$\mu ACTL^+$: A temporal logic for UML statecharts diagrams

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

Istituto di Scienza e Tecnologie dell'Informazione - ISTI - C.N.R., Pisa

1 Introduction

We present here the use UML statecharts for the design and the specification of the dynamic behavior of the airport system. A statechart diagram is defined for each class of the model, providing a complete operational description of the behavior of all the objects of the class. The full system is then represented by a set of class objects. The UML semantics [**?**,1,9, 11] 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 [3] in which the states represent the variuos system configurations and the edges the possible evolutions of a system configuration. The topology of the system is 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 (the <<move>> operations).

The verification of the system is done with a prototypal "on-the-fly" model checker (UMC) (cf. [7]) for UML statecharts. On-the-fly verification means intuitively that, when the model checker has to verify the validity of a certain temporal logic formula on one state, it tries to give an answer by observing the internal state properties (e.g. the values of its attributes) and than by checking recursively the validity of the necessary subformulas on the necessary next states. 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 (cf. [2, 4]). The logic supported by UMC, $\mu ACTL^+$ (cf. [7]) is an extension of the temporal logic $\mu ACTL$, (cf. [5]) which has the full power of μ-calculus (cf. [8]). 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.

2 $\mu ACTL^+$

In this Section we describe the $\mu ACTL^+$ logic. Before to introduce it we give a slightly different definition of Labelled Transition Systems [10], and of Doubly Labelled Transition Systems [3]. The latter will be used as semantic models for $\mu ACTL^+$ formulae.

Definition 1 (Labelled Transition System) *A Labelled Transition System (LTS in short) is a 4-tuple* $(Q, q_0, Act^* \cup \{\texttt{tau}\}, R)$*, where:*

- Q *is a set of states;*
- q_0 *is the initial state;*
- Act *is a finite set of observable events.* `tau` *is the unobservable event.* e *ranges over* Act*, observable events have the form "*$target.event(args)$*";*
- Act^* *is a set of finite sequences of observable events.* es *ranges over* Act^* *and* es *is equal to* $e_1; \ldots; e_n$*;*
- $Act^* \cup \texttt{tau}$ *is the set of evolution labels.* α *ranges over* $Act^* \cup \texttt{tau}$*.* α *denotes or a sequence* es *of observable events or* `tau`*;*
- $R \subseteq Q \times (Act^* \cup \{\texttt{tau}\}) \times Q$ *is the transition relation. Whenever* $(q, \alpha, q') \in R$ *we will write* $q \xrightarrow{\alpha} q'$*.*

Definition 2 (Doubly Labelled Transition System) *A Doubly Labelled Transition System (*L^2TS *in short) is a 5-tuple* $(Q, q_0, Act^* \cup \{\texttt{tau}\}, R, \mathcal{L})$*, where* $(Q, q_0, Act^* \cup \{\texttt{tau}\}, R)$ *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 AP will typically have the form of VAR=value.

Before defining the syntax of $\mu ACTL^+$ we introduce an auxiliary logic of events.

Definition 3 (Evolution formulae) *Given a set of observable events* Act*, the language* $\mathcal{EF}$ *of the evolution formulae on* $Act \cup \{\texttt{tau}\}$ *is defined as follows (here square parenthesis are used to denote optional parts):*

$$\chi ::= tt \mid [target.]event[(args)] \mid \tau \mid \neg\chi \mid \chi \wedge \chi$$

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

- $\alpha \models tt$ *always*
- $\alpha \models [target.]event[(args)]$ *iff* $\alpha = e^1; \ldots; e^n$ *and exists* i *in* $1 \ldots n$ *such that* $e^i = target.event.(args)$
- $\alpha \models \tau$ *iff* $\alpha = \texttt{tau}$
- $\alpha \models\sim \chi$ *iff not* $\alpha \models \chi$
- $\alpha \models \chi \wedge \chi'$ *iff* $\alpha \models \chi$ *and* $\alpha \models \chi'$

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

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

Definition 5 ($\mu ACTL^+$ syntax)

$$\phi ::= true \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid ASSERT(VAR = value) \mid AX\chi\phi \mid EX\chi\phi \mid EF\phi] \mid EF_\chi\phi \mid \mu Y.\phi(Y) \mid Y$$

where Y ranges over a set of variables, *state formulae* are ranged over by ϕ, $EX\chi$ is the *indexed existential next* operator and EF is the *eventually* operator.

2.1 $\mu ACTL^+$ semantics

The formal semantic of $\mu ACTL^+$ is given over Doubly Labelled Transition Systems. Informally, a formula is true on an L^2TS, if the set of evolutions of the L^2TS verifies what the formula states. We hence say that the basic predicate $ASSERT(VAR = value)$is true if and only if in the current configuration (state) the attribute VAR has value equal to $value$. The formula $EX\chi\phi$ holds if there is a successor of the current configuration (state) which is reachable with an evolution satisfying χ and in which the formula ϕ holds.

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

- $q \models ASSERT(VAR = value)$ *if and only if* $(VAR = value) \in \mathcal{L}(q)$;
- $q \models tt$ *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* $q \models \phi'$;
- $q \models EX\{\chi\}\phi$ *if and only if there exists* q' *such that* $q \xrightarrow{\alpha} q'$, $q' \models \phi$ *and* $\alpha \models \chi$;
- $q \models AX\{\chi\}\phi$ *if and only if for all successor states* q_i *of* q, $q \xrightarrow{\alpha} q_i$, $q_i \models \phi$, $\alpha \models \chi$ *and* q *is not a final states;*
- $q \models EF\ \phi$ *if and only if there exist a sequence of states* $q_0, \ldots, q_n$ *and* $\alpha_1, \ldots, \alpha_n$, *with* $n \geq 0$, *such that* $q = q_0 \xrightarrow{\alpha_1} q_1 \ldots \xrightarrow{\alpha_n} q_n$ *and* $q_n \models \phi$.
- $q \models EF\{\chi\}\phi$ *if and only if there exist* $q_0, \ldots, q_n$ *with* $n \geq 0$, *such that:* $q = q_0 \xrightarrow{\alpha_1} q_1 \ldots \xrightarrow{\alpha_n} q_n$ *and* $q_n \models \phi$ *and for each* i $\alpha_i \models \chi$;
- $q \models \mu Y.\phi(Y)$ *if and only if* $q \models \bigvee_{n \geq 0} \phi^n(false)$, *where* $\phi^0(Y) = Y$ *and* $\phi^{n+1}(Y) = \phi(\phi^n(Y))$.

Several useful derived modalities can be defined, starting from the basic ones. In particular, 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. $\nu Y.\phi(Y)$ for $\neg\mu Y.\neg\phi(\neg Y)$; ν is called the maximal fixpoint operator. Note that having the fixed point operator defined in $\mu ACTL^+$ EF and EF_χ could be defined from it. We have preferred to directly introduce them to make easier the use of the logic to express properties of systems.

Following the above syntax we will write using $\mu ACTL^+$ formulae such as:

```
EX {Chart.my_event} true
```

that means: in the current configuration the system can perform an evolution in which a state machine sends the signal *myevent* to the state machine Chart. Or the formula:

```
AG ((EX {my_event}true) -> ASSERT(object.attribute=v))
```

meaning that the signal *myevent* can be sent, only when the object attribute has value v.

2.2 Model checking of $\mu ACTL^{+}$

Coming back to the airport example, let us consider an extremely simplified version of the system composed of two airports, two passengers (one at each airport), and one plane. The plane is supposed to carry exaclty one passenger and flyes (if it has passengers) between the two airports. Departing passengers try to check in at the airport and than board the plane. We contemplate only one observable action performed by the passengers during the flight, namely the consumption of a meal. The complete dynamic behaviour of the objects of classes Passenger, Airport and Plane, is shown in Fig. 1, and Fig. 2, in the form of statecharts diagrams.

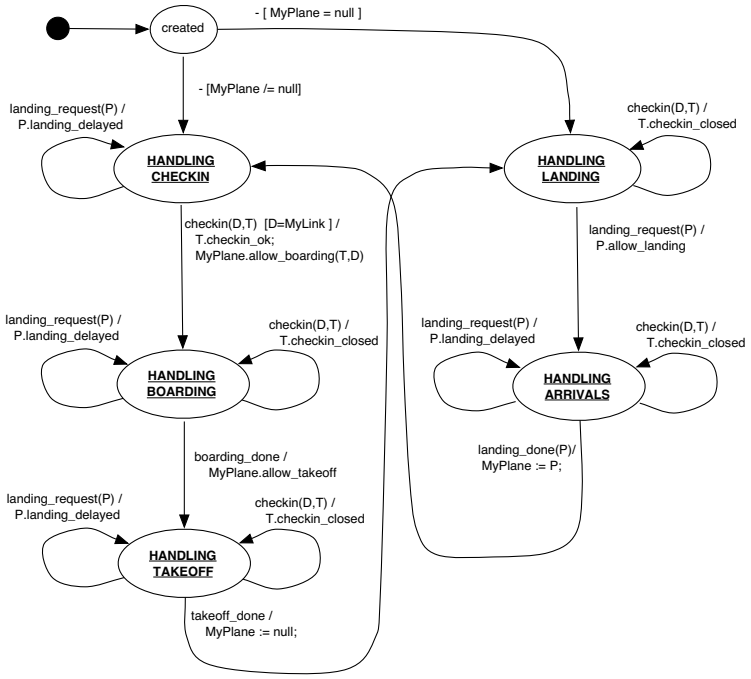

Fig. 1. Airport statemachine

The initial deployment of the system is defined by the following declarations

```
OBJECT     CLASS     INITIAL VALUE FOR ATTRIBUTES
Airport1 : Airport   (MyLink => Airport2, MyPlane => Plane1)
Airport2 : Airport   (MyLink => Airport1)
Traveler1: Passenger (atLoc => Airport1, Destination => Airport2)
Traveler2: Passenger (atLoc => Airport2, Destination => Airport1)
Plane1   : Plane     (atLoc => Airport1)
```

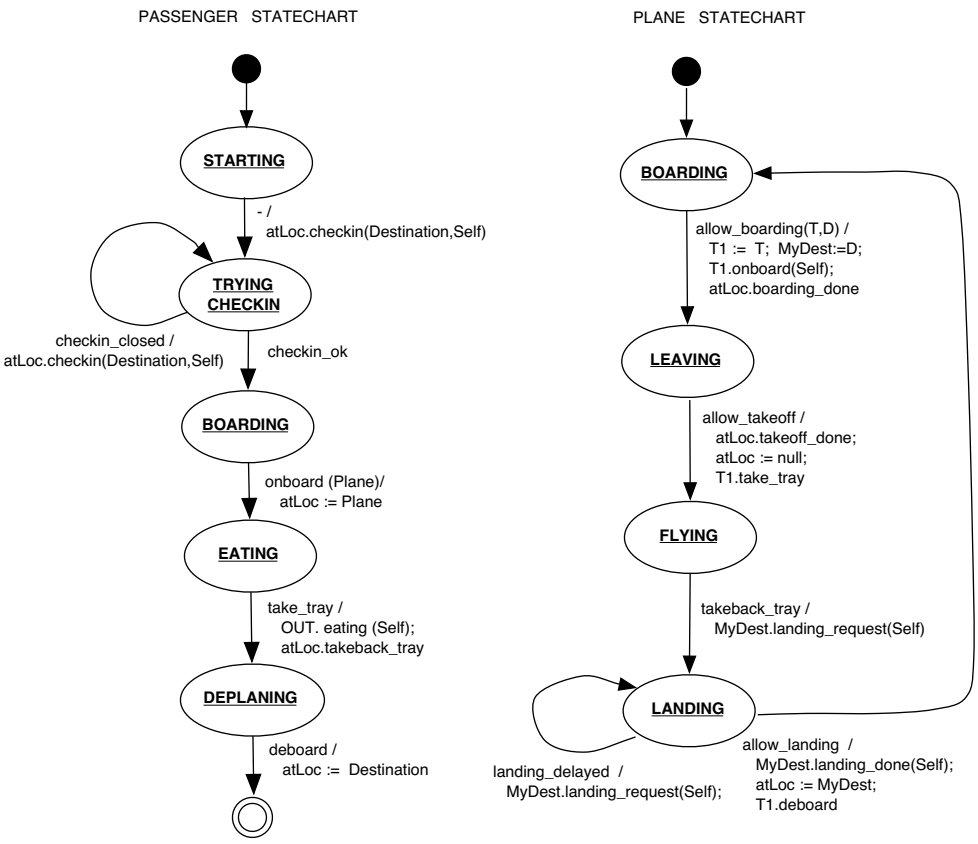

Fig. 2. Plane and passenger statemachines

An example of property which can be verified over this syestem is the following: It is always true that *Traveler1 can eat* only while he/she is flying on Plane1. This property can be written in $\mu ACTL^{+}$ as:

```
   AG ( (EX {eating(Traveler1} true) ->
( ASSERT (Traveler1.atLoc=Plane1) & ASSERT (Plane1.atLoc=null))
```

We whish to point out that that the development activity of UMC is still in progress and we have reported here some preliminary results on its aplication to the airport case study. Indeed several aspects of UML statcharts are not currently supported (e.g. the execution of "synchronous call" operarations, the use of "deferred events", the use of "history states"), and the logic itself needs to be better investigated, (e.g. its relation with localities). Work in this direction is planned in the next future.

References

1. M. von der Beeck, Formalization of UML-Statecharts, UML 2001 Confrence, LNCS 2185, Springer-Verlag, pp. 406-421, 2001.
2. G. Bhat, R. Cleaveland, O. Grumberg, Efficient on-the-fly Model checking for CTL*, in Proceedings of Symposium on Logics in Computer Science,pp. 388-397, IEEE, 1995.
3. R. De Nicola, F. W. Vaandrager. Three logics for branching bisimulation Journal of ACM. Vol.42 No.2,pp458-487,ACM, 1995.
4. J.-C. Fernandez, C. Jard, T. Jron, C. Viho, Using on-the-fly verification techniques for the generation of test suites , in Proceedings of Conference on Computer-Aided Verification (CAV '96), LNCS 1102, pp. 348-359, Springer, 1996.
5. A. Fantechi, S. Gnesi, F. Mazzanti, R.Pugliese, E. Tronci, A Symbolic Model Checker for ACTL, International Workshop on Current Trends in Applied Formal Methods, LNCS 1641, Springer - Verlag, 1999.
6. S.Gnesi and F.Mazzanti, On the fly Verification of Networks of Automata, in Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99), CSREA Press, 1999.
7. S.Gnesi and F.Mazzanti, On the fly model checking of communicating UML State Machines, Isti Tecnical Report, 2003.
8. D.Kozen, Results on the propositional $\mu - calculus$, Theoretical Computer Science, 27:333-354, 1983.
9. D. Latella, I. Majzik, and M. Massink. Towards a formal operational semantics of UML statechart diagrams, in Proceedings of IFIP TC6/WG6.1 FMOODS 99, pages 331–347. Kluwer Academic Publishers, 1999.
10. R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
11. R. Wieringa and J. Broersen. A minimal transition system semantics for lightweight class and behavioral diagrams. ICSE98 Workshop on Precise Semantics for Software Modeling Techniques, 1998