A MODEL CHECKING VERIFICATION ENVIRONMENT FOR UML STATECHARTS [*]

Stefania Gnesi and Franco Mazzanti
Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo"
ISTI - CNR, Pisa
[stefania.gnesi, franco.mazzanti]@isti.cnr.it

In this paper we present the state/event-based temporal logic $\mu UCTL$ which is a logic oriented towards a natural description of dynamic properties of UML models. This logic allows to specify the basic properties that a runtime system configuration should satisfy and to combine these basic predicates with logic and temporal operators which allow to take into consideration also the events performed by the system when evolving from one system configuration to another. Doubly Labelled Transition Systems are the semantic domain for $\mu UCTL$. The logic is supported by a prototypical verification environment under development at ISTI built around the "on the fly" UMC model checker.

1. Introduction

Most of the specification languages used to describe the dynamic behaviour of systems are either state-based or event-based. In the first case the description of a system is centered around the internal properties of the states; in the latter the description of system is centered around the events performed while moving from one state to the next. Indeed both paradigms are important for the specification of real systems and hence specification languages should cover both.
The Unified Modeling Language (UML) is a graphical modeling language for object-oriented software and systems [25]; 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 modeling.
The UML definition (as rigorously, but informally described in [25]) allows to associate a statechart diagram to each object composing the system, and defines the semantics of a statechart in terms of a state machine. All the possible system behaviours can be described as the possible evolutions of a set of communicating state machines. These system evolutions can be formally represented as a Doubly Labelled Transition Systems [6] in which the states represent the various system configurations and the edges the possible evolutions of a system configuration.
After a system has been modeled 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

[*]Work partially founded by FET-IST 016004 project *SENSORIA*.

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 if we want to take advantage of all the opportunities given by this description language.
In this paper we present the state/event-based temporal logic $\mu UCTL$ which is a logic oriented towards a natural description of dynamic properties of UML models. This logic allows to specify the basic properties that a runtime system configuration should satisfy, and to combine these basic predicates with logic and temporal operators which allow to take into consideration also the events performed by the system when evolving from one system configuration to another.
Experimentation with the $\mu UCTL$ logic and UML models is being carried out at ISTI in the context of the SENSORIA project, and a prototypical verification enviroment for $\mu UCTL$ and UML (centered around the UMC [30, 9] on-the-fly model-checker) is under development.
The paper is organized as follows. In Section 2 some preliminary definitions are given. The interpretation 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 modeled as UML statecharts is shown. We conclude the paper with a comparison with some related works, and with some conclusions.

2. Preliminaries

We give here a slightly different definition of Labelled Transition Systems [23] and of Doubly Labelled Transition Systems [6]. The latter will be used as interpretation domain for 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 set of observable events. e ranges over Act*
- *Act^* is the set of transition labels. α ranges over Act^*.*
  α denotes either a sequence $es = e_1; \ldots; e_n$ 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 labelled by sequences of events (possibly of length 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 relations labelled 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*

- *$\mathcal{L}$ is a labelling function $\mathcal{L} : Q \longrightarrow 2^{AP}$ such that $\mathcal{L}(q)$ is the subset of all atomic propositions in AP that are true in q.*

Atomic propositions will typically have the form of expressions like $VAR = value$, where VAR is a variable over a set of variables $\mathcal{V}$ and $value$ is a valuation of the variable over a set $\mathcal{D}$. A proposition $VAR = value$ (representing a structural property of a system configuration, like the fact that a certain attribute of an object has a certain value) will be true in a state q if the valuation of VAR in q is equal to $value$. 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 and 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 currently active sub states (w.r.t. a corresponding statechart diagram). In fact, according to UML semantics [25] the behaviour of an object is modeled 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 suspension over a synchronous operation.
We refer to [25] for the precise definition of state machines and run-to-completion steps. Here we only suggest a possible formalization of a UML system as a L^2TS, in which the states represent the collection of state machine configurations, and the transitions represent the possible state machine evolutions. The set of atomic propositions of the model will represent the set of structural properties of a system configuration which we are interested to observe (e.g. the values of the object attributes). The set of observable events of ther model represent what we are interested to observe about the activities being executed during the evolution from one system configuration to the next (e.g. which object is currently evolving, which signals are being sent). It is definitely outside the purpose of this paper to give a formal semantics of UML statecharts in terms of L^2TS . The idea we want to promote here, is that L^2TS are an intuitive and natural structure for formally reasoning on the evolutions of a UML system.
Several approaches have been proposed in the literature for the definition of a formal semantics of UML Statechart Diagrams, e.g. [1, 19, 27, 28, 29]. All these approaches either use Kripke structures or respectively labelled transition systems as a semantic model for the description of the dynamic behaviour of a UML system. We believe instead, that doubly labelled transitions system are far more intuitive, flexible and expressive structures for this purpose.
Actually, we are interested in modeling a system as a collection of evolving and communicating objects. UML does not specify the overall behaviour of a system composed by

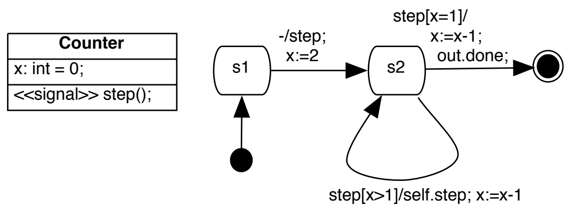

Figure 1: A simple statechart diagram

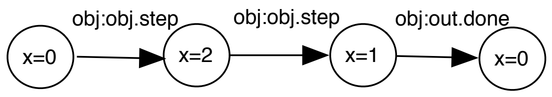

Figure 2: The L^2TS associated to the statechart diagram of Fig. 1

a set of communicating state machines evolving in parallel; in particular UML intentionally does not specify the reliability and delay of communications between state machines, nor 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[1] (i.e. state machines are supposed to evolve in interleaving). For example under these assumptions, if we consider a system composed by two equal objects, obj1 and obj2, as described by the statechart diagram of Fig. 1, the associated L^2TS is shown in Fig. 3.

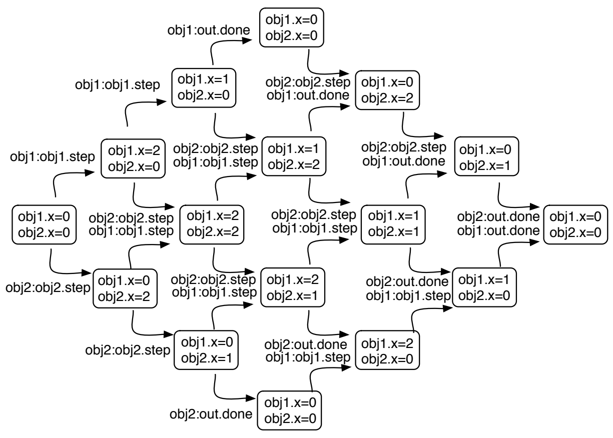

Figure 3: The L^2TS associated to the composition of $obj1$ and $obj2$

Observable events have here 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). Atomic propositions on states are instead expressions like $obj.attr = value$. We suppoose that the user is given the possibility to specify in detail which events and/or which object attributes should be "observed" from the point of view of the model semantics.

[1]A statechart transition might involve a sequence of evolution steps if it include synchronous calls to other objects.

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 [6] and the branching time state-based logic CTL [5].
We will then show in the next section that $\mu UCTL$ is suitable to express the behavioural properties of systems modeled as mobile UML communicating state machines.
Before defining the syntax of $\mu UCTL$ we introduce the two auxiliary logics of events and of state predicates.

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

$$\chi \quad ::= \quad tt \,|\, e \,|\, \tau \,|\, \neg\chi \,|\, \chi \wedge \chi$$

Definition 4 (Event formulae semantics) *The satisfaction relation $\models$ for event formulae $(\alpha \models \chi$ and $\alpha \in Act^*)$ 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* $\alpha = \varepsilon$
- $\alpha \models \neg\chi$ *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 $\neg(\neg\chi \wedge \neg\chi')$.

Definition 5 (State predicates) *The language $\mathcal{SP}$ of the state predicates $(p \in SP)$ is defined as follows:*

$$\mathcal{SP} \quad ::= \quad VAR_1\, relop\, VAR_2 \,|\, VAR\, relop\, value$$

where $relop ::= \ =, \neq, <, >, \leq, \geq$, $VAR_i \in \mathcal{V}$ and $value \in \mathcal{D}$.

Definition 6 (State predicates semantics) *The satisfaction relation $\models$ for state predicates $(\mathcal{L}(q) \models p, p \in \mathcal{SP})$ is defined as follows:*

- $\mathcal{L}(q) \models VAR\ relop\ value$ *iff* $(VAR = value') \in \mathcal{L}(q)$ *and* $value'\ relop\ value$
- $\mathcal{L}(q) \models VAR_1\, relop\, VAR_2$ *iff* $(VAR_1 = value_1) \in \mathcal{L}(q)$ *and* $(VAR_2 = value_2) \in \mathcal{L}(q)$ *and* $value_1\, relop\, value_2$

Definition 7 ($\mu UCTL$ syntax)

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

where p ranges a set of state predicates, Y ranges over a set of variables, *state formulae* are ranged over by ϕ, $<\chi>$ is the *strong next* operator.
The formal semantic of $\mu UCTL$ is given over Doubly Labelled Transition Systems. A formula is true on an L^2TS, if it holds on its initial state, q_0. We hence say that the basic predicate p is true if and only if it belongs to the predicates which are true in the current state. The formula $<\chi> \phi$ 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, providing that all occurrences of Y within ϕ fall under an even number of negations in ϕ .

Definition 8 ($\mu UCTL$ semantics) *Let $\mathcal{M} = (Q, q_0, Act^*, R, \mathcal{L})$ a Doubly Labelled Transition System the satisfaction relation for a $\mu UCTL$ formula by a state q, notation $q \models \phi$, is defined inductively in the following way:*

- *$q \models p$ if and only if $\mathcal{L}(q) \models p$;*
- *$q \models true$ 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 $[\chi]\phi$ stands for $\neg <\chi> \neg\phi$, $EF\phi$ stands for $\mu Y.(\phi \vee <tt> Y)$. EF 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 $\neg EF \neg\phi$. $AG\phi$ holds if and only if the formula ϕ holds in all the configurations reachable from the current state.
Also the weak box and diamond operators and the indexed Until modalitities of the ACTL [6] logic may be derived. In particular:

$$\begin{aligned}
[[\chi']] &= \nu Y.(([\chi']true) \vee [\tau]Y) \\
<<\chi'>> &= \mu Y.((<\chi'> true) \vee <\tau> Y) \\
E[\phi_\chi U \phi'] &= \mu Y.(\phi' \vee (\phi) \wedge <\chi \vee \tau> Y)). \\
E[\phi_\chi U_{\chi'} \phi'] &= \mu Y.(\phi \wedge ((<\chi'> \phi') \vee <\chi \vee \tau> Y)))
\end{aligned}$$

The meaning of the existential indexed until modality, $E[\phi_\chi U_{\chi'} \phi']$, is that there exists a path on the model such that any state of it is reached with an action in $\chi \vee \{\tau\}$ and all the states satisfy the formula ϕ until a state is reached with an action satisfying χ' and the formula ϕ' holds in that state.
$E[\phi_\chi U \phi']$ means instead that there exists a path on the L^2TS such that any state of it is reached with an action in $\chi \vee \{\tau\}$ and all the states satisfy the formula ϕ until a state is reached where the formula ϕ' holds in that state.
Note that $\mu UCTL$ has the same expressive power of the propositional μ-calculus [16] 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 CTL[5], and (weak) action based properties, expressible instead in the Hennesy - Milner logic plus recursion [18] and ACTL[6].
As it is well known, logics such as $\mu UCTL$ include both linear time (i.e. LTL [17]) and branching time logics (i.e. CTL, CTL^* [5], ACTL, $ACTL^*$ [6]) 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.

5. Model checking UML State Machines

We have developed a prototypical ”on the fly” model checking tool for $\mu UCTL$, called UMC [30, 9]. UMC takes as input a set of statechart diagram descriptions (describing the dynamic behaviour of the classes of the system), an object diagram describing the initial system configuration, a set of ”observation criteria” stating the object attributes and the communication event we are interested to observe, and a $\mu UCTL$ formula representing the property we want to verify. The formula is then evaluated ”on the fly”, incrementally generating the L^2TS as required by the evaluation algorithm. In this way (depending on the formula and on the L^2TS structure), only a fragment of the overall state space might be needed to be generated and analysed to be able to produce the correct result.
For example, for the evaluation of the formula $< act > SUBFORMULA$, we observe the possible system evolutions from the initial state and check whether any evolution satisfying act exist. If no such evolution exists the formula is discharged as $false$, otherwise for all (and only) existing evolutions satisfying act we recursively check the validity of $SUBFORMULA$ in the reached system configuration. The above description is just a major simplification of the actual algorithm with the only purpose of giving a flavour of the ”on the fly” approach. It is outside the purpose of this paper to give an exhaustive description of the UMC on-the-fly evaluation algorithm which, in any case, is in the style of [31].
Model checking procedures like the above are also called local [4] in contrast with those called global [5] where the whole state space is explored to check the validity of a formula. 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.
Indeed, since a naive ‘’depth first” evaluation algorithm in the case of infinite state machines might fail to find the correct result, UMC adopts a ‘’bounded” model checking approach [2]; i.e. the evaluation is started assuming a certain value as maximum 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.
In the following we briefly summarize the assumptions and limitations under which UMC currently works.

——————— **At system level** ———————

- A system is constituted by fixed static set of objects (no dynamic active object creation).
- A possible object evolution becomes a possible system evolution. (i.e. the parallel composition of objects is modeled through interleaving).
- The communications (exchange of signals and call operations) between objects of the system occur without delays, loss, or re-orderings.
- Communication is direct and one-to-one (no broadcasts).
- The events queues associated to objects are simple FIFO queues.

———————— **At the level of class statechart** ———————-

- The standard UML transition priority mechanism is taken into account.
- The standard UML ”run to completion step” semantics is considered.
- The only supported types for the class attributes and event parameters are boolean, integer and object.
- The only actions which can appear in state transitions are assignments (to local variables), sending of signals, synchronous (function) call operations (and return stmts).
- Parameters of signals and operation calls have only ”in” mode.
- Composite and parallel states are supported.
- History / Deep History, and Synch states are not supported.
- Fork, Join transitions and completion transitions are supported.
- Dynamic choice / Static choice composite transitions are not supported.
- Deferred events are supported.
- Change events, Time events are not supported.
- Statechart refinements, statechart inheritance and sub charts are not supported.
- State Enter/ Exit / Do activities are not supported.

None of the above limitations is *intrinsic* to the tool, and future versions of the prototype are likely to overcome them.
If we consider again the system composed by two equal objects, obj1 and obj2, described in Fig. 1 we may check on it properties such as:

- $< obj1 : obj1.step > true$
  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.
- $AG((< obj1 : obj1.step > true) \rightarrow (obj1.x = 0))$
  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. Related Works

State/event-based logics have been recently described by some authors with different purposes than our. In [14] a state/event-based logic for Petri Nets has been introduced. In [13] 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 [3]. As far as we now, UMC is the only ”on the fly” tool supporting model checking of a state/event based temporal logic with fixed point operators.
Linear time model checking of UML Statechart Diagrams is addressed in [19], [7] and [24]. In [8] a simple (branching time) model-checking approach to the formal verification of UML Statechart Diagrams was presented exploiting the ”classical” model checking facilities provided by the AMC model checker available in JACK. We are currently aware of

three available tools for model checking UML systems described as sets of communicating state machines. HUGO [21],[26] and vUML[20] take the approach of translating the model into the Promela language using SPIN [10] as the underlying verification engine. We have not had direct experience with these tools, but clearly in this case the properties to be verified need to be mapped into LTL logic. While vUML is restricted to deadlock checking, HUGO is mainly intended to verify whether certain specified collaborations are indeed feasible for a set of UML state machines. In both cases, the UML coverage of the tools is wider than ours because it includes history states, and internal state activities. A timed version of HUGO (called HUGO/RT [12]) has also been developed, which maps into the UPPAAL verification engine, instead than into SPIN. A third interesting approach is that one adopted in the ongoing UMLAUT [11] project. In this case a UML execution engine has been developed, adopting the Open Caesar standard interface of the CADP environment. In this way all the CADP verifications tools (including the ”on the fly” Evaluator tool [22]) can be applied also to this new engine.

7. Conclusions

To naturally express properties on the dynamic behaviour of systems described by UML statecharts we have defined a state/event-based temporal logic $\mu UCTL$ whose semantic domains are Doubly Labelled Transition Systems. Using this logic we are able to directly model and verify both the dynamic structural properties of UML system configurations and the observable events generated by system during the UML evolutions steps. A protypical ”on the fly” model checker for $\mu UCTL$, UMC is under development at ISTI. The current state of this model checker (now at version 3.1) is still that of a running prototype, useful for investigating and experiencing possible improvements of the state of art, and definitely usable for didactic purposes, while performance aspects and capability of handling extremely large systems are two aspects to which has not yet been devoted the needed attention as a widely usable tool would need.

Acknowledgments We thank Alessandro Fantechi for his helpful suggestions.

8. References

[1] 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.

[2] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu, *Symbolic Model Checking without BDDs*, TACAS’99, LNCS 1579, Springer-Verlag 1999.

[3] S. Chaki, E. Clarke, J. Ouaknine, N. Sharygina, and N. Sinha. *State/event-based software model checking*, In Proc. of IFM LNCS 2999, 2004.

[4] R. Cleaveland, *Tableu -based Model checking in Propositional μ-calculus*, Acta Informatica 27, 725-747.

[5] 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.

[6] R. De Nicola, F. W. Vaandrager. *Three logics for branching bisimulation*, Journal of ACM. Vol.42 No.2,pp458-487,ACM, 1995.

[7] Gallardo, M. M., Merino P. and Pimentel E. *Debugging UML Designs with Model Checking*, In Journal of Object Technology, vol. 1, no.2, 2002, pages 101-117.

[8] S. Gnesi, D. Latella, and M. Massink. *Model checking UML statechart diagrams using JACK*, In A. Williams, editor, Fourth IEEE International High-Assurance Systems Engineering Symposium, pages 46–55. IEEE Computer Society Press, 1999.

[9] 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.

[10] G.J. Holzmann, *The SPIN Model Checer*, IEEE TSE 23 (1997), pp. 279-295.

[11] W.M. Ho and Le Guennec A. Pennaneach F. Jezequel, J. M. *Umlaut: an extendible UML transformation framework*, INRIA-RR-3775, 1999.

[12] A. Knapp, S. Merz, and C. Rauh, *Model Checking Timed UML State Machines and Collaborations*, FTRTFT 2002: 7th International Symposium on Formal Tech-niques in Real-Time and Fault Tolerant Systems. Springer LNCS, 2002.

[13] M. Huth, R. Jagadeesan, and D. Schimidt. *Modal transition systems: A foundation for three valued program analysis*, In LNCS, volume 2028, page 155. Springer, 2001.

[14] E. Kindler and T. Vesper. *ESTL: A temporal logic for events and states*, Lecture Notes in Computer Science, 1420:365-383, 1998.

[15] A. Knapp, S. Merz, M. Wirsing, *On Refinement of Mobile UML State Machines*, Proc. AMAST 2004, LNCS volume 3116 LNCS,Springer - Verlag, July 2004.

[16] D.Kozen, *Results on the propositional* $\mu - calculus$, Theoretical Computer Science, **27**, pp 333-354, 1983.

[17] 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.

[18] K. G. Larsen, *Proof Systems for Satisfiability in Hennessy-Milner Logic with Recursion*, Theoretical Computer Science **72** 2, pp. 265-288, 1990.

[19] 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.

[20] J. Lilus, I. Porres Paltor, *vUML: a Tool for Verifying UML Models*, 14th IEEE International Conference on Automated Software Engineering, (ASE’99), pp.255-258 1999.

[21] *Project Hugo*, http://www.pst.informatik.uni-muenchen.de/projekte/hugo/.

[22] R. Mateescu, M. Sighireanu: *Efficient On-the-fly Model-Checking for regular Al-ternation-Free m-Calculus*, Science of Computer Programming 46(3) 2003.

[23] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[24] Mikk E., Lakhnech Y., Siegel M. and Holzmann G. J., *Implementing Stateacharts in Promela/SPIN*, Proc. of WIFT’98, 1998.

[25] Object Management Group. *UML 2.0 Superstructure Specification*, ptc/03-08-02, OMG, 2003.

[26] T. Schaefer, A. Knapp and S. Merz, *Model Checking UML State Machines and Collaborations*, Software Model Checking Workshop, 55(3) Electronic Notes in Theoretical Computer Science, Paris 2001.

[27] 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.

[28] W.Damm, B Josko, A. Pnueli, and A. Votintseva, *Understanding UML: A Formal Semantics of Concurrency and Communication in Real-Time UML Formal Methods for Components and Objects: First International Symposium, FMCO 2002, Leiden, The Netherlands, November 5-8, 2002, LNCS, Vol. 2852, 2003.*

[29] G.Reggio, M.Cerioli, and E.Astesiano Towards a Rigorous Semantics of UML Supporting Its Multiview Approach in Proceedings of 4th International Conference on Fundamental Approaches to Software Engineering (FASE 2001), Genova, Italy 2001, LNCS 2029

[30] F.Mazzanti UMC User Guide, Version 2.5, ISTI Technical Report 2003-TR-22 , September 2003 http://fmt.isti.cnr.it/ mazzanti/publications/UMC-UGV25.pdf

[31] C. Stirling , Walker, Local model checking in the modal mu-calculus, TCS Vol. 89, 1991