$\mu UCTL$: A Temporal Logic for UML Statecharts*

S. Gnesi[1], F. Mazzanti[1]

Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo"
ISTI- CNR
Pisa, Italy

Abstract. In this paper we present the state/event-based temporal logic $\mu UCTL$ that makes possible the description of properties on UML model evolutions and assertions on explicit local state variables of UML state machines. This logic allows both to specify the basic properties that a state should satisfy, and to combine these basic predicates with advanced logic or temporal operators. Doubly Labelled Transition Systems are the semantic domain for $\mu UCTL$ where states are labelled by sets of propositions that hold in them and transitions by events performed. The logic we propose here is then applied to verify properties over the dynamic behaviour of a mobile system modelled as extended UML statecharts.

1 Introduction

Most of the specification languages used to describe systems are either state-based or event-based. In the first case systems are characterized by their states and by the transitions between states; in the latter by the events they perform moving from one state to another. Indeed both paradigms are important for the specification of real systems and hence specification languages should cover both.

The Unified Modelling Language (UML) is a graphical modelling language for object-oriented software and systems [18, 23]; it has been specifically designed for visualizing, specifying, constructing and documenting several aspects - or views- of systems. Different diagrams are used for the description of the different views. In this paper we focus on UML Statechart Diagrams, which are meant for describing dynamic aspects of system behaviour and cover both different paradigms of modelling.

The UML semantics [23] associates to each active object a state machine, and the possible system behaviours are defined by the possible evolutions of a set of communicating state machines. All the possible system evolutions can be formally represented as a Doubly Labelled Transition Systems [11] in which the states represent the various system configurations and the edges the possible evolutions of a system configuration.

* Work partially founded by IST-2001-32747 project*AGILE*.

After a system has been modelled it is also useful to provide formal tools to check the validity of properties over the system under specification. Temporal logics have been widely recognized as a useful formalism to express liveness (something good eventually happens) and safety (nothing bad can happen) properties of complex systems (with and without fairness constraints). Most of the commonly used temporal logics deal only with one of the two paradigms (states/events) hence we often speak about state-based and event (or action)-based logics. In the case of systems described in UML this is no more sufficient.

In this paper we present the state/event-based temporal logic $\mu UCTL$ that makes possible the description of properties on UML models. This logic allows both to specify the basic properties that a state should satisfy, and to combine these basic predicates with advanced temporal operators dealing with the events performed. Doubly Labelled Transition Systems are the semantic domain for $\mu UCTL$.

State/event-based logics have been recently described by some authors with different purposes than our. In [15] a state/event-based logic for Petri Nets has been introduced. In [14] a modal logic without fixed point operator and interpreted over a modal version of Doubly Labelled Transition Systems, called Kripke MTS, is defined. A state/event extension of a linear time temporal logic and a model checking framework for it has been presented in [7].

The logic we propose here can also be applied to describe properties of mobile systems modelled as extended UML statecharts. This extension has been proposed in the framework of mobile extensions of UML [1, 2] adding concepts to describe mobile computations. Here we consider an extension of core UML state machines by primitives that designate the location of objects and their moves within a network [16].

The verification of such properties over the model of the system has been done using the prototypical model checker, UMC, for $\mu UCTL$ and UML statecharts (cf. [13]).

The paper is organized as follows. In Section 2 some preliminary definitions are given. The semantics of UML state machines over Doubly Labelled Transition Systems is presented in Section 3. In Section 4 we present syntax and semantics of the $\mu UCTL$ logic. In Section 5 the use of $\mu UCTL$ and its model checker to describe and verify properties on systems modelled as UML statecharts is shown. In Section 6 a case study on the application of $\mu UCTL$ to mobile systems modelled as extended UML statecharts is reported. Finally, Section 7 concludes the paper.

2 Preliminaries

Before to present the semantics of UML state machines and to introduce $\mu UCTL$, we give a slightly different definition of Labelled Transition Systems [22] and of Doubly Labelled Transition Systems [11]. The latter will be used as semantic models UML state machines and for $\mu UCTL$ formulae.

Definition 1 (Labelled Transition System) *A Labelled Transition System (LTS in short) is a 4-tuple* (Q, q_0, Act^*, R)*, where:*

- Q *is a set of states;*
- q_0 *is the initial state;*
- Act *is a finite set of observable events.*
  e *ranges over* Act*.*
- Act^* *is the set of finite sequences of observable events;* es *ranges over* Act^* *and* es *can be written as* $e_1; \ldots; e_n$*;*
- Act^* *is the set of transition labels.* α *ranges over* Act^**.* α *denotes or a sequence* es *of observable events or the empty sequence* ϵ*;*
- $R \subseteq Q \times (Act^*) \times Q$ *is the transition relation. Whenever* $(q, \alpha, q') \in R$ *we will write* $q \xrightarrow{\alpha} q'$*.*

Note that the main difference between this definition of LTS and the classical one lies in the labels of transitions. Here transitions are labeled by sequences of events (eventually of lenght 0) while usually they are labelled by single observable or unobservable events. Equivalences defined on states of LTSs can be extended to deal with transition realations labeled as sequences of events.

Definition 2 (Doubly Labelled Transition System) *A Doubly Labelled Transition System (*L^2TS *in short) is a 5-tuple* $(Q, q_0, Act^*, R, \mathcal{L})$*, where* (Q, q_0, Act^*, R) *is an LTS and* $\mathcal{L} : Q \longrightarrow 2^{AP}$ *is a labelling function that associates a set of atomic propositions* AP *to each state of the LTS.*

Atomic propositions, $p \in AP$ will typically have the form of expressions like $VAR = value$. L^2TS can be projected naturally on both LTSs and Kripke structures and equivalences defined on states of LTSs and Kripke structures can be lifted over L^2TS.

3 L^2TS semantics for UML State Machines

According to the UML paradigm a dynamic system is constituted by a set of evolving and communicating objects. Each object has a set of local attributes, an event pool collecting the set of events which need to be processed, and a set of active states inside a corresponding statechart diagram. In fact, according to UML semantics [23] (UML Superstructure 2.0 Draft Adopted Specification, Section 15.3.12) the behaviour of an object is modelled as a traversal of a graph of state nodes interconnected by one or more joined transitions that are triggered by the dispatching of series of events. The concept of state machine is used to express precisely the behaviour of objects. The so-called run-to-completion step defines the passage between two configurations of the state machine. The run-to-completion assumption states that an event can be taken from the pool and dispatched only when the processing of the previous event is fully completed. During a run-to-completion step a sequence of activities can be executed, which include the change of the value of some local attribute, the sending of a signal to some object, the activation or deactivation of some node of the statechart. We refer to [UML 15.3.12] for the precise definition of state machines and run-to-completion steps. Here we only show a possible formalization of a state machine as a L^2TS, in which the states represent the state machine configurations, and the transitions represent the state machine steps. Several approaches have been proposed in the literature for the definition of a formal semantics of UML Statechart Diagrams, e.g.[23, 3, 4, 21, 24], all these approach either use Kripke structures or respectively labelled transition systems as a semantic model for the description of the dynamic behavior of an UML system. We believe instead, that doubly labelled transitions system are far more intutive, flexible and expressive structures for this purpose.

The labelling associated to the L^2TS states describes the structural properties of state machine configurations that we are interested to observe, and the labelling associated to the transitions of the L^2TS describes the actions being performed during a run-to-completion step which we are interested to observe. In this paper we suppose that the labelling of a state shows the current values of the object attributes in that state machine configuration, and the labelling of a transition shows the sequence of signals generated by the run-to-completion step; moreover we suppose that event pools support a FIFO policy (this is an aspect which UML intentionally leaves unspecified).
For example, given the statechart diagram of Fig. 1, the L^2TS associated to the corresponding object *obj* is the one shown in Fig. 2.

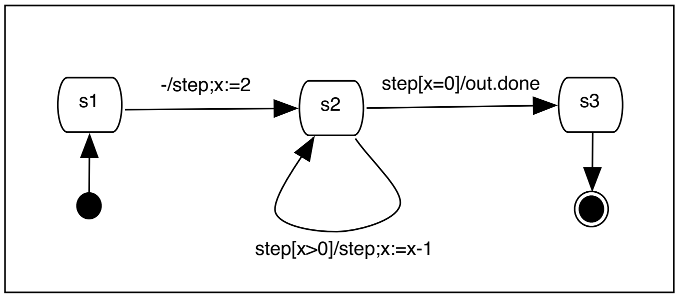

Fig. 1. A simple statechart diagram

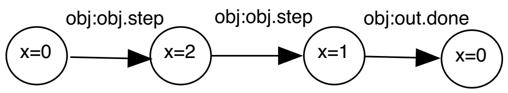

Fig. 2. The L^2TS associated to the statechart diagram of Fig. 1

Actually, we are interested in modelling a system as a collection of evolving and communicating objects. UML does not specify the overall behaviour of a system composed by a set of set communicating state machines evolving in parallel; in particular UML intentionally does not specify the reliability and delay of communications between state machines, not the degree of parallelism with which they evolve. Here we make the assumption that communications are instantaneous and loss-less, and that a system evolution is constituted by a single state machine evolution (i.e. state machines are supposed to evolve in interleaving).

For example, if we consider a system composed by two equals objects, obj1 and obj2, as described by the statechart diagram of Fig. 1, the associated L^2TS is shown in Fig. 3

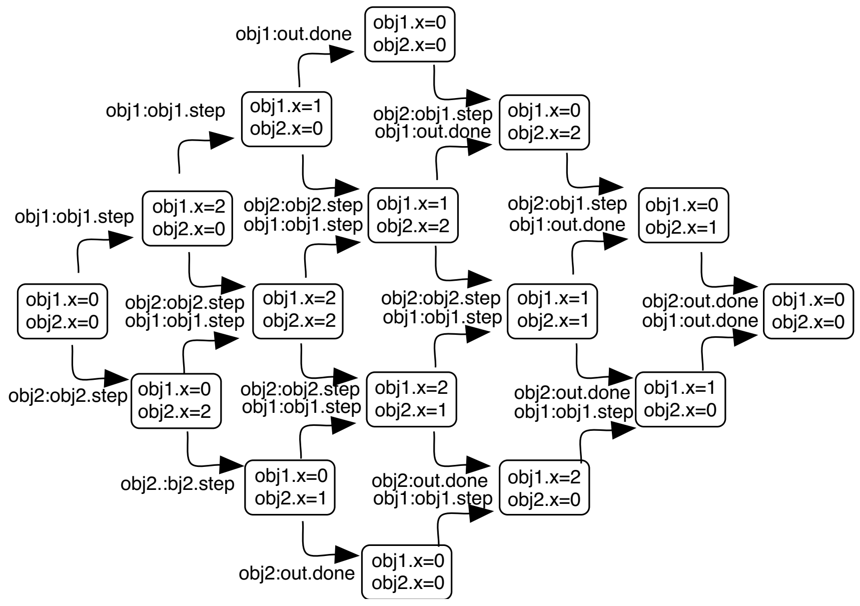

Fig. 3. The L^2TS associated to the composition of $obj1$ and $obj2$

Notice that system events have the form: $source : target.signal(args)$ where source is the name of the evolving object (sending the signal), target is the object which is the destination of the signal, signal the signal name and args its parameters (if any). Moreover, since events may be more in general sequences of events, the transition labels of L^2TSs associated to UML statecharts may have the form: $source1 : target1.signal1(args); source2 : target2.signal2(args); \ldots$.

4 $\mu UCTL$

In this Section we present syntax and semantics of the $\mu UCTL$ logic. This logic, *action and state based*, allows to reason both on states properties and to describe the behaviour of systems that perform actions during their working time. It includes both the branching time action-based logic ACTL [11] and the branching time state-based logic CTL [10].

We will then show in the next section that $\mu UCTL$ is suitable to express the behavioural properties of systems modelled as mobile UML communicating state machines.

Before defining the syntax of $\mu UCTL$ we introduce an auxiliary logic of events.

Definition 3 (Event formulae) *Given a set of observable events Act, the language $\mathcal{EF}$ of the event formulae on $Act \cup \{\tau\}$ is defined as follows:*

$$\chi ::= tt \mid e \mid \tau \mid \neg\chi \mid \chi \wedge \chi$$

Definition 4 (Event formulae semantics) *The satisfaction relation $\models$ for event formulae ($\alpha \models \chi$) is defined as follows:*

- $\alpha \models tt$ *always*
- $\alpha \models e$ *iff* $\alpha = e^1; \ldots; e^n$ *and exists i in* $\{1 \ldots n\}$ *such that* $e^i = e$
- $\alpha \models \tau$ *iff* $\alpha = \epsilon$
- $\alpha \models \neg\chi$ *iff not* $\alpha \models \chi$
- $\alpha \models \chi \wedge \chi'$ *iff* $\alpha \models \chi$ *and* $\alpha \models \chi'$

As usual, *ff* abbreviates $\neg tt$ and $\chi \vee \chi'$ abbreviates $\neg(\neg\chi \wedge \neg\chi')$.

$\mu UCTL$ is a branching time temporal logic of state formulae (denoted in the following by ϕ),

Definition 5 ($\mu UCTL$ syntax)

$$\phi ::= true \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid p \mid \ <\chi>\phi \mid \mu Y.\phi(Y) \mid Y$$

where Y ranges over a set of variables, *state formulae* are ranged over by ϕ, $<\chi>$ is the *strong next* operator.

4.1 $\mu UCTL$ semantics

The formal semantic of $\mu UCTL$ is given over Doubly Labelled Transition Systems. Informally, a formula is true on an L^2TS, if the set of transitions of the L^2TS verifies what the formula states. We hence say that the basic predicate p is true if and only if it belong to the predicates which are true in the current state. in a state q if $q(VAR) = value$. The formula $<\chi>\phi$ (strong next) holds if there exists a next state performing an event satisfying χ and in which the formula ϕ holds. $\mu Y.\phi(Y)$ is the minimal fixed point operator.

Definition 6 ($\mu UCTL$ **semantics)** *The satisfaction relation for $\mu UCTL$ formulae is defined in the following way:*

- $q \models p$ *if and only if* $(p) \in \mathcal{L}(q)$*;*
- $q \models true$ *holds always;*
- $q \models \neg\phi$ *if and only if not* $q \models \phi$*;*
- $q \models \phi \wedge \phi'$ *if and only if* $q \models \phi$ *and* $q \models \phi'$*;*
- $q \models < \chi > \phi$ *if and only if there exists* q' *such that* $q \xrightarrow{\alpha} q'$, $q' \models \phi$ *and* $\alpha \models \chi$*;*
- $q \models \mu Y.\phi(Y)$ *if and only if* $q \models \bigvee_{n \geq 0} \phi^n(false)$*, where* $\phi^0(Y) = Y$ *and* $\phi^{n+1}(Y) = \phi(\phi^n(Y))$*.*

As usual, $false$ abbreviates $\neg true$, $\phi \vee \phi'$ abbreviates $\neg(\neg\phi \wedge \neg\phi')$ and $\phi - > \phi'$ abbreviates $\neg\phi \vee \phi'$. $\nu Y.\phi(Y)$ stands for $\neg\mu Y.\neg\phi(\neg Y)$; ν is called the maximal fixpoint operator. Several useful derived modalities can be defined, starting from the basic ones. In particular $[\chi]\phi$ for $\neg < \chi > \neg\phi$, $EF\phi = \mu Y.(\phi \vee < tt > Y)$ for any χ, the "eventually" temporal operator. It holds if and only if the formula ϕ holds in at least one configuration reachable from the current state. Then we will write $AG\phi$ for $\neg EF \neg\phi$; the "forall" temporal operator. It holds if and only if the formula ϕ holds in all the configurations reachable from the current state.

Note that $\mu UCTL$ has the same expressive power of the propositional μ-calculus [17] when an arbitrary nesting of μ and ν fixed points are used. The main difference between $\mu UCTL$ and μ-calculus lies in the syntax extension that allows both state based properties, that is those definable in the propositional μ-calculus, and action based properties, expressible instead in the Hennesy - Milner logic plus recursion [20] to be expressed.

As it is well known, logics such as $\mu UCTL$ include both linear time (i.e. LTL [19]) and branching time logics (i.e. CTL, CTL^* [5, 9], ACTL, $ACTL^*$ [11]) and it has also been widely discussed in the literature that these logics have different expressive power in terms of the properties definable in them.

We introduce now a subset of $\mu UCTL$, $UCTL$ including both ACTL and CTL operators, that corresponds to the alternation free fragment of $\mu UCTL$. This subset is particularly interesting in verification purposes since for it efficient model checking tools exist (cf. [13]). The $UCTL$ operators could have been defined as derived operators from the $\mu UCTL$ ones. However, classical *Until* operators are not easily derivable in a state/event based framework [15], hence we have preferred to directly introduce them to make easier the understanding of their meaning.

The syntax of $UCTL$ formulae is defined by the following grammar:

Definition 7 ($UCTL$ **syntax)**

$$\phi ::= true \mid p \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid A\pi \mid E\pi$$

$$\pi ::= X\chi\phi \mid \phi\,_{\chi}U\ \phi \mid \phi\,_{\chi}U_{\chi'}\ \phi$$

state formulae *are ranged over by* ϕ, path formulae *are ranged over by* γ, E *and* A *are* path quantifiers, X *and* U *are indexed* next *and* until *operators.*

In order to present the *UCTL* semantics, we need to introduce the notion of paths over L^2TS .

Definition 8 (paths) *Let* $\mathcal{A} = (Q, q_0, Act^*, R, \mathcal{L}))$ *be a* L^2TS.

- *σ is a path from $r_0 \in Q$ if either $\sigma = r_0$ (the empty path from r_0) or σ is a (possibly infinite) sequence $(r_0, \alpha_1, r_1)(r_1, \alpha_2, r_2)\ldots$ such that $(r_i, \alpha_{i+1}, r_{i+1}) \in R$.*
- *The concatenation of paths is denoted by juxtaposition. The concatenation $\sigma_1\sigma_2$ is a partial operation: it is defined only if σ_1 is finite and its last state coincides with the initial state of σ_2. The concatenation of paths is associative and has identities. Actually, $\sigma_1(\sigma_2\sigma_3) = (\sigma_1\sigma_2)\sigma_3$, and if r_0 is the first state of σ and r_n is its last state, then we have $r_0\sigma = \sigma r_n = \sigma$.*
- *A path σ is called maximal if either it is infinite or it is finite and its last state has no successor states. The set of the maximal paths from r_0 will be denoted by $\Pi(r_0)$.*
- *If σ is infinite, then $|\sigma| = \omega$.*
  If $\sigma = r_0$, then $|\sigma| = 0$.
  If $\sigma = (r_0, \alpha_1, r_1)(r_1, \alpha_2, r_2)\ldots(r_n, \alpha_{n+1}, r_{n+1})$, $n \geq 0$, then $|\sigma| = n + 1$. Moreover, we will denote the i^{th} state in the sequence, i.e. r_i, by $\sigma(i)$.

Definition 9 (*UCTL* semantics) *The satisfaction relation for UCTL formulae is defined in the following way:*

- *$q \models true$ holds always;*
- *$q \models p$ if and only if $p \in \mathcal{L}(q)$;*
- *$q \models \neg\phi$ if and only if not $q \models \phi$;*
- *$q \models \phi \wedge \phi'$ if and only if $q \models \phi$ and $q \models \phi'$;*
- *$\sigma \models X\{\chi\}\phi$ iff $\sigma = (\sigma(0), \alpha_1, \sigma(1))\sigma'$, and $\alpha_1 \models \chi$, and $\sigma(1) \models \phi$*
- *$\sigma \models [\phi\{\chi\}U\phi']$ iff there exists $i \geq 0$ such that $\sigma(i) \models \phi'$, and for all $0 \leq j < i$: $\sigma = \sigma'(\sigma(j), \alpha_{j+1}, \sigma(j+1))\sigma''$ implies $\sigma(j) \models \phi$, and $\alpha_{j+1} = \epsilon$ or $\alpha_{j+1} \models \chi$*
- *$\sigma \models [\phi\{\chi\}U\{\chi'\}\phi']$ iff there exists $i \geq 1$ such that $\sigma = \sigma'(\sigma(i-1), \alpha_i, \sigma(i))\sigma''$, and $\sigma(i) \models \phi'$, and $\sigma(i-1) \models \phi$, and $\alpha_i \models \chi'$, and for all $0 < j < i$: $\sigma = \sigma'_j(\sigma(j-1), \alpha_j, \sigma(j))\sigma''_j$ implies $\sigma(j-1) \models \phi$ and $\alpha_j = \epsilon$ or $\alpha_j \models \chi$*

Also for UCTL a set of derived operators can be derived, in particular:

- $EF\phi$ stands for $E[true_{\{true\}}U\phi]$.
- $< \chi > \phi$ stands for $E[true_{\{false\}}U\{\chi\}\phi]$.
- $EF_{\{e\}}\phi$ stands for $E[true_{\{e\}}U\phi]$.

4.2 $\mu UCTL$ Model Checking

We have developed an "on the fly" model checking tool for $\mu UCTL$, called UMC. The basic idea behind UMC is that, given a L^2TS state, the validity of a formula on that state can be evaluated analyzing the transitions allowed in that state, and analyzing the validity of some sub-formula in only some of the next reachable states, in a recursive way. In this way (depending on the formula) only a fragment of the overall state space might need to be generated and analysed to be able to produce the correct result. Model checking procedure like the above are also called local [8] in contrast with those called global [10] where the whole state space is explored to check the validity of a formula. The complexity results of UMC are those expected [12]. We have linear time complexity for the evaluation of $UCTL$ formulae and exponential worst case time complexity for the full $\mu UCTL$.

5 Model checking UML State Machines

UMC can be applied to check the validity of $\mu UCTL/UCTL$ formulae over a set of communicating (i.e. exchanging signals) UML State machines. The "on the fly" approach seems to be particularly promising when applied to UML state machines (or groups of communicating state machines) because it can easily be extended also to the case of potentially infinite state space, as it may happen for UML state machines [13].

Indeed, since a naive ''depth first" evaluation algoritm in the case of infinite state machines might fail to find the correct result, UMC adoptes a ''bounded" model checking approach [6]; i.e. the evaluation is started assuming a certain value as maximun depth limit of the model generation. In this case if a result of the evaluation of a formula is found remaining within the requested depth, then the result holds for the whole system, otherwise the depth limit is increased and the evaluation restarted. This approach, initially introduced in UMC to overcome the problem of infinite state machines, happens to be quite useful also for another reason. Setting a small initial depth limit, and a small automatic increment of it at each re-evaluation failure, when we finally find a result we can have a reasonable (almost minimal) explanation for it, and this could be very useful also in the case of finite states machines.

If we consider again the system composed by two equals objects, obj1 and obj2, described in Fig. 1 we may check on it properties such as:

- $EX_{\{obj1:obj1.step\}}true$
  that means: in the initial configuration obj1 can perform an evolution in which it sends the signal *step* to itself. This property is checked to be true on the model of the system described in Fig. 3.

- $AG((EX_{\{obj1:obj1.step\}}true)->(obj1.x=0))$
  meaning that the event $\{obj1 : obj1.step\}$ can be sent, only when the object attribute has value 0. This is false on the system described in Fig. 3.

- $EF(\nu Y. < \tau > (Y))$
  meaning that there exists an infinite cyclic empty sequence. This is false on the system described in Fig. 3.

6 A case study: Mobile UML State Machines

The $\mu UCTL$ logic and its model checker can also be applied to verify properties of mobile systems modelled as extended UML statecharts. This extension has been proposed in the framework of mobile extensions of UML [1, 2] adding concepts to describe mobile computation. Here we consider an extension of core UML state machines by primitives that designate the location of objects and their moves within a network [16].

In this framework it is possible to describe the dynamic behaviour of systems where component objects are characterized by a location and that can move from a place to another. The topology of a system is then modelled by an "atLoc" attribute, associated to each class, which represents its locality. Mobility is realized by all the operations which update the atLoc attribute of an object. This simple extension of UML statecharts allows mobile systems to be described quite naturally. A statechart diagram is defined for each class of the model, providing a complete operational description of the behaviour of all the objects of the class. The whole system is then represented by a set of class objects.

Example 1. Let us consider a topology of interconnected "places". Each "place" may have up to four links to other places (these links are named N, S, W, E); moreover each place has its own unique id. We have also a "traveler", which has an "atLoc" attribute representing his current location, and an id. The goal of the traveler is to move around the topology, searching for a place with a matching id. The behavior of the "Traveller" class is specified by the statechart diagram shown in Fig. 4 and the behavior of the "Place" class is shown in Fig. 5.

The topology of the places is shown in Fig. 6.

Using a textual notation, the initial system deployment is specified as a collection of five objects:
$P1 : Place(N => null, S => P3, W => P2, E => P2, id => 1)$
$P2 : Place(N => null, S => P4, W => P1, E => P1, id => 2)$
$P3 : Place(N => P1, S => null, W => P4, E => P4, id => 3)$
$P4 : Place(N => P2, S => null, W => P3, E => P3, id => 4)$
$T : Traveler(myid => 4, atLoc => P1)$

Starting from this description we may derive the L^2TS which represent the dynamic behaviour of this system, in which now the states describe the state machine configurations also in relation with the "atLoc" attribute.
The initial fragment of the L^2TS modelling the system is sketched in Fig. 7 The whole L^2TS model consists in 103 states and 110 transitions.

On the above systems we may express some properties as the following:

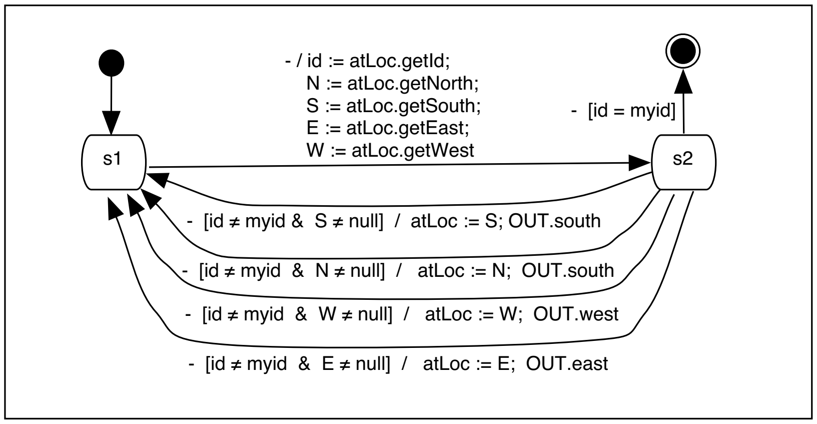
- / id := atLoc.getId;
N := atLoc.getNorth;
S := atLoc.getSouth;
E := atLoc.getEast;
W := atLoc.getWest
- [id = myid]
s1
s2
- [id ≠ myid & S ≠ null] / atLoc := S; OUT.south
- [id ≠ myid & N ≠ null] / atLoc := N; OUT.south
- [id ≠ myid & W ≠ null] / atLoc := W; OUT.west
- [id ≠ myid & E ≠ null] / atLoc := E; OUT.east

Fig. 4. The traveller statechart

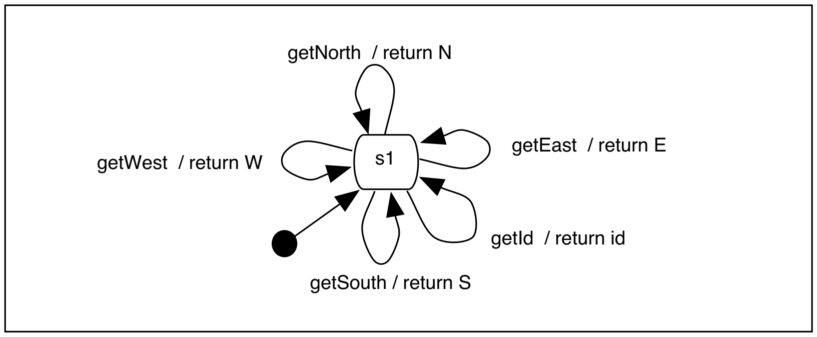

Fig. 5. The place statechart

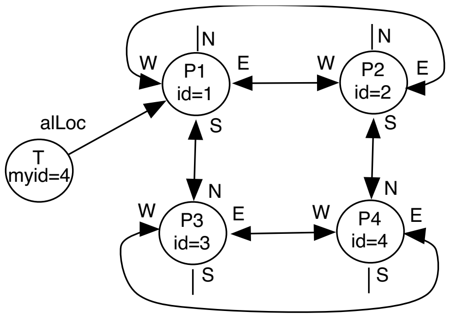

Fig. 6. The topology of the places

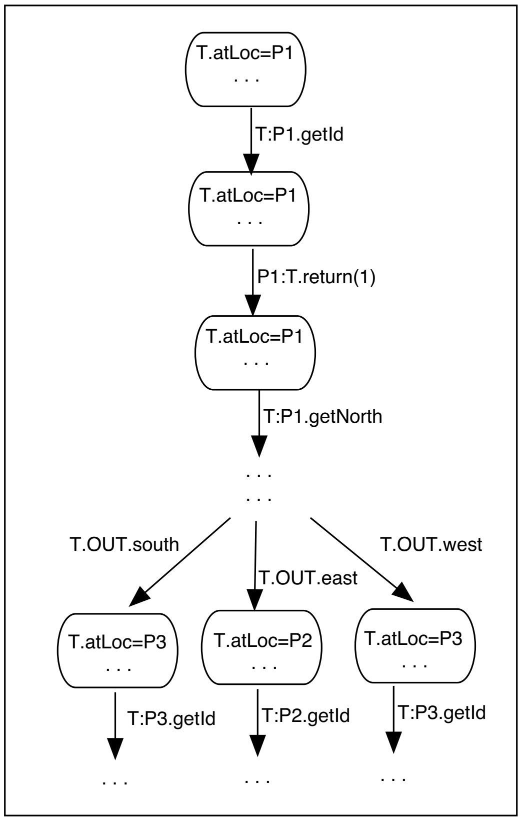

Fig. 7. The associated L^2TS

- There is a system evolution in which the traveler finally reaches place P4: $EF(T.atLoc = P4)$.

- There is a system evolution in which the traveller never reaches place P4: $EG(T.atLoc \neq P4)$.

- Starting from the initial configuration, the traveller can perform an infinite sequence of moves towards east: $\nu Y. < T : OUT.east > (Y)$.

- In no way the traveler can perform an infinite sequence of moves towards north: $AG\neg(\nu Y. < T : OUT.north > (Y))$.

– It is not possible to reach place P4 unless we pass from place P2 or P3: $\neg E[\neg(T.atLoc = P2) \wedge \neg(T.atLoc = P3) true U (T.atLoc = P4)]$

We may check that all the above properties are true on our system.

7 Conclusions

The need to define state/event-based logics relies on the fact that for the verification of concurrent software quite often it is necessary to specify both state information and the evolution in time by events/actions and hence semantic models should take both the views in consideration. Doubly Labelled Transition Systems are one of these semantics models. UML is a graphical modelling language for object-oriented software and systems may be designed for visualizing, specifying, building and documenting several aspects - or views- of them. The UML semantics [23] associates to each active object a state machine, and the possible system behaviours are defined by the possible evolutions of these communicating state machines. All the possible system evolutions can be formally represented as a Doubly Labelled Transition Systems in which the states represent the various system configurations and the edges the possible evolutions of a system configuration. To express properties on the dynamic behaviour of systems described as UML statecharts we have defined a state/event-based temporal logic $\mu UCTL$. The fact of being able to state structural properties of system configurations (state attributes and predicates) and not just events, opens the door to the modelling and verification of several structural properties of parallel systems, like topologic issues, state invariants, and mobility issues. We have hence shown that this logic can be also employed to deal with an extension of core UML state machines by primitives that designate the location of objects and their moves within a network and we have applied to them the "on the fly" model checker developed for $\mu UCTL$.

Acknowledgments We thank Alessandro Fantechi for his suggestions.

References

1. L. Andrade et al., AGILE: Software architecture for mobility, Recent Trends in Algebraic Develeopment Techniques–16th International Workshop, WADT 2002 LNCS 2755. Springer, 2003.
2. H. Baumeister, N.Koch, P, Kosiuczenko, P. Stevens, and M. Wirsing. UML for global computing. In Global Computing. Programming Environments, Languages, Security, and Analysis of Systems, LNCS 2874, Springer-Verlag, 2003.
3. M. von der Beeck, Formalization of UML-Statecharts, UML 2001 Confrence, LNCS 2185, Springer-Verlag, pp. 406-421, 2001.
4. M. von der Beeck, A structured operational semantics for UML-statecharts, Software and Systems Modeling, **Vol. 1, No. 2** Springer-Verlag, pp. 130-141, 2002.

5. Ben–Ari, M., Pnueli, A., Manna, Z. (1983). The Temporal Logic of Branching Time. *Acta Informatica* **20**, pp. 207 – 226.
6. A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu, Symbolic Model Checking without BDDs, TACAS'99, LNCS 1579, Springer-Verlag 1999.
7. S. Chaki, E. Clarke, J. Ouaknine, N. Sharygina, and N. Sinha. State/event-based software model checking. In Proc. of IFM. LNCS 2999, 2004.
8. R. Cleaveland, Tableu -based Model checking in Propositional μ-calculus, *Acta Informatica* **27**, 725-747.
9. E.M. Clarke and E.A. Emerson. Synthesis of synchronization skeletons for branching time temporal logic. In Logic of Programs: Workshop, Yorktown Heights, NY, May 1981 LNCS131, Springer-Verlag. 1981
10. Clarke, E.M., Emerson, E.A., Sistla, A.P. Automatic Verification of Finite State Concurrent Systems using Temporal Logic Specifications. *ACM Toplas*, **Vol. 8 No.. 2**, pp. 244–263, 1986.
11. R. De Nicola, F. W. Vaandrager. Three logics for branching bisimulation Journal of ACM. Vol.42 No.2,pp458-487,ACM, 1995.
12. E. A. Emerson and C. Lei. Efficient Model Checking in Fragments of the Propositional Mu-Calculus (Extended Abstract). LICS 1986: Pages 267-278
Using on-the-fly verification techniques Proceedings of Conference on Computer-Aided Verification (CAV '96),
F.Mazzanti, On the fly Verification of Networks Conference on Parallel and Distributed Processing
13. S.Gnesi and F.Mazzanti, On the fly model checking of communicating UML State Machines, Second ACIS International Conference on Software Engineering Research, Management and Applications (SERA2004), pp. 331–338, 2004.
14. M. Huth, R. Jagadeesan, and D. Schimidt. Modal transition systems: A foundation for threevalued program analysis. In LNCS, volume 2028, page 155. Springer, 2001.
15. E. Kindler and T. Vesper. ESTL: A temporal logic for events and states. Lecture Notes in Computer Science, 1420:365383, 1998.
16. A. Knapp, S. Merz, M. Wirsing, On Refinement of Mobile UML State Machines, Proc. AMAST 2004, LNCS volume 3116 LNCS,Springer - Verlag, July 2004.
17. D.Kozen, Results on the propositional $\mu-calculus$, Theoretical Computer Science, **27**, pp 333-354, 1983.
18. I. Jacobson, , G. Booch, J. Rumbaugh, The Unified Modeling Language Reference Manual. Addison-Wesley, 1999.
19. L. Lamport, "Sometime" is Sometimes "Not Never" - On the Temporal Logic of Programs. Seventh Annual ACM Symposium on Principles of Programming Languages POPL, 174-185, 1980.
20. K. G. Larsen, Proof Systems for Satisfiability in Hennessy-Milner Logic with Recursion, Theoretical Computer Science **72** 2, pp. 265-288, 1990.
21. D. Latella, I. Majzik, and M. Massink. Automatic verification of a behavioural subset of UML statechart diagrams using the SPIN model-checker. Formal Aspects of Computing. The International Journal of Formal Methods. Springer, **711(6)**, pp. 637–664, 1999.
22. R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.
23. Object Management Group. Unified Modeling Language Specification, Version 1.5. Speci- fication, OMG, 2003.
24. R. Wieringa and J. Broersen. A minimal transition system semantics for lightweight class and behavioral diagrams. ICSE'98 Workshop on Precise Semantics for Software Modeling Techniques, 1998.