector and from the connector to each state  $O_j \in O$ .

–  $Synch = \{(I, O) \mid I \subset S, O \subset S\}$ . The relation is drawn by two black rectangles (fork and join connectors) and by a circle of a synch-state.

A transition semantics of a UML-statechart, in general, was defined in [4] as a computational tree. A node of this tree is a vector  $n = (s, v, q)$ , where  $s \subset S$  of the  $SchD$ ,  $v$  is a tuple of values of all attributes of objects from the object diagram,  $q$  is a queue of operation calls from the  $SchD$ .

2. *To enable the specification of properties* of computational sub-trees, we define specification classes. Specification classes are stereotyped. They can be related to a set of traditional classes. Each stereotype of specification has its own intuitive name and a formal representation by a parameterized TCTL formula [3]. The TCTL variant that we use [3,7] contains location predicates and reset quantifiers over variables.

### 3 Transformation Rules

We transform a tuple of UML class, object and statechart diagrams into XTG, the original language of the Prototype Model Checker.

#### 3.1 XTG

XTG is a formalism for describing real-time systems.

An XTG is a tuple  $G = (V, L, l_0, T)$ , where

–  $V$  is a finite set of variables of the following

$$DataType = \{Integer, Enumeration, Real, DenseTime\}.$$

*DenseTime* is a type for representing clocks (as nonnegative real that increases continuously).

–  $L$  is a finite set of locations (nodes).  $l_0 \in L$  is an initial location;

–  $T$  is a finite set of transitions (arrows with labels).  $T = \{(l_i, l_j, c, up, ur)\}$ , where  $l_i, l_j \in L$ ,  $c \in BooleanExpression(V)$ ,  $up \in ValueAssignment$  of the variables  $V$ ,  $ur \in \{Urgent, UnUrgent\}$ . An urgent transition is represented by an arrow with a black dot.

A state in the time computation tree semantics of the XTG is defined [7] by a location and the values of variables in the location  $(l, \rho)$ . A transition from a state  $(l, \rho)$  is enabled if  $c(\rho) = True$  after the substitution of the variable values. Time can pass in state as long as its invariant is satisfied and no urgent transitions are enabled. Time is not allowed to progress while a transition marked as urgent is enabled. The parallel composition of XTG is defined [7] by a synchronization mechanism that is based on a form of value passing CCS [5]. A synchronization is a pair of labels specifying that the transition marked by *syn!* in an XTG is executed simultaneously with a transition marked by *syn?* in another XTG.

#### 3.2 Transformation Steps

We make the transformation of the UML specification into XTG in three steps.

- First, we represent the hierarchical states from the statecharts introducing the CCS synchronization. The result of this transformation we name the flat statecharts. A *flat statechart* is a UML statechart which does not contain hierarchical states and uses the CCS synchronization mechanism. The parallel composition of two flat statecharts is a flat statechart such that the set of its transitions is defined by rules of XTG parallel composition.
- Second, we transform the flat statecharts defining the *run to completion* [6] semantics of operation calls and semantics of signals using the synchronization.
- The last step means the transformation of fork-join connectors and synch states of the flat statescharts to XTG.

### From Hierarchical Statecharts to Flat Statecharts

*XOR state with the History mark.* The transition to the XOR state (fig.1) means the transition to the initial state  $a_1$  among substates  $a_1, a_2$ . The transition from the XOR state  $c_1$  means the transition from the current state  $a_i$ . Only one substate from the  $a$ -set can be the current substate. The history label means

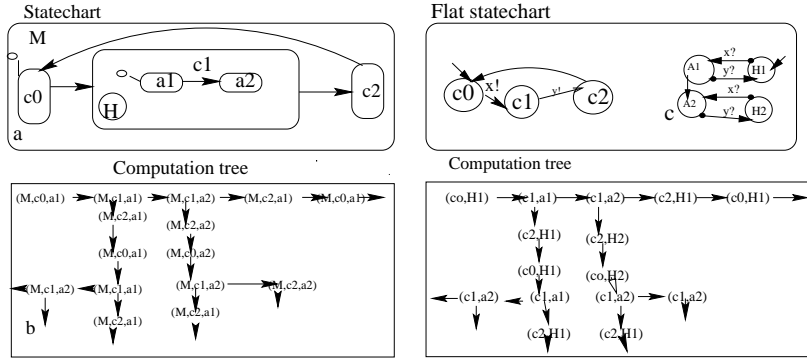


Fig. 1. XOR states

memorizing of the current substate of the XOR state and starting the next computation inside of the XOR state from this substate. The semantics is shown by the computation tree (fig.1b).

We represent the statechart (fig.1) by two XTG: the external graph ( $c_0, c_1, c_2$ ) and internal one ( $a_1, a_2$ ). The history label means that there is a history state  $H_i$  for each  $a_i$ . States  $c_0, H_1$  are initial for the system. The transition to the XOR state is replaced by two synchronous transitions:  $(c_0, c_1, x!)$  and  $(H_i, a_i, x?)$ . The value of  $i$  depends on the last active state of the internal graph. The transition from the XOR state is replaced by pair of synchronous states  $(c_1, c_2, y!)$  and  $(a_i, H_i, y?)$ .

*AND state.* The transition to the AND state  $C$ (fig.2a) corresponds to transitions to both initial states of substatecharts of  $A$  and  $B$ . The arrow from AND state  $C$  means the transitions from the current states of  $A$  and  $B$ . The transition from each state of  $A, B$  is possible. The corresponding computation tree is shown in fig.2b. In the flat statechart representation there are three sub-graphs (fig. 2c). There is a projection of the computation tree (fig. 2d) of the flat statechart to the computation tree of the statechart (fig.2b). In this projection one transition to AND state  $(M, X, A_1, B_1) \rightarrow (M, C, A_1, B_1)$  is modeled by two transitions in the computation tree of flat statechart  $(X, A_0, B_0) \rightarrow (X_1, A_1, B_0) \rightarrow (C, A_1, B_1)$ . A transition from AND state is modeled by a subtree: for example,  $(M, C, A_2, B_2) \rightarrow (M, Y, A_1, B_1)$  (fig. 2b) is represented by  $(C, A_2, B_1) \rightarrow (Y_1, A_0, B_1) \rightarrow (Y, A_0, B_0)$  and  $(C, A_2, B_1) \rightarrow (Y_1, A_2, B_0) \rightarrow (Y, A_0, B_0)$ . *AND*

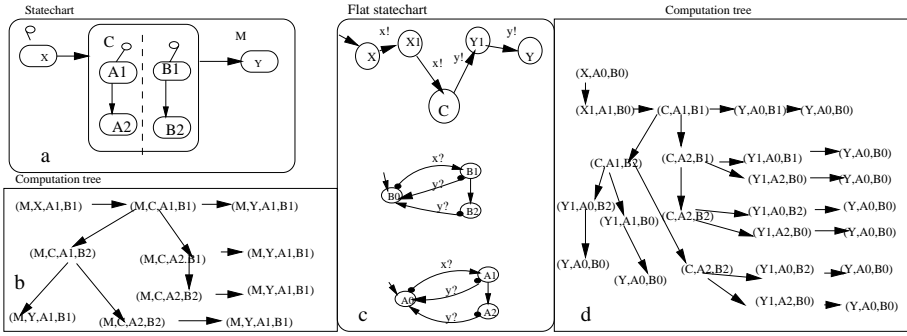


Fig. 2. AND states

state with the history label are transformed to flat statecharts using the combination of rules given in two previous cases.

### Transformation of Operation Calls and Signals

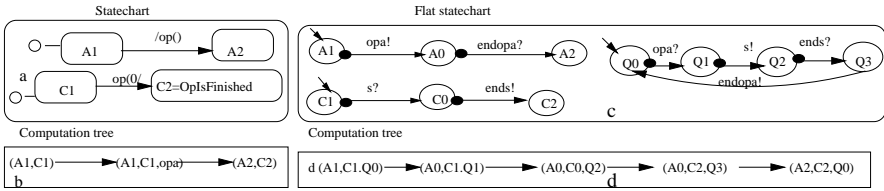


Fig. 3. Operation in statecharts

Let an *operation*  $op$  be called by the object  $A$  and be fulfilled by the object  $C$  (fig.3a). If the operation call is shown in UML statechart, it means that the operation will run to completion. The corresponding computation tree is represented in the fig.3b. The operation call of the object  $A$  is shown by the label  $opa$ . The time between the operation call and the operation end [6] is the subject of the specification in real-time systems and it can be shown by updates of clock attributes at the moment of the operation call. When an operation is called by several classes then there is a time between the operation call and the moment of the begin of the operation. To represent the semantics of the operation calls we extend graphs of objects  $A, C$  by states  $A_0, C_0$  and use an additional flat statechart  $Q$  with four states to model each operation call (fig.3c.) The internal state of this graph  $Q_1$  allows to wait for the moment of the begin of the operation. This moment is shown by synchronization  $s$  with the graph  $C$ . When the operation has been fulfilled, graph  $Q$  is synchronized by  $end_{opA}$  with graph  $A$ . The computation tree of the flat statechart is shown in fig.3d. There is a

projection of this tree to the computation tree of the statechart: three transitions  $(A_0, C_1, Q_1) \rightarrow (A_0, C_0, Q_2) \rightarrow (A_0, C_2, Q_3) \rightarrow (A_2, C_2, Q_0)$  in the computation tree of the flat statechart correspond to one transition  $(A_1, C_1, opa) \rightarrow (A_1, C_2)$  in the computation tree of the statechart.

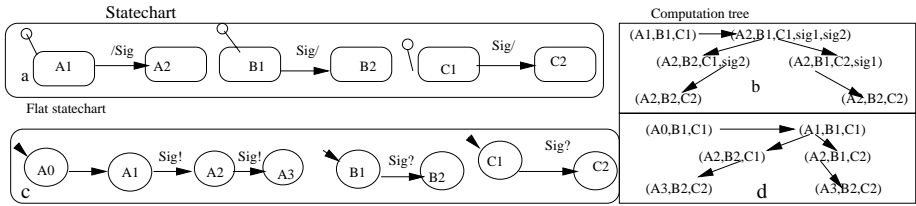


Fig. 4. Signal in statecharts

Let a *signal* be sent to two objects fig.4a. The moment of sending is a subject of specification in real time systems. The reactions to the signal can come in different moments. To represent the behaviour of a signal by flat statecharts (fig.4 b) we extend the set of states to fix the moment of the signal sending and to send 2 synchronizations *Sig!*. We can see in fig.4b,d that the computation tree of the flat statecharts and the computation tree of the statechart are equal.

### Transformation of Flat Statecharts

Assume now that all XOR, AND states, operation calls and signals have been transformed and a flat statechart has represented a system specification.

In fig.5 there is a statechart with a *fork-join* connector and its XTG representation. The semantics of the join-fork statechart is the following. First, all

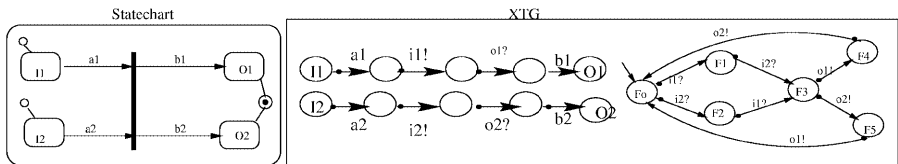


Fig. 5. Fork-Join connector

transitions to the connector have to be finished. The order of the transitions is nondeterministic. When all transitions to connector have been finished then the transitions from the connector are enabled. The order of the transitions is nondeterministic.

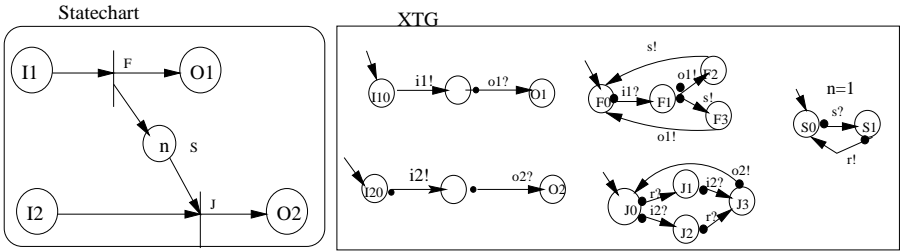


Fig. 6. Synch state

*Synch state.* The graph that is defined by means of *Synch*-relation is a statechart fig. 6. The semantics of this statechart uses the semantics of connectors and the synch state. The last one guarantees that one region of statechart leaves a particular state before another region can enter a particular state. If a synch state is marked by number 1 then one input to synch state corresponds to one output from it. This case is shown in the fig. 6. If a synch is marked by number  $n$  then one input to synch-state can correspond to  $n$  outputs from it. An unlimited synch-state is marked by symbol  $*$ .

*In* and *Out* labels of the state mean that all edges to the state are marked by *In* and all edges from the state are marked by *Out*.

## 4 Conclusion

Analyzing the transformation rules of our UML specification into the input language XTG of the PMC toolset we can conclude that there is a projection of the XTG computation tree into the computation tree of the corresponding UML specification. So, the properties of computational sub-trees specified at the UML level are visible at the XTG level. The TCTL variant that we use [3,7] contains state predicates and reset quantifiers over clocks. For example, we would like to specify a *Deadline* for the state  $y$  for the statechart with an AND-state (fig. 2), i.e. if we are situated in state  $x$  of the class  $X$  we should achieve state  $y$  till the deadline  $T$ . The stereotype *Deadline* has the clock attribute  $t$ , the parameters: deadline  $T$ , two state predicates  $X@x$ ,  $X@y$  and the following TCTL formula:  $AG(X@x \Rightarrow (t := 0).(AF(X@y \wedge t \leq T)))$ .

The PMC tool is able to verify temporal properties of systems with unlimited sets of traces. The combination of the model checker power and the UML specification power allows to apply formal methods into every day practice of design.

## References

1. G. Booch, J. Rubaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, Amsterdam, 1999.

2. B. P. Douglass. *Real-Time UML. Developing Efficient Objects for Embedded Systems*. Addison-Wesley, 1998.
3. E.E. Roubtsova and J.van Katwijk and W.J. Toetenel and C. Pronk and R.C.M.de Rooij. The Specification of Real-Time Systems in UML. *MCTS2000*, <http://www.elsevier.nl/locate/entcs/volume39.html> , 2000.
4. J. Lilius and I.P.Palor. Formalising UML StateMachines for Model Checking. *UML'99. Beyond the Standard, LNCS 1723*, pages 430–445, 1999.
5. R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
6. OMG. *Unified Modeling Language Specification v.1.3*. ad/99-06-10, <http://www.rational.com/uml/resources/documentation/index.jsp>, June 1999.
7. R.F. Lutje Spelberg and W.J. Toetenel and M. Ammerlaan. Partition Refinement In Real-Time Model Checking. *Formal Techniques in Real-Time and Fault-Tolerant Systems. LNCS* , 1486:143–157, 1998.