Formalisation of Quantitative UML models Using Continuous Time Markov Chains.

Nawal Addouche, Christian Antoine, Jacky Montmain

FORMALISATION OF QUANTITATIVE UML MODELS USING CONTINUOUS TIME MARKOV CHAINS

Nawal Addouche, Christian Antoine, Jacky Montmain

URC CEA/EMA-LGI2P Research Center of Ales School of Mines
Parc scientifique - Georges Besse, 30035 Nîmes, Cedex 01 France
nawal.addouche@ema, christian.antoine@ema.fr, jacky.montmain@ema.fr

Abstract: In this paper, a quantitative modelling with UML is presented and a translation from concurrent extended UML statecharts into continuous time Markov chains (CTMCs) is proposed. These are widely used in the context of performance and reliability evaluation of various systems. Our aim is to carry out a probabilistic model checking of properties related to dependability of probabilistic systems. A semantics based on stochastic clocks is used to easily translate from UML statecharts augmented with stochastic time to CTMCs. Temporal probabilistic properties related to system dependability are specified with continuous stochastic logic (CSL).

Keywords: Probabilistic systems, UML, CTMCs, Dependability properties, Probabilistic model checking.

1. INTRODUCTION

Recently, the analysis of functional system requirements in combination with quantitative aspects of system behaviour have come into focus. Several approaches have already been explored to introduce a quantitative information in the dynamic UML models. A stochastic extension of UML statechart diagrams is proposed in (Gnesi *et al.,* 2000). It is based on a set of stochastic clocks which can be used as guards for transitions. The clock value is given by a random variable with specified distribution function. A probabilistic extension of UML statecharts is presented in (Jansen *et al.*, 2002). The probabilistic UML statecharts describe probabilistic choice and nondeterministic system behaviour. Their formal semantics is given in terms of Markov decision process as defined in (Kwiatkowska, 2003). To evaluate system performance, other approaches are also proposed. Dynamic UML models are formalised with stochastic Petri nets in (King and Pooley, 1999; Merseguer and Campos, 2002) or with stochastic process algebra in (Pooley, 1999; Canevet *et al.*, 2002). In order to take into account either real-time constraints and probabilistic behaviour (undesired signals, lost signals, etc.), a profile called DAMRTS (Dependability Analysis Models for Real-Time Systems) is defined by (Addouche *et al*., 2004). A translation from UML statecharts used in this profile to probabilistic timed automata is also proposed.

In this paper, another approach intended to quantitative dependability analysis is presented. Adding to evaluate system performance using the several existing tools, the formalisation of combined collaboration statechart diagrams using continuous time Markov chains contains tree fault information. Introduction of this type of information allows to verify formally properties related to system dependability. In our works, we focus on UML statechart diagrams, which allow to describe dynamic aspects of system behaviour. In section 2, semantics of extended UML statecharts using stochastic clocks is presented. Dependability information excerpted from faults trees analysis (failures with their causes) are included in these statecharts. A set of extended UML statecharts describing the behaviour of an assembly chain is presented in section 3. In this example, the activities duration are considered as distributed exponentially. That make possible to translate UML models into continuous time Markov chains as given in section 4. To verify formally temporal probabilistic properties, we finally propose to use the probabilistic model checker Prism. In section 5, the formal model which represents the chain example is given with CTMCs. Dependability properties are specified with CSL and some results are presented before the conclusion.

2. EXTENDED UML STATECHARTS

In UML, each class of the class diagram has an optional statechart which describes the behaviour of its instances (the objects). This statechart receives events from other ones and reacts to them. The reactions can include the sending of new events to other objects and the execution of internal methods on the object. Communications between system components are generally modelled as events. In proposed UML statecharts, exchanged signals, orders

and random events (e.g. undesired and lost signals) are represented as events. Syntax of UML statecharts defined in the standard UML of the (OMG, 2003) is extended with integrating rates on transitions. A dependability information related to faults trees analysis are also introduced as defined in (Addouche *et al*., 2004). Semantics of extended statecharts is presented in section 2.2.

2.1 Syntax

This section describes informal interpretation of extended UML statecharts. The graphical representation is based on a set of nodes and a set of edges. An edge is presented by the following syntax*:*
Edge:= Event [Guard] / Action.
Event: = Event name.
Guard: = Boolean Expression.
Action: = Operation name[Rate].
Event represents either received signals or orders or random events. *Guard* is a boolean expression that represents AND-composition and OR-composition states of different objects. The compositions can be a particular qualitative information which represent the causes of undesirable events. In this case, when guard is TRUE, a mode of failure appears on the system. This type of information is available in dependability analysis based on faults trees. *Action* expresses operation execution or sending messages to other objects. The considered time on duration is stochastic and exponentially distributed.

2.2 Semantics

The semantic model used in our extended UML statecharts is inspired by stochastic automata defined in (D'Argenio *et al*., 1999). These are a variant of timed automata proposed by (Alur and Dill, 1994). Stochastic automata are defined with a set of random clocks and a clock setting function that determines for each state, called *location*, with clocks are set to which value. The transitions of stochastic automata, called *edges*, are labelled by an action and a finite set of clocks. A stochastic automaton can perform a transition from location *s* to location *s'* labelled by action *a* and clock set *C* by performing action *a* as soon as all the clocks in the set *C* have expired. A global time is assumed and all clocks are decreased at the same speed. Immediately after the transition takes place all clocks associated to *s'* by the clock setting function are randomly set according to their probability distributions.

Stochastic automata has been chosen by (D'Argenio *et al.*, 1999) to express semantics of stochastic process algebra and by (Jansen *et al*., 2003) to extend UML statecharts with randomly varying duration associated to arbitrary probability distribution (e.g. exponential, uniform, etc.). In our case, the random variables are exponentially distributed. Translation from proposed UML statecharts to continuous time Markov chains requires this choice. UML statechart diagrams are a variant of classical statecharts of (Harel, 1987). Semantics of UML statecharts is defined in (Latella *et al*., 1999). The associated extension is defined by adding to each action name, a distribution function that determines the stochastic timing of the transition. A single extended UML statechart consists of :

- A finite set of nodes with a tree structure, described by the function children_i : $Nodes_i \rightarrow \mathbf{P}(Nodes_i)$
- A finite set of events,
- A set of guard expressions. Guard expressions are boolean combinations,
- A set of actions. A set of clocks is defined on actions. We consider that actions are executed in their correspondent states,
- A E-edge, an extended edge with event e, guard g, action a and set of clocks is denoted as an arrow $\xrightarrow{e[g]/a,C}$. C is as set of stochastic clocks assigned to action a.

As in the case of stochastic automata, when entering a state, the clocks listed in the state are set to values which are determined by random variables associated to the clocks. We also assume a global time and all clocks decreased at the same time.

3. EXAMPLE : ASSEMBLY CHAIN OF MICRO-MOTORS

This example presents an automated chain for assembly of electrical micro-motors. It is excerpted from a European project named PABADIS (Plant Automation Based on DIstributed Systems) and presented in (Lüder *et al*., 2004). This one deals with flexible and reconfigurable system designed for production of different types of micro-motors. Fig 1 presents the controlled system. Micro-motors consist to stators and rotors. The firsts are transported to assembly robots, on pallets via a conveyor system and seconds are available into stocks near each robot. A set of pallets containing stators moves along the conveyor belt. These are detected by pallet sensors PSi at different levels of the conveyor system.

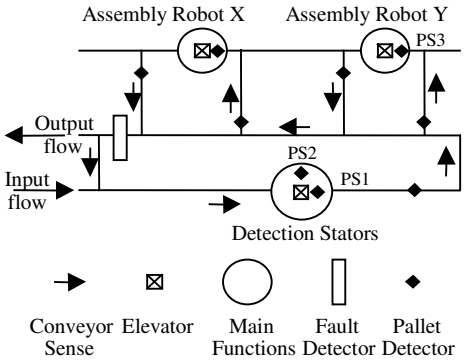

Fig. 1. Assembly chain

When assembly of micro-motors is completed, the pallets then move into a fault detection station where a camera detects the possible assembly faults. Set of PLC (Programmable Logic Controller) and PC composes the control system. Let us consider the mode of failure "*Assembly fault*". Among the causes of this mode of failure, there are undesired signals sent by the sensors, undesired orders sent by the controller and material failures of elevator.

To represent dynamic aspects of the system, extended UML statecharts and collaboration diagrams are used. Combination of these two diagrams allows to represent all system interactions. Indeed, the collaboration model describes external interactions between objects whereas UML statecharts diagrams represent how an instance of a class reacts to an event occurrence.

3.1. Collaboration diagram

Interactions between objects of classes are presented in fig 2. Exchanged messages describe signals sent from *Sensors* (S.PPi for *sensor* i) and *Fault detector* objects (S.Fault) to *controller*. They also represent orders sent from *Controller* to *Robot* and *Elevator* objects. Our example presents a distributed system such that several PLC interact to control the system functioning. To simplify, one *controller* object is presented in the collaboration diagram.

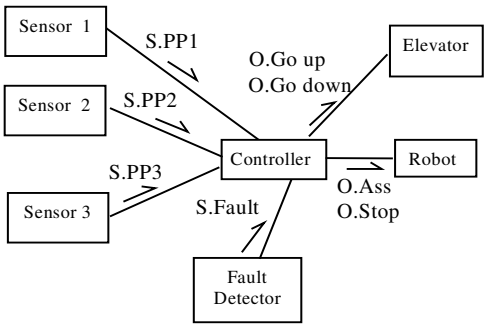

Fig. 2. Collaboration diagram

3.2 Statechart diagrams

*Robo*t and *Controller* objects behaviour are respectively modelled as given in fig 3 and 4. In *Robot* statechart, orders are modelled as events. In the example: "O.Ass[PS1.Ds OR PS2.Ds]/ Assembly()[0,03] ", guard expresses that the edge is enabled if one of sensors PS2 or PS3 is in degraded state *Ds*. The rate of executing assembly tasks represents constant of exponential distribution function associated to the clock C=EXP (0,03). The guard "S.Active AND E.Active" represents the condition to leave the state *Emergency stop*: sensors and elevator must be in the state *Active* of their respective UML statecharts. Among the malfunctions of controller, sending of undesired orders or lost orders are modelled in *Controller* statechart as a fault which arrives randomly.

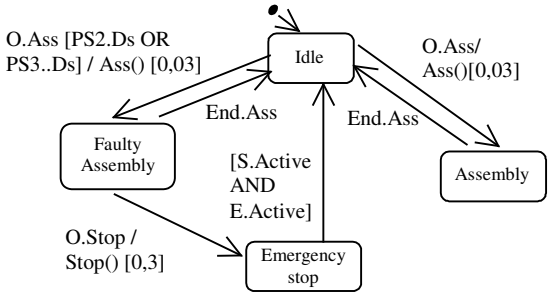

Fig. 3. Robot statechart

In the example "UO/ Send.O ()[0.001]", an undesired order (UO) can arrives randomly. After which an order with a rate of 0.001 is sent to the corresponding component of controlled system. The received sensor signals are presented as events and the sending of orders as actions. The edge "SPP1/ O.Go up()", expresses that when PS1 detects a pallet, the order *go up* is sent from controller to elevator.

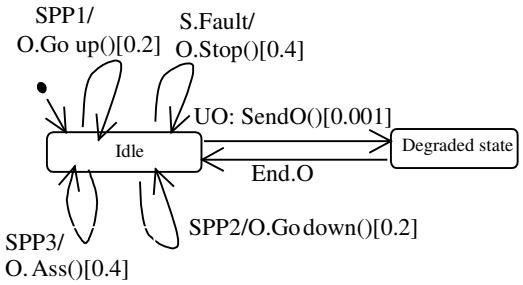

Fig. 4. Controller statechart

4. TRANSLATION OF UML MODELS WITH CTMCs

System behaviour is composed by a set of scenario. A scenario is composed by signal, order and executing tasks. It is presented with a set of combined collaboration and extended statecharts diagrams. Formalisation of this behavioural UML models is given as follow:

- Each scenario is modelled by a set of combined collaboration extended statecharts diagrams. Number of combined diagrams varies according scenario,
- At any time the combined diagram must have each of its objects in exactly one state. These combinations of active states are called "marking" and represent in a continuous time Markov chains the reachable states,
- It is assumed that the rates associated to sent messages are exponentially distributed.

Let us consider the following scenario : when a pallet arrives in the detection station and sensor PS2 send an undesired signal to controller. This make erroneous, position stator data sent to the controller.

Then a fault assembly will appear and will be detected in fault detection station.

Translation of behavioural UML models to its correspondent continuous time Markov chain is explained in fig 5 and 6 such that first one presents a scenario of assembly chain example described above with behavioural UML models and second one shows a possible translation to CTMCs.

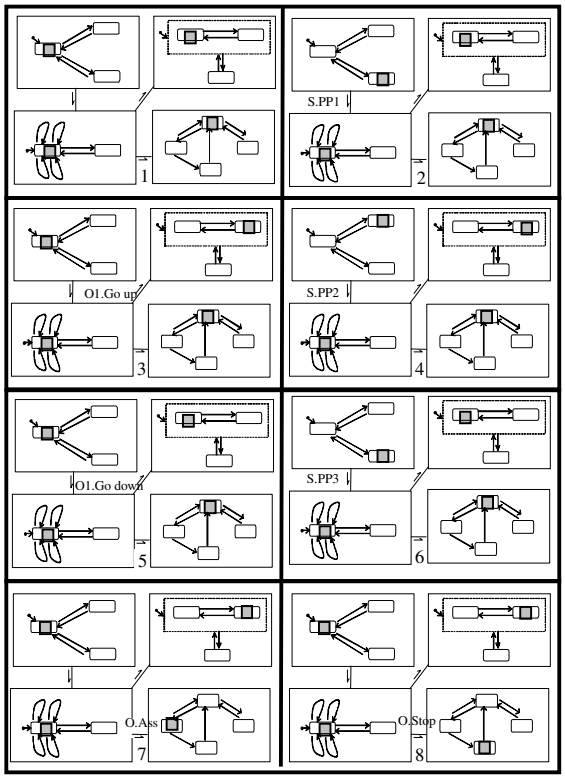

Fig. 5. Direct marking of combined collaboration statechart diagrams

4.1 Application

The fig 5 describes a faulty assembly due to sending of undesired signal by sensor 2. This scenario is described by eight combined diagrams. In each one from up to down and from right to left are respectively presented, statecharts of *sensor i*, *elevator i, controller* and *robot* objects. To simplify fig 5, *Default detector* object is deliberately omitted, one *sensor* statecharts and one elevator statecharts are shown, index (i) used on signals and orders gives which sensor or elevator is modelled in combined diagrams. Transition from a marking to another results of exchanging signals or orders between objects. Active states of each marking are grey tented in the figure.

The scenario of faulty assembly begins in marking 1 and takes end in marking 8. From marking 1 to marking 5 tasks of stators detection station (go up, go down) are executed with undesired signal sent from sensor 2 to controller. Malfunctioning of sensors 2 leads erroneous detection of stator position on pallet. Consequently, robot do not put rotors in good pallet compartments. From marking 5 to marking 8, the faulty assembly and the emergency stop are modelled.

If we assume an initial marking, such as all objects are in *Idle* state and *elevator* in *Down* state, it is possible to derive all markings by following the exchanging messages between objects. The CTMC reachable states are thus formed. Fig 6 shows the continuous time Markov chain derived from model of fig 5. The numbers on reachable states are related to numbers on markings of fig 5.

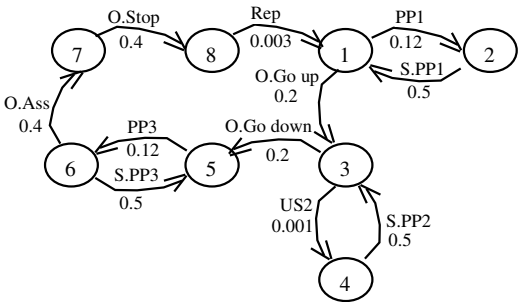

Fig. 6. Continuous time Markov chain

4.2 Principle of translation

A continuous time Markov chain is composed by set of states and set of edges subject to an exponentially distributed random delay. Each state of CTMC represents a combination of active states of extended UML statecharts. A CTMC edge is fired when an action is executed by an extended UML statechart. Rates associated to actions are directly translated as rate of firing edges. In fig 6, the rates are related either to orders sent by controller (O.Stop, O.Ass, etc.) or to signals sent by sensors (SPP1, SPP2 and SPP3). They are also related to stochastic events such as random passage of pallets (PP1 and PP3), operator intervention or repairing system (Rep). Guards are used to model AND-composition and OR-composition of degraded or normal states. Abnormal or normal functioning scenarios are obtained according to whether these guards are true or false. So, Guards are implicitly expressed in the CTMC.

4.3 Comments

The global system behaviour is obtained from the set of continuous time Markov chains related to different scenario. Combination of these models gives global continuous time Makov chains. This method represents an intuitive proposition for formalisation. It is not applied in this paper because it is not necessary to construct it while Prism tool is used for probabilistic model checking. This one is based on a modular modelling of the system. It does not require a global CTMC but rather a set of CTMCs, where each represents one “module” of the system, as given in section 5.1.

5. PROBABILISTIC MODEL CHECKING

The model checking is an algorithmic method designed to check if a system satisfies the specifications. A model checker is a software tool which introduces a system model (e.g. automata) and specifications (e.g. logic formula), as well as the return *yes* or *no* depending on whether the system satisfy or not the specifications. In a probabilistic model checking, the return *yes* or *no* are replaced by probabilities evaluations. The probabilities are introduced in two cases:

- The first case is related to probabilistic systems, i.e. which include probabilistic information such as the mean time between failures of a material device or in transmission protocols, the probability of message transmission,
- The second case concerns non-probabilistic systems whose complexity makes exhaustive verification practically impossible. So, a probabilistic description is used for unavailable information or for the information whose the too complex representation would be not exploitable.

To verify dependability properties, the model checker Prism is adopted (Kwiatkowska *et al.*, 2002). This tool is designed for analysis of probabilistic models and supports various models such as Markov decision processes, discrete time Markov chains and continuous time Markov chains. Prism is a tool developed at the University of Birmingham which supports the model checking described before. The tool takes as input a description of a system written in Prism language. It constructs the model from this description and computes the set of reachable states. It accepts specification in either the logic PCTL or CSL (Kwiatkowska, 2003) depending on the model type. It then determines which states of the system satisfy each specification.

5.1 Model analysis

In order to model check a system with Prism tool, it must be specified in the Prism language, based on the Reactive Modules formalism of (Alur and Henzinger, 1996). This formal model is designed for concurrent systems and represents synchronous and asynchronous components in a uniform framework that supports compositional and hierarchical design and verification.

The fundamental components of Prism language are *modules* and *variables*. A system is composed of a number of modules which can interact with each other. A module contains a number of *local variables*. The values of these variables at any given time constitute state of the module. Global state of the system is determined by local states of all modules.

The translation from UML statechart diagrams to reactive modules is given as follow :

- The *modules* are defined for each UML object,
- The states of extended UML statecharts are modelled by one or several local variables (integer or boolean),
- The signals, orders and random events are given by one local boolean variable that determine their presence,
- The rates associated to actions (as given in our extended statecharts) are modelled by a set of constants proposed in reactive modules to assign stochastic information to the transition,
- The guards of extended UML statecharts are expressed with constraints. These are predicates over the local variables of other modules and are proposed in Prism language in order to condition the transition firing,
- The actions of UML models are also defined as actions in reactive modules.

The behaviour model of assembly chain is proposed such that each presented object in collaboration diagram of section 3.1 is taken into account. Part of the model is presented in fig. 7.

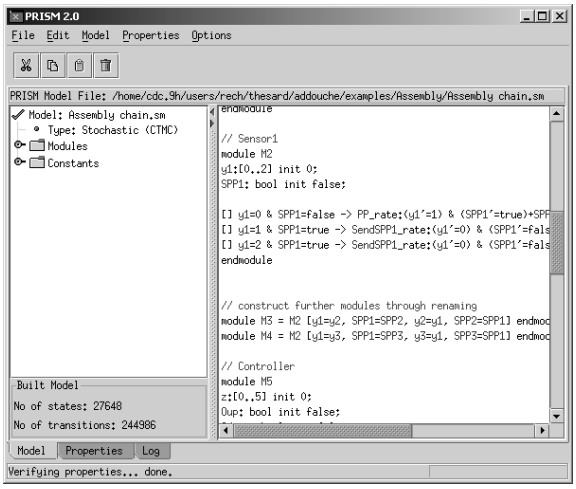

Fig. 7. Prism interface: Editing a model

5.2. Properties in Prism specification language

Some probabilistic properties related to our example are presented. Their informal specifications is given as follow:

Property 1: “In the long run, the probability the robot carries out a faulty assembly is less than 1%”.

Property 2: “In initial state, the probability that the robot remains in emergency stop until the elevator and the sensors are reactivated is at least 0,95”.

Property 3: “Elevator remains in down position less than k units of time until the sensor1 detects a pallet presence with a probability $\geq p$ ”.

These dependability requirements are formally specified with the temporal logic CSL. The following are their Prism specification language.

Property 1: S<0.1 [(r=2)]

Property 2: "init"⇒ P>0.95[(r=3) U (e=0) & (y1=0) & (y2=0) & (y3=0)]

Property 3: P=? [(e=0)$U^{\geq K}$ SPP1= true]

5.3. Experimental results

The results of our experiments are shown in fig. 8. Dependability properties 1 and 2 are verified (true). The verification of property 3 is presented with a curve. The CSL requirement are evaluated for increasing time points *k* and the boundary probabilities *p* at which the requirement turns from being true to being false are calculated. We plotted a graph generated by Prism where a pair (t, p) above a plot the requirement is FALSE, while for pairs below it is TRUE.

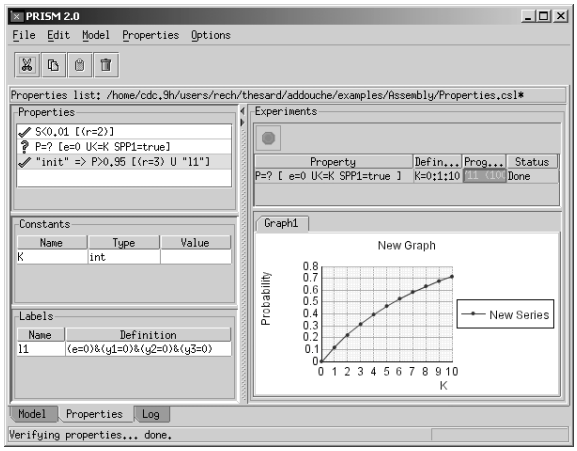

Fig. 8. Prism interface: Properties specification

6. CONCLUSION

The paper presents UML models to analyse the system dependability. The aim is to verify formally dependability properties of probabilistic systems. Extended UML statecharts are proposed with semantics related to stochastic automata. Duration of activities are expressed with stochastic clocks. The time is exponentially distributed, that make possible translation to continuous time Markov chains. As example, an assembly chain is described with extended statecharts. A translation method from combined collaboration statechart diagrams to CTMCs is proposed. Using the Prism tool, some dependability properties related to the example are specified with the temporal logic CSL and verified by probabilistic model checking.

REFERENCES

Addouche, N, C. Antoine and J. Montmain. (2004). UML Models for Dependability Analysis of Real-Time System. In: *Proc of SMC'04*, The Hague, The Netherlands.

Alur, R and D. Dill. (1994). A theory of timed automata. *Theoretical Computer Science*, **126(2)**, 183-235.

Alur, R and T. Henzinger. (1996). Reactive Modules. In: *Proc of LICS'96*, pp. 207-218, IEEE Computer Society Press, New Jersey.

Canevet, C, S. Gilmore, J. Hillston and P. Stevens. (2002). Performance modelling with UML and stochastic process algebra. In*: Proc of UKPEW'02*, pp. 16. The Univ of Glascow, UK.

D'Argenio, P.R, J-P. Katoen and E. Brinksma. (1999). Specification and Analysis of Soft Real-Time Systems: Quantity and Quality. In: *Proc. of RTSS'99*, pp.104-114. IEEE Society Press, Phoenix, Arizona, USA.

Gnesi, S, D.Latella and M. Massink. (2000). A Stochastic Extension of a Behavioural Subset of UML Statechart Diagrams. In: *Proc of HASE'00*, Albuquerque, New Mexico.

Harel, D. (1987). Statcharts: A visual formalism for complex systems. *Science of Computer Programming*. Elsevier, **8(3)**, 231-274.

Jansen, D.N, H. Hermanns and J-P Kaoten. (2002). A Probabilistic Extension of UML Statecharts: Specification and Verification. In: *Proc of FTRTFT'02*, pp. 355-374. Oldenburg, Germany.

Jansen, D.N, H. Hermanns and J-P Kaoten (2003). QoS-oriented Extension of UML Statecharts. In: *UML 2003*, (Perdita Stevens et al.), pp. 76-91., LNCS, 2863, San Fransisco, USA.

King, P and R. Pooley. (1999). Using UML to derive stochastic Petri nets models. In : *Proc of UKPEW'99*, pp. 45-56, The Univ of Bristol.

Kwiatkowska, M, G. Norman and D. Parker. (2002), "Prism: Probabilistic Model Checker", In*: Proc of TOOLS'02*, (T. Field et al), pp. 200-204, London, UK.

Kwiatkowska, M. (2003). Model Checking for Probability and Time : From Theory to Practice. In: *Proc of LICS'03*, pp. 351-360, IEEE Computer Society Press, Ottawa, Canada.

Latella, D, I. Majzik, and M. Massink. (1999), Towards a formal operational semantics of UML statechart diagrams. In: *Proc of FMOODS'99*, pp. 331-347, Florence, Italy.

Lüder, A, J. Peschke, T. Sauter, S. Deter and D, Diep. (2004), Distributed intelligence for plant automation on multi-agent systems: the PABADIS approach*, Production Planning and Control*, **15(2)**, pp. 201-212.

Merseguer, J and J. Campos. (2002). A Compositional Semantics for UML State Machines Aimed at Performance Evaluation. In: *Proc of WODES'02*, pp. 295-302, IEEE Computer Society Press, Zaragoza, Spain.

OMG (2003) Unified Modeling Language Specification. v.1.5, *OMG Document Formal /* 03-03-01.

Pooley, R. (1999). Using UML to Derive Stochastic Process Algebra Models. In: *Proc of UKPEW'99*, (J.T. Bradley and N.J. Davies), pp. 23-33. The Univ of Bristol.